

# Приветствуем в документации по консоли Windows!

Статья • 12.07.2023

В разделах в левой части этой страницы вы найдете сведения о концепциях, интерфейсах API и связанных функциях, структурах и других способах для программного управления консолью Windows и взаимодействия с ней.

# Путеводитель по экосистеме консоли и терминала Windows

Статья • 12.07.2023

В этом документе изложен общий план развития консоли Windows и Терминала Windows. В нем рассматриваются следующие аспекты:

- место консоли Windows и Терминала Windows в экосистеме приложений командной строки в Windows и других операционных системах;
- прошлый и будущий план создания этой платформы и развития продуктов, функций и стратегий в рамках этого процесса.

Основное внимание на текущем этапе развития консоли и терминала в корпорации Майкрософт уделяется созданию первоклассных функциональных возможностей терминала для разработчиков на платформе Windows, а также [поэтапному отказу от классических API-интерфейсов консоли Windows и их замене последовательностями виртуальных терминалов с помощью псевдоконсоли](#).

[Терминал Windows](#) наглядно демонстрирует переход к таким первоклассным возможностям, привлекая разработчиков к [сотрудничеству в проектах с открытым кодом](#), поддерживая всевозможные сочетания приложений для командной строки и размещения в терминале и объединение экосистемы Windows со всеми остальными платформами.

## Определения

Прежде чем продолжить, рекомендуем ознакомиться с [определениями](#) общей терминологии, используемой в этой области, в том числе с такими понятиями, как [приложения командной строки \(или консоли\)](#), [стандартные дескрипторы \(STDIN, STDOUT, STDERR\)](#), [устройства телетайпа и псевдотерминалов](#), [клиенты и серверы](#), [подсистема консоли](#), [узел консоли](#), [псевдоконсоль](#) и [терминал](#).

## Architecture

Общая архитектура системы состоит из четырех частей: клиент, устройство, сервер и терминал.

## Command-Line Communication



## Клиент

Клиент — это приложение командной строки, использующее текстовый интерфейс для ввода команд пользователем (вместо пользовательского интерфейса с управлением мышью) и возвращающее текстовое представление результата. В Windows API-интерфейс консоли обеспечивает уровень взаимодействия между клиентом и устройством. (Это также может быть стандартный обработчик консоли с API-интерфейсами для управления устройствами.)

## Устройство

Устройство выполняет функцию промежуточного уровня взаимодействия для обработки сообщений между двумя процессами: клиентом и сервером. В Windows это драйвер консоли. На других платформах это устройство телетайпа или псевдотерминала. Если вся транзакция представляет собой обычный текст или содержит [последовательности виртуальных терминалов](#), в качестве такого канала связи могут использоваться другие устройства, например файлы, каналы и сокеты. Однако с [API-интерфейсами консоли Windows](#) такой подход не работает.

## Сервер

Сервер интерпретирует запрошенные вызовы API или сообщения от клиента. В классическом режиме работы Windows сервер также создает пользовательский интерфейс для вывода выходных данных на экран. Сервер дополнительно собирает входные данные для отправки их в ответных сообщениях на клиент через драйвер, например терминал, присоединенный к тому же модулю. При использовании модуля [псевдоконсоли](#) он вместо этого выполняет только функцию ретранслятора, предоставляющего подключенному терминалу эту информацию из [последовательности виртуальных терминалов](#).

## Терминал

Терминал — это конечный уровень, который обеспечивает графическое отображение и интерактивные службы для пользователя. Он отвечает за запись

входных данных и их кодирование в виде [последовательностей виртуальных терминалов](#), которые в итоге поступают на `STDIN` клиента. Он также будет получать и декодировать *последовательности виртуальных терминалов*, поступающие от `STDOUT` клиента, для вывода их на экран.

## Дополнительные подключения

Далее можно также создать дополнительные подключения, присоединив цепочку приложений с различными ролями к одной из конечных точек. Например, сеанс SSH выполняет две роли. Он функционирует как **терминал** для приложения командной строки, выполняемого на одном устройстве, и как **клиент**, перенаправляющий все полученные сведения, на другом устройстве. Эту цепочку можно продлить на произвольное число устройств в любых контекстах, что обеспечивает гибкие возможности для реализации различных сценариев.

На платформах, отличных от Windows, роли **сервера** и **терминала** реализованы в одном блоке, так как нет необходимости в переходном уровне для обеспечения совместимости между набором API и [последовательностями виртуальных терминалов](#).

## Продукты Майкрософт

Все продукты командной строки Microsoft Windows теперь доступны на сайте GitHub в репозитории с открытым кодом [microsoft/terminal](#) ↗ .

## Узел консоли Windows

Это традиционный пользовательский интерфейс Windows для приложений командной строки. Он обрабатывает все операции обслуживания через API консоли,ываемые из любого подключенного приложения командной строки. Консоль Windows также обрабатывает представление графического пользователяского интерфейса от имени всех этих приложений. Она находится в системном каталоге в виде `conhost.exe` или `openconsole.exe` в форме программы с открытым кодом. Она входит в состав операционной системы Windows. Она также включена в другие продукты Майкрософт, созданные на основе репозитория с открытым кодом, чтобы реализовать более актуальную инфраструктуру [псевдоконсоли](#). В соответствии с приведенными выше определениями она может выполнять либо традиционную общую роль сервера и терминала, либо функционировать только в качестве сервера с использованием предпочтительной инфраструктуры [псевдоконсоли](#).

## Терминал Windows

Это новый интерфейс Windows для приложений командной строки. Терминал Windows может служить основным примером использования [псевдоконсоли](#) для разделения взаимодействия при обслуживании через API и работе с текстовым приложением по тому же принципу, что и на всех остальных платформах, отличных от Windows.

Терминал Windows — это основной текстовый пользовательский интерфейс для Windows. Он демонстрирует возможности экосистемы и способствует дальнейшему сближению Windows с другими платформами. Терминал Windows также может служить примером создания надежного и сложного современного приложения с учетом прежних наработок для всего спектра API-интерфейсов и платформ Windows. В соответствии с приведенными выше определениями этот продукт выполняет роль терминала.

## Основные исторические вехи

Основные исторические вехи развития подсистемы консоли можно разделить на две части: реализация до 2014 года и вся работа, проделанная после 2014 года, когда было определено новое направление развития командной строки в эпохе Windows 10.

### Начальная реализация

[1989–1990 гг.] Изначально система узла консоли была реализована в виде эмуляции среды DOS в операционной системе Windows. Ее код был тесно связан и объединен с [командной строкой cmd.exe](#), которая является представлением этой среды DOS. Функции и права кода системы узла консоли такие же, как у интерпретатора (оболочки) командной строки. Эта система также предоставляет базовый уровень служб для других служебных программ командной строки, которые работают аналогично CMD.

### DBCS для ККЯ

[1997–1999 гг.] В этот период была внедрена поддержка [двухбайтовой кодировки](#) (DBCS), которая обеспечила поддержку ККЯ (китайского, корейского и японского языков). Это привело к разделению методов записи и чтения в консоли на две ветви: одна использовалась для создания отдельных "западных" версий, обрабатывающих однобайтовые символы, а другая — для альтернативных

"восточных" версий, в которых для представления огромного массива символов требуется два байта. Такое разделение на ветви предусматривало расширение представления ячейки в среде консоли, ширина которой теперь могла составлять 1 или 2 ячейки. Представление с 1 ячейкой узкое (больше в высоту, чем в ширину), а с 2 ячейками — широкое (полная ширина или квадрат, в который можно вписать обычные китайские, японские и корейские иероглифы).

## Безопасность и изоляция

[2005–2009 гг.] Так как подсистема консоли работает внутри критического системного процесса `csrss.exe`, подключение разных клиентских приложений с разными уровнями доступа к одному суперкритическому и привилегированному процессу было определено как особенно опасное. На этом этапе подсистема консоли была разделена на клиент, драйвер и серверные приложения. Каждое приложение может работать в собственном контексте, что позволяет сузить круг его прав и задач. Такая изоляция повышает общую надежность системы, так как никакой сбой в подсистеме консоли больше не может повлиять на другие критически важные функции процесса.

## Улучшение взаимодействия с пользователем

[2014–2016 гг.] После длительного периода несистематического обновления подсистемы консоли всевозможными командами разработчиков была сформирована новая отдельная команда, отвечавшая за улучшения ее возможностей. В этот период были внесены следующие улучшения: выбор строки, плавное изменение размера окна, изменение размещения текста, копирование и вставка, поддержка высоких значений DPI и ориентация на Юникод, в том числе сведение отдельных алгоритмов управления хранением и потоками для "западных" и "восточных" языков в единый алгоритм.

## Клиент виртуального терминала

[2015–2017 гг.] Наряду с выходом подсистемы [Windows для Linux](#) в качестве мер корпорации Майкрософт по улучшению работы [Docker в Windows](#) и внедрению [OpenSSH](#) в качестве основной технологии удаленного выполнения в командной строке в узле консоли были представлены исходные реализации [последовательностей виртуальных терминалов](#). Это позволило имеющейся консоли выполнять функцию терминала, подключенного непосредственно к таким собственным приложениям Linux в соответствующих средах, преобразовывая

графические и текстовые атрибуты для просмотра на экране и возвращая введенные пользователем данные на требуемом "диалекте".

## Сервер виртуального терминала

[2018 г.] За последние двадцать лет сторонние производители создали альтернативы исходному узлу консоли, которые помогают разработчикам работать еще эффективнее, а также поддерживают настройку дополнительных возможностей и предлагают интерфейсы с вкладками. Однако этим приложениям по-прежнему требовалось запускать и скрывать окно узла консоли. Они присоединяются в качестве дополнительного "клиентского" приложения, извлекающего данные из буфера путем циклического опроса в процессе работы основного клиентского приложения командной строки. Их задача — выполнять функцию терминала так, как на других платформах, но уже в мире Windows, где замена терминалов не поддерживалась.

В этот период была внедрена инфраструктура [псевдоконсоли](#). Псевдоконсоль позволяет любому приложению запускать узел консоли в неинтерактивном режиме и выполнять функцию конечного терминального интерфейса для пользователя. Основным ограничением развития в этом направлении было обещание обеспечить дальнейшую совместимость Windows в отношении обслуживания всех опубликованных [API-интерфейсов консоли Windows](#) в неопределенном будущем, предоставив при этом для замены интерфейс размещения на сервере, способный обеспечить такие же возможности, как на всех других платформах. Решением этой задачи стала [последовательность виртуальных терминалов](#). По сути, это зеркальное отражение того, чтобы было сделано на этапе реализации клиента — проекты [псевдоконсоли](#), которые будут отображаться на экране как *последовательности виртуальных терминалов* для делегированного узла и интерпретировать ответы, преобразовывая их в последовательности ввода в формате Windows для использования клиентским приложением.

## План развития на будущее

### Приложения терминала

[С 2019 г. по сегодняшний день] Это эпоха открытого кода для подсистемы консоли с ориентацией на новый Терминал Windows. В рамках конференции Microsoft Build в мае 2019 г. было объявлено о выходе Терминала Windows, полностью размещенного на GitHub в репозитории [microsoft/terminal](#) ↗. Основным приоритетом этого периода будет создание приложения "Терминал Windows" на

основе оптимизированной платформы для [псевдоконсоли](#), которое обеспечит первоклассные функциональные возможности терминала для разработчиков на платформе Windows.

**Терминал Windows** не только демонстрирует возможности платформы, в том числе технологию интерфейса [WinUI](#), модель упаковки [MSIX](#) и архитектуру компонентов [C++/WinRT](#), но и позволяет выполнить ее проверку. С помощью Терминала Windows организация Windows сможет открывать и развивать платформу приложений по мере необходимости, что позволит разработчикам работать еще эффективнее. Реализация уникального набора требований для опытных пользователей и разработчиков в Терминале Windows обеспечит соответствие платформы Windows современным требованиям и ожиданиям.

Для операционной системы Windows это означает [отказ от стандартного пользовательского интерфейса узла классической консоли](#) и переход на [Терминал Windows](#), [ConPTY](#) и [последовательности виртуальных терминалов](#).

Наконец, на этом этапе предполагается поддержка всего спектра стандартных функциональных возможностей, как в Терминале Windows, так и во всех альтернативных терминалах.

## Библиотека поддержки клиентов

[Будущее] Учитывая наличие поддержки [последовательностей виртуальных терминалов](#) и соответствующей документации на стороне клиента, мы настоятельно рекомендуем разработчикам служебных программ командной строки Windows отказаться от классических API Windows в пользу последовательностей виртуальных терминалов, чтобы воспользоваться преимуществами унифицированной экосистемы при работе со всеми платформами. Однако в этой картине недостает одной существенной детали. Другие платформы имеют широкий спектр вспомогательных библиотек на стороне клиента для обработки входных данных, например [readline](#), и графического отображения, например [ncurses](#). Чтобы добавить эту недостающую часть плана будущего развития, необходимо определить, какие возможности предлагает эта экосистема и как ускорить внедрение последовательностей виртуальных терминалов в приложения командной строки Windows вместо классических API-интерфейсов консоли.

## Транзитная передача последовательности

**[Будущее]** Совместное использование реализаций клиента и сервера виртуальных терминалов обеспечивает поддержку всевозможных сочетаний приложений для командной строки и размещения в терминале. Эта комбинация может взаимодействовать как с классическими API-интерфейсами консоли Windows, так и с [последовательностями виртуальных терминалов](#), однако преобразование в совместимый классический метод Windows и обратно в более универсальный метод виртуальных терминалов сопряжено с дополнительными временными затратами.

Когда применение *последовательностей виртуальных терминалов* и UTF-8 в Windows получит широкое распространение на рынке, задание преобразования и интерпретации для узла консоли можно будет при необходимости отключить. После этого узел консоли станет обычным средством обработки вызовов API и будет ретранслировать вызовы с устройств в основное приложение через [псевдоконсоль](#). Это изменение повысит производительность и обеспечит максимальную поддержку "диалекта" последовательностей, используемого для обмена данными между клиентским приложением и терминалом. Благодаря этому изменению появится возможность реализовать дополнительные сценарии взаимодействия и (наконец) привести мир Windows в соответствие с семейством всех остальных платформ, на которых реализуются приложения командной строки.

# Определения

Статья • 23.10.2023

Этот документ содержит определения конкретных слов и фраз в этом пространстве и используется в качестве ссылки на этот набор документов.

## Приложения командной строки

Приложения командной строки или иногда называемые "консольными приложениями" и /или "клиентами" подсистемы консоли, являются программами, которые работают главным образом в потоке текстовых или символьных сведений. Как правило, они не содержат собственных элементов пользовательского интерфейса и делегируют выходные данные и отображение и роли ввода и взаимодействия в размещенное приложение. Приложения командной строки получают поток текста на стандартном дескрипторе ввода, `STDIN` который представляет ввод клавиатуры пользователя, обрабатывает эти сведения, а затем отвечает потоком текста на стандартные выходные данные `STDOUT` для отображения на мониторе пользователя. Конечно, это изменилось со временем для дополнительных устройств ввода и удаленных сценариев, но та же базовая философия остается той же: клиенты командной строки работают с текстом и кто-то другой управляет отображением и вводом.

## Стандартные дескрипторы

Стандартные дескрипторы — это ряд, `STDIN`, `STDOUT` и `STDERR`, как часть пространства процессов при запуске. Они представляют собой место для принятия информации на пути и отправки обратно на выход (включая специальное место для сообщения об ошибках). Для приложений командной строки они всегда должны существовать при запуске приложения. Они наследуются от родительского элемента автоматически, задаются явным образом родительским или автоматически создаются операционной системой, если ни один из них не указан или разрешен. Для классических приложений Windows они могут быть пустыми при запуске. Однако они могут быть неявно или явно унаследованы от родительского или выделенного, присоединенного и освобожденного во время выполнения самим приложением.

Стандартные дескрипторы не подразумевают конкретный тип присоединенного устройства. Однако в случае приложений командной строки устройство чаще всего является консольным устройством, файлом (от перенаправления в оболочке) или

каналом (из оболочки, подключающей выходные данные одной служебной программы к входным данным следующего). Он также может быть сокетом или любым другим типом устройства.

## TTY/PTY

На платформах, отличных от Windows, устройства TTY и PTY представляют соответственно истинное физическое устройство или созданное по программному обеспечению псевдо-устройство, которое является тем же понятием, что и сеанс консоли Windows: канал, в котором обмениваются данными между клиентским приложением командной строки и серверным приложением интерактивного взаимодействия или физическим приложением клавиатуры или устройством для отображения текста.

## Клиенты и серверы

В этом пространстве мы называем "клиентами" как приложения, которые выполняют обработку информации и выполняют команды. Приложения "server" — это приложения, которые отвечают за пользовательский интерфейс и являются работниками для перевода входных и выходных данных в стандартные формы от имени клиентов.

## Консольная подсистема

Это термин catch-all, представляющий все модули, влияющие на операции консоли и командной строки. Он специально относится к флагу, который является частью заголовка переносимого исполняемого файла, который указывает, является ли начальное приложение командной строкой или консольным приложением (и должно иметь стандартные дескрипторы для запуска) или приложение windows (и не требует их).

Узел консоли, клиентские приложения командной строки, драйвер консоли, область API консоли, псевдоконсольная инфраструктура, терминалы, таблицы свойств конфигурации, механизмы и заглушки внутри загрузчика процесса, а также все служебные программы, связанные с работой этих форм приложений, считаются принадлежащими этой группе.

## Узел консоли

Узел консоли Windows или `conhost.exe` — это серверное приложение для всех API консоли Windows, а также классический пользовательский интерфейс Windows для работы с приложениями командной строки. Полное содержимое этого двоичного файла, как сервера API, так и пользовательского интерфейса, исторически принадлежало Windows `csrss.exe`, критическому системному процессу и было разорвано для целей безопасности и изоляции. В дальнейшем `conhost.exe` будет отвечать за обслуживание и перевод вызовов API, но компоненты пользовательского интерфейса предназначены для делегирования через псевдоконсоль терминалу.

## Псевдоконсоль

Это имитация Windows псевдотерминального или PTY с других платформ. Он пытается сопоставить общую философию интерфейсов PTY, предоставляя простой двунаправленный канал обмена данными на основе текста, но он дополняет его в Windows большим уровнем совместимости, чтобы перевести ширину приложений Windows, написанных до этого изменения философии проектирования с классической области консольного API в простую форму связи с текстовым каналом. Терминалы могут использовать псевдоконсоль для владения элементами пользовательского интерфейса от узла консоли, `conhost.exe` оставляя его в ответственности за обслуживание, перевод и совместимость.

## Терминал

Терминал — это модуль пользовательского интерфейса и взаимодействия для приложения командной строки. Сегодня это программное представление того, что раньше было физическим устройством с монитором дисплея, клавиатурой и двунаправленным последовательным каналом связи. Он отвечает за сбор входных данных от пользователя в различных формах, перевод его и кодировку и любую специальную информацию команды в один текстовый поток и отправку его в PTY для передачи в `STDIN` канал клиентского приложения командной строки. Он также отвечает за получение обратной информации через PTY, которая была получена из канала клиентского приложения `STDOUT`, декодируя все специальные сведения в полезных данных, выкладывая все тексты и дополнительные команды, а также представлять это графически для конечного пользователя.

# Классические API консоли и виртуальные последовательности терминалов

Статья • 12.07.2023

Мы рекомендуем заменить классические API консоли Windows на последовательности виртуальных терминалов. В этой статье описываются различия между первым и вторым вариантами, а также рассматриваются причины этой рекомендации.

## Определения

Под классической контактной зоной [API консоли Windows](#) имеется в виду серия функциональных интерфейсов на языке C, помещенных в `kernel32.dll` и имеющих в имени слово `Console`.

[Последовательности виртуальных терминалов](#) определяются как язык команд, входящий в стандартные потоки ввода и вывода. Последовательности виртуальных терминалов используют непечатаемые escape-символы, чтобы передавать команды, чередуемые с обычным печатаемым текстом.

## История

Консоль Windows предоставляет обширную контактную зону API для клиентских приложений командной строки, с помощью которой можно управлять как буфером отображения вывода, так и буфером пользовательского ввода. Однако другие платформы, не относящиеся к Windows, никогда не использовали подход на основе API в среде командной строки, предпочитая последовательности виртуальных терминалов, входящие в стандартные потоки ввода и вывода. (*В то время корпорация Microsoft поддерживала подобное решение в ранних версиях DOS и Windows, предлагая драйвер с именем ANSI.SYS.*)

С другой стороны, [последовательности виртуальных терминалов](#) (для разных диалектов) отвечают за операции среды командной строки на всех остальных платформах. Эти последовательности возникли из [стандарта ECMA](#) и серии расширений от разных поставщиков, начиная с терминалов Digital Equipment Corporation и Tektronix и заканчивая более современными и распространенными терминалами наподобие [xterm](#). В домене последовательностей виртуальных

терминалов существует множество расширений, причем одни последовательности поддерживаются значительно шире, чем другие. Можно с уверенностью сказать, что эти последовательности стали мировым стандартом в качестве языка команд для командной строки, чье общеизвестное подмножество поддерживается практически каждым терминалом и клиентским приложением командной строки.

## Поддержка межплатформенности

Последовательности виртуальных терминалов изначально поддерживаются на разных платформах, благодаря чему приложения для терминала и служебные программы командной строки можно легко переносить между версиями и вариантами операционных систем, за исключением Windows.

Напротив, API консоли Windows поддерживаются только в Windows. Чтобы перенести служебные программы командной строки с одной платформы на другую, на пути между Windows и виртуальным терминалом должен быть создан обширный адаптер или библиотека преобразования.

## Удаленный доступ

Последовательности виртуальных терминалов наиболее полезные при использовании удаленного доступа. Они не требуют дополнительных работ для передачи или выполнения удаленных вызовов процедур, которые так необходимы для настройки стандартного подключения удаленной командной строки. Чтобы перенести всю информацию, необходимую приложению, которое сообщает последовательности удаленному узлу, достаточно просто подключить транспортный канал ввода и вывода (или же один двухсторонний канал) через канал, сокет, файл, последовательный порт или любое другое устройство.

API консоли Windows, в свою очередь, доступны только на локальном компьютере, и при любой попытке подключиться к ним удаленно кроме простого канала придется создавать целый промежуточный слой для удаленных вызова и передачи.

## Разделение задач

Некоторые API консоли Windows предоставляют низкоуровневый доступ к буферам ввода и вывода или вспомогательные функции для интерактивных командных строк. Сюда могут входить псевдонимы и журнал команд, запрограммированные в подсистеме консоли и среде узла, а не в самом клиентском приложении командной строки.

Другие платформы, напротив, передают ответственность за хранение текущего состояния приложения и за удобство его использования служебной программе командной строки или самой оболочке.

Благодаря тому, как консоль Windows обрабатывает эту обязанность в узле консоли и API, написание приложения с упомянутыми функциями становится легче и быстрее, при этом снимается обязанность запоминания состояния знаков или управления вспомогательными функциями редактирования. Однако из-за этого такие действия практически невозможно удаленно использовать между платформами, версиями или сценариями, поскольку существуют различные варианты реализации и доступности. Этот же способ обработки обязанности делает так, что последнее интерактивное взаимодействие приложений командной строки Windows начинает полностью зависеть от реализации, приоритетов и цикла выпуска узла консоли.

Например, такие расширенные функции редактирования строк, как выделение синтаксиса и комплексные критерии выбора, возможны только в том случае, когда приложение командной строки обрабатывает изменения редактирования самостоятельно. Консоль, в отличие от клиентского приложения, никогда не будет обладать достаточным контекстом для полного понимания этих сценариев.

Другие платформы, в свою очередь, используют **последовательности виртуальных терминалов** для обработки этих действий и самого взаимодействия между виртуальными терминалами посредством клиентской библиотеки с возможностью повторного использования, например [readline](#) и [ncurses](#). Конечный терминал отвечает только за отображение информации и прием входных данных через двухсторонний канал связи.

## Команды, которые выполняются в обратном направлении

С помощью консоли Windows некоторые действия можно выполнить в неестественном для потоков ввода и вывода направлении. Это избавляет приложения командной строки Windows от необходимости управления собственными буферами. Кроме того, приложения командной строки Windows получают возможность выполнять расширенные операции, такие как имитация или внедрение входных данных от имени пользователя, а также считывание записей отдельных журналов.

Приложения Windows, работающие в определенном пользовательском контексте на одном компьютере, получают дополнительные возможности и вектор для преодоления уровней безопасности и полномочий, а также доменов в

определенных сценариях. В эти сценарии включены операции между контекстами для одного компьютера или в различных контекстах для другого компьютера или среды.

Другие платформы, использующие **последовательности виртуальных терминалов**, не поддерживают это действие. Мы рекомендуем перейти с классической консоли Windows на последовательности виртуальных терминалов, чтобы обеспечить совместимость и безопасность.

## Прямой доступ к окну

Контактная зона API консоли Windows предоставляет точный дескриптор окна для окна размещения. Благодаря этому служебная программа командной строки может выполнять дополнительные операции с окном, имея доступ к широкому диапазону API Win32, разрешенных для дескриптора окна. Эти API Win32 способны управлять состоянием окна, его рамкой, значком или другими свойствами.

Для других платформ, которые используют **последовательность виртуальных терминалов**, также существует небольшой набор команд, которые можно выполнить для окна. Эти команды могут выполнять такие действия, как изменение размера окна или отображаемого заголовка, однако их нужно выполнять в той же полосе и в том же элементе управления, что и остальной поток.

По мере развития Windows увеличивалось количество элементов управления безопасностью и ограничений для дескрипторов окон. Менялись также характер и доступность дескриптора окна в приложении для каждого отдельного элемента пользовательского интерфейса, особенно с увеличением поддержки форм-факторов и платформ устройств. Из-за этого прямой доступ к окну для приложений командной строки становится ненадежным ввиду постоянного развития платформы и ее возможностей.

## Юникод

UTF-8 — это кодировка для данных в Юникоде, которую используют почти на всех современных платформах, поскольку она обеспечивает оптимальный баланс между переносимостью, размером хранилища и временем обработки. Однако в Windows в качестве основной кодировки для данных в Юникоде изначально выбрана кодировка UTF-16. Поддержка UTF-8 в Windows становится все более значительной, однако использование этих форматов Юникода не исключает возможности использования других кодировок.

Платформа консоли Windows поддерживала ранее и будет дальше поддерживать все существующие кодовые страницы и кодировки. Используйте UTF-16 для максимальной совместимости между версиями Windows и при необходимости выполняйте алгоритмический перевод с помощью UTF-8. Для консольной системы предусматривается расширенная поддержка UTF-8.

Поддержку UTF-16 можно активировать в консоли, не совершая дополнительных настроек и используя вариант *W* всех API-интерфейсов консоли. Она является более вероятным выбором для приложений, уже хорошо знакомых с UTF-16 через взаимодействие с `wchar_t` и вариантом *W* других функций и продуктов платформы Microsoft и Windows.

Поддержку UTF-8 в консоли можно активировать с помощью варианта *A* API-интерфейсов консоли для дескрипторов консоли, задав для кодовой страницы значения `65001` или `CP_UTF8` с помощью методов `SetConsoleOutputCP` и `SetConsoleCP` соответственно. Предварительная настройка кодовых страниц необходима только в том случае, если для компьютера в настройках для приложений, не поддерживающих Юникод, в разделе "Регион" панели управления не выбрано значение Use Unicode UTF-8 for worldwide language support (Использовать Unicode UTF-8 для поддержки всех языков).

#### ⓘ Примечание

На данный момент UTF-8 поддерживается полностью в стандартном потоке вывода при использовании методов `WriteConsole` и `WriteFile`. Поддержка потока ввода зависит от режима ввода и будет совершенствоваться с течением времени. Примечательно, что стандартные режимы "обработанного" ввода не полностью поддерживают UTF-8. Текущее состояние этого механизма можно найти на сайте GitHub в разделе [microsoft/terminal#7777 ↗](https://github.com/microsoft/terminal/issues/7777). Решение проблемы заключается в использовании алгоритмически реализуемого UTF-16 для считывания ввода с помощью методов `ReadConsoleW` или `ReadConsoleInputW`, пока не будут решены оставшиеся вопросы.

## Рекомендации

При разработке всех новых и текущих приложений в Windows в качестве способа взаимодействия с терминалом рекомендуется использовать **последовательности виртуальных терминалов**. Это позволит согласовать клиентские приложения

командной строки Windows со стилем разработки приложений на других платформах.

## Особенности использования API консоли Windows

Для настройки первоначальной среды все еще нужно будет использовать ограниченное подмножество API-интерфейсов консоли Windows. Платформа Windows по-прежнему отличается от других платформ обработкой процессов, сигналов, устройств и кодировок.

- Управление стандартными дескрипторами процесса будет по-прежнему реализовано с помощью методов [GetStdHandle](#) и [SetStdHandle](#).
- Конфигурация режимов консоли на дескрипторе для использования поддержки последовательности виртуальных терминалов будет обрабатываться с помощью методов [GetConsoleMode](#) и [SetConsoleMode](#).
- Объявление кодовой страницы или UTF-8 осуществляется с помощью методов [SetConsoleOutputCP](#) и [SetConsoleCP](#).
- Для объединения сеансов или выхода из сеанса работы с консольным устройством может потребоваться определенный уровень управления общим процессом и использование методов [AllocConsole](#), [AttachConsole](#) и [FreeConsole](#).
- Передача сигналов и их обработка будет по-прежнему осуществляться с помощью методов [SetConsoleCtrlHandler](#), [HandlerRoutine](#), а также [GenerateConsoleCtrlEvent](#).
- Обмен данными с дескрипторами консольного приложения можно выполнять с помощью методов [WriteConsole](#) и [ReadConsole](#). Их также можно использовать в средах выполнения языков программирования в виде: – Среда выполнения C (CRT): [потоковый ввод-вывод](#), например `printf`, `scanf`, `putc`, `getc` или [другие уровни функций ввода-вывода](#). – Стандартная библиотека C++ (STL): объекты [iostream](#), например `cout` и `cin`. – Среда выполнения .NET: [System.Console](#), например `Console.WriteLine`.
- Для приложений, которые должны получать сведения об изменениях размера окна, по-прежнему понадобится использовать метод [ReadConsoleInput](#), чтобы эти сведения содержали ключевые события, поскольку метод [ReadConsole](#) не будет принимать их во внимание.

- Для приложений, выполняющих операции по отрисовке столбцов, сеток или заполнению экрана, поиск размера окна все равно необходимо выполнить с помощью метода [GetConsoleScreenBufferInfo](#). В сеансе [псевдоконсоли](#) размеры буфера и окна будут совпадать.

## Планы на будущее и псевдоконсоль

Мы не планируем удалять API консоли Windows с платформы.

Напротив, узел консоли Windows предоставил технологию [псевдоконсоли](#) для преобразования существующих вызовов приложений командной строки Windows в последовательности виртуальных терминалов и их последующей переадресации в другую среду размещения в удаленном режиме или на другие платформы.

Это преобразование не безупречно. Для него нужно, чтобы окно узла консоли поддерживало смоделированную среду того, что пользователь может увидеть в Windows. Затем реплика этой смоделированной среды проецируется на узел [псевдоконсоли](#). Все вызовы API консоли Windows выполняются в смоделированной среде, чтобы отвечать потребностям устаревшего клиентского приложения командной строки. На окончательный узел распространяются только эффекты.

Поэтому при использовании приложений командной строки, требующих полной совместимости между платформами и поддержки всех новых функций и сценариев как в Windows, так и в других приложениях, рекомендуется перейти на последовательность виртуальных терминалов и настроить архитектуру приложений командной строки в соответствии с требованиями всех платформ.

Дополнительные сведения об этом переходе Windows для приложений командной строки см. в [стратегии развития экосистемы](#).

# Сведения о приложениях с текстовым режимом

Статья • 23.10.2023

Приложения режима символов (или командной строки):

1. [Необязательно] Чтение данных из стандартных входных данных (stdin)
2. Сделать "работа"
3. [Необязательно] Запись данных в стандартные выходные данные (stdout) или стандартная ошибка (stderr)

Приложения режима символов взаимодействуют с конечным пользователем с помощью приложения "консоль" (или "терминал"). Консоль преобразует входные данные пользователя с клавиатуры, мыши, сенсорного экрана, пера и т. д., а также отправляет его в stdin приложения в режиме символов. Консоль также может отображать текстовые выходные данные приложения в режиме символов на экране пользователя.

В Windows консоль встроена и предоставляет широкий API, с помощью которого приложения режима символов могут взаимодействовать с пользователем. Однако в последнее время команда консоли поощряет разработку всех приложений в режиме символов с помощью последовательностей виртуальных терминалов [по классическим вызовам API](#) для обеспечения максимальной совместимости между Windows и другими операционными системами. Дополнительные сведения об этом переходе и компромиссах, связанных с этим, можно найти в нашем обсуждении [классических API](#) и [виртуальных последовательностей терминалов](#).

- Консоли
- Методы ввода и вывода
- Кодовые страницы консоли
- Обработчики команд управления в консоли
- Псевдонимы в консоли
- Безопасность и права доступа для буфера консоли
- Проблемы с приложением консоли

# Методы ввода и вывода

Статья • 23.10.2023

Существует два различных подхода к вводу-выводу консоли, выбор которого зависит от того, сколько гибкости и контроля требуется приложению.

Высокоуровневый подход обеспечивает простой поток ввода-вывода символов, но ограничивает доступ к входным и экранным буферам консоли. Низкоуровневый подход требует, чтобы разработчики писали больше кода и выбирайте один из более широких диапазонов функций, но он также дает приложению большую гибкость.

## ⓘ Примечание

Подход низкого уровня не рекомендуется для новых и текущих разработок. Приложениям, нуждающимся в функциональных возможностях низкоуровневой консоли ввода-вывода, рекомендуется использовать последовательности виртуальных терминалов и изучить нашу документацию по классическим функциям и виртуальному терминалу и стратегии экосистемы.

Приложение может использовать функции ввода-вывода файлов, [ReadFile](#) и [WriteFile](#), а также функции консоли, [ReadConsole](#) и [WriteConsole](#) для высокоуровневого ввода-вывода, предоставляющего косвенный доступ к входным и экранным буферам консоли. Высокоуровневый фильтр входных функций и обработка данных в входном буфере консоли для возврата входных данных в виде потока символов, дис карта при изменении размера мыши и буфера. Аналогичным образом высокоуровневые выходные функции записывают поток символов, отображаемых в текущем расположении курсора в буфере экрана. Приложение управляет тем, как эти функции работают, задав режимы ввода-вывода консоли.

Низкоуровневые функции ввода-вывода обеспечивают прямой доступ к входным и экранным буферам консоли, что позволяет приложению получать доступ к событиям ввода и изменения размера буфера и расширенным сведениям для событий клавиатуры. Низкоуровневые выходные функции позволяют приложению считывать или записывать в указанное число последовательных символьных ячеек в буфере экрана, а также читать или записывать в прямоугольные блоки символьных ячеек в указанном расположении в буфере экрана. Режимы ввода консоли влияют на низкоуровневые входные данные, позволяя приложению определить, помещаются ли события изменения размера мыши и буфера в

входной буфер. Режимы вывода консоли не влияют на низкоуровневые выходные данные.

Высокоуровневые и низкоуровневые методы ввода-вывода не являются взаимоисключающими, и приложение может использовать любое сочетание этих функций. Как правило, приложение использует один подход или другой исключительно, и мы рекомендуем сосредоточиться на одной конкретной парадигме для оптимальных результатов.

### 💡 Совет

Идеальное ориентированное приложение будет сосредоточиться на высокоуровневых методах и дополнительных потребностях с **виртуальными последовательностями терминалов** через высокоуровневые методы ввода-вывода при необходимости избежать использования низкоуровневых функций ввода-вывода полностью.

В следующих разделах описываются режимы консоли и высокоуровневые и низкоуровневые функции ввода-вывода.

- Консольные режимы
- Высокоуровневая консоль ввода-вывода
- Режимы консоли высокого уровня
- Функции ввода и вывода консоли высокого уровня
- Последовательности виртуального терминала в консоли
- Классические функции и последовательности виртуальных терминалов
- Стратегия развития экосистемы
- Низкоуровневый ввод-вывод консоли
- Режимы консоли низкого уровня
- Низкоуровневые функции ввода консоли
- Функции вывода консоли низкого уровня

# Кодовые страницы консоли

Статья • 23.10.2023

Кодовая страница — это сопоставление 256 кодов символов с отдельными символами. Разные кодовые страницы включают разные специальные символы, как правило, настроенные для языка или группы языков.

Связанные с каждой консолью: две кодовых страницы: одна для входных данных и одна для выходных данных. Консоль использует свою кодовую страницу ввода для перевода ввода клавиатуры в соответствующее значение символа. Она использует свою выходную кодовую страницу для перевода значений символов, написанных различными выходными функциями, в изображения, отображаемые в окне консоли. Приложение может использовать [функции SetConsoleCP](#) и [GetConsoleCP](#) для задания и получения входных кодовых страниц консоли и [функций SetConsoleOutputCP](#) и [GetConsoleOutputCP](#), чтобы задать и получить свои выходные кодовые страницы.

Идентификаторы кодовых страниц, доступных на локальном компьютере, хранятся в реестре в следующем разделе:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage`

Сведения об использовании функций реестра для определения доступных кодов см. в разделе "[Реестр](#)".

## 💡 Совет

Рекомендуется для всех новых и обновленных приложений командной строки, чтобы избежать кодовых страниц и использования **Юникода**.

Форматированный текст UTF-16 можно отправить в семейство API консоли *W*.

Форматированный текст UTF-8 можно отправить в семейство интерфейсов

API консоли после того, как кодовая страница должна иметь значение **65001** (`CP_UTF8`) с функциями `SetConsoleCP` и `SetConsoleOutputCP`.

# Обработчики команд управления в консоли

Статья • 23.10.2023

Каждый консольный процесс имеет собственный список функций обработчика элементов управления, вызываемых системой, когда процесс получает [сигнал CTRL+C, CTRL+BREAK](#) или [CTRL+CLOSE](#). Изначально список обработчиков элементов управления для каждого процесса содержит только функцию обработчика по умолчанию, которая вызывает [функцию ExitProcess](#). Консольный процесс может добавлять или удалять дополнительные функции HandlerRoutine, вызывая [функцию SetConsoleCtrlHandler](#). Эта функция не влияет на списки обработчиков элементов управления для других процессов. Когда консольный процесс получает любой из сигналов управления, он вызывает функции обработчика на последней зарегистрированной, первой вызываемой основе, пока один из обработчиков не возвращает `TRUE`. Если ни один из обработчиков не возвращает значение `TRUE`, вызывается обработчик по умолчанию.

Параметр `dwCtrlType` функции определяет, какой сигнал управления был получен, а возвращаемое значение указывает, был ли обработан сигнал.

Новый поток запускается внутри клиентского процесса командной строки для запуска подпрограмм обработчика. Дополнительные сведения о значениях времени ожидания и действии этого потока см. в [документации по функции HandlerRoutine](#).

Пример функции обработчика элементов управления см. в разделе "[Регистрация функции обработчика элементов управления](#)".

Обратите внимание, что вызов [AttachConsole](#), [AllocConsole](#) или [FreeConsole](#) сбрасывает таблицу обработчиков элементов управления в процессе клиента в исходное состояние.

# Псевдонимы в консоли

Статья • 12.07.2023

## ⓘ Важно!

В этом документе описаны функциональные возможности платформы консоли, которые больше не являются частью [нашей стратегии развития экосистемы](#). Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие варианты использования в неопределенном будущем. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию см. в нашем [документе о классической консоли и виртуальном терминале](#).

Псевдонимы консоли используются для сопоставления исходных строк с целевыми строками. Например, можно определить псевдоним консоли, который сопоставляет "test" с "cd \a\_very\_long\_path\test". При вводе "test" в командной строке подсистема консоли расширяет псевдоним и выполняет указанную команду cd.

Чтобы определить псевдоним консоли, используйте [Doskey.exe](#) для создания макроса или функцию [AddConsoleAlias](#). В следующем примере используется [Doskey.exe](#).

```
doskey test=cd \a_very_long_path\test
```

Следующий вызов [Метода AddConsoleAlias](#) создает тот же псевдоним консоли:

```
C:\>AddConsoleAlias( TEXT("test"),
    TEXT("cd \\\<a_very_long_path>\\\test"),
    TEXT("cmd.exe"));
```

Чтобы добавить параметры в макрос псевдонима консоли с помощью [Doskey.exe](#), используйте параметры [\\$1](#) пакета через [\\$9](#). Дополнительные сведения о специальных кодах, которые можно использовать в определениях макросов Doskey, см. в справке командной строки для [Doskey.exe](#) или [Doskey](#) в TechNet.

Все экземпляры исполняемого файла, запущенного в одном окне консоли, используют все определенные псевдонимы консоли. Несколько экземпляров одного исполняемого файла, запущенных в разных окнах консоли, не используют общие псевдонимы консоли. Разные исполняемые файлы, работающие в одном окне консоли, не используют псевдонимы консоли.

Чтобы получить целевую строку для указанной исходной строки и исполняемого файла, используйте функцию [GetConsoleAlias](#) . Чтобы получить все псевдонимы для указанного исполняемого файла, используйте функцию [GetConsoleAliases](#) .

Чтобы получить имена всех псевдонимов, для которых определены псевдонимы консоли, используйте функцию [GetConsoleAliasExes](#) .

# Безопасность и права доступа для буфера консоли

Статья • 23.10.2023

Модель безопасности Windows позволяет управлять доступом к буферам ввода консоли и буферам экрана консоли. Дополнительные сведения о безопасности см. в статье "[Модель управления доступом](#)".

## Дескрипторы безопасности объектов консоли

При вызове функции [CreateFile](#) или [CreateConsoleScreenBuffer](#) можно указать **дескриптор безопасности** для буферов ввода и экрана консоли. При указании **NULL** объект получает дескриптор безопасности по умолчанию. Списки управления доступом в дескрипторе безопасности по умолчанию для буфера консоли получены из первичного или олицетворения маркера создателя.

Маркеры, возвращаемые [CreateFile](#), [CreateConsoleScreenBuffer](#) и [GetStdHandle](#), имеют права доступа **GENERIC\_READ** и **GENERIC\_WRITE**.

Допустимые права доступа включают **GENERIC\_READ** и **GENERIC\_WRITE**[универсальные права](#) доступа.

Значение	Значение
GENERIC_READ (0x800000000000L)	Запрашивает доступ на чтение к буферу экрана консоли, что позволяет процессу считывать данные из буфера.
GENERIC_WRITE (0x400000000000L)	Запрашивает доступ на запись к буферу экрана консоли, что позволяет процессу записывать данные в буфер.

### ⓘ Примечание

**консольные приложения** универсальная платформа Windows и те, которые с более низким уровнем целостности, чем подключенная консоль, будут запрещены как считывать выходной буфер, так и записывать в входной буфер, даже если дескрипторы безопасности, указанные выше, обычно разрешают его. Дополнительные сведения см. в разделе "[Неправильный способ глаголов](#)" ниже.

# Команды, которые выполняются в обратном направлении

Некоторые операции с объектами консоли будут отклонены даже в том случае, если объект имеет дескриптор безопасности, который, как указано, специально разрешает чтение или запись. Это особенно касается приложений командной строки, работающих в контексте с уменьшенными привилегиями, которые совместно используют сеанс консоли, созданный приложением командной строки в более неизбежном контексте.

Термин "неправильный способ глаголов" предназначен для применения к операции, которая является обратным потоком для одного из объектов консоли. В частности, обычный поток для выходного буфера записывается, а обычный поток для входного буфера считывается. Таким образом, "неправильный способ" будет чтение выходного буфера или запись входного буфера. Это функции, описанные в [документации по функциям консоли](#) нижнего уровня.

Два сценария, в которых это можно найти:

1. [универсальная платформа Windows консольных приложений](#). Так как это двоюродные братья других приложений универсальная платформа Windows, они содержат обещание, что они изолированы от других приложений и предоставляют пользователям гарантии вокруг последствий их работы.
2. Любое консольное приложение намеренно запущено с более низким [уровнем целостности](#), чем существующий сеанс, который можно выполнить с [помощью меток или обработки маркеров во время CreateProcess](#).

Если обнаружен любой из этих сценариев, консоль применит флаг "неправильные команды" к подключению приложения командной строки и отклонит вызовы следующих API, чтобы уменьшить поверхность взаимодействия между уровнями:

[ReadConsoleOutput](#)

[ReadConsoleOutputCharacter](#)

[ReadConsoleOutputAttribute](#)

[WriteConsoleInput](#)

Отклоненные вызовы получат код ошибки с отказом доступа, как если разрешение на чтение или запись было отклонено дескрипторами безопасности объекта.

# Проблемы с приложением консоли

Статья • 23.10.2023

8-разрядные функции консоли используют кодовую страницу OEM. Все остальные функции используют кодовую страницу ANSI по умолчанию. Это означает, что строки, возвращаемые функциями консоли, могут не обрабатываться правильно другими функциями и наоборот. Например, если `FindFirstFileA` возвращает строку, содержащую определенные расширенные символы ANSI, `WriteConsoleA` не будет отображать строку должным образом.

Лучшее долгосрочное решение для консольного приложения — использовать [Юникод](#). Консоль примет кодировку UTF-16 в варианте W API или кодировки UTF-8 в варианте API-интерфейсов A после использования `SetConsoleCP` и `SetConsoleOutputCP 65001` (`CP_UTF8` константа) для кодовой страницы UTF-8.

За исключением этого решения консольное приложение должно использовать [функцию SetFileApisToOEM](#). Эта функция изменяет соответствующие функции файлов, чтобы они создали строки наборов символов OEM, а не строки набора символов ANSI.

Ниже приведены функции файлов:

[CopyFile](#)

[.CreateDirectory](#)

[CreateFile](#)

[Createprocess](#)

[DeleteFile](#)

[FindFirstFile](#)

[FindNextFile](#)

[GetCurrentDirectory](#)

[GetDiskFreeSpace](#)

[GetDriveType](#)

[GetFileAttributes](#)

[GetFullPathName](#)

[GetModuleFileName](#)

[GetModuleHandle](#)

[GetSystemDirectory](#)

[GetTempFileName](#)

[GetTempPath](#)

[GetVolumeInformation](#)

[GetWindowsDirectory](#)

[Loadlibrary](#)

[LoadLibraryEx](#)

[MoveFile](#)

[MoveFileEx](#)

[Openfile](#)

[RemoveDirectory](#)

[SearchPath](#)

[SetCurrentDirectory](#)

[SetFileAttributes](#)

При работе с командными строками консольное приложение должно получить командную строку в форме Юникода и преобразовать ее в форму OEM, используя соответствующие функции символов в OEM. Обратите внимание, что *argv* использует набор символов ANSI.

# Консоли

Статья • 23.10.2023

Консоль — это приложение, которое предоставляет службы ввода-вывода приложениям в режиме символов.

Консоль состоит из входного буфера и одного или нескольких буферов экрана. Входной буфер содержит очередь входных записей, каждая из которых содержит сведения о входном событии. Очередь ввода всегда включает события нажатия клавиши и выпуска ключей. Он также может включать события мыши (движения указателя и нажатия кнопки и выпуски) и события, во время которых действия пользователя влияют на размер активного буфера экрана. Буфер экрана — это двумерный массив символов и данных о цвете для вывода в окне консоли. Любое количество процессов может совместно использовать консоль.

## 💡 Совет

Более широкая идея консоли и их связь с терминалами и клиентскими приложениями командной строки можно найти в [стратегии](#) экосистемы.

- [Создание консоли](#)
- [Подключение к консоли](#)
- [Закрытие консоли](#)
- [Дескрипторы консоли](#)
- [Входной буфер консоли](#)
- [Буферы экрана консоли](#)
- [Размер буфера окна и экрана](#)
- [Выбор консоли](#)
- [Прокрутка буфера экрана](#)

# Создание консоли

Статья • 12.07.2023

Система создает новую консоль при запуске *процесса консоли* (процесса в символьном режиме, начальной точкой которого является функция `main`).

Например, система создает новую консоль при запуске обработчика команд `cmd.exe`. Когда обработчик команд запускает новый процесс консоли, пользователь может указать, что должна сделать система: создать новую консоль для нового процесса или наследовать консоль обработчика команд.

Процесс может создать консоль с помощью одного из следующих методов:

- Графический пользовательский интерфейс (GUI) или процесс консоли может использовать функцию [CreateProcess](#) с флагом `CREATE_NEW_CONSOLE` для создания процесса консоли с новой консолью. По умолчанию процесс консоли наследует родительскую консоль, но при этом нет гарантии, что входные данные будут получены процессом, для которого они предназначены.
- Процесс консоли или графического пользовательского интерфейса, в настоящее время не подключенный к консоли, может использовать для создания новой консоли функцию [AllocConsole](#). (Процессы GUI не прикрепляются к консоли при их создании. Процессы консоли не привязываются к консоли, если они созданы с помощью [CreateProcess](#) с `DETACHED_PROCESS`.)

Обычно процесс использует для создания консоли функцию [AllocConsole](#), если возникает ошибка, требующая взаимодействия с пользователем. Например, процесс графического пользовательского интерфейса может создать консоль при возникновении ошибки, которая не позволяет использовать обычный графический интерфейс. Или же процесс консоли, который обычно не взаимодействует с пользователем, может создать консоль для отображения ошибки.

Процесс также может создать консоль, указав флаг `CREATE_NEW_CONSOLE` в вызове функции [CreateProcess](#). Этот метод приводит к созданию новой консоли, которая доступна дочернему, но не родительскому процессу. Отдельные консоли позволяют без конфликтов взаимодействовать с пользователем как дочерним, так и родительским процессам. Если этот флаг не указан при создании процесса консоли, оба процесса подключаются к одной консоли. При этом нет гарантии, что нужный процесс получит предназначенные для него входные данные. Приложения могут предотвратить такую путаницу, создав дочерние процессы, которые не наследуют дескрипторы входного буфера, или включив одновременно только один

дочерний процесс для наследования дескриптора входного буфера и в то же время запретив родительскому процессу считывать входные данные консоли до тех пор, пока не завершится работа дочернего процесса.

Создание новой консоли приводит к созданию нового окна консоли, а также отдельных буферов ввода-вывода для [вывода сведений на экран](#) и [ввода сведений от пользователя](#). Процесс, связанный с новой консолью, использует функцию [GetStdHandle](#) для получения дескрипторов буфера входных данных и экранного буфера новой консоли. Такие дескрипторы позволяют процессу получить доступ к консоли.

Если процесс использует функцию [CreateProcess](#), он может указать структуру [STARTUPINFO](#), элементы которой управляют характеристиками первой новой консоли (при ее наличии), созданной для этого дочернего процесса. Структура [STARTUPINFO](#), указанная в вызове функции [CreateProcess](#), влияет на созданную консоль, если задан флаг [CREATE\\_NEW\\_CONSOLE](#). Она также влияет на созданную консоль, если дочерний процесс в дальнейшем использует функцию [AllocConsole](#). Можно задать следующие характеристики консоли:

- размер нового окна консоли в символьных ячейках;
- расположение нового окна консоли в пиксельных координатах экрана;
- размер нового экранного буфера консоли в символьных ячейках;
- атрибуты цвета для текста и фона нового экранного буфера консоли;
- отображаемое имя для заголовка нового окна консоли.

Система использует значения по умолчанию, если значения [STARTUPINFO](#) не указаны. Дочерний процесс может использовать функцию [GetStartupInfo](#), чтобы определить значения в своей структуре [STARTUPINFO](#).

Процесс не может изменить расположение своего окна консоли на экране, но вы можете использовать следующие функции консоли, чтобы задать или получить другие свойства в структуре [STARTUPINFO](#).

Функция	Описание
<a href="#">GetConsoleScreenBufferInfo</a>	Получает данные о размере окна, размере экранного буфера и атрибутах цвета.
<a href="#">SetConsoleWindowInfo</a>	Изменяет размер окна консоли.
<a href="#">SetConsoleScreenBufferSize</a>	Изменяет размер экранного буфера консоли.
<a href="#">SetConsoleTextAttribute</a>	Задает атрибуты цвета.
<a href="#">SetConsoleTitle</a>	Задает заголовок окна консоли.

Функция	Описание
<a href="#">GetConsoleTitle</a>	Получает заголовок окна консоли.

Процесс может использовать функцию [FreeConsole](#), чтобы отключиться от наследованной консоли или от консоли, созданной функцией [AllocConsole](#).

Процесс может использовать функцию [AttachConsole](#), чтобы подключиться к другому существующему сеансу консоли после использования [FreeConsole](#) для отключения от собственного сеанса (или при отсутствии подключения к сеансу).

# Подключение к консоли

Статья • 23.10.2023

Процесс может использовать [функцию AttachConsole](#) для подключения к консоли в качестве клиента. Процесс можно подключить к одной консоли.

Консоль может содержать множество процессов, подключенных к нему. Чтобы получить список процессов, подключенных к консоли, вызовите [функцию GetConsoleProcessList](#).

Сведения о псевдоконсолях [см. в разделе "Подключение как сервер"](#).

# Закрытие консоли

Статья • 23.10.2023

Процесс может использовать [функцию FreeConsole](#) для отсоединения от консоли. Если другие процессы совместно используют консоль, консоль не уничтожается, но процесс, который называется `FreeConsole`, не может ссылаться на него. После вызова `FreeConsole` процесс может использовать [AllocConsole](#) для создания новой консоли или [AttachConsole](#) для подключения к другой консоли.

Консоль закрывается, когда последний процесс, подключенный к нему, завершается или вызывает [FreeConsole](#).

# Дескрипторы консоли

Статья • 12.07.2023

Процесс консоли получает доступ к входным буферам и буферам экрана консоли с помощью дескрипторов. Процесс может открыть один из этих дескрипторов с помощью функций [GetStdHandle](#), [CreateFile](#) или [CreateConsoleScreenBuffer](#).

Функция [GetStdHandle](#) предоставляет механизм получения связанных с процессом дескрипторов стандартного ввода (`STDIN`), стандартного вывода (`STDOUT`) и стандартных ошибок (`STDERR`). Во время создания консоли эти дескрипторы создаются системой. Изначально `STDIN` является дескриптором входного буфера консоли, а `STDOUT` и `STDERR` являются дескрипторами активного буфера экрана консоли. Но функция [SetStdHandle](#) может перенаправить стандартные дескрипторы путем изменения дескриптора, связанного с `STDIN`, `STDOUT` или `STDERR`. Так как стандартные дескрипторы родительского процесса наследуются любым дочерним процессом, последующие вызовы метода [GetStdHandle](#) возвращают перенаправленный дескриптор. Дескриптор, возвращенный с помощью [GetStdHandle](#), может ссылаться на элемент, отличный от ввода-вывода в консоли. Например, перед созданием дочернего процесса родительский процесс может использовать [SetStdHandle](#), чтобы задать канал `STDIN`, наследуемый дочерним процессом, в качестве канала дескриптора. Когда дочерний процесс вызывает [GetStdHandle](#), он получает дескриптор канала. Это означает, что родительский процесс может управлять стандартными дескрипторами дочернего процесса. Дескрипторы, возвращаемые с помощью [GetStdHandle](#), имеют права доступа `GENERIC_READ | GENERIC_WRITE`, если только с помощью метода [SetStdHandle](#) не был задан стандартный дескриптор с более ограниченным доступом.

Значение дескрипторов, возвращаемых с помощью [GetStdHandle](#), не равно 0, 1 или 2, поэтому стандартные предопределенные константы потока в `Stdio.h` (`STDIN`, `STDOUT` и `STDERR`) нельзя использовать в функциях, для которых требуется дескриптор консоли.

Функция [CreateFile](#) позволяет процессу получить дескриптор для входного буфера консоли и активного буфера экрана, даже если `STDIN` и `STDOUT` были перенаправлены. Чтобы открыть дескриптор для входного буфера консоли, укажите значение `CONIN$` при вызове [CreateFile](#). Укажите значение `CONOUT$` при вызове [CreateFile](#), чтобы открыть дескриптор для активного буфера экрана консоли. [CreateFile](#) позволяет указать доступ только для чтения и записи в возвращаемом дескрипторе.

Функция [CreateConsoleScreenBuffer](#) создает буфер экрана и возвращает дескриптор. Этот дескриптор можно использовать в любой функции, которая принимает дескриптор для вывода в консоли. Новый буфер экрана не активен (не отображается), пока его дескриптор не будет указан в вызове функции [SetConsoleActiveScreenBuffer](#). Обратите внимание, что изменение активного буфера экрана не влияет на дескриптор, возвращаемый методом [GetStdHandle](#). Аналогичным образом использование [SetStdHandle](#) для изменения дескриптора `STDOUT` не влияет на активный буфер экрана.

Дескрипторы консоли, возвращенные методом [CreateFile](#) и [CreateConsoleScreenBuffer](#), можно использовать в любой из консольных функций, которым необходим дескриптор для входного буфера консоли или буфера экрана консоли. Дескрипторы, возвращенные методом [GetStdHandle](#), могут использоваться функциями консоли, если они не были перенаправлены для ссылки на элементы, отличные от ввода-вывода в консоли. Но если стандартный дескриптор был перенаправлен для ссылки на файл или канал, этот дескриптор может использоваться только функциями [ReadFile](#) и [WriteFile](#). С помощью метода [GetFileType](#) можно определить тип устройства, на который ссылается этот дескриптор. Дескриптор консоли представлен как `FILE_TYPE_CHAR`.

Процесс может использовать функцию [DuplicateHandle](#) для дублирования дескриптора консоли, который имеет не такой уровень доступа или режим наследования, как у исходного дескриптора. Но обратите внимание, что процесс может дублировать дескриптор консоли только для собственного использования. Он отличается от других типов дескрипторов (таких как файл, канал или объекты мьютекса), для которых [DuplicateHandle](#) может создать дубликат, допустимый для другого процесса. Доступ к консоли должен быть общим во время [создания](#) другого процесса или запрашиваться другим процессом с помощью механизма [AttachConsole](#).

Процесс может закрыть дескриптор консоли с помощью функции [CloseHandle](#).

# Входной буфер консоли

Статья • 23.10.2023

Каждая консоль имеет входной буфер, содержащий очередь входных записей событий. Когда окно консоли имеет фокус клавиатуры, консоль форматирует каждое событие ввода (например, одно нажатие клавиши, перемещение мыши или нажатие кнопки мыши) в качестве входной записи, которая помещает его в входной буфер консоли.

Приложения могут получить доступ к входной буферу консоли косвенно с помощью [функций](#) ввода-вывода консоли высокого уровня или напрямую с помощью [низкоуровневых функций](#) ввода-вывода консоли. Высокоуровневый фильтр входных функций и обработка данных во входном буфере, возвращая только поток входных символов. Функции ввода низкого уровня позволяют приложениям считывать входные записи непосредственно из входного буфера консоли или помещать входные записи в входной буфер. Чтобы открыть дескриптор входного буфера консоли, укажите значение `CONIN$` в вызове [функции CreateFile](#).

Входная запись — это структура, содержащая сведения о типе события, которое произошло (клавиатура, мышь, изменение размера окна, фокус или событие меню), а также конкретные сведения о событии. Элемент `EventType` в [структуре INPUT\\_RECORD](#) указывает, какой тип события содержится в записи.

События фокуса и меню помещаются в входной буфер консоли для внутреннего использования системой и должны игнорироваться приложениями.

## События клавиатуры

События клавиатуры создаются при нажатии или освобождении любого клавиши; сюда входят клавиши управления. Однако клавиша ALT имеет особое значение для системы при нажатии и освобождении без сочетания с другим символом, и он не передается в приложение. Кроме того, сочетание клавиш CTRL+C не передается, если входной дескриптор находится в обработанном режиме.

Если входное событие является нажатием клавиш, [элемент события](#) в [INPUT\\_RECORD](#) является `KEY_EVENT_RECORD` структурой, содержащей следующие сведения:

- Логическое значение, указывающее, был ли нажатием или освобожден ключ.

- Число повторов, которое может быть больше одного, если ключ удерживается.
- Код виртуального ключа, определяющий заданный ключ независимо от устройства.
- Код виртуальной проверки, указывающий значение, зависящее от устройства, созданное оборудованием клавиатуры.
- Переведенный символ Юникода™ или ANSI.
- Переменная флага, указывающая состояние клавиш управления (КЛАВИШИ ALT, CTRL, SHIFT, NUM LOCK, SCROLL LOCK и CAPS LOCK) и указывает, был ли нажат расширенный ключ. Расширенные ключи для клавиатуры IBM® 101 и 102-клавиш являются INS, DEL, HOME, END, PAGE UP, PAGE DOWN и со стрелками в кластерах слева от числовой клавиатуры и разделения (/) и КЛАВИШ ВВОД на числовой клавиатуре.

## События мыши

События мыши создаются всякий раз, когда пользователь перемещает мышь или нажимает или освобождает одну из кнопок мыши. События мыши помещаются в входной буфер только в том случае, если выполнены следующие условия:

- Для режима ввода консоли задано значение **ENABLE\_MOUSE\_INPUT** (режим по умолчанию).
- В окне консоли фокус клавиатуры.
- Указатель мыши находится в границах окна консоли.

Если входное событие является событием мыши, элемент **события в INPUT\_RECORD** является **MOUSE\_EVENT\_RECORD** структурой, содержащей следующие сведения:

- Координаты указателя мыши с точки зрения строки и столбца символьной ячейки в системе координат буфера экрана консоли.
- Переменная флага, указывающая состояние кнопок мыши.
- Переменная флага, указывающая состояние клавиш управления (ALT, CTRL, SHIFT, NUM LOCK, SCROLL LOCK и CAPS LOCK) и указывающая, был ли нажат расширенный ключ. Расширенные ключи для клавиатуры IBM 101 и 102-клавиш являются INS, DEL, HOME, END, PAGE UP, PAGE DOWN и со стрелками в кластерах слева от числовой клавиатуры и разделения (/) и КЛАВИШ ВВОД на числовой клавиатуре.
- Переменная флаг, указывающая, было ли событие обычным событием нажатия кнопки или выпуска кнопки, событием перемещения мыши или вторым щелчком события двойного щелчка.

### ⓘ Примечание

Координаты положения мыши находятся в буфере экрана консоли, а не в окне консоли. Буфер экрана может быть прокручен относительно окна, поэтому верхний левый угол окна не обязательно является координатой (0,0) буфера экрана консоли. Чтобы определить координаты мыши относительно системы координат окна, вычитает координаты источника окна из координат положения мыши. Используйте функцию [GetConsoleScreenBufferInfo](#), чтобы определить координаты источника окна.

Элемент `dwButtonState` структуры [MOUSE\\_EVENT\\_RECORD](#) имеет бит, соответствующий каждой кнопке мыши. Бит равен 1, если кнопка вниз и 0, если кнопка находится вверх. Событие выпуска кнопки обнаруживается значением 0 для члена `dwEventFlags` [MOUSE\\_EVENT\\_RECORD](#) и изменением бита кнопки от 1 до 0. Функция [GetNumberOfConsoleMouseButtons](#) извлекает количество кнопок мыши.

## События изменения размера буфера

Меню окна консоли позволяет пользователю изменять размер активного буфера экрана; это изменение создает событие изменения размера буфера. События изменения размера буфера помещаются в входной буфер, если входной режим консоли имеет значение [ENABLE\\_WINDOW\\_INPUT](#) (то есть режим по умолчанию отключен).

Если входное событие является событием изменения размера буфера, элемент [события INPUT\\_RECORD](#) представляет собой [WINDOW\\_BUFFER\\_SIZE\\_RECORD](#) структуру, содержащую новый размер буфера экрана консоли, выраженный в столбцах и строках символьных ячеек.

Если пользователь уменьшает размер буфера экрана консоли, все данные в дискарача части буфера теряются.

Изменения размера буфера экрана консоли в результате вызовов [приложения функции SetConsoleScreenBufferSize](#) не создаются как события изменения размера буфера.

# Буферы экрана консоли

Статья • 12.07.2023

Буфер экрана — это двумерный массив символов и данных о цвете для вывода в окне консоли. Консоль может иметь несколько буферов экрана. Буфер активного экрана отображается на экране.

Система создает буфер экрана каждый раз при создании новой консоли. Чтобы открыть дескриптор для активного буфера экрана консоли, укажите значение `CONOUT$` в вызове к функции [CreateFile](#). Процесс может использовать функцию [CreateConsoleScreenBuffer](#) для создания дополнительных буферов экрана консоли. Новый буфер экрана не активен, пока его дескриптор не будет указан в вызове функции [SetConsoleActiveScreenBuffer](#). Но доступ к буферам экрана для чтения и записи можно получить независимо от того, активны они или неактивны.

Каждый буфер экрана содержит собственный двумерный массив записей с данными о символах. Данные о каждом символе хранятся в структуре [CHAR\\_INFO](#), которая определяет символ Юникода или ANSI, а также цвета переднего плана и фона для отображения символа.

Некоторые свойства, связанные с буфером экрана, можно задать независимо для каждого буфера. Это означает, что изменение активного буфера экрана может привести к значительному изменению внешнего вида окна консоли. Свойства, связанные с буфером экрана, включают следующие:

- Размер буфера экрана, в символьных строках и столбцах.
- Атрибуты текста (цвета переднего плана и фона для отображения текста, записываемого функцией [WriteFile](#) или [WriteConsole](#)).
- Размер и расположение окна (прямоугольная область буфера экрана консоли, отображаемого в окне консоли).
- Позиция, внешний вид и видимость курсора.
- Режимы вывода (`ENABLE_PROCESSED_OUTPUT` и `ENABLE_WRAP_AT_EOL_OUTPUT`). Дополнительные сведения о режимах вывода консоли см. в статье [Высокоуровневые режимы консоли](#).

Созданный буфер экрана содержит пробелы в каждой из позиций. Его курсор отображается и размещается в точке начала буфера (0,0), при этом окно располагается таким образом, что его верхний левый угол находится в месте начала буфера. Размер буфера экрана консоли, размер окна, атрибуты текста и внешний вид курсора определяются пользователем или системными параметрами по умолчанию. Чтобы получить текущие значения различных свойств, связанных с

буфером экрана консоли, используйте функции [GetConsoleScreenBufferInfo](#), [GetConsoleCursorInfo](#) и [GetConsoleMode](#).

Приложения, изменяющие любые свойства буфера экрана консоли, должны либо создать собственный буфер экрана, либо сохранить состояние наследуемого буфера экрана во время запуска и восстановить его при выходе. Такое скоординированное поведение гарантирует, что изменения не будут затрагивать другие приложения, совместно использующие один сеанс консоли.

### 💡 Совет

Рекомендуется использовать **режим альтернативного буфера**, а не создавать второй буфер экрана для этой цели. **Режим альтернативного буфера** обеспечивает повышенную совместимость для удаленных устройств и с другими платформами. Дополнительные сведения см. в статье о [классических API консоли и виртуальном терминале](#).

## Внешний вид и расположение курсора

Курсор буфера экрана может быть видимым или скрытым. Если он отображается, курсор может иметь самый разный вид — от горизонтальной линии внизу ячейки до полного заполнения символьной ячейки. Чтобы получить сведения о внешнем виде и видимости курсора, используйте функцию [GetConsoleCursorInfo](#). Эта функция сообщает, виден ли курсор, и описывает вид курсора с указанием процентного отношения заполнения им символьной ячейки. Чтобы задать внешний вид и видимость курсора, используйте функцию [SetConsoleCursorInfo](#).

[Высокоуровневые функции ввода-вывода консоли](#) записывают символы в текущее расположение курсора и переводят его в следующую позицию. Чтобы определить текущую позицию курсора в системе координат буфера экрана, используйте функцию [GetConsoleScreenBufferInfo](#). Вы можете использовать функцию [SetConsoleCursorPosition](#), чтобы задавать позицию курсора и таким образом управлять размещением текста, который записывается высокогуровневыми функциями или выводится ими. Если переместить курсор, текст в новом расположении будет перезаписан.

### ⓘ Примечание

Не рекомендуется использовать низкоуровневые функции для поиска позиции курсора. При необходимости используйте [последовательности виртуального](#)

терминала для запрашивания этой позиции для расширенных макетов.

Дополнительные сведения о предпочтительных последовательностях виртуального терминала см. в документе о [классических функциях и виртуальном терминале](#).

Расположение, внешний вид и видимость курсора задаются для каждого буфера экрана независимо.

## Атрибуты символов

Атрибуты символов можно разделить на два класса: Color и DBCS. Приведенные ниже атрибуты определены в файле заголовка `WinCon.h`.

attribute	Значение
FOREGROUND_BLUE	Текст содержит синий цвет.
FOREGROUND_GREEN	Текст содержит зеленый цвет.
FOREGROUND_RED	Текст содержит красный цвет.
FOREGROUND_INTENSITY	Для цвета текста изменена интенсивность.
BACKGROUND_BLUE	Фон содержит синий цвет.
BACKGROUND_GREEN	Фон содержит зеленый цвет.
BACKGROUND_RED	Фон содержит красный цвет.
BACKGROUND_INTENSITY	Для цвета фона изменена интенсивность.
COMMON_LVB.LEADING_BYTE	Начальный байт.
COMMON_LVB.TRAILING_BYTE	Конечный байт.
COMMON_LVB_GRID_HORIZONTAL	Верхний горизонтальный.
COMMON_LVB_GRID_LVERTICAL	Левый вертикальный.
COMMON_LVB_GRID_RVERTICAL	Правый вертикальный.
COMMON_LVB_REVERSE_VIDEO	Обратить атрибуты переднего плана и фона.
COMMON_LVB_UNDERSCORE	Знак подчеркивания.

Атрибуты переднего плана задают цвет текста. Атрибуты фона задают цвет, используемый для заполнения фона ячейки. Другие атрибуты используются с [DBCS](#).

Приложение может сочетать константы переднего плана и фона для отображения различных цветов. Например, следующее сочетание приводит к отображению светло-голубого текста на синем фоне.

```
BACKGROUND_BLUE | FOREGROUND_GREEN | FOREGROUND_INTENSITY | BACKGROUND_BLUE
```

Если константа фона не указана, фон будет черным. Если не указана ни одна из констант переднего плана, текст будет черным. Например, следующее сочетание приводит к отображению черного цвета на белом фоне. Красный, зеленый и синий цвета задаются для фона, который сочетается с белым фоном. Для переднего плана не указаны цвета флага, поэтому он является черным.

```
BACKGROUND_BLUE | BACKGROUND_GREEN | BACKGROUND_RED
```

Каждая символьная ячейка буфера экрана хранит атрибуты цвета для цветов, используемых при отрисовке переднего плана (текста) и фона такой ячейки. Приложение может задавать данные о цвете отдельно для каждой символьной ячейки, сохраняя данные в элементе **Attributes** структуры [CHAR\\_INFO](#) для каждой ячейки. Текущие текстовые атрибуты для каждого буфера экрана используются для символов, которые последовательно записываются или выводятся высокоуровневыми функциями.

Приложение может использовать функцию [GetConsoleScreenBufferInfo](#), чтобы определить текущие атрибуты текста для буфера экрана, и функцию [SetConsoleTextAttribute](#), чтобы задать атрибуты символов. Изменение атрибутов буфера экрана не влияет на отображение уже записанных символов. Такие атрибуты текста не влияют на символы, записанные низкоуровневыми функциями ввода-вывода консоли (например, [WriteConsoleOutput](#) или [WriteConsoleOutputCharacter](#)), которые явно задают атрибуты для каждой записываемой ячейки или оставляют атрибуты без изменений.

#### ⓘ Примечание

Мы не рекомендуем использовать низкоуровневые функции для изменения атрибутов текста (в том числе атрибутов по умолчанию). Чтобы задать атрибуты текста, используйте [последовательности виртуального терминала](#). Дополнительные сведения о предпочтительных последовательностях виртуального терминала см. в документе о [классических функциях](#) и [виртуальном терминале](#).

## Атрибуты шрифтов

Функция [GetCurrentConsoleFont](#) получает сведения о текущем шрифте консоли. В структуре [CONSOLE\\_FONT\\_INFO](#) хранятся такие сведения, как ширина и высота каждого символа в шрифте.

Функция [GetConsoleFontSize](#) получает данные о размере шрифта, используемые конкретным буфером экрана консоли.

#### ⓘ Примечание

Мы не рекомендуем использовать функции для поиска и изменения данных шрифтов. Придерживайтесь нейтральности шрифтов в приложениях командной строки, чтобы обеспечить совместимость с другими платформами, а также совместимость со средами размещения, которые разрешают пользователям настраивать шрифты. Дополнительные сведения о предпочтениях пользователей и средах размещения (в том числе о терминалах), см. в статье о [стратегии развития экосистемы](#).

# Консольные режимы

Статья • 23.10.2023

Связанный с каждым буфером ввода консоли — это набор режимов ввода, влияющих на операции ввода. Аналогичным образом каждый буфер экрана консоли имеет набор режимов вывода, влияющих на выходные операции. Режимы ввода можно разделить на две группы: те, которые влияют на высокоуровневые входные функции и те, которые влияют на низкоуровневые входные функции. Режимы вывода влияют только на приложения, использующие высокоуровневые выходные функции.

Функция [GetConsoleMode](#) сообщает текущий входной режим входного буфера консоли или текущий выходной режим буфера экрана. Функция [SetConsoleMode](#) задает текущий режим входного буфера консоли или буфера экрана. Если консоль имеет несколько буферов экрана, выходные режимы каждого из них могут отличаться. Приложение может изменять режимы ввода-вывода в любое время. Дополнительные сведения о режимах консоли, влияющих на высокоуровневые и низкоуровневые операции ввода-вывода, см . [В режимах консоли](#) высокого уровня и [режимах](#) консоли низкого уровня.

Приложение командной строки должно ожидать, что другие приложения командной строки могут изменять режим консоли в любое время и не восстанавливать его в исходной форме перед возвратом элемента управления. Кроме того, рекомендуется, чтобы все приложения командной строки захватывали начальный режим консоли при запуске и пытались восстановить его при выходе, чтобы обеспечить минимальное влияние на другие приложения командной строки, подключенные к той же консоли.

Функция [GetConsoleDisplayMode](#) сообщает, находится ли текущая консоль в полноэкранном режиме.

# Группы процессов консоли

Статья • 23.10.2023

Когда процесс использует [функцию CreateProcess](#) для создания нового консольного процесса, он может указать флаг `CREATE_NEW_PROCESS_GROUP`, чтобы сделать новый процесс корневым процессом группы процессов консоли. Группа процессов включает все процессы, которые являются потомками корневого процесса.

Процесс может использовать [функцию GenerateConsoleCtrlEvent](#) для отправки сигнала CTRL+C или CTRL+BREAK всем процессам в группе консольных процессов. Сигнал получается только теми процессами в группе, которая подключена к той же консоли, что и процесс, который называется `GenerateConsoleCtrlEvent`.

# Размер буфера окна и экрана

Статья • 12.07.2023

Размер буфера экрана выражается в виде сетки координат на основе символьных ячеек. Ширина — это количество символьных ячеек в каждой строке, а высота — количество строк. С каждым буфером экрана связано окно, определяющее размер и расположение прямоугольной части буфера экрана консоли, отображаемой в окне консоли. Окно буфера экрана определяется путем указания координат символьных ячеек верхней левой и нижней правой ячеек прямоугольника окна.

## ① Примечание

В мире последовательностей виртуальных терминалов размер окна и размер буфера экрана фиксируются в одном и том же значении. Терминал обрабатывает любую область прокрутки, эквивалентную консоли с размером буфера экрана, превышающим размер окна. Это содержимое принадлежит терминалу и, как правило, больше не является частью адресной области. Дополнительные сведения см. в нашем сравнении [классических функций консоли и последовательностей виртуальных терминалов](#).

Буфер экрана может быть любого размера и ограничен только доступной памятью. Размеры окна буфера экрана не могут превышать соответствующие размеры буфера экрана консоли или максимальное окно, которое может поместиться на экране в зависимости от текущего размера шрифта (контролируется исключительно пользователем).

Функция [GetConsoleScreenBufferInfo](#) возвращает следующие сведения о буфере экрана и его окне:

- Текущий размер буфера экрана консоли
- Текущее расположение окна
- Максимальный размер окна с учетом текущего размера буфера экрана, текущего размера шрифта и размера экрана

Функция [GetLargestConsoleWindowSize](#) возвращает максимальный размер окна консоли на основе текущего шрифта и размеров экрана. Этот размер отличается от максимального размера окна, возвращаемого [командой GetConsoleScreenBufferInfo](#), тем, что размер буфера экрана консоли игнорируется.

Чтобы изменить размер буфера экрана, используйте функцию [SetConsoleScreenBufferSize](#). Эта функция завершается ошибкой, если любое из

измерений указанного размера меньше соответствующего измерения окна консоли.

Чтобы изменить размер или расположение окна буфера экрана, используйте функцию [SetConsoleWindowInfo](#). Эта функция завершается ошибкой, если указанные координаты оконного угла превышают ограничения буфера экрана консоли или экрана. Изменение размера окна активного буфера экрана изменяет размер окна консоли, отображаемого на экране.

Процесс может изменить режим ввода консоли, чтобы включить входные данные окна, чтобы процесс мог получать входные данные при изменении пользователем размера буфера экрана консоли. Если приложение включает вход в окно, оно может использовать [GetConsoleScreenBufferInfo](#) для получения размера окна и буфера экрана при запуске. Затем эти сведения можно использовать для определения способа отображения данных в окне. Если пользователь изменяет размер буфера экрана консоли, приложение может ответить, изменив способ отображения данных. Например, приложение может настроить способ переноса текста в конце строки при изменении количества символов в строке. Если приложение не включает входные данные окна, оно должно либо использовать унаследованные размеры окна и буфера экрана, либо задать для них требуемый размер во время запуска и восстановить унаследованные размеры при выходе. Дополнительные сведения о режиме ввода окна см. в разделе [Режимы низкоуровневой консоли](#).

# Выбор консоли

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Приложению специальных возможностей требуется сведения о выборе пользователя в консоли. Чтобы получить текущий выбор консоли, вызовите [функцию GetConsoleSelectionInfo](#). Структура [CONSOLE\\_SELECTION\\_INFO](#) содержит сведения о выборе, например привязке, координатах и состоянии.

# Устаревший режим консоли

Статья • 12.07.2023

Устаревший консольный режим — это средство обеспечения совместимости, разработанное для облегчения работы пользователей со старыми программами командной строки в Windows 10. Для любой программы командной строки, которая не отображается или не работает правильно в консоли Windows 10 по умолчанию, этот режим предоставляет простое решение для возврата системы в предыдущую версию среды размещения консоли.

## Использование устаревшего режима консоли

Чтобы использовать устаревший режим консоли, сначала откройте любое окно размещения консоли. Обычно это делается путем запуска одного из интерпретаторов команд [CMD](#) или [PowerShell](#).

Щелкните правой кнопкой мыши заголовок окна приложения и выберите пункт меню `Properties`. Выберите первую вкладку `Options`. Затем установите флажок в нижней части страницы с описанием `Use legacy console`. Нажмите кнопку `OK`, чтобы применить.

Чтобы отменить параметр, вернитесь в то же меню страницы свойств и снимите флажок, а затем нажмите кнопку `OK`.

### ⓘ Примечание

Этот параметр глобально применяется ко всем сессиям, которые запускаются после изменения параметра. Уже открытые сессии не будут изменены.

## Различия между режимами

Группа разработки узла консоли стремится минимизировать различия между устаревшими и текущими режимами консоли, чтобы как можно большее количество пользователей могли запускать наиболее актуальную версию. Если у вас возникла ошибка, требующая использования устаревшей консоли, которая не описана здесь, обратитесь за помощью в группу разработчиков на странице [microsoft/terminal](#) ↗ репозитория GitHub или через [Центр отзывов](#).

# 16-разрядные приложения в 32-разрядной версии Windows

Некоторые 16-разрядные приложения в 32-разрядной версии Windows используют технологию виртуальной машины под названием [NTVDM](#). Часто эти приложения для работы используют графический режим буферизации экрана в сочетании со средой размещения консоли. Только устаревший интерфейс консоли обеспечивает поддержку этих графических режимов буферизации и дополнительную поддержку API консоли, что необходимо для работы этих приложений. При запуске одного из этих приложений система автоматически выберет устаревшую консольную среду.

## Внедрение IME

В устаревшем узле консоли внедрена часть предложения IME внутри окна размещения путем резервирования строки в нижней части экрана для предложений. В текущей среде узла консоли это действие делегируется подсистеме IME, чтобы отобразить наложенное окно над узлом консоли с предложениями. В среде, в которой наложение окон невозможно (например, с определенными средствами удаленного взаимодействия), может потребоваться устаревший узел консоли.

## Различия в API

Основное известное отличие между устаревшим и текущим — это реализация UTF-8. Устаревший узел обеспечивает крайне примитивную и часто неправильную поддержку UTF-8 с [кодовой страницей 65001](#). Текущий же узел консоли содержит добавочные улучшения в каждом выпуске Windows 10 для улучшения этой поддержки. Приложения, которые пытаются полагаться на прогнозированные "известные неправильные" интерпретации UTF-8 из устаревшей консоли, будут получать разные ответы по мере улучшения поддержки.

О других различиях в API-интерфейсах необходимо сообщить на странице [репозитория GitHub](#) или через [Центр отзывов](#) для рассмотрения и возможных исправлений.

# Псевдоконсолы

Статья • 23.10.2023

Псевдоконсол — это тип устройства, позволяющий приложениям стать узлом для приложений в режиме символов.

Это в отличие от типичного сеанса консоли, в котором операционная система создаст окно размещения от имени приложения в режиме символа для обработки графических выходных данных и ввода пользователем.

При использовании псевдоконсоля окно размещения не создается. Приложение, которое делает псевдоконсол, должно стать ответственным за отображение графических выходных данных и сбор входных данных пользователя. Кроме того, данные можно ретранслировать дальше другому приложению, ответственному за эти действия, в более позднюю точку в цепочке.

Эта функция предназначена для сторонних приложений "окна терминала", которые будут существовать на платформе или для перенаправления действий в режим символов в удаленный сеанс "окно терминала" на другом компьютере или даже на другой платформе.

Обратите внимание, что базовый сеанс консоли по-прежнему будет создан от имени приложения, запрашивающего псевдоконсол. Все правила сеансов консоли по-прежнему применяются, включая возможность подключения к сеансу нескольких клиентских приложений в режиме символов.

Чтобы обеспечить максимальную совместимость с существующим миром псевдотерминальных функций, информация, предоставляемая по каналу псевдоконсоля, всегда будет закодирована в UTF-8. Это не влияет на кодовую страницу или кодировку подключенных клиентских приложений. Перевод будет происходить внутри псевдоконсоля системы по мере необходимости.

Пример начала работы можно найти в [разделе "Создание псевдоконсоля сеанса"](#).

Дополнительные справочные сведения о псевдоконсолях можно найти в записи блога о объявлении: Командная строка Windows: [Знакомство с псевдоконсолью Windows \(ConPTY\)](#).

# Справочник по консоли

Статья • 23.10.2023

В следующих разделах описывается API консоли:

- [Функции консоли](#)
- [Структуры консоли](#)
- [Консоль WinEvents](#)

# Использование консоли

Статья • 12.07.2023

В следующих примерах показано, как использовать функции консоли:

- Использование высокоуровневых функций ввода и вывода
- Чтение и запись блоков символов и атрибутов
- Чтение событий входного буфера
- Очистка экрана
- Прокрутка окна буфера экрана
- Прокрутка содержимого буфера экрана
- Регистрация функции обработчика элемента управления

# ФУНКЦИИ ВВОДА И ВЫВОДА КОНСОЛИ ВЫСОКОГО УРОВНЯ

Статья • 23.10.2023

Функции `ReadFile` и `WriteFile` или функции `ReadConsole` и `WriteConsole` позволяют приложению читать входные данные консоли и записывать выходные данные консоли в виде потока символов. `ReadConsole` и `WriteConsole` ведут себя точно так же, как `ReadFile` и `WriteFile`, за исключением того, что они могут использоваться как функции с широкими символами (в которых текстовые аргументы должны использовать Юникод) или как функции ANSI (в которых текстовые аргументы должны использовать символы из набора символов Windows). Приложения, которые должны поддерживать один набор источников для поддержки Юникода или набора символов ANSI, должны использовать `ReadConsole` и `WriteConsole`.

`ReadConsole` и `WriteConsole` можно использовать только с дескрипторами консоли; `ReadFile` и `WriteFile` можно использовать с другими дескрипторами (например, с файлами или каналами). Сбой `ReadConsole` и `WriteConsole`, если используется со стандартным дескриптором, который был перенаправлен и больше не является дескриптором консоли.

Чтобы получить ввод клавиатуры, процесс может использовать `ReadFile` или `ReadConsole` с дескриптором входного буфера консоли или использовать `ReadFile` для чтения входных данных из файла или канала, если `STDIN` он был перенаправлен. Эти функции возвращают только события клавиатуры, которые можно преобразовать в символы ANSI или Юникода. Входные данные, которые можно вернуть, включают сочетания клавиш управления. Функции не возвращают события клавиатуры, включающие клавиши функции или клавиши со стрелками. Входные события, созданные мышью, окном, фокусом или вводом меню, не отображаются карта.

Если режим ввода строки включен (режим по умолчанию), `ReadFile` и `ReadConsole` не возвращаются в вызывающее приложение до нажатия клавиши ВВОД. Если режим ввода строки отключен, функции не возвращаются до тех пор, пока не будет доступен хотя бы один символ. В любом режиме все доступные символычитываются до тех пор, пока не будут доступны никакие клавиши или указанное число символов было прочитано. Непрочитанные символы буферизованы до следующей операции чтения. Функции сообщают общее количество символов, которые фактически считаются. Если включен режим ввода эхо, символы, считываемые этими функциями, записываются в активный буфер экрана в текущей позиции курсора.

Процесс может использовать [WriteFile](#) или [WriteConsole](#) для записи в активный или неактивный буфер экрана, или использовать [WriteFile](#) для записи в файл или канал, если STDOUT был перенаправлен. Обработанный режим вывода и оболочка в режиме вывода EOL определяют способ записи или эхо символов в буфер экрана.

Символы, написанные [writeFile](#) или [WriteConsole](#), или эхом [ReadFile](#) или [ReadConsole](#), вставляются в буфер экрана в текущей позиции курсора. По мере записи каждого символа позиция курсора перемещается к следующей ячейке символов; Однако поведение в конце строки зависит от оболочки буфера экрана консоли в режиме вывода EOL.

Дополнительные сведения о расположении курсора можно найти с помощью [последовательностей виртуальных терминалов](#), в частности в [категории состояния](#) запроса для поиска текущей позиции и [категории размещения](#) курсора для задания текущей позиции. Кроме того, приложение может использовать функцию [GetConsoleScreenBufferInfo](#) для определения текущей позиции курсора и функции [SetConsoleCursorPosition](#), чтобы задать положение курсора. Однако механизм последовательностей виртуальных терминалов предпочтителен для всех новых и текущих разработок. Дополнительные сведения о стратегии, лежащей в основе этого решения, можно найти в [классических функциях](#) и [документации по стратегии развития виртуального терминала и экосистемы](#).

Пример использования функций ввода-вывода консоли высокого уровня см. в разделе "[Использование высокоуровневых входных и выходных функций](#)".

# Использование высокоуровневых функций ввода и вывода

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли](#) и в [документе виртуального терминала](#).

В следующем примере используются функции ввода-вывода консоли высокого уровня для операций ввода-вывода консоли. Дополнительные сведения о функциях ввода-вывода консоли высокого уровня см. в разделе "[Высокоуровневый ввод-вывод](#) консоли".

В примере предполагается, что режимы ввода-вывода по умолчанию изначально применяются для первых вызовов функций `ReadFile` и `WriteFile`. Затем режим ввода изменяется, чтобы включить автономный входной режим и режим ввода эхо для второго вызова `ReadFile` и `WriteFile`. Функция `SetConsoleTextAttribute` используется для задания цветов, в которых будет отображаться последующий текст. Перед выходом программа восстанавливает исходный режим ввода консоли и атрибуты цвета.

Функция примера `NewLine` используется при отключении режима ввода строки. Он обрабатывает каретки, перемещая позицию курсора в первую ячейку следующей строки. Если курсор уже находится в последней строке буфера экрана консоли, содержимое буфера экрана консоли прокручивается вверх по одной строке.

C

```
#include <windows.h>

void NewLine(void);
void ScrollScreenBuffer(HANDLE, INT);
```

```
HANDLE hStdout, hStdin;
CONSOLE_SCREEN_BUFFER_INFO csbiInfo;

int main(void)
{
    LPSTR lpszPrompt1 = "Type a line and press Enter, or q to quit: ";
    LPSTR lpszPrompt2 = "Type any key, or q to quit: ";
    CHAR chBuffer[256];
    DWORD cRead, cWritten, fdwMode, fdwOldMode;
    WORD wOldColorAttrs;

    // Get handles to STDIN and STDOUT.

    hStdin = GetStdHandle(STD_INPUT_HANDLE);
    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hStdin == INVALID_HANDLE_VALUE ||
        hStdout == INVALID_HANDLE_VALUE)
    {
        MessageBox(NULL, TEXT("GetStdHandle"), TEXT("Console Error"),
                  MB_OK);
        return 1;
    }

    // Save the current text colors.

    if (! GetConsoleScreenBufferInfo(hStdout, &csbiInfo))
    {
        MessageBox(NULL, TEXT("GetConsoleScreenBufferInfo"),
                  TEXT("Console Error"), MB_OK);
        return 1;
    }

    wOldColorAttrs = csbiInfo.wAttributes;

    // Set the text attributes to draw red text on black background.

    if (! SetConsoleTextAttribute(hStdout, FOREGROUND_RED |
                                  FOREGROUND_INTENSITY))
    {
        MessageBox(NULL, TEXT("SetConsoleTextAttribute"),
                  TEXT("Console Error"), MB_OK);
        return 1;
    }

    // Write to STDOUT and read from STDIN by using the default
    // modes. Input is echoed automatically, and ReadFile
    // does not return until a carriage return is typed.
    //
    // The default input modes are line, processed, and echo.
    // The default output modes are processed and wrap at EOL.

    while (1)
    {
        if (! WriteFile(
            hStdout, // output handle
```

```

        lpszPrompt1,           // prompt string
        lstrlenA(lpszPrompt1), // string length
        &cWritten,             // bytes written
        NULL) )                // not overlapped
    {
        MessageBox(NULL, TEXT("WriteFile"), TEXT("Console Error"),
            MB_OK);
        return 1;
    }

    if (! ReadFile(
        hStdin,      // input handle
        chBuffer,    // buffer to read into
        255,         // size of buffer
        &cRead,       // actual bytes read
        NULL) )      // not overlapped
        break;
    if (chBuffer[0] == 'q') break;
}

// Turn off the line input and echo input modes

if (! GetConsoleMode(hStdin, &fdwOldMode))
{
    MessageBox(NULL, TEXT("GetConsoleMode"), TEXT("Console Error"),
        MB_OK);
    return 1;
}

fdwMode = fdwOldMode &
~(ENABLE_LINE_INPUT | ENABLE_ECHO_INPUT);
if (! SetConsoleMode(hStdin, fdwMode))
{
    MessageBox(NULL, TEXT("SetConsoleMode"), TEXT("Console Error"),
        MB_OK);
    return 1;
}

// ReadFile returns when any input is available.
// WriteFile is used to echo input.

NewLine();

while (1)
{
    if (! WriteFile(
        hStdout,          // output handle
        lpszPrompt2,      // prompt string
        lstrlenA(lpszPrompt2), // string length
        &cWritten,        // bytes written
        NULL) )          // not overlapped
    {
        MessageBox(NULL, TEXT("WriteFile"), TEXT("Console Error"),
            MB_OK);
        return 1;
    }
}

```

```
    }

    if (! ReadFile(hStdin, chBuffer, 1, &cRead, NULL))
        break;
    if (chBuffer[0] == '\r')
        NewLine();
    else if (! WriteFile(hStdout, chBuffer, cRead,
        &cWritten, NULL)) break;
    else
        NewLine();
    if (chBuffer[0] == 'q') break;
}

// Restore the original console mode.

SetConsoleMode(hStdin, fdwOldMode);

// Restore the original text colors.

SetConsoleTextAttribute(hStdout, wOldColorAttrs);

return 0;
}

// The NewLine function handles carriage returns when the processed
// input mode is disabled. It gets the current cursor position
// and resets it to the first cell of the next row.

void NewLine(void)
{
    if (! GetConsoleScreenBufferInfo(hStdout, &csbiInfo))
    {
        MessageBox(NULL, TEXT("GetConsoleScreenBufferInfo"),
            TEXT("Console Error"), MB_OK);
        return;
    }

    csbiInfo.dwCursorPosition.X = 0;

    // If it is the last line in the screen buffer, scroll
    // the buffer up.

    if ((csbiInfo.dwSize.Y-1) == csbiInfo.dwCursorPosition.Y)
    {
        ScrollScreenBuffer(hStdout, 1);
    }

    // Otherwise, advance the cursor to the next line.

    else csbiInfo.dwCursorPosition.Y += 1;

    if (! SetConsoleCursorPosition(hStdout,
        csbiInfo.dwCursorPosition))
    {
        MessageBox(NULL, TEXT("SetConsoleCursorPosition"),
            TEXT("SetConsoleCursorPosition")));
    }
}
```

```
        TEXT("Console Error"), MB_OK);
    return;
}

void ScrollScreenBuffer(HANDLE h, INT x)
{
    SMALL_RECT srctScrollRect, srctClipRect;
    CHAR_INFO chiFill;
    COORD coordDest;

    srctScrollRect.Left = 0;
    srctScrollRect.Top = 1;
    srctScrollRect.Right = csbiInfo.dwSize.X - (SHORT)x;
    srctScrollRect.Bottom = csbiInfo.dwSize.Y - (SHORT)x;

    // The destination for the scroll rectangle is one row up.

    coordDest.X = 0;
    coordDest.Y = 0;

    // The clipping rectangle is the same as the scrolling rectangle.
    // The destination row is left unchanged.

    srctClipRect = srctScrollRect;

    // Set the fill character and attributes.

    chiFill.Attributes = FOREGROUND_RED|FOREGROUND_INTENSITY;
    chiFill.Char.AsciiChar = (char)' ';

    // Scroll up one line.

    ScrollConsoleScreenBuffer(
        h,                  // screen buffer handle
        &srctScrollRect, // scrolling rectangle
        &srctClipRect,   // clipping rectangle
        coordDest,        // top left destination cell
        &chiFill);       // fill character and color
}
```

# Режимы консоли High-Level

Статья • 12.07.2023

На поведение высокоуровневых функций консоли влияет режимы ввода и вывода консоли. Все следующие режимы ввода консоли включены для входного буфера консоли при создании консоли:

- Режим ввода строки
- Режим обработанных входных данных
- Режим эхо-ввода

Оба следующих режима вывода консоли включены для буфера экрана консоли при его создании:

- Режим обработанных выходных данных
- Упаковка в режиме вывода EOL

Все три режима ввода, а также режим обработанных выходных данных предназначены для совместной работы. Лучше всего включить или отключить все эти режимы в группе. Если все включены, приложение находится в режиме готовности, что означает, что большая часть обработки обрабатывается для приложения. Если все отключены, приложение находится в режиме "необработанный", что означает, что входные данные не отфильтрованы, а любая обработка остается за приложением.

Приложение может использовать функцию [GetConsoleMode](#) для определения текущего режима входного буфера консоли или буфера экрана. Вы можете включить или отключить любой из этих режимов, используя следующие значения в функции [SetConsoleMode](#). Обратите внимание, что установка режима вывода для одного буфера экрана не влияет на режим вывода других буферов экрана.

Если в качестве входного дескриптора используется параметр *hConsoleHandle*, режим может быть одним из следующих. При создании консоли все режимы ввода, кроме **ENABLE\_WINDOW\_INPUT** и **ENABLE\_VIRTUAL\_TERMINAL\_INPUT**, включены по умолчанию.

Значение	Значение
<b>ENABLE_ECHO_INPUT</b> 0x0004	Символы, считанные функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , записываются в активный буфер экрана по мере ввода в консоли. Этот режим можно использовать только в том случае, если также включен режим <b>ENABLE_LINE_INPUT</b> .

Значение	Значение
ENABLE_INSERT_MODE 0x0020	Если этот режим включен, вводимый в окне консоли текст будет вставлен по текущему расположению курсора, а весь последующий текст не будет перезаписан. Если этот режим отключен, весь последующий текст будет перезаписан.
ENABLE_LINE_INPUT 0x0002	Функция <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> возвращает значение только в том случае, если считан символ возврата каретки. Если этот режим отключен, функции возвращают значение при доступности одного или нескольких символов.
ENABLE_MOUSE_INPUT 0x0010	Если указатель мыши находится в границах окна консоли и окно находится в фокусе для ввода текста с клавиатуры, события мыши, связанные с ее перемещением и нажатиями кнопок, помещаются во входной буфер. Такие события удаляются функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , даже если этот режим включен. Функцию <a href="#">ReadConsoleInput</a> можно использовать для считывания входных записей <a href="#">MOUSE_EVENT</a> из входного буфера.
ENABLE_PROCESSED_INPUT 0x0001	Нажатие клавиш CTRL+C обрабатывается системой и не помещается во входной буфер. Если входной буфер считывается функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , нажатия других управляющих клавиш обрабатываются системой и не возвращаются в буфере <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> . Если также включен режим <a href="#">ENABLE_LINE_INPUT</a> , символы стирания назад, возврата каретки и перевода строки обрабатываются системой.
ENABLE_QUICK_EDIT_MODE 0x0040	Этот флаг позволяет пользователю использовать мышь для выбора и редактирования текста. Чтобы включить этот режим, используйте <code>ENABLE_QUICK_EDIT_MODE   ENABLE_EXTENDED_FLAGS</code> . Чтобы отключить этот режим, используйте <code>ENABLE_EXTENDED_FLAGS</code> без этого флага.
ENABLE_WINDOW_INPUT 0x0008	Действия пользователей, которые приводят к изменению буфера экрана консоли, регистрируются во входном буфере консоли. Сведения о таких событиях могут быть считаны приложениями из входного буфера с помощью функции <a href="#">ReadConsoleInput</a> , но не могут быть считаны приложениями, использующими <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> .

Значение	Значение
ENABLE_VIRTUAL_TERMINAL_INPUT 0x0200	<p>Этот флаг указывает модулю обработки виртуального терминала преобразовать ввод пользователя, полученный в окне консоли, в <b>последовательности виртуального терминала консоли</b>, которые вспомогательное приложение может получить с помощью функций <a href="#">WriteFile</a> или <a href="#">WriteConsole</a>.</p> <p>Обычно этот флаг рекомендуется использовать вместе с флагом ENABLE_VIRTUAL_TERMINAL_PROCESSING для выходного дескриптора, чтобы обеспечить подключение к приложению, которое обменивается данными исключительно через последовательности виртуального терминала.</p>

Если в качестве дескриптора буфера экрана используется параметр *hConsoleHandle*, режим может быть одним из следующих. При создании буфера экрана оба режима вывода включены по умолчанию.

Значение	Значение
ENABLE_PROCESSED_OUTPUT 0x0001	<p>Символы, записываемые функцией <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> либо выводимые функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a>, анализируются для управляющих последовательностей ASCII, после чего выполняется нужное действие.</p> <p>Обрабатываются символы стирания назад, возврата каретки и перевода строки. Этот режим следует включить, если используются управляющие последовательности или задано значение <b>ENABLE_VIRTUAL_TERMINAL_PROCESSING</b>.</p>
ENABLE_WRAP_AT_EOL_OUTPUT 0x0002	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> либо выводе с помощью функции <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> курсор перемещается в начало следующей строки, если он достиг конца текущей строки. Это приводит к тому, что строки, отображаемые в окне консоли, автоматически прокручиваются вверх, если курсор переходит далее с последней строки в окне. Кроме того, содержимое буфера экрана консоли также прокручивается вверх (с удалением верхней строки в буфере экрана консоли), если курсор переходит далее с последней строки в буфере экрана консоли.</p> <p>Если этот режим отключен, последний символ в</p>

Значение	Значение
	строке будет перезаписан последующими символами.
<b>ENABLE_VIRTUAL_TERMINAL_PROCESSING</b> 0x0004	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> символы обрабатываются для VT100 и аналогичных управляющих символьных последовательностей, которые управляют перемещением курсора, цветом и режимом шрифта, а также другими операциями, доступными для выполнения через существующие API-интерфейсы консоли.</p> <p>Дополнительные сведения см. в статье <a href="#">Последовательности виртуального терминала консоли</a>.</p> <p>При использовании этого флага убедитесь, что установлен флаг <b>ENABLE_PROCESSED_OUTPUT</b>.</p>
<b>DISABLE_NEWLINE_AUTO_RETURN</b> 0x0008	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> к переносу в конце строки добавляется дополнительное состояние, которое задерживает перемещение курсора и помещает операции прокрутки в буфер.</p> <p>Как правило, если флаг <b>ENABLE_WRAP_AT_EOL_OUTPUT</b> установлен и текст достигает конца строки, курсор немедленно переходит на следующую строку, а содержимое буфера прокручивается на одну строку. Но если задан этот флаг, курсор не переходит на следующую строку, а операция прокрутки не выполняется. Записанный символ будет выведен в последней позиции строки, а курсор будет располагаться над этим символом (как в случае с отключенным флагом <b>ENABLE_WRAP_AT_EOL_OUTPUT</b>). Но следующий печатаемый символ будет выведен таким образом, как если бы флаг <b>ENABLE_WRAP_AT_EOL_OUTPUT</b> был включен. Перезапись при этом не выполняется. В частности, курсор быстро переходит на следующую строку, при необходимости выполняется прокрутка, символ выводится, а курсор передвигается еще на одну позицию.</p> <p>Обычно этот флаг рекомендуется использовать вместе с флагом <b>ENABLE_VIRTUAL_TERMINAL_PROCESSING</b>, что позволяет оптимально сымитировать эмулятор</p>

Значение	Значение
	терминала, в котором запись последнего символа на экране (в правом нижнем углу) без немедленной прокрутки является желаемым поведением.
ENABLE_LVB_GRID_WORLDWIDE 0x0010	<p>API-интерфейсы для записи атрибутов символов, в том числе <a href="#">WriteConsoleOutput</a> и <a href="#">WriteConsoleOutputAttribute</a>, позволяют использовать флаги <a href="#">атрибутов символов</a> для изменения цвета переднего плана и фона текста. Кроме того, некоторые флаги DBCS указываются с префиксом COMMON_LVB. Исторически эти флаги работали только в кодовых страницах DBCS для китайского, японского и корейского языков.</p> <p>За исключением флагов начальных и конечных байтов оставшиеся флаги, описывающие отрисовку строки и обратный видеовывод (смена местами цветов переднего плана и фона), можно использовать и с другими языками для выделения определенных частей выходных данных.</p> <p>Установка этого флага режима консоли позволяет использовать эти атрибуты для каждой кодовой страницы любого языка.</p> <p>По умолчанию он отключен для сохранения совместимости с известными приложениями, которые исторически игнорируют такие флаги на компьютерах без поддержки китайского, японского и корейского языков для хранения битов в таких полях (случайно или с собственными целями).</p> <p>Обратите внимание, что использование режима ENABLE_VIRTUAL_TERMINAL_PROCESSING может привести к установке флагов сетки LVB и обратного видеовывода, хотя этот флаг не включается, если подключенное приложение запрашивает видеовывод с подчеркиванием или инвертированием через <a href="#">последовательности виртуального терминала консоли</a>.</p>

# Высокоуровневая консоль ввода-вывода

Статья • 23.10.2023

Высокоуровневые функции ввода-вывода предоставляют простой способ чтения потока символов из входных данных консоли или записи потока символов в выходные данные консоли. Операция чтения высокого уровня получает входные символы из входного буфера консоли и сохраняет их в указанном буфере. Операция записи высокого уровня принимает символы из указанного буфера и записывает их в буфер экрана в текущем расположении курсора, перемещая курсор по мере записи каждого символа.

Высокоуровневый ввод-вывод предоставляет выбор между функциями [ReadFile](#) и [WriteFile](#), а также функциями [ReadConsole](#) и [WriteConsole](#). Они идентичны, за исключением двух важных различий. Функции консоли поддерживают использование символов Юникода или символов ANSI, заданных с помощью вариантов A и W каждой функции; Функции ввода-вывода файлов не поддерживают Юникод, за исключением UTF-8, заданных константой `CP_UTF8` для функций [SetConsoleCP](#) и [SetConsoleOutputCP](#) перед использованием. Кроме того, функции ввода-вывода файлов можно использовать для доступа к файлам, каналам и последовательным устройствам связи; Функции консоли можно использовать только с дескрипторами консоли. Это различие важно, если приложение использует стандартные дескрипторы, которые могли быть перенаправлены.

При использовании любого набора функций высокого уровня приложение может управлять цветами текста и фона, используемыми для отображения символов, которые впоследствии записываются в буфер экрана с предпочтаемым механизмом с помощью [последовательностей виртуальных терминалов](#).

Приложение также может использовать режимы консоли, влияющие на высокоуровневый ввод-вывод консоли, чтобы включить или отключить следующие свойства:

- Эхо ввода клавиатуры в активный буфер экрана
- Входные данные строки, в которых операция чтения не возвращается до нажатия клавиши ВВОД
- Автоматическая обработка ввода клавиатуры для обработки возврата каретки, CTRL+C и других входных данных
- Автоматическая обработка выходных данных для обработки упаковки линий, возврата каретки, внутренних пространств и других выходных сведений

Дополнительные сведения см. в следующих разделах:

- Консольные режимы
- Режимы консоли высокого уровня
- Функции ввода и вывода консоли высокого уровня
- Классические API и виртуальные последовательности терминалов

# Низкоуровневые функции ввода консоли

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использований для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли](#) и в [документе виртуального терминала](#).

Буфер низкоуровневых входных функций консоли содержит входные записи, которые могут включать сведения о клавиатуре, мыши, изменении размера буфера, фокусе и событиях меню. Низкоуровневые функции обеспечивают прямой доступ к входной буферу, в отличие от высокоуровневых функций, которые фильтруют и обрабатывают данные входного буфера карта все, кроме ввода клавиатуры.

Существует пять низкоуровневых функций для доступа к входной буферу консоли:

- [ReadConsoleInput](#)
- [PeekConsoleInput](#)
- [GetNumberOfConsoleInputEvents](#)
- [WriteConsoleInput](#)
- [FlushConsoleInputBuffer](#)

Функции [ReadConsoleInput](#), [PeekConsoleInput](#) и [WriteConsoleInput](#) используют [структурку INPUT\\_RECORD](#) для чтения из входного буфера или записи.

Ниже приведены описания низкоуровневых входных функций консоли.

Function	Description
<a href="#">ReadConsoleInput</a>	Считывает и удаляет входные записи из входного буфера. Функция не возвращается до тех пор, пока не будет доступна по крайней мере одна запись для чтения.

Function	Description
	Затем все доступные записи передаются в буфер вызывающего процесса, пока не будут доступны никакие записи или указанное количество записей было прочитано. Непрочитанные записи остаются в входном буфере для следующей операции чтения. Функция сообщает общее количество записей, которые были прочитаны. Пример использования <a href="#">ReadConsoleInput</a> см. в разделе <a href="#">"Чтение входных буферных событий"</a> .
<a href="#">PeekConsoleInput</a>	Считывает без удаления ожидающих входных записей в входном буфере. Все доступные записи до указанного числа копируются в буфер вызывающего процесса. Если записи недоступны, функция возвращается немедленно. Функция сообщает общее количество записей, которые были прочитаны.
<a href="#">GetNumberOfConsoleInputEvents</a>	Определяет количество непрочитанных входных записей в входном буфере.
<a href="#">WriteConsoleInput</a>	Помещает входные записи в входной буфер за любыми ожидающими записями в буфере. Входной буфер динамически растет, если это необходимо, чтобы хранить столько записей, сколько записываются. Чтобы использовать эту функцию, указанный дескриптор входного буфера должен иметь право <b>GENERIC_WRITE</b> доступа.
<a href="#">FlushConsoleInputBuffer</a>	Dis карты все непрочитанные события в входном буфере. Чтобы использовать эту функцию, указанный дескриптор входного буфера должен иметь право <b>GENERIC_WRITE</b> доступа.

Поток процесса приложения может выполнять операцию ожидания, чтобы ждать, пока входные данные будут доступны в входном буфере. Чтобы инициировать операцию ожидания, укажите дескриптор входного буфера в вызове любой из [функций](#) ожидания. Эти функции могут возвращаться при сигнале состояния одного или нескольких объектов. Состояние дескриптора входных данных консоли сигнализирует о наличии непрочитанных записей в входном буфере. Состояние сбрасывается до не сигнального, когда входной буфер становится пустым. Если нет входных данных, вызывающий поток вводит эффективное состояние ожидания, потребляя очень мало времени процессора, ожидая выполнения условий операции ожидания.

# ФУНКЦИИ ВЫВОДА КОНСОЛИ НИЗКОГО УРОВНЯ

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использований для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли](#) и в [документе виртуального терминала](#).

Функции вывода консоли низкого уровня обеспечивают прямой доступ к символьным ячейкам буфера экрана. Один набор функций считывает или записывает в последовательные ячейки, начиная с любого расположения в буфере экрана консоли. Другой набор функций считывает или записывает прямоугольные блоки ячеек.

Следующие функции считаются или записываются в указанное число последовательных ячеек символов в буфере экрана, начиная с указанной ячейки.

Function	Description
<a href="#">ReadConsoleOutputCharacter</a>	Копирует строку символов Юникода или ANSI из буфера экрана.
<a href="#">WriteConsoleOutputCharacter</a>	Записывает строку символов Юникода или ANSI в буфер экрана.
<a href="#">ReadConsoleOutputAttribute</a>	Копирует строку атрибутов цвета текста и фона из буфера экрана.
<a href="#">WriteConsoleOutputAttribute</a>	Записывает строку атрибутов цвета текста и фона в буфер экрана.
<a href="#">FillConsoleOutputCharacter</a>	Записывает один символ Юникода или ANSI в указанное число последовательных ячеек в буфере экрана.

Function	Description
<a href="#">FillConsoleOutputAttribute</a>	Записывает сочетание атрибута цвета текста и фона в указанное число последовательных ячеек в буфере экрана.

Для всех этих функций при обнаружении последней ячейки строки считывание или запись оболочки вокруг первой ячейки следующей строки. Когда появится последняя строка буфера экрана консоли, функции записи не карта все незаписанные символы или атрибуты, а функции чтения сообщают о количестве символов или атрибутов, фактически записанных.

Следующие функциичитываются из или записываются в прямоугольные блоки символьных ячеек в указанном расположении в буфере экрана.

Function	Description
<a href="#">ReadConsoleOutput</a>	Копирует данные символов и цветов из указанного блока ячеек буфера экрана в заданный блок в целевом буфере.
<a href="#">WriteConsoleOutput</a>	Записывает данные символа и цвета в указанный блок ячеек буфера экрана из заданного блока в исходном буфере.

Эти функции обрабатывают буферы экрана и исходные или конечные буферы как двухмерные массивы структур CHAR\_INFO ([содержащие данные атрибута символа](#) и цвета для каждой ячейки). Функции указывают ширину и высоту в ячейках символов исходного или целевого буфера, а указатель на буфер обрабатывается как указатель на ячейку источника (0,0) двухмерного массива. Функции используют [строку SMALL\\_RECT](#), чтобы указать, какой прямоугольник для доступа к буферу экрана консоли, а координаты левой верхней левой ячейки в исходном или целевом буфере определяют расположение соответствующего прямоугольника в этом буфере.

Эти функции автоматически закрепят указанный прямоугольник буфера экрана, чтобы поместиться в границы буфера экрана консоли. Например, если прямоугольник задает правые координаты нижнего справа (столбец 100, строка 50) и буфер экрана консоли составляет всего 80 столбцов, координаты обрезаются таким образом, чтобы они были (столбец 79, строка 50). Аналогичным образом этот настроенный прямоугольник снова обрезается, чтобы поместиться в границы исходного или целевого буфера. Указаны координаты буфера экрана фактического прямоугольника, считываемого или записанного. Пример использования этих функций см. в разделе "[Чтение и запись блоков символов и атрибутов](#)".

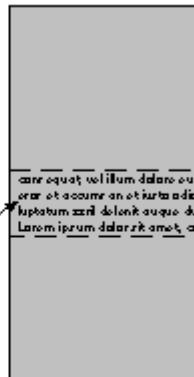
На рисунке показана [операция ReadConsoleOutput](#), в которой происходит вырезка, когда блок считывается из буфера экрана консоли, и снова при

копировании блока в целевой буфер. Функция сообщает фактический прямоугольник буфера экрана, скопированный из него.

### Screen buffer

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt lobortis nisl ut aliquam. Ut enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquam consetetur sadipscing elitr, sed diam nonummy nibh euismod tincidunt lobortis nisl ut aliquam consetetur sadipscing elitr. Ut enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquam consetetur sadipscing elitr. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusamus et iusto odio dignissim qui blanditiis praesentium voluptatum simili dolent aquo dui dolore te fugiat nulla facilisi. Nam liber tempor cum soluta nobis officia idem optime canique nihil imperdiet doming id quod maxim placet facere purius arum.  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt lobortis nisl ut aliquam. Ut enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquam consetetur sadipscing elitr, sed diam nonummy nibh euismod tincidunt lobortis nisl ut aliquam consetetur sadipscing elitr. Ut enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquam consetetur sadipscing elitr. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusamus et iusto odio dignissim qui blanditiis praesentium voluptatum simili dolent aquo dui dolore te fugiat nulla facilisi. Nam liber tempor cum soluta nobis officia idem optime canique nihil imperdiet doming id quod maxim placet facere purius arum.

### Destination buffer



Clipped region

Clipped region

# Низкоуровневый ввод-вывод консоли

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на **последовательности виртуальных терминалов** для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической **консоли и в документе виртуального терминала**.

Функции низкоуровневой консоли ввода-вывода расширяют управление приложениями над вводом-выводом консоли, обеспечивая прямой доступ к входным и экранным буферам консоли. Эти функции позволяют приложению выполнять следующие задачи:

- Получение входных данных о событиях изменения размера мыши и буфера
- Получение расширенных сведений о событиях ввода клавиатуры
- Запись входных записей в входной буфер
- Чтение входных записей без их удаления из входного буфера
- Определение количества ожидающих событий в входном буфере
- Очистка входного буфера
- Чтение и запись строк символов Юникода или ANSI в указанном расположении в буфере экрана
- Чтение и запись строк атрибутов цвета текста и фона в указанном расположении буфера экрана
- Чтение и запись прямоугольных блоков символьных и цветовых данных в указанном расположении буфера экрана
- Запись одного символа Юникода или ANSI или сочетания атрибутов цвета текста и фона в указанное число последовательных ячеек, начиная с указанного расположения буфера экрана

Дополнительные сведения см. в следующих разделах:

- Консольные режимы
- Режимы консоли низкого уровня

- Низкоуровневые функции ввода консоли
- Функции вывода консоли низкого уровня

# Режимы консоли низкого уровня

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Типы событий ввода, сообщаемые в входном буфере консоли, зависят от режимов ввода мыши и окна консоли. Обработанный режим ввода консоли определяет, как система обрабатывает сочетание клавиш CTRL+C. Чтобы задать или получить состояние входных режимов консоли, приложение может указать дескриптор входного буфера консоли в вызове [функции SetConsoleMode или GetConsoleMode](#). Следующие режимы используются с дескрипторами ввода консоли.

Режим	Description
ENABLE_MOUSE_INPUT	Определяет, передаются ли события мыши в входном буфере. По умолчанию входные данные мыши включены, а входные данные окна отключены. Изменение любого из этих режимов влияет только на входные данные, возникающие после установки режима; Ожидающие события мыши или окна в входном буфере не удаляются. Указатель мыши отображается независимо от режима мыши.
ENABLE_WINDOW_INPUT	Определяет, передаются ли события изменения размера буфера в входном буфере. По умолчанию входные данные мыши включены, а входные данные окна отключены. Изменение любого из этих режимов влияет только на входные данные, возникающие после установки режима; Ожидающие события мыши или окна в входном буфере не удаляются. Указатель мыши отображается независимо от режима мыши.
ENABLE_PROCESSED_INPUT	Управляет обработкой входных данных для приложений с помощью высокоуровневых функций ввода-вывода консоли.

Режим	Description
	<p>Однако если включен обработанный режим ввода, сочетание клавиш CTRL+C не сообщается в входном буфере консоли. Вместо этого он передается в соответствующую функцию обработчика элементов управления. Дополнительные сведения об обработчиках управления см. в разделе "<a href="#">Обработчики элементов управления консолью</a>".</p>

Режимы вывода буфера экрана не влияют на поведение низкоуровневых выходных функций.

# Чтение и запись блоков символов и атрибутов

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли](#) и в [документе виртуального терминала](#).

Функция [ReadConsoleOutput](#) копирует прямоугольный блок данных атрибута символа и цвета из буфера экрана консоли в целевой буфер. Функция обрабатывает буфер назначения как двухмерный массив [структур CHAR\\_INFO](#). [Аналогичным образом](#) функция [WriteConsoleOutput](#) копирует прямоугольный блок данных атрибута символа и цвета из исходного буфера в буфер экрана консоли. Дополнительные сведения о чтении или записи в прямоугольные блоки ячеек буфера экрана см. в разделе "[Методы ввода и вывода](#)".

В следующем примере функция [CreateConsoleScreenBuffer](#) [используется](#) для создания нового буфера экрана. [После того как функция SetConsoleActiveScreenBuffer](#) делает этот активный буфер экрана, блок символов и атрибутов цвета копируется из двух верхних двух строк буфера экрана STDOUT во временный буфер. Затем данные копируются из временного буфера в новый активный буфер экрана. После завершения работы приложения с помощью нового буфера экрана он вызывает [SetConsoleActiveScreenBuffer](#) для восстановления исходного буфера экрана STDOUT.

C

```
#include <windows.h>
#include <stdio.h>

int main(void)
{
    HANDLE hStdout, hNewScreenBuffer;
```

```
SMALL_RECT srctReadRect;
SMALL_RECT srctWriteRect;
CHAR_INFO chiBuffer[160]; // [2][80];
COORD coordBufSize;
COORD coordBufCoord;
BOOL fSuccess;

// Get a handle to the STDOUT screen buffer to copy from and
// create a new screen buffer to copy to.

hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
hNewScreenBuffer = CreateConsoleScreenBuffer(
    GENERIC_READ |           // read/write access
    GENERIC_WRITE,
    FILE_SHARE_READ |        //
    FILE_SHARE_WRITE,         // shared
    NULL,                    // default security attributes
    CONSOLE_TEXTMODE_BUFFER, // must be TEXTMODE
    NULL);                  // reserved; must be NULL
if (hStdout == INVALID_HANDLE_VALUE ||
    hNewScreenBuffer == INVALID_HANDLE_VALUE)
{
    printf("CreateConsoleScreenBuffer failed - (%d)\n", GetLastError());
    return 1;
}

// Make the new screen buffer the active screen buffer.

if (! SetConsoleActiveScreenBuffer(hNewScreenBuffer) )
{
    printf("SetConsoleActiveScreenBuffer failed - (%d)\n",
GetLastError());
    return 1;
}

// Set the source rectangle.

srctReadRect.Top = 0;      // top left: row 0, col 0
srctReadRect.Left = 0;
srctReadRect.Bottom = 1; // bot. right: row 1, col 79
srctReadRect.Right = 79;

// The temporary buffer size is 2 rows x 80 columns.

coordBufSize.Y = 2;
coordBufSize.X = 80;

// The top left destination cell of the temporary buffer is
// row 0, col 0.

coordBufCoord.X = 0;
coordBufCoord.Y = 0;

// Copy the block from the screen buffer to the temp. buffer.
```

```

fSuccess = ReadConsoleOutput(
    hStdout,           // screen buffer to read from
    chiBuffer,         // buffer to copy into
    coordBufSize,     // col-row size of chiBuffer
    coordBufCoord,   // top left dest. cell in chiBuffer
    &srctReadRect); // screen buffer source rectangle
if (! fSuccess)
{
    printf("ReadConsoleOutput failed - (%d)\n", GetLastError());
    return 1;
}

// Set the destination rectangle.

srctWriteRect.Top = 10;    // top lt: row 10, col 0
srctWriteRect.Left = 0;
srctWriteRect.Bottom = 11; // bot. rt: row 11, col 79
srctWriteRect.Right = 79;

// Copy from the temporary buffer to the new screen buffer.

fSuccess = WriteConsoleOutput(
    hNewScreenBuffer, // screen buffer to write to
    chiBuffer,       // buffer to copy from
    coordBufSize,   // col-row size of chiBuffer
    coordBufCoord,  // top left src cell in chiBuffer
    &srctWriteRect); // dest. screen buffer rectangle
if (! fSuccess)
{
    printf("WriteConsoleOutput failed - (%d)\n", GetLastError());
    return 1;
}
Sleep(5000);

// Restore the original active screen buffer.

if (! SetConsoleActiveScreenBuffer(hStdout))
{
    printf("SetConsoleActiveScreenBuffer failed - (%d)\n",
GetLastError());
    return 1;
}

return 0;
}

```

# Чтение данных о событиях входного буфера

Статья • 13.12.2023

Функцию `ReadConsoleInput` можно использовать для прямого доступа к входной буферу консоли. При создании консоли входные данные мыши включены, а входные данные окна отключены. Чтобы процесс получил все типы событий, в этом примере функция `SetConsoleMode` **позволяет** включить входные данные окна и мыши. Затем он переходит в цикл, который считывает и обрабатывает 100 событий ввода консоли. Например, сообщение "Событие клавиатуры" отображается, когда пользователь нажимает клавишу и отображается сообщение "Событие мыши" при взаимодействии пользователя с мышью.

C

```
#include <windows.h>
#include <stdio.h>

HANDLE hStdin;
DWORD fdwSaveOldMode;

VOID ErrorExit(LPCSTR);
VOID KeyEventProc(KEY_EVENT_RECORD);
VOID MouseEventProc(MOUSE_EVENT_RECORD);
VOID ResizeEventProc(WINDOW_BUFFER_SIZE_RECORD);

int main(VOID)
{
    DWORD cNumRead, fdwMode, i;
    INPUT_RECORD irInBuf[128];
    int counter=0;

    // Get the standard input handle.

    hStdin = GetStdHandle(STD_INPUT_HANDLE);
    if (hStdin == INVALID_HANDLE_VALUE)
        ErrorExit("GetStdHandle");

    // Save the current input mode, to be restored on exit.

    if (! GetConsoleMode(hStdin, &fdwSaveOldMode) )
        ErrorExit("GetConsoleMode");

    // Enable the window and mouse input events.

    fdwMode = ENABLE_WINDOW_INPUT | ENABLE_MOUSE_INPUT;
    if (! SetConsoleMode(hStdin, fdwMode) )
        ErrorExit("SetConsoleMode");
```

```
// Loop to read and handle the next 100 input events.

while (counter++ <= 100)
{
    // Wait for the events.

    if (! ReadConsoleInput(
        hStdin,          // input buffer handle
        irInBuf,         // buffer to read into
        128,             // size of read buffer
        &cNumRead) ) // number of records read
    ErrorExit("ReadConsoleInput");

    // Dispatch the events to the appropriate handler.

    for (i = 0; i < cNumRead; i++)
    {
        switch(irInBuf[i].EventType)
        {
            case KEY_EVENT: // keyboard input
                KeyEventProc(irInBuf[i].Event.KeyEvent);
                break;

            case MOUSE_EVENT: // mouse input
                MouseEventProc(irInBuf[i].Event.MouseEvent);
                break;

            case WINDOW_BUFFER_SIZE_EVENT: // scrn buf. resizing
                ResizeEventProc( irInBuf[i].Event.WindowBufferSizeEvent
);
                break;

            case FOCUS_EVENT: // disregard focus events
                break;

            case MENU_EVENT: // disregard menu events
                break;

            default:
                ErrorExit("Unknown event type");
                break;
        }
    }
}

// Restore input mode on exit.

SetConsoleMode(hStdin, fdwSaveOldMode);

return 0;
}

VOID ErrorExit (LPSTR lpszMessage)
{
    fprintf(stderr, "%s\n", lpszMessage);
```

```
// Restore input mode on exit.

SetConsoleMode(hStdin, fdwSaveOldMode);

ExitProcess(0);
}

VOID KeyEventProc(KEY_EVENT_RECORD ker)
{
    printf("Key event: ");

    if(ker.bKeyDown)
        printf("key pressed\n");
    else printf("key released\n");
}

VOID MouseEventProc(MOUSE_EVENT_RECORD mer)
{
#ifndef MOUSE_HWHEELED
#define MOUSE_HWHEELED 0x0008
#endif
    printf("Mouse event: ");

    switch(mer.dwEventFlags)
    {
        case 0:

            if(mer.dwButtonState == FROM_LEFT_1ST_BUTTON_PRESSED)
            {
                printf("left button press \n");
            }
            else if(mer.dwButtonState == RIGHTMOST_BUTTON_PRESSED)
            {
                printf("right button press \n");
            }
            else
            {
                printf("button press\n");
            }
            break;
        case DOUBLE_CLICK:
            printf("double click\n");
            break;
        case MOUSE_HWHEELED:
            printf("horizontal mouse wheel\n");
            break;
        case MOUSE_MOVED:
            printf("mouse moved\n");
            break;
        case MOUSE_WHEELED:
            printf("vertical mouse wheel\n");
            break;
        default:
            printf("unknown\n");
    }
}
```

```
        break;
    }
}

VOID ResizeEventProc(WINDOW_BUFFER_SIZE_RECORD wbsr)
{
    printf("Resize event\n");
    printf("Console screen buffer is %d columns by %d rows.\n",
wbsr.dwSize.X, wbsr.dwSize.Y);
}
```

# Очистка экрана

Статья • 23.10.2023

Существует три способа очистки экрана в консольном приложении.

## Пример 1

### 💡 Совет

Это рекомендуемый метод, использующий **последовательности виртуальных терминалов** для всех новых разработок. Дополнительные сведения см. в обсуждении классических **ИНТЕРФЕЙСов API консоли и последовательностей виртуальных терминалов**.

Первый метод — настроить приложение для последовательностей выходных данных виртуального терминала, а затем вызвать команду "очистить экран".

C

```
#include <windows.h>

int main(void)
{
    HANDLE hStdOut;

    hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

    // Fetch existing console mode so we correctly add a flag and not turn
    off others
    DWORD mode = 0;
    if (!GetConsoleMode(hStdOut, &mode))
    {
        return ::GetLastError();
    }

    // Hold original mode to restore on exit to be cooperative with other
    command-line apps.
    const DWORD originalMode = mode;
    mode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;

    // Try to set the mode.
    if (!SetConsoleMode(hStdOut, mode))
    {
        return ::GetLastError();
    }
```

```

// Write the sequence for clearing the display.
DWORD written = 0;
PCWSTR sequence = L"\x1b[2J";
if (!WriteConsoleW(hStdOut, sequence, (DWORD)wcslen(sequence), &written,
NULL))
{
    // If we fail, try to restore the mode on the way out.
    SetConsoleMode(hStdOut, originalMode);
    return ::GetLastError();
}

// To also clear the scroll back, emit L"\x1b[3J" as well.
// 2J only clears the visible window and 3J only clears the scroll back.

// Restore the mode on the way out to be nice to other command-line
applications.
SetConsoleMode(hStdOut, originalMode);

return 0;
}

```

Дополнительные варианты этой команды можно найти в документации по последовательности виртуальных терминалов в [разделе "Удаление в дисплее"](#).

## Пример 2

Второй метод — написать функцию, чтобы прокрутить содержимое экрана или буфера и задать заливку для обнаруженного пространства.

Это соответствует поведению командной строки `cmd.exe`.

```

C

#include <windows.h>

void cls(HANDLE hConsole)
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    SMALL_RECT scrollRect;
    COORD scrollTarget;
    CHAR_INFO fill;

    // Get the number of character cells in the current buffer.
    if (!GetConsoleScreenBufferInfo(hConsole, &csbi))
    {
        return;
    }

    // Scroll the rectangle of the entire buffer.
    scrollRect.Left = 0;

```

```

scrollRect.Top = 0;
scrollRect.Right = csbi.dwSize.X;
scrollRect.Bottom = csbi.dwSize.Y;

// Scroll it upwards off the top of the buffer with a magnitude of the
entire height.
scrollTarget.X = 0;
scrollTarget.Y = (SHORT)(0 - csbi.dwSize.Y);

// Fill with empty spaces with the buffer's default text attribute.
fill.Char.UnicodeChar = TEXT(' ');
fill.Attributes = csbi.wAttributes;

// Do the scroll
ScrollConsoleScreenBuffer(hConsole, &scrollRect, NULL, scrollTarget,
&fill);

// Move the cursor to the top left corner too.
csbi.dwCursorPosition.X = 0;
csbi.dwCursorPosition.Y = 0;

SetConsoleCursorPosition(hConsole, csbi.dwCursorPosition);
}

int main(void)
{
    HANDLE hStdout;

    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

    cls(hStdout);

    return 0;
}

```

## Пример 3

Третий метод — написать функцию для программного очистки экрана с помощью функций [FillConsoleOutputCharacter](#) и [FillConsoleOutputAttribute](#).

В следующем примере кода демонстрируется этот метод.

C

```

#include <windows.h>

void cls(HANDLE hConsole)
{
    COORD coordScreen = { 0, 0 };      // home for the cursor
    DWORD cCharsWritten;

```

```
CONSOLE_SCREEN_BUFFER_INFO csbi;
DWORD dwConSize;

// Get the number of character cells in the current buffer.
if (!GetConsoleScreenBufferInfo(hConsole, &csbi))
{
    return;
}

dwConSize = csbi.dwSize.X * csbi.dwSize.Y;

// Fill the entire screen with blanks.
if (!FillConsoleOutputCharacter(hConsole,           // Handle to console
screen buffer
                                (TCHAR)' ',        // Character to write
to the buffer
                                dwConSize,         // Number of cells to
write
                                coordScreen,       // Coordinates of first
cell
                                &cCharsWritten)) // Receive number of
characters written
{
    return;
}

// Get the current text attribute.
if (!GetConsoleScreenBufferInfo(hConsole, &csbi))
{
    return;
}

// Set the buffer's attributes accordingly.
if (!FillConsoleOutputAttribute(hConsole,           // Handle to console
screen buffer
                                csbi.wAttributes, // Character
attributes to use
                                dwConSize,         // Number of cells to
set attribute
                                coordScreen,       // Coordinates of
first cell
                                &cCharsWritten)) // Receive number of
characters written
{
    return;
}

// Put the cursor at its home coordinates.
SetConsoleCursorPosition(hConsole, coordScreen);
}

int main(void)
{
    HANDLE hStdout;
```

```
hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

cls(hStdout);

return 0;
}
```

# Прокрутка буфера экрана

Статья • 12.07.2023

## ⓘ Важно!

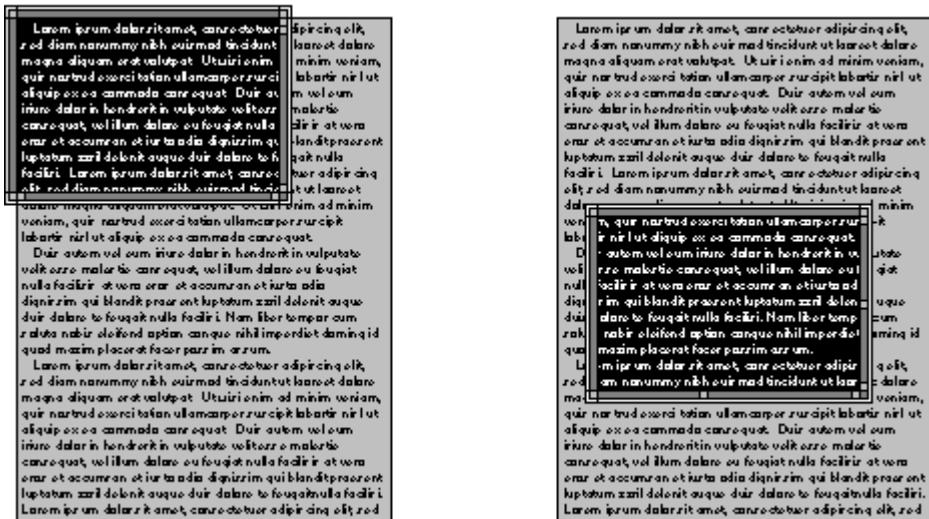
В этом документе описаны функции платформы консоли, которые больше не являются частью нашей [экосистемы](#). Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие варианты использования на неопределенный срок. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении можно найти в [классической консоли и в документе виртуального терминала](#).

В окне консоли отображается часть активного буфера экрана. Каждый буфер экрана поддерживает собственный прямоугольник текущего окна, указывающий координаты верхних левых и нижних правых символьных ячеек, которые будут отображаться в окне консоли. Чтобы определить текущий прямоугольник окна буфера экрана, используйте [командлет GetConsoleScreenBufferInfo](#). При создании буфера экрана верхний левый угол окна находится в левом верхнем углу буфера экрана консоли в (0,0).

Прямоугольник окна может изменяться для отображения различных частей буфера экрана консоли. Прямоугольник окна буфера экрана может измениться в следующих ситуациях:

- При [вызове Метода SetConsoleWindowInfo](#) для указания нового прямоугольника окна он прокручивает представление буфера экрана консоли, изменяя положение прямоугольника окна без изменения размера окна. Примеры прокрутки содержимого окна см. в разделе [Прокрутка окна буфера экрана](#).

## Window



## Screen buffer

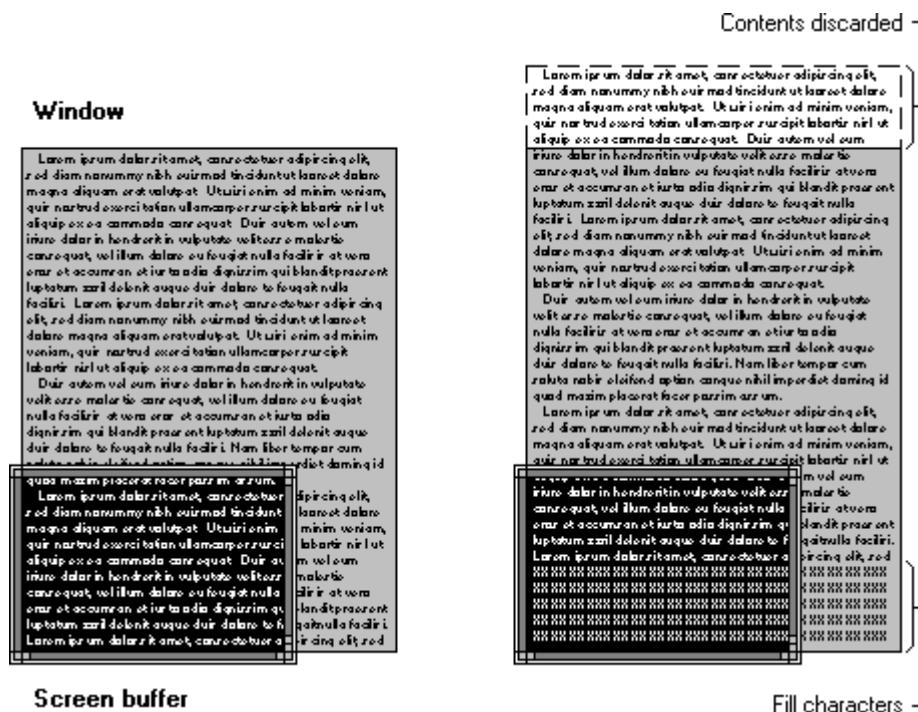
- При использовании функции **WriteFile** для записи в буфер экрана с включенной оболочкой в конце строки (EOL) прямоугольник окна сдвигается автоматически, поэтому курсор всегда отображается.
- Когда функция **SetConsoleCursorPosition** задает новое положение курсора, которое находится за пределами текущего прямоугольника окна, прямоугольник окна автоматически сдвигается для отображения курсора.
- Когда пользователь изменяет размер окна консоли или использует полосы прокрутки окна, прямоугольник окна активного буфера экрана может измениться. Это изменение не отображается как событие изменения размера окна во входном буфере.

В каждой из этих ситуаций прямоугольник окна сдвигается, чтобы отобразить другую часть буфера экрана консоли, но содержимое буфера экрана консоли остается в том же положении. Следующие ситуации могут привести к смещению содержимого буфера экрана консоли:

- При **вызове функции ScrollConsoleScreenBuffer** прямоугольный блок копируется из одной части буфера экрана в другую.
- При использовании **WriteFile** для записи в буфер экрана с включенной оболочкой в режиме вывода EOL содержимое буфера экрана консоли автоматически прокручивается при обнаружении конца буфера экрана консоли. При прокрутке верхняя строка буфера экрана консоли удаляется.

**ScrollConsoleScreenBuffer** указывает перемещаемый прямоугольник буфера экрана консоли и новые верхние левые координаты, в которые копируется прямоугольник. Эта функция может прокручивать часть или все содержимое буфера экрана консоли.

На рисунке показана операция **ScrollConsoleScreenBuffer**, которая прокручивает все содержимое буфера экрана консоли на несколько строк. Содержимое верхних строк удаляется, а нижние строки заполняются указанными символами и цветом.



Эффекты **ScrollConsoleScreenBuffer** можно ограничить, указав необязательный прямоугольник обрезки, чтобы содержимое буфера экрана консоли за пределами прямоугольника обрезки не изменялось. Результатом вырезки является создание подокна (прямоугольник обрезки), содержимое которого прокручивается без влияния на остальную часть буфера экрана консоли. Пример использования **ScrollConsoleScreenBuffer** см. в разделе [Прокрутка содержимого буфера экрана](#).

# Прокрутка содержимого буфера экрана

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использований для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Функция [ScrollConsoleScreenBuffer](#) перемещает блок символьных ячеек из одной части буфера экрана в другую часть того же буфера экрана. Функция задает верхние и нижние правые ячейки исходного прямоугольника, которые будут перемещены, и координаты назначения нового расположения для левой верхней ячейки. Данные символа и цвета в исходных ячейках перемещаются в новое расположение, а все ячейки, оставшиеся пустыми при перемещении, заполняются указанным символом и цветом. Если указан прямоугольник обрезки, ячейки за ее пределами остаются неизменными.

[ScrollConsoleScreenBuffer](#) можно использовать для удаления строки, указав координаты первой ячейки в строке в качестве координат назначения и указав прокручиваемый прямоугольник, содержащий все строки под линией.

В следующем примере показано использование прямоугольника с вырезкой для прокрутки только нижних 15 строк буфера экрана консоли. Строки в указанном прямоугольнике прокручиваются по одной строке одновременно, а верхняя строка блока не карта. Содержимое буфера экрана консоли за пределами прямоугольника вырезки остается неизменным.

C

```
#include <windows.h>
#include <stdio.h>
```

```
int main( void )
{
    HANDLE hStdout;
    CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
    SMALL_RECT srctScrollRect, srctClipRect;
    CHAR_INFO chiFill;
    COORD coordDest;
    int i;

    printf("\nPrinting 20 lines for reference. ");
    printf("Notice that line 6 is discarded during scrolling.\n");
    for(i=0; i<=20; i++)
        printf("%d\n", i);

    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

    if (hStdout == INVALID_HANDLE_VALUE)
    {
        printf("GetStdHandle failed with %d\n", GetLastError());
        return 1;
    }

    // Get the screen buffer size.

    if (!GetConsoleScreenBufferInfo(hStdout, &csbiInfo))
    {
        printf("GetConsoleScreenBufferInfo failed %d\n", GetLastError());
        return 1;
    }

    // The scrolling rectangle is the bottom 15 rows of the
    // screen buffer.

    srctScrollRect.Top = csbiInfo.dwSize.Y - 16;
    srctScrollRect.Bottom = csbiInfo.dwSize.Y - 1;
    srctScrollRect.Left = 0;
    srctScrollRect.Right = csbiInfo.dwSize.X - 1;

    // The destination for the scroll rectangle is one row up.

    coordDest.X = 0;
    coordDest.Y = csbiInfo.dwSize.Y - 17;

    // The clipping rectangle is the same as the scrolling rectangle.
    // The destination row is left unchanged.

    srctClipRect = srctScrollRect;

    // Fill the bottom row with green blanks.

    chiFill.Attributes = BACKGROUND_GREEN | FOREGROUND_RED;
    chiFill.Char.AsciiChar = (char)' ';

    // Scroll up one line.
```

```
if(!ScrollConsoleScreenBuffer(
    hStdout,           // screen buffer handle
    &srctScrollRect, // scrolling rectangle
    &srctClipRect,   // clipping rectangle
    coordDest,        // top left destination cell
    &chiFill))        // fill character and color
{
    printf("ScrollConsoleScreenBuffer failed %d\n", GetLastError());
    return 1;
}
return 0;
}
```

## См. также

[Прокрутка окна буфера экрана](#)

[Прокрутка буфера экрана](#)

# Прокрутка окна буфера экрана

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Функцию [SetConsoleWindowInfo](#) можно использовать для прокрутки содержимого буфера экрана в окне консоли. Эта функция также может изменить размер окна. Функция может указать новые верхние и правые углы окна экрана консоли в качестве абсолютных координат буфера экрана или указать изменения из текущих координат окна. Функция завершается ошибкой, если указанные координаты окна находятся вне границ буфера экрана консоли.

В следующем примере прокручивается представление буфера экрана консоли вверх, изменив координаты окна, возвращаемые [функцией GetConsoleScreenBufferInfo](#). Функция `ScrollByAbsoluteCoord` демонстрирует, как указать абсолютные координаты, а `ScrollByRelativeCoord` функция демонстрирует, как указать относительные координаты.

C

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

HANDLE hStdout;

int ScrollByAbsoluteCoord(int iRows)
{
    CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
    SMALL_RECT srctWindow;

    // Get the current screen buffer size and window position.
```

```

if (! GetConsoleScreenBufferInfo(hStdout, &csbiInfo))
{
    printf("GetConsoleScreenBufferInfo (%d)\n", GetLastError());
    return 0;
}

// Set srctWindow to the current window size and location.

srctWindow = csbiInfo.srWindow;

// Check whether the window is too close to the screen buffer top

if ( srctWindow.Top >= iRows )
{
    srctWindow.Top -= (SHORT)iRows;      // move top up
    srctWindow.Bottom -= (SHORT)iRows;   // move bottom up

    if (! SetConsoleWindowInfo(
        hStdout,                  // screen buffer handle
        TRUE,                     // absolute coordinates
        &srctWindow))           // specifies new location
    {
        printf("SetConsoleWindowInfo (%d)\n", GetLastError());
        return 0;
    }
    return iRows;
}
else
{
    printf("\nCannot scroll; the window is too close to the top.\n");
    return 0;
}

int ScrollByRelativeCoord(int iRows)
{
    CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
    SMALL_RECT srctWindow;

    // Get the current screen buffer window position.

    if (! GetConsoleScreenBufferInfo(hStdout, &csbiInfo))
    {
        printf("GetConsoleScreenBufferInfo (%d)\n", GetLastError());
        return 0;
    }

    // Check whether the window is too close to the screen buffer top

    if (csbiInfo.srWindow.Top >= iRows)
    {
        srctWindow.Top -= (SHORT)iRows;      // move top up
        srctWindow.Bottom -= (SHORT)iRows;   // move bottom up
        srctWindow.Left = 0;                // no change
        srctWindow.Right = 0;               // no change
    }
}

```

```

    if (! SetConsoleWindowInfo(
        hStdout,           // screen buffer handle
        FALSE,             // relative coordinates
        &srctWindow))     // specifies new location
    {
        printf("SetConsoleWindowInfo (%d)\n", GetLastError());
        return 0;
    }
    return iRows;
}
else
{
    printf("\nCannot scroll; the window is too close to the top.\n");
    return 0;
}
}

int main( void )
{
    int i;

    printf("\nPrinting twenty lines, then scrolling up five lines.\n");
    printf("Press any key to scroll up ten lines; ");
    printf("then press another key to stop the demo.\n");
    for(i=0; i<=20; i++)
        printf("%d\n", i);

    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

    if(ScrollByAbsoluteCoord(5))
        _getch();
    else return 0;

    if(ScrollByRelativeCoord(10))
        _getch();
    else return 0;
}

```

## См. также

[Прокрутка содержимого буфера экрана](#)

[Прокрутка буфера экрана](#)

# Сигналы CTRL+C и CTRL+BREAK

Статья • 12.07.2023

Сочетания клавиш `CTRL + C` и `CTRL + BREAK` получают специальную обработку, выполняемую консольными процессами. Если у окна консоли есть фокус клавиатуры, `CTRL + C` или `CTRL + BREAK` по умолчанию обрабатывается как сигнал (SIGINT или SIGBREAK), а не как ввод с клавиатуры. Эти сигналы по умолчанию передаются во все консольные процессы, подключенные к консоли. (Отключенные процессы не затрагиваются. См. статью [Создание консоли](#).) Система создает новый поток в каждом клиентском процессе для обработки события. Поток вызывает исключение, если выполняется отладка процесса. Отладчик может обработать исключение или продолжить работу с необработанным исключением.

Сочетание клавиш `CTRL + BREAK` всегда обрабатывается как сигнал, но приложение может изменить поведение `CTRL + C` по умолчанию двумя способами, чтобы предотвратить вызов функций обработчика:

- Функция [SetConsoleMode](#) может отключить режим ввода `ENABLE_PROCESSED_INPUT` для входного буфера консоли, поэтому `CTRL+C` отображается как ввод с клавиатуры, а не как сигнал.
- Если [SetConsoleCtrlHandler](#) вызывается со значениями `NULL` и `TRUE` для своих параметров, то вызывающий процесс игнорирует сигналы `CTRL+C`. Нормальная обработка `CTRL+C` восстанавливается путем вызова [SetConsoleCtrlHandler](#) со значениями `NULL` и `FALSE`. Этот атрибут игнорирования или неигнорирования сигналов `CTRL+C` наследуется дочерними процессами, но его можно включить или отключить в любом процессе, не затрагивая имеющиеся процессы.

Дополнительные сведения о том, как обрабатываются эти сигналы, в том числе время ожидания, см. в документации по функции обратного вызова [HandlerRoutine](#).

# CTRL+CLOSE Signal

Статья • 23.10.2023

Система создает сигнал CTRL+CLOSE, когда пользователь закрывает консоль. Все процессы, подключенные к консоли, получают сигнал, предоставляя каждому процессу возможность очистки перед завершением. Когда процесс получает этот сигнал, функция обработчика может выполнить одно из следующих действий после выполнения любых операций очистки:

- Вызов ExitProcess для завершения процесса.
- Возвращает значение **FALSE**. Если ни одна из зарегистрированных функций обработчика не возвращает значение **TRUE**, обработчик по умолчанию завершает процесс.
- Возвращает значение **TRUE**. В этом случае другие функции обработчика не вызываются и процесс завершается.

# Регистрация функции обработчика команд управления

Статья • 13.12.2023

## Базовый пример обработчика элементов управления

Это пример [функции SetConsoleCtrlHandler](#), которая используется для установки обработчика элементов управления.

При получении сигнала CTRL+C обработчик элемента управления возвращает **ЗНАЧЕНИЕ TRUE**, указывающее, что он обрабатывает сигнал. Это предотвращает вызов других обработчиков элементов управления.

При получении сигнала CTRL\_CLOSE\_EVENT обработчик элемента управления возвращает **ЗНАЧЕНИЕ TRUE**, а процесс завершается.

При получении сигнала CTRL\_BREAK\_EVENT, CTRL\_LOGOFF\_EVENT или CTRL\_SHUTDOWN\_EVENT обработчик управления возвращает **ЗНАЧЕНИЕ FALSE**. Это приводит к передаче сигнала в следующую функцию обработчика элементов управления. Если никакие другие обработчики управления не были зарегистрированы или ни один из зарегистрированных обработчиков не возвращает значение **TRUE**, будет использоваться обработчик по умолчанию, в результате чего процесс завершается.

### ⓘ Примечание

Вызов **AttachConsole**, **AllocConsole** или **FreeConsole** приведет к сбросу таблицы обработчиков элементов управления в клиентском процессе до исходного состояния. Обработчики необходимо повторно зарегистрировать при изменении сеанса присоединенной консоли.

C

```
// CtrlHandler.cpp : This file contains the 'main' function. Program
execution begins and ends there.
//

#include <windows.h>
#include <stdio.h>
```

```
BOOL WINAPI CtrlHandler(DWORD fdwCtrlType)
{
    switch (fdwCtrlType)
    {
        // Handle the CTRL-C signal.
    case CTRL_C_EVENT:
        printf("Ctrl-C event\n\n");
        Beep(750, 300);
        return TRUE;

        // CTRL-CLOSE: confirm that the user wants to exit.
    case CTRL_CLOSE_EVENT:
        Beep(600, 200);
        printf("Ctrl-Close event\n\n");
        return TRUE;

        // Pass other signals to the next handler.
    case CTRL_BREAK_EVENT:
        Beep(900, 200);
        printf("Ctrl-Break event\n\n");
        return FALSE;

    case CTRL_LOGOFF_EVENT:
        Beep(1000, 200);
        printf("Ctrl-Logoff event\n\n");
        return FALSE;

    case CTRL_SHUTDOWN_EVENT:
        Beep(750, 500);
        printf("Ctrl-Shutdown event\n\n");
        return FALSE;

    default:
        return FALSE;
    }
}

int main(void)
{
    if (SetConsoleCtrlHandler(CtrlHandler, TRUE))
    {
        printf("\nThe Control Handler is installed.\n");
        printf("\n -- Now try pressing Ctrl+C or Ctrl+Break, or");
        printf("\n     try logging off or closing the console...\n");
        printf("\n(...waiting in a loop for events...)\n\n");

        while (1) {}
    }
    else
    {
        printf("\nERROR: Could not set control handler");
        return 1;
    }
}
```

```
    return 0;  
}
```

## Пример скрытого окна прослушивания

Примечания, если загружается библиотека gdi32.dll или user32.dll, SetConsoleCtrlHandler не вызывается для событий CTRL\_LOGOFF\_EVENT и CTRL\_SHUTDOWN\_EVENT . Указанное решение — создать скрытое окно, если окно еще не существует, вызвав метод CreateWindowEx с параметром dwExStyle значение 0 и прослушивать сообщения WM\_QUERYENDSESSION и WM\_ENDSESSION окна. Если окно уже существует, добавьте два сообщения в существующую процедуру окна.

Дополнительные сведения о настройке окна и его цикле обмена сообщениями можно найти в разделе "Создание окна".

C++

```
// CtrlHandler.cpp : This file contains the 'main' function. Program  
execution begins and ends there.  
  
#include <Windows.h>  
#include <stdio.h>  
  
BOOL WINAPI CtrlHandler(DWORD fdwCtrlType)  
{  
    switch (fdwCtrlType)  
    {  
        // Handle the CTRL-C signal.  
        case CTRL_C_EVENT:  
            printf("Ctrl-C event\n\n");  
            Beep(750, 300);  
            return TRUE;  
  
        // CTRL-CLOSE: confirm that the user wants to exit.  
        case CTRL_CLOSE_EVENT:  
            Beep(600, 200);  
            printf("Ctrl-Close event\n\n");  
            return TRUE;  
  
        // Pass other signals to the next handler.  
        case CTRL_BREAK_EVENT:  
            Beep(900, 200);  
            printf("Ctrl-Break event\n\n");  
            return FALSE;  
  
        case CTRL_LOGOFF_EVENT:  
            Beep(1000, 200);
```

```
    printf("Ctrl-Logoff event\n\n");
    return FALSE;

case CTRL_SHUTDOWN_EVENT:
    Beep(750, 500);
    printf("Ctrl-Shutdown event\n\n");
    return FALSE;

default:
    return FALSE;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_QUERYENDSESSION:
        {
            // Check `lParam` for which system shutdown function and handle events.
            // See https://learn.microsoft.com/windows/win32/shutdown/wm-queryendsession
            return TRUE; // Respect user's intent and allow shutdown.
        }
        case WM_ENDSESSION:
        {
            // Check `lParam` for which system shutdown function and handle events.
            // See https://learn.microsoft.com/windows/win32/shutdown/wm-endsession
            return 0; // We have handled this message.
        }
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
}

int main(void)
{
    WNDCLASS sampleClass{ 0 };
    sampleClass.lpszClassName = TEXT("CtrlHandlerSampleClass");
    sampleClass.lpfnWndProc = WindowProc;

    if (!RegisterClass(&sampleClass))
    {
        printf("\nERROR: Could not register window class");
        return 2;
    }

    HWND hwnd = CreateWindowEx(
        0,
        sampleClass.lpszClassName,
        TEXT("Console Control Handler Sample"),

```

```
    0,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    nullptr,
    nullptr,
    nullptr,
    nullptr
);

if (!hwnd)
{
    printf("\nERROR: Could not create window");
    return 3;
}

ShowWindow(hwnd, SW_HIDE);

if (SetConsoleCtrlHandler(CtrlHandler, TRUE))
{
    printf("\nThe Control Handler is installed.\n");
    printf("\n -- Now try pressing Ctrl+C or Ctrl+Break, or");
    printf("\n      try logging off or closing the console...\n");
    printf("\n(...waiting in a loop for events...)\\n\\n");

    // Pump message loop for the window we created.
    MSG msg{};
    while (GetMessage(&msg, nullptr, 0, 0) > 0)
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return 0;
}
else
{
    printf("\nERROR: Could not set control handler");
    return 1;
}
}
```

# Последовательности виртуального терминала в консоли

Статья • 12.07.2023

Последовательности виртуальных терминалов — это управляющие последовательности символов, которые могут управлять перемещением курсора, цветом консоли и другими операциями при записи в поток вывода.

Последовательности можно также получить из входного потока в ответ на последовательность сведений запроса потока вывода или в виде кодировки входных данных пользователя (если установлен соответствующий режим).

Для настройки такого поведения можно использовать функции [GetConsoleMode](#) и [SetConsoleMode](#). В конце этого документа приведен пример предлагаемого способа включения поведения виртуальных терминалов.

Поведение следующих последовательностей основано на VT100 и производных технологиях эмулятора терминала, в частности эмулятора терминала xterm.

Дополнительные сведения о последовательностях терминала можно найти по адресу <http://vt100.net> и <http://invisible-island.net/xterm/ctlseqs/ctlseqs.html>.

## Выходные последовательности

Следующие последовательности терминалов перехватываются узлом консоли при записи в поток вывода, если флаг ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING установлен на дескрипторе буфера экрана с помощью функции [SetConsoleMode](#). Обратите внимание, что флаг DISABLE\_NEWLINE\_AUTO\_RETURN также может быть полезен при эмуляции положения курсора и прокрутки других эмуляторов терминала по отношению к символам, записанным в последний столбец в любой строке.

## Простое позиционирование курсора

Во всех следующих описаниях ESC всегда имеет шестнадцатеричное значение 0x1B. В последовательностях терминала не должно быть пробелов. Отдельные последовательности терминала могут быть разделены в любой позиции символов или байтов между несколькими последовательными вызовами [WriteFile](#) или [WriteConsole](#), но рекомендуется включать всю последовательность в один вызов. Чтобы узнать, как использовать эти последовательности на практике, ознакомьтесь с [примером](#) в конце этой статьи.

В следующей таблице описаны простые escape-последовательности с одной командой действия непосредственно после символа ESC. Эти последовательности не имеют параметров и вступают в силу немедленно.

Все команды в этой таблице обычно эквивалентны вызову API консоли [SetConsoleCursorPosition](#) для размещения курсора.

Перемещение курсора будет ограничено текущим окном просмотра в буфере. Прокрутка (если доступна) не будет выполняться.

Последовательность	Сокращение	Поведение
ESC M	RI	Обратный индекс — выполняет обратную операцию \n, перемещает курсор вверх на одну строку, сохраняет горизонтальное положение, при необходимости прокручивает буфер*
ESC 7	DECSC	Сохранение положения курсора в памяти**
ESC 8	DECSR	Восстановление положения курсора из памяти**

#### ① Примечание

\* Если заданы поля прокрутки, зарезервированный экземпляр внутри полей прокрутит только содержимое полей и оставит окно просмотра без изменений. (См. раздел "Поля прокрутки".)

\*\*Значение не будет сохранено в памяти до первого использования команды сохранения. Единственный способ получить доступ к сохраненному значению — выполнить команду восстановления.

## Позиционирование курсора

В указанных ниже таблицах приведены последовательности типа CSI (начала управляющей последовательности). Все последовательности CSI начинаются с ESC (0x1B) и [ (левая скобка, 0x5B) и могут содержать параметры переменной длины, чтобы указать дополнительные сведения для каждой операции. Это будет представлено в виде сокращения <n>. Каждая таблица ниже сгруппирована по функциональности. Ниже таблицы указаны примечания, объясняющие, как работает эта группа.

Для всех параметров применяются следующие правила, если не указано иное:

- <n> представляет расстояние для перемещения и является необязательным параметром;
- если параметр <n> опущен или равен 0, он будет рассматриваться как 1;
- <n> не может превышать 32 767 (максимальное короткое значение);
- <n> не может быть отрицательным.

Все команды в этом разделе обычно эквивалентны вызову API консоли

### [SetConsoleCursorPosition](#).

Перемещение курсора будет ограничено текущим окном просмотра в буфере. Прокрутка (если доступна) не будет выполняться.

Последовательность	Код	Описание	Поведение
ESC [ <n> A	CUU	Перемещение курсора вверх	Перемещение курсора вверх на <n> позиций
ESC [ <n> B	CUD	Перемещение курсора вниз	Перемещение курсора вниз на <n> позиций
ESC [ <n> C	CUF	Перемещение курсора вперед	Перемещение курсора вперед (вправо) на <n> позиций
ESC [ <n> D	CUB	Перемещение курсора назад	Перемещение курсора назад (влево) на <n> позиций
ESC [ <n> E	CNL	Перемещение курсора на следующую строку	Перемещение курсора вниз на <n> строк от текущей позиции
ESC [ <n> F	CPL	Перемещение курсора на предыдущую строку	Перемещение курсора вверх на <n> строк от текущей позиции
ESC [ <n> G	CHA	Перемещение курсора по горизонтали (абсолютное позиционирование)	Курсор перемещается на <n>-ю позицию по горизонтали в текущей строке
ESC [ <n> d	VPA	Вертикальное положение строки (абсолютное позиционирование)	Курсор перемещается на <n>-ю позицию по вертикали в текущем столбце
ESC [ <y> ; <x> H	CUP	Позиция курсора	*Курсор перемещается в x<>; <Координата y> в

Последовательность	Код	Описание	Поведение
			окне просмотра, где <х> — это столбец <линии у>.
ESC [ <y> ; <x> f	HVP	Горизонтально-вертикальная позиция	*Курсор перемещается в x<>; <Координата у> в окне просмотра, где <х> — это столбец <линии у>.
ESC [ s	ANSISYSSC	Сохранение курсора — эмуляция Ansi.sys	**Без параметров выполняет операцию сохранения курсора, например DECSC

### ① Примечание

Параметры \*<х и у> имеют те же ограничения, что <и <n>> выше. Если параметры <х> и <у> не указаны, им будут присвоены значения 1;1.

\*\*ANSI.sys историческую документацию можно найти по адресу <https://msdn.microsoft.com/library/cc722862.aspx> и реализована для удобства и совместимости.

## Видимость курсора

Следующие команды управляют видимостью курсора и его мигающим состоянием. Последовательности DECTCEM обычно эквивалентны вызову API консоли [SetConsoleCursorInfo](#) для переключения видимости курсора.

Последовательность	Код	Описание	Поведение
ESC [ ? 12 h	ATT160	Текстовый курсор: включение мигания	Запуск мигания курсора
ESC [ ? 12 l	ATT160	Текстовый курсор: отключение мигания	Остановка мигания курсора
ESC [ ? 25 h	DECTCEM	Текстовый курсор: включение режима отображения	Отображение курсора

Последовательность	Код	Описание	Поведение
ESC [ ? 25 l	DECTCEM	Текстовый курсор: включение скрытого режима	Скрытие курсора

### 💡 Совет

Последовательности включения заканчиваются символом строчной буквы Н (h), а последовательности отключения — символом L (l).

## Фигура курсора

Следующие команды управляют и позволяют настраивать фигуру курсора.

Последовательность	Код	Описание	Поведение
ESC [ 0 SP q	DECSCUSR	Фигура пользователя	Фигура курсора по умолчанию, настроенная пользователем
ESC [ 1 SP q	DECSCUSR	Мигающий блок	Мигающая фигура курсора блока
ESC [ 2 SP q	DECSCUSR	Устойчивый блок	Фигура устойчивого блочного курсора
ESC [ 3 SP q	DECSCUSR	Мигание подчеркивания	Мигающая фигура курсора подчеркивания
ESC [ 4 SP q	DECSCUSR	Устойчивое подчеркивание	Фигура курсора устойчивого подчеркивания
ESC [ 5 SP q	DECSCUSR	Мерцающая полоса	Фигура курсора мигает линейчатой полосой
ESC [ 6 SP q	DECSCUSR	Устойчивый столбик	Фигура курсора устойчивой линейчатой полосы

### ⚠ Примечание

SP — символ пробела (0x20) в промежуточной позиции, за которым следует q (0x71) в конечной позиции.

## Позиционирование окна просмотра

Все команды в этом разделе обычно эквивалентны вызову API консоли [ScrollConsoleScreenBuffer](#) для перемещения содержимого буфера консоли.

**Внимание!** Имена команд недостоверны. Прокрутка — это направление, в котором текст перемещается во время операции, а не возможное направление перемещения окна просмотра.

Последовательность	Код	Описание	Поведение
ESC [ <n> S	Kоличество SU	Прокрутка вверх	Прокрутка текста вверх на <n> позиций. Это действие также называется сдвигом вниз, новые строки появляются в нижней части экрана.
ESC [ <n> T	SD	Прокрутка вниз	Прокрутка вниз на <n> позиций. Это действие также называется сдвигом вверх, новые строки появляются в верхней части экрана.

Текст перемещается, начиная со строки, в которой расположен курсор. Если курсор находится в средней строке окна просмотра, то прокрутка вверх приведет к перемещению нижней половины окна просмотра и вставке пустых строк в нижней части. Прокрутка вниз приведет к перемещению верхней половины строк окна просмотра и вставке новых строк в верхней части.

Кроме того, обратите внимание, что прокрутка вверх и вниз также влияет на поля прокрутки. Прокрутка вверх и вниз не влияет на строки за пределами полей прокрутки.

Значение по умолчанию для <n> равно 1 (его можно опустить).

## Изменение текста

Все команды в этом разделе обычно эквивалентны вызову API-интерфейсов консоли [FillConsoleOutputCharacter](#), [FillConsoleOutputAttribute](#) и [ScrollConsoleScreenBuffer](#) для изменения содержимого буфера текста.

Последовательность	Код	Описание	Поведение
ESC [ <n> @	ICH	Вставка символа	Вставка <n> пробелов в текущую позицию курсора со сдвигом всего существующего текста вправо. Текст, который выходит за экран справа, удаляется.
ESC [ <n> P	DCH	Удаление символа	Удаление <n> символов в текущей позиции курсора со сдвигом пробелов с правого края

Последовательность	Код	Описание	Поведение
ESC [ <n> X	ECH	Стереть символ	Очистка <n> символов из текущей позиции курсора с заменой их символом пробела.
ESC [ <n> L	IL	Вставить строки	Вставка <n> строк в буфер в позицию курсора. Стока, на которой находится курсор, и строки под ней будут сдвинуты вниз.
ESC [ <n> M	DL	Удалить строку	Удаляет <n> строк из буфера, начиная со строки, в которой находится курсор.

### ① Примечание

Для IL и DL затрагиваются только строки в полях прокрутки (см. раздел "Поля прокрутки"). Если поля не заданы, то границами полей по умолчанию является текущее окно просмотра. Если линии будут сдвинуты ниже полей, они удалятся. При удалении строк в нижней части полей вставляются пустые строки, а строки за пределами окна просмотра никогда не затрагиваются.

Значение по умолчанию для <n> для каждой последовательности равно 0 (если оно опущено).

Для следующих команд параметр <n> имеет три допустимых значения:

- 0 — очищает все данные с текущей позиции курсора (включительно) до конца строки или экрана;
- 1 — очищает все данные с начала строки или экрана до текущей позиции курсора (включительно);
- 2 — очищает всю строку или экран.

Последовательность	Код	Описание	Поведение
ESC [ <n> J	ED	Стереть на экране	Заменяет весь текст в текущем окне просмотра или на экране, указанный параметром <n>, пробелами.
ESC [ <n> K	EL	Стереть в строке	Заменить весь текст в строке с курсором, указанный параметром <n>, пробелами

## Форматирование текста

Все команды в этом разделе обычно эквивалентны вызову API консоли [SetConsoleTextAttribute](#) для изменения форматирования всех будущих операций

записи в буфер текста, выводимого на консоль.

Отличие этой команды заключается в том, что позиция `<n>`, указанная ниже, может принимать от 0 до 16 параметров, разделенных точкой с запятой.

Если параметры не указаны, все они обрабатываются как единственный параметр 0.

Последовательность	Код	Описание	Поведение
ESC [ <n> m	SGR	Установка графического отображения	Установка формата экрана и текста, как указано <code>&lt;n&gt;</code>

Указанная ниже таблица значений может использоваться для `<n>`, чтобы отобразить различные режимы форматирования.

Режимы форматирования применяются слева направо. Если применить конкурирующие параметры форматирования, крайний правый параметр будет приоритетным.

Для параметров, задающих цвета, будут использованы цвета, указанные в таблице цветов консоли, которую можно изменить с помощью API

[SetConsoleScreenBufferInfoEx](#). Если таблица изменена так, что "синяя" позиция в таблице отображает оттенок красного цвета RGB, то все вызовы **синего цвета переднего плана** будут отображать этот красный цвет до тех пор, пока не будет указано иное.

Значение	Описание	Поведение
0	По умолчанию	Возвращает все атрибуты в состояние по умолчанию до изменения
1	Глубокий/яркий	Применяет флаг яркости/интенсивности к цвету переднего плана
22	Неглубокий/неяркий	Удаляет флаг яркости/интенсивности для цвета переднего плана
4	Underline	Добавляет подчеркивание
24	Нет подчеркивания	Удаляет подчеркивание
7	Отрицательное число	Меняет местами цвет переднего плана и фона
27	Позитив (не негатив)	Возвращает передний план и фон в обычное состояние

<b>Значение</b>	<b>Описание</b>	<b>Поведение</b>
30	Черный цвет переднего плана	Применяет неглубокий/яркий черный цвет для переднего плана
31	Красный цвет переднего плана	Применяет неглубокий/яркий красный цвет для переднего плана
32	Зеленый цвет переднего плана	Применяет неглубокий/яркий зеленый цвет для переднего плана
33	Желтый цвет переднего плана	Применяет неглубокий/яркий желтый цвет для переднего плана
34	Синий цвет переднего плана	Применяет неглубокий/яркий синий цвет для переднего плана
35	Пурпурный цвет переднего плана	Применяет неглубокий/яркий пурпурный цвет для переднего плана
36	Голубой цвет переднего плана	Применяет неглубокий/яркий голубой цвет для переднего плана
37	Белый цвет переднего плана	Применяет неглубокий/яркий белый цвет для переднего плана
38	Дополнительные цвета для переднего плана	Применяет дополнительные значения цвета для переднего плана (см. сведения ниже)
39	Цвет текста по умолчанию для переднего плана	Применяет для переднего плана только цвета по умолчанию (см. 0)
40	Черный цвет фона	Применяет неглубокий/яркий черный цвет для фона
41	Красный цвет фона	Применяет неглубокий/яркий красный цвет для фона
42	Зеленый цвет фона	Применяет неглубокий/яркий зеленый цвет для фона
43	Желтый цвет фона	Применяет неглубокий/яркий желтый цвет для фона
44	Синий цвет фона	Применяет неглубокий/яркий синий цвет для фона
45	Пурпурный цвет фона	Применяет неглубокий/яркий пурпурный цвет для фона
46	Голубой цвет фона	Применяет неглубокий/яркий голубой цвет для фона

<b>Значение</b>	<b>Описание</b>	<b>Поведение</b>
47	Белый цвет фона	Применяет неглубокий/яркий белый цвет для фона
48	Дополнительные цвета для фона	Применяет дополнительные значения цвета для фона (см. сведения ниже)
49	Цвет текста по умолчанию для фона	Применяет для фона только цвета по умолчанию (см. 0)
90	Ярко-черный цвет переднего плана	Применяет глубокий/яркий черный цвет для переднего плана
91	Ярко-красный цвет переднего плана	Применяет глубокий/яркий красный цвет для переднего плана
92	Ярко-зеленый цвет переднего плана	Применяет глубокий/яркий зеленый цвет для переднего плана
93	Ярко-желтый цвет переднего плана	Применяет глубокий/яркий желтый цвет для переднего плана
94	Ярко-синий цвет переднего плана	Применяет глубокий/яркий синий цвет для переднего плана
95	Ярко-пурпурный цвет переднего плана	Применяет глубокий/яркий пурпурный цвет для переднего плана
96	Ярко-голубой цвет переднего плана	Применяет глубокий/яркий голубой цвет для переднего плана
97	Ярко-белый цвет переднего плана	Применяет глубокий/яркий белый цвет для переднего плана
100	Ярко-черный цвет фона	Применяет глубокий/яркий черный цвет для фона
101	Ярко-красный цвет фона	Применяет глубокий/яркий красный цвет для фона
102	Ярко-зеленый цвет фона	Применяет глубокий/яркий зеленый цвет для фона
103	Ярко-желтый цвет фона	Применяет глубокий/яркий желтый цвет для фона
104	Ярко-синий цвет фона	Применяет глубокий/яркий синий цвет для фона
105	Ярко-пурпурный цвет фона	Применяет глубокий/яркий пурпурный цвет для фона

<b>Значение</b>	<b>Описание</b>	<b>Поведение</b>
106	Ярко-голубой цвет фона	Применяет глубокий/яркий голубой цвет для фона
107	Ярко-белый цвет фона	Применяет глубокий/яркий белый цвет для фона

## Дополнительные цвета

Некоторые эмуляторы виртуального терминала поддерживают палитру, превышающую 16 цветов, которые предоставляет консоль Windows. Для этих дополнительных цветов консоль Windows выберет ближайший подходящий цвет из существующей 16-цветной таблицы для отображения. В отличие от приведенных выше стандартных значений SGR, после первоначального обозначения дополнительные значения будут использовать дополнительные параметры в соответствии с таблицей ниже.

<b>Подпоследовательность</b>	<b>Описание</b>
<b>SGR</b>	
38 ; 2 ; <r> ; <g> ; <b>	Задайте для цвета переднего плана значение RGB, указанное в <параметрах r>, <g>, <b> *
48 ; 2 ; <r> ; <g> ; <b>	Задайте для цвета фона значение RGB, указанное в <параметрах r>, <g>, <b> *
38 ; 5 ; <s>	Задайте для цвета <переднего плана индекс s> в таблице цветов 88 или 256*
48 ; 5 ; <s>	Задайте для цвета <фона индекс s> в таблице цветов 88 или 256*

\*Цветовые палитры 88 и 256, поддерживаемые внутренне для сравнения, основаны на эмуляторе терминала xterm. В настоящее время невозможно изменить таблицы сравнения/округления.

## Цвета экрана

Приведенная ниже команда позволяет приложению установить для значения палитры цветов экрана любое значение RGB.

Значения RGB должны быть в шестнадцатеричном формате в диапазоне от `0` до `ff` и разделяться символом косой черты (например, `rgb:1/24/86`).

Обратите внимание, что эта последовательность является последовательностью OSC "Команда операционной системы", а не CSI, как многие другие перечисленные последовательности, и поэтому начинается с "\x1b]", а не "\x1b[". Как последовательности OSC, они заканчиваются строковым признаком конца , представленным как `<ST>` и передаваемым с помощью `ESC \` (0x1B 0x5C). Вместо признака конца строки можно использовать `BEL` (0x7), но рекомендуется использовать более длинную форму.

Последовательность	Описание	Поведение
<code>ESC ] 4 ; &lt;i&gt; ; rgb : &lt;r&gt; / &lt;g&gt; / &lt;b&gt;&lt;ST&gt;</code>	Изменение цветов экрана	Задает индекс палитры цветов экрана <code>&lt;i&gt;</code> для значений RGB, указанных в <code>&lt;r&gt;</code> , <code>&lt;g&gt;</code> , <code>&lt;b&gt;</code>

## Изменения режима

Существуют последовательности, управляющие режимами ввода. Существует два разных набора режимов ввода: режим клавиш курсора и режим клавиш дополнительной клавиатуры. Режим клавиш курсора управляет последовательностями, которые сформированы клавишами со стрелками, а также клавишами Home и End. Режим клавиш дополнительной клавиатуры управляет в основном последовательностями, созданными клавишами цифровой клавиатуры, а также функциональными клавишами.

Каждый из этих режимов представляет собой простые логические параметры: режим клавиш курсора — "Обычный" (по умолчанию) или "Приложение", а режим клавиш дополнительной клавиатуры — "Числовой" (по умолчанию) или "Приложение".

Последовательности, создаваемые в этих режимах, см. в разделах Клавиши курсора и Функциональные клавиши NumPad & .

Последовательность	Код	Описание	Поведение
<code>ESC =</code>	<code>DECKPAM</code>	Включение режима приложения для дополнительной клавиатуры	Клавиши дополнительной клавиатуры будут создавать последовательности в режиме приложения.
<code>ESC &gt;</code>	<code>DECKPNM</code>	Включение числового режима для дополнительной клавиатуры	Клавиши дополнительной клавиатуры будут создавать последовательности в числовом режиме.

Последовательность	Код	Описание	Поведение
ESC [ ? 1 ч	DECCKM	Включение режима приложения для клавиш курсора	Клавиши дополнительной клавиатуры будут создавать последовательности в режиме приложения.
ESC [ ? 1 l	DECCKM	Выключение режима приложения для клавиш курсора (использование обычного режима)	Клавиши дополнительной клавиатуры будут создавать последовательности в числовом режиме.

## Состояние запроса

Все команды в этом разделе обычно эквивалентны вызову API консоли Get\* для получения сведений о состоянии текущего состояния буфера консоли.

### ⓘ Примечание

Эти запросы будут выдавать свои ответы во входной поток консоли сразу после распознавания в выходном потоке при настройке ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING. Флаг ENABLE\_VIRTUAL\_TERMINAL\_INPUT не применяется к командам запроса, так как предполагается, что приложение, выполняющее запрос, всегда будет получать ответ.

Последовательность	Код	Описание	Поведение
ESC [ 6 n	DECXCPR	Отчет о позиции курсора	Выведите позицию курсора в виде: ESC [ <r> ; <c> R Где <r> = строка курсора и <c> = столбец курсора
ESC [ 0 c	DA	Атрибуты устройства	Сообщает об идентификаторе терминала. Выдаст "\x1b[?1;0c", указывающее "VT101 без параметров".

## Символы табуляции

Хотя консоль Windows традиционно ожидает, что вкладки будут содержать только восемь символов в ширину, приложения \*nix, использующие определенные последовательности, могут управлять расположением табуляции в окнах консоли для оптимизации перемещения курсора приложением.

Следующие последовательности позволяют приложению задавать расположения позиции табуляции в окне консоли, удалять их, а также перемещаться между ними.

<b>Последовательность</b>	<b>Код</b>	<b>Описание</b>	<b>Поведение</b>
ESC H	HTS	Установка горизонтальной табуляции	Задает позицию табуляции в текущем столбце, в котором находится курсор.
ESC [ <n> I	CHT	Горизонтальная табуляция курсора (вперед)	Перемещение курсора к следующему столбцу (в той же строке) в соответствии с позицией табуляции. Если больше нет позиций табуляции, выполняется перемещение к последнему столбцу в строке. Если курсор находится в последнем столбце, выполняется перемещение к первому столбцу следующей строки.
ESC [ <n> Z	CBT	Обратная табуляция курсора	Перемещение курсора к предыдущему столбцу (в той же строке) в соответствии с позицией табуляции. Если больше нет позиций табуляции, выполняется перемещение курсора к первому столбцу. Если курсор находится в первом столбце, то он не перемещается.
ESC [ 0 g	TBC	Очистка позиции табуляции (текущий столбец)	Очищает позицию табуляции в текущем столбце, если она есть. В противном случае ничего не происходит.
ESC [ 3 g	TBC	Очистка позиции табуляции (все столбцы)	Очищает все установленные позиции табуляции.

- Для CHT и CBT <n> является необязательным параметром (по умолчанию равно 1), указывающим, сколько раз следует передвинуть курсор в указанном направлении.
- Если позиции табуляции не заданы через HTS, то CHT и CBT будут рассматривать первый и последний столбцы окна как единственныe позиции табуляции.
- Использование HTS для установки табуляции также приведет к переходу консоли к следующей позиции табуляции на выходе символа TAB (0x09, "\t") так же, как и CHT.

## Назначение набора символов

Следующие последовательности позволяют программе изменять сопоставление активного набора символов. Это позволяет программе выдавать 7-битные символы кодировки ASCII, но отображать их как другие глифы на экране терминала. В настоящее время единственными поддерживаемыми наборами символов являются ASCII (по умолчанию) и специальный набор графических символов в кодировке DEC. Список всех символов, представленных в наборе графических символов в кодировке DEC, см. по этому адресу <http://vt100.net/docs/vt220-rm/table2-4.html>.

<b>Последовательность</b>	<b>Описание</b>	<b>Поведение</b>
ESC ( 0	Назначение набора символов — рисование линии DEC	Включает режим рисования линии DEC
ESC ( B	Назначение набора символов — US ASCII	Включает режим ASCII (по умолчанию)

Обратите внимание, что режим рисования линии DEC используется для очерчивания границ в консольных приложениях. В следующей таблице показано, какой символ ASCII соответствует символу рисования линии.

<b>Hex</b>	<b>ASCII</b>	<b>Рисование линии DEC</b>
0x6a	j	↙
0x6b	k	↖
0x6c	l	Γ
0x6d	m	L
0x6e	n	+
0x71	q	—
0x74	t	
0x75	u	⊣
0x76	v	⊥
0x77	w	⊤
0x78	x	

## Поля прокрутки

Следующие последовательности позволяют программе настроить область прокрутки экрана, на которую влияют операции прокрутки. Это подмножество строк, которые корректируются при прокрутке экрана в противном случае, например на "\n" или ri. Эти поля также влияют на строки, измененные при выполнении операций вставки строки (IL), удаления строки (DL), прокрутки вверх (SU) и прокрутки вниз (SD).

Поля прокрутки могут особенно пригодиться для отображения части экрана, которая не прокручивается при заполнении остальной части экрана. Например, заголовок окна вверху или строка состояния в нижней части приложения.

Для DECSTBM есть два необязательных параметра: <t> и <b>. Они используются для указания строк, представляющих верхние и нижние строки области прокрутки (включительно). Если параметры не указаны, <t> по умолчанию равно 1, а <b> — текущей высоте окна просмотра.

Поля прокрутки задаются для каждого буфера, поэтому важно, чтобы альтернативный и основной буфер поддерживали отдельные параметры полей прокрутки (таким образом полноэкранное приложение в альтернативном буфере не навредит полям основного буфера).

Последовательность	Код	Описание	Поведение
ESC [ <t> ; <b> r	DECSTBM	Установка области прокрутки	Задает поля прокрутки VT в окне просмотра.

## ЗАГОЛОВОК ОКНА

Указанные ниже команды позволяют приложению задать для заголовка окна консоли значение указанного параметра <string>. Допустимое количество символов в строке не должно превышать 255. Это эквивалентно вызову SetConsoleTitle с заданной строкой.

Обратите внимание, что эти последовательности являются последовательностями OSC "Команда операционной системы", а не CSI, как многие другие перечисленные последовательности, и как таковые начинаются с "\x1b]", а не "\x1b[". Как последовательности OSC, они заканчиваются *строковым признаком конца*, представленным как <ST> и передаваемым с помощью ESC \ (0x1B 0x5C). Вместо признака конца строки можно использовать BEL (0x7), но рекомендуется использовать более длинную форму.

Последовательность	Описание	Поведение
ESC ] 0 ; <Строка> <ST>	Установка заголовка окна	Задает для заголовка окна консоли значение <string>.
ESC ] 2 ; <Строка> <ST>	Установка заголовка окна	Задает для заголовка окна консоли значение <string>.

Завершающий символ здесь является символом "Колокол", "\x07"

## Альтернативный буфер экрана

\* Приложения в стиле Nix часто используют альтернативный буфер экрана, чтобы они могли изменять все содержимое буфера, не влияя на приложение, которое их запустило. Альтернативный буфер точно соответствует размеру окна без области обратной прокрутки.

В качестве примера такого поведения рассмотрим запуск Vim из Bash. Vim использует весь экран для редактирования файла, а затем возвращается к Bash, не затрагивая исходный буфер.

Последовательность	Описание	Поведение
ESC [ ? 1 0 4 9 h	Использование альтернативного буфера экрана	Переключается на новый альтернативный буфер экрана.
ESC [ ? 1 0 4 9 l	Использование основного буфера экрана	Переключается на основной буфер.

## Ширина окна

Для управления шириной окна консоли можно использовать следующие последовательности. Они примерно эквивалентны вызову API консоли SetConsoleScreenBufferSizeEx для установки ширины окна.

Последовательность	Код	Описание	Поведение
ESC [ ? 3 h	DECCOLM	Установка для числа столбцов значения 132	Задает ширину консоли в 132 столбца.
ESC [ ? 3 l	DECCOLM	Установка для числа столбцов значения 80	Задает ширину консоли в 80 столбцов.

# Программный сброс

Для сброса некоторых свойств к значениям по умолчанию можно использовать указанную ниже последовательность. Следующие свойства сбрасываются к значениям по умолчанию (также перечислены последовательности, управляющие этими свойствами):

- Видимость курсора: видимый (DECTEM).
- Цифровая клавиатура: числовой режим (DECNKM).
- Режим клавиш курсора: обычный режим (DECCKM).
- Верхнее и нижнее поле: верхнее = 1, нижнее = высоте консоли (DECSTBM).
- Набор символов: US ASCII.
- Графическое отображение: по умолчанию/выкл. (SGR).
- Сохранение состояния курсора: начальная позиция (0,0) (DECSC).

Последовательность	Код	Описание	Поведение
ESC [ ! p	DECSTR	Программный сброс	Сбрасывает некоторые параметры терминала к значениям по умолчанию.

## Входные последовательности

Следующие последовательности терминала создаются узлом консоли во входном потоке, если флаг ENABLE\_VIRTUAL\_TERMINAL\_INPUT установлен на дескрипторе буфера ввода с помощью флага SetConsoleMode.

Существует два внутренних режима, определяющих, какие последовательности выдаются для заданных клавиш ввода: режим клавиш курсора и режим клавиш дополнительной клавиатуры. Они подробно описаны в разделе "Изменения режима".

## Клавиши курсора

Ключ	Обычный режим	Режим приложения
Стрелка вверх	ESC [ A	ESC O A
Стрелка вниз	ESC [ B	ESC O B
Стрелка вправо	ESC [ C	ESC O C
СТРЕЛКА ВЛЕВО	ESC [ D	ESC O D

<b>Ключ</b>	<b>Обычный режим</b>	<b>Режим приложения</b>
Домашняя страница	ESC [ H	ESC O H
Конец	ESC [ F	ESC O F

Кроме того, если нажать клавишу Ctrl с любой из этих клавиш, будут выданы указанные ниже последовательности, независимо от режима клавиш курсора:

<b>Ключ</b>	<b>Любой режим</b>
CTRL+СТРЕЛКА ВВЕРХ	ESC [ 1 ; 5 A
CTRL+СТРЕЛКА ВНИЗ	ESC [ 1 ; 5 B
CTRL+СТРЕЛКА ВПРАВО	ESC [ 1 ; 5 C
CTRL+СТРЕЛКА ВЛЕВО	ESC [ 1 ; 5 D

## Функциональные клавиши NumPad &

<b>Ключ</b>	<b>Последовательность</b>
Отмена	0x7f (DEL)
Пауза	0x1a (SUB)
ESC	0x1b (ESC)
Вставить	ESC [ 2 ~
Удалить	ESC [ 3 ~
PAGE UP	ESC [ 5 ~
PAGE DOWN	ESC [ 6 ~
F1	ESC O P
F2	ESC O Q
F3	ESC O R
F4	ESC O S
F5	ESC [ 1 5 ~
F6	ESC [ 1 7 ~
F7	ESC [ 1 8 ~

<b>Ключ</b>	<b>Последовательность</b>
F8	ESC [ 1 9 ~
F9	ESC [ 2 0 ~
F10	ESC [ 2 1 ~
F11	ESC [ 2 3 ~
F12	ESC [ 2 4 ~

## Модификаторы

Клавиша Alt обрабатывается с помощью префикса escape-последовательности: ESC <c>, где <c> — это символ, передаваемый операционной системой. Alt+Ctrl обрабатывается аналогичным образом, за исключением того, что у операционной системы будет предварительно сдвинута клавиша <c> на соответствующий управляющий символ, который будет передаваться в приложение.

Обычно передача Ctrl выполняется точно так же, как и при получении от системы. Обычно это один символ, который сдвинут вниз в зарезервированное пространство управляющего символа (0x0-0x1f). Например, ctrl+@ (0x40) становится NUL (0x00), CTRL+[ (0x5b) — ESC (0x1b) и т. д. Некоторые сочетания клавиш CTRL обрабатываются специально в соответствии со следующей таблицей:

<b>Ключ</b>	<b>Последовательность</b>
CTRL+ПРОБЕЛ	0x00 (NUL)
CTRL+СТРЕЛКА ВВЕРХ	ESC [ 1 ; 5 A
CTRL+СТРЕЛКА ВНИЗ	ESC [ 1 ; 5 B
CTRL+СТРЕЛКА ВПРАВО	ESC [ 1 ; 5 C
CTRL+СТРЕЛКА ВЛЕВО	ESC [ 1 ; 5 D

### ⓘ Примечание

Левый **ctrl**+ правый **alt** обрабатывается как AltGr. При одновременном отображении они будут удалены, а значение символа Юникода, представленного системой, будет передано в целевой объект. Система будет предварительно преобразовывать значения AltGr в соответствии с текущими входными параметрами системы.

# Примеры

## Пример последовательностей терминала SGR

Следующий код содержит несколько примеров форматирования текста.

```
C

#include <stdio.h>
#include <wchar.h>
#include <windows.h>

int main()
{
    // Set output mode to handle virtual terminal sequences
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut == INVALID_HANDLE_VALUE)
    {
        return GetLastError();
    }

    DWORD dwMode = 0;
    if (!GetConsoleMode(hOut, &dwMode))
    {
        return GetLastError();
    }

    dwMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
    if (!SetConsoleMode(hOut, dwMode))
    {
        return GetLastError();
    }

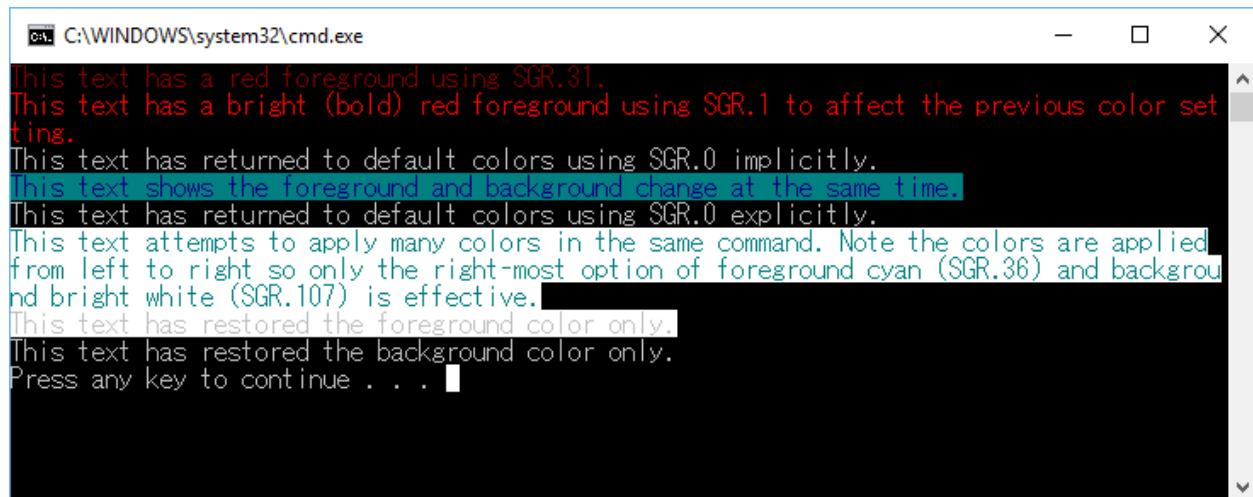
    // Try some Set Graphics Rendition (SGR) terminal escape sequences
    wprintf(L"\x1b[31mThis text has a red foreground using SGR.31.\r\n");
    wprintf(L"\x1b[1mThis text has a bright (bold) red foreground using
SGR.1 to affect the previous color setting.\r\n");
    wprintf(L"\x1b[mThis text has returned to default colors using SGR.0
implicitly.\r\n");
    wprintf(L"\x1b[34;46mThis text shows the foreground and background
change at the same time.\r\n");
    wprintf(L"\x1b[0mThis text has returned to default colors using SGR.0
explicitly.\r\n");
    wprintf(L"\x1b[31;32;33;34;35;36;101;102;103;104;105;106;107mThis text
attempts to apply many colors in the same command. Note the colors are
applied from left to right so only the right-most option of foreground cyan
(SGR.36) and background bright white (SGR.107) is effective.\r\n");
    wprintf(L"\x1b[39mThis text has restored the foreground color
only.\r\n");
    wprintf(L"\x1b[49mThis text has restored the background color
only.\r\n");
}
```

```
    return 0;  
}
```

### ⓘ Примечание

В предыдущем примере строка "`\x1b[31m`" является реализацией **ESC [ <n> m** с **<n>** равной 31.

На следующем рисунке показаны выходные данные предыдущего примера кода.



The screenshot shows a Windows Command Prompt window titled "cmd C:\WINDOWS\system32\cmd.exe". The window contains the following text:

```
This text has a red foreground using SGR.31.  
This text has a bright (bold) red foreground using SGR.1 to affect the previous color setting.  
This text has returned to default colors using SGR.0 implicitly.  
This text shows the foreground and background change at the same time.  
This text has returned to default colors using SGR.0 explicitly.  
This text attempts to apply many colors in the same command. Note the colors are applied from left to right so only the right-most option of foreground cyan (SGR.36) and background bright white (SGR.107) is effective.  
This text has restored the foreground color only.  
This text has restored the background color only.  
Press any key to continue . . .
```

## Пример включения обработки виртуального терминала

В следующем коде приведен пример рекомендуемого способа включения обработки виртуального терминала для приложения. Цель примера — продемонстрировать, что:

- Существующий режим всегда должен извлекаться через `GetConsoleMode` и анализироваться, прежде чем он будет задан с помощью `SetConsoleMode`.
- Проверка того, возвращает ли `SetConsoleMode`, а `GetLastError` возвращает `ERROR_INVALID_PARAMETER` является текущим механизмом, который можно определить при запуске в системе нижнего уровня. Приложение, получающее `ERROR_INVALID_PARAMETER` с одним из новых флагов режима консоли в битовом поле, должно корректно ухудшить поведение и повторить попытку.

```
C
```

```
#include <stdio.h>  
#include <wchar.h>  
#include <windows.h>
```

```
int main()
{
    // Set output mode to handle virtual terminal sequences
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut == INVALID_HANDLE_VALUE)
    {
        return false;
    }
    HANDLE hIn = GetStdHandle(STD_INPUT_HANDLE);
    if (hIn == INVALID_HANDLE_VALUE)
    {
        return false;
    }

    DWORD dwOriginalOutMode = 0;
    DWORD dwOriginalInMode = 0;
    if (!GetConsoleMode(hOut, &dwOriginalOutMode))
    {
        return false;
    }
    if (!GetConsoleMode(hIn, &dwOriginalInMode))
    {
        return false;
    }

    DWORD dwRequestedOutModes = ENABLE_VIRTUAL_TERMINAL_PROCESSING |
DISABLE_NEWLINE_AUTO_RETURN;
    DWORD dwRequestedInModes = ENABLE_VIRTUAL_TERMINAL_INPUT;

    DWORD dwOutMode = dwOriginalOutMode | dwRequestedOutModes;
    if (!SetConsoleMode(hOut, dwOutMode))
    {
        // we failed to set both modes, try to step down mode gracefully.
        dwRequestedOutModes = ENABLE_VIRTUAL_TERMINAL_PROCESSING;
        dwOutMode = dwOriginalOutMode | dwRequestedOutModes;
        if (!SetConsoleMode(hOut, dwOutMode))
        {
            // Failed to set any VT mode, can't do anything here.
            return -1;
        }
    }

    DWORD dwInMode = dwOriginalInMode | dwRequestedInModes;
    if (!SetConsoleMode(hIn, dwInMode))
    {
        // Failed to set VT input mode, can't do anything here.
        return -1;
    }

    return 0;
}
```

# Пример выбора функций юбилейного обновления

Следующий пример представляет собой более надежный пример кода, использующий разнообразные escape-последовательности для работы с буфером, с акцентом на возможности, добавленные в юбилейном обновлении для Windows 10.

В этом примере используется альтернативный буфер экрана, выполняется управление позициями табуляции, установка полей прокрутки и изменение набора символов.

```
C

// System headers
#include <windows.h>

// Standard library C-style
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#define ESC "\x1b"
#define CSI "\x1b["

bool EnableVTMode()
{
    // Set output mode to handle virtual terminal sequences
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut == INVALID_HANDLE_VALUE)
    {
        return false;
    }

    DWORD dwMode = 0;
    if (!GetConsoleMode(hOut, &dwMode))
    {
        return false;
    }

    dwMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
    if (!SetConsoleMode(hOut, dwMode))
    {
        return false;
    }
    return true;
}

void PrintVerticalBorder()
{
    printf(ESC "(0"); // Enter Line drawing mode
    printf(CSI "104;93m"); // bright yellow on bright blue
```

```

        printf("x"); // in line drawing mode, \x78 -> \u2502 "Vertical Bar"
        printf(CSI "0m"); // restore color
        printf(ESC "(B"); // exit line drawing mode
    }

void PrintHorizontalBorder(COORD const Size, bool fIsTop)
{
    printf(ESC "(0"); // Enter Line drawing mode
    printf(CSI "104;93m"); // Make the border bright yellow on bright blue
    printf(fIsTop ? "1" : "m"); // print left corner

    for (int i = 1; i < Size.X - 1; i++)
        printf("q"); // in line drawing mode, \x71 -> \u2500 "HORIZONTAL
SCAN LINE-5"

    printf(fIsTop ? "k" : "j"); // print right corner
    printf(CSI "0m");
    printf(ESC "(B"); // exit line drawing mode
}

void PrintStatusLine(const char* const pszMessage, COORD const Size)
{
    printf(CSI "%d;1H", Size.Y);
    printf(CSI "K"); // clear the line
    printf(pszMessage);
}

int __cdecl wmain(int argc, WCHAR* argv[])
{
    argc; // unused
    argv; // unused
    //First, enable VT mode
    bool fSuccess = EnableVTMode();
    if (!fSuccess)
    {
        printf("Unable to enter VT processing mode. Quitting.\n");
        return -1;
    }
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut == INVALID_HANDLE_VALUE)
    {
        printf("Couldn't get the console handle. Quitting.\n");
        return -1;
    }

    CONSOLE_SCREEN_BUFFER_INFO ScreenBufferInfo;
    GetConsoleScreenBufferInfo(hOut, &ScreenBufferInfo);
    COORD Size;
    Size.X = ScreenBufferInfo.srWindow.Right -
ScreenBufferInfo.srWindow.Left + 1;
    Size.Y = ScreenBufferInfo.srWindow.Bottom -
ScreenBufferInfo.srWindow.Top + 1;

    // Enter the alternate buffer
    printf(CSI "?1049h");
}

```

```

// Clear screen, tab stops, set, stop at columns 16, 32
printf(CSI "1;1H");
printf(CSI "2J"); // Clear screen

int iNumTabStops = 4; // (0, 20, 40, width)
printf(CSI "3g"); // clear all tab stops
printf(CSI "1;20H"); // Move to column 20
printf(ESC "H"); // set a tab stop

printf(CSI "1;40H"); // Move to column 40
printf(ESC "H"); // set a tab stop

// Set scrolling margins to 3, h-2
printf(CSI "3;%dr", Size.Y - 2);
int iNumLines = Size.Y - 4;

printf(CSI "1;1H");
printf(CSI "102;30m");
printf("Windows 10 Anniversary Update - VT Example");
printf(CSI "0m");

// Print a top border - Yellow
printf(CSI "2;1H");
PrintHorizontalBorder(Size, true);

// // Print a bottom border
printf(CSI "%d;1H", Size.Y - 1);
PrintHorizontalBorder(Size, false);

wchar_t wch;

// draw columns
printf(CSI "3;1H");
int line = 0;
for (line = 0; line < iNumLines * iNumTabStops; line++)
{
    PrintVerticalBorder();
    if (line + 1 != iNumLines * iNumTabStops) // don't advance to next
line if this is the last line
        printf("\t"); // advance to next tab stop
}

PrintStatusLine("Press any key to see text printed between tab stops.", size);
wch = _getwch();

// Fill columns with output
printf(CSI "3;1H");
for (line = 0; line < iNumLines; line++)
{
    int tab = 0;
    for (tab = 0; tab < iNumTabStops - 1; tab++)
    {

```

```

        PrintVerticalBorder();
        printf("line=%d", line);
        printf("\t"); // advance to next tab stop
    }
    PrintVerticalBorder(); // print border at right side
    if (line + 1 != iNumLines)
        printf("\t"); // advance to next tab stop, (on the next line)
}

PrintStatusLine("Press any key to demonstrate scroll margins", Size);
wch = _getwch();

printf(CSI "3;1H");
for (line = 0; line < iNumLines * 2; line++)
{
    printf(CSI "K"); // clear the line
    int tab = 0;
    for (tab = 0; tab < iNumTabStops - 1; tab++)
    {
        PrintVerticalBorder();
        printf("line=%d", line);
        printf("\t"); // advance to next tab stop
    }
    PrintVerticalBorder(); // print border at right side
    if (line + 1 != iNumLines * 2)
    {
        printf("\n"); //Advance to next line. If we're at the bottom of
the margins, the text will scroll.
        printf("\r"); //return to first col in buffer
    }
}

PrintStatusLine("Press any key to exit", Size);
wch = _getwch();

// Exit the alternate buffer
printf(CSI "?10491");

}

```

# Создание сеанса псевдоконсоли

Статья • 12.07.2023

Псевдоконсоль Windows (иногда также псевдо консоль, conpty или псевдотерминал Windows) — это механизм, предназначенный для создания внешнего узла для действий подсистемы с поддержкой символьного режима, заменяющего часть взаимодействия пользователя окна узла консоли по умолчанию.

Процесс реализации сеанса псевдоконсоли немного отличается от сеанса традиционной консоли. Сеансы традиционной консоли запускаются автоматически, когда операционная система распознает начало запуска приложения с поддержкой символьного режима. В отличие от них сеанс псевдоконсоли и каналы связи необходимо создавать, используя приложение размещения, до создания процесса с размещением дочернего приложения с поддержкой символьного режима. Дочерний процесс будет по-прежнему создаваться с помощью функции [CreateProcess](#), но уже с дополнительными сведениями, которые позволяют операционной системе настроить соответствующую среду.

Дополнительные сведения об этой системе см. в [записи блога о первоначальном объявлении](#).

Полные примеры использования псевдоконсоли см. в каталоге примеров раздела [Терминал Windows](#) нашего репозитория GitHub.

## Подготовка каналов связи

Сначала надо объединить в пару синхронные каналы связи, которые будут предоставлены в процессе создания сеанса псевдоконсоли для двухстороннего обмена данными с размещенным приложением. Система псевдоконсоли обрабатывает эти каналы, используя функции [ReadFile](#) и [WriteFile](#) с [синхронным вводом-выводом](#). Дескрипторы файла или устройства ввода-вывода, такие как файловый поток или канал, допустимы, только если для асинхронной передачи данных не требуется структура типа [OVERLAPPED](#).

### Предупреждение

Чтобы предотвратить возникновение состояний гонки и взаимоблокировок, настоятельно рекомендуется обслуживать каждый канал связи в отдельном

потоке, который поддерживает собственные состояния буфера клиента и очередь обмена сообщениями внутри приложения. Обслуживание всех действий псевдоконсоли в одном потоке может привести к взаимоблокировке, при которой один из буферов связи заполняется и переходит в режим ожидания действия, пока вы пытаетесь отправить запрос на блокировку в другой канал.

## Создание псевдоконсоли

Настроив каналы связи, определите точку завершения "считывания" канала ввода и точку завершения "записи" канала вывода. Эта пара дескрипторов предоставляется для создания объекта при вызове [CreatePseudoConsole](#).

Во время создания необходимо использовать размер, представленный измерениями X и Y (в количестве символов). Эти измерения будут применяться к экрану дисплея для окна итоговой (терминальной) презентации. Значения используются для создания буфера оперативной памяти в системе псевдоконсоли.

Размер буфера предоставляет ответы клиентским приложениям с поддержкой символьного режима, которые проверяют данные с помощью таких [функций консоли на стороне клиента](#), как [GetConsoleScreenBufferInfoEx](#), и определяет структуру и расположение текста при использовании клиентами таких функций, как [WriteConsoleOutput](#).

И в заключение при создании псевдоконсоли предоставляется поле флагов для выполнения специальных функций. Если вам не нужно использовать специальные функции, задайте для параметра 0 в качестве значения по умолчанию.

В настоящее время для запроса наследования позиции курсора из сеанса консоли, уже присоединенного к вызывающей стороне API псевдоконсоли, можно использовать только один специальный флаг. Он предназначен для более сложных сценариев, где приложение размещения, подготавливающее сеанс псевдоконсоли, само по себе является клиентским приложением с поддержкой символьного режима, но уже в среде другой консоли.

Ниже приведен пример фрагмента кода, в котором для установки пары каналов связи и создания псевдоконсоли используется функция [CreatePipe](#).

C

```
HRESULT SetUpPseudoConsole(COORD size)
{
```

```
HRESULT hr = S_OK;

// Create communication channels

// - Close these after CreateProcess of child application with
pseudoconsole object.
HANDLE inputReadSide, outputWriteSide;

// - Hold onto these and use them for communication with the child
through the pseudoconsole.
HANDLE outputReadSide, inputWriteSide;

if (!CreatePipe(&inputReadSide, &inputWriteSide, NULL, 0))
{
    return HRESULT_FROM_WIN32(GetLastError());
}

if (!CreatePipe(&outputReadSide, &outputWriteSide, NULL, 0))
{
    return HRESULT_FROM_WIN32(GetLastError());
}

HPCON hPC;
hr = CreatePseudoConsole(size, inputReadSide, outputWriteSide, 0, &hPC);
if (FAILED(hr))
{
    return hr;
}

// ...

}
```

### ⓘ Примечание

Этот фрагмент кода неполон. Он предназначен только для демонстрации этого конкретного вызова. Временем существования функции **HANDLE** следует управлять соответствующим образом. Если временем существования функции **HANDLE** не управлять должным образом, то могут возникнуть ситуации взаимоблокировок, особенно для синхронных вызовов ввода-вывода.

По окончании вызова [CreateProcess](#) для создания клиентского приложения с поддержкой символьного режима, присоединенного к псевдоконсоли, заданные при создании дескрипторы следует убрать из этого процесса. Это снизит число ссылок для объекта базового устройства, а также позволит операциям ввода-вывода правильно обнаруживать поврежденный канал в случае закрытия сеансом псевдоконсоли копии ее дескрипторов.

# Подготовка к созданию дочернего процесса

Следующим этапом является подготовка структуры [STARTUPINFOEX](#), ответственной за передачу информации псевдоконсоли во время запуска дочернего процесса.

Эта структура способна предоставить сложную информацию о запуске, в том числе об атрибутах создания процесса и потока.

Используйте [InitializeProcThreadAttributeList](#) в двойном вызове, чтобы сначала рассчитать число байтов, необходимых для хранения списка, выделить запрошенную память, а затем повторить вызов, предоставив непрозрачный указатель памяти, чтобы настроить этот список в качестве списка атрибутов.

После этого вызовите [UpdateProcThreadAttribute](#) для передачи списка инициализированных атрибутов с флагом **PROC\_THREAD\_ATTRIBUTE\_PSEUDOCONSOLE**, дескриптором псевдоконсоли и размером последнего.

```
C

HRESULT PrepareStartupInformation(HPCON hpc, STARTUPINFOEX* psi)
{
    // Prepare Startup Information structure
    STARTUPINFOEX si;
    ZeroMemory(&si, sizeof(si));
    si.StartupInfo.cb = sizeof(STARTUPINFOEX);

    // Discover the size required for the list
    size_t bytesRequired;
    InitializeProcThreadAttributeList(NULL, 1, 0, &bytesRequired);

    // Allocate memory to represent the list
    si.lpAttributeList =
        (PPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(GetProcessHeap(), 0, bytesRequired);
    if (!si.lpAttributeList)
    {
        return E_OUTOFMEMORY;
    }

    // Initialize the list memory location
    if (!InitializeProcThreadAttributeList(si.lpAttributeList, 1, 0,
&bytesRequired))
    {
        HeapFree(GetProcessHeap(), 0, si.lpAttributeList);
        return HRESULT_FROM_WIN32(GetLastError());
    }

    // Set the pseudoconsole information into the list
    if (!UpdateProcThreadAttribute(si.lpAttributeList,
```

```

        0,
        PROC_THREAD_ATTRIBUTE_PSEUDOCONSOLE,
        hpc,
        sizeof(hpc),
        NULL,
        NULL))
{
    HeapFree(GetProcessHeap(), 0, si.lpAttributeList);
    return HRESULT_FROM_WIN32(GetLastError());
}

*psi = si;

return S_OK;
}

```

## Создание размещенного процесса

Далее вызовите [CreateProcess](#), передав исполняемому файлу структуру [STARTUPINFOEX](#) и путь, а при необходимости и дополнительные сведения о конфигурации. Для оповещения системы о том, что в расширенных сведениях содержится ссылка псевдоконсоли, при вызове следует установить флаг [EXTENDED\\_STARTUPINFO\\_PRESENT](#).

C

```

HRESULT SetUpPseudoConsole(COORD size)
{
    // ...

    PCWSTR childApplication = L"C:\\windows\\system32\\cmd.exe";

    // Create mutable text string for CreateProcessW command line string.
    const size_t charsRequired = wcslen(childApplication) + 1; // +1 null
    terminator
    PWSTR cmdLineMutable = (PWSTR)HeapAlloc(GetProcessHeap(), 0,
    sizeof(wchar_t) * charsRequired);

    if (!cmdLineMutable)
    {
        return E_OUTOFMEMORY;
    }

    wcscpy_s(cmdLineMutable, charsRequired, childApplication);

    PROCESS_INFORMATION pi;
    ZeroMemory(&pi, sizeof(pi));

    // Call CreateProcess

```

```
if (!CreateProcessW(NULL,
                    cmdLineMutable,
                    NULL,
                    NULL,
                    FALSE,
                    EXTENDED_STARTUPINFO_PRESENT,
                    NULL,
                    NULL,
                    &siEx.StartupInfo,
                    &pi))
{
    HeapFree(GetProcessHeap(), 0, cmdLineMutable);
    return HRESULT_FROM_WIN32(GetLastError());
}

// ...
}
```

### ⓘ Примечание

Если закрыть сеанс псевдоконсоли, пока размещенный процесс запускается и устанавливает соединение, в клиентском приложении может отобразиться диалоговое окно ошибки. Такое же диалоговое окно ошибки появляется, если предоставить размещенному процессу при запуске недопустимый дескриптор псевдоконсоли. Кодом инициализации размещенного процесса два эти обстоятельства воспринимаются как идентичные. При сбое всплывающее диалоговое окно из размещенного клиентского приложения будет содержать код ошибки `0xc0000142` и локализованное сообщение, содержащее сведения о сбое инициализации.

## Взаимодействие с сеансом псевдоконсоли

После успешного создания процесса приложение размещения может завершить запись канала ввода для отправки сведений о взаимодействии пользователя в псевдоконсоль, а также завершить чтение канала вывода для получения сведений о графическом представлении из псевдоконсоли.

Решение об обработке будущих действий полностью зависит от приложения размещения. Приложение размещения способно запустить окно в другом потоке, чтобы собрать входные данные о взаимодействии пользователя и сериализовать их в точку завершения записи канала ввода для псевдоконсоли и размещенного приложения с поддержкой символьного режима. Другой поток также может быть запущен, чтобы приостановить точку завершения чтения канала вывода для

псевдоконсоли, декодировать текст и сведения о [последовательности виртуального терминала](#), а затем вывести их на экран.

Потоки также используются для передачи данных из каналов псевдоконсоли в другой канал или устройство, при этом для передачи удаленной информации в другой процесс или компьютер используется сеть, однако предотвращается локальное перекодирование информации.

## Изменение размеров псевдоконсоли

В ходе выполнения может возникнуть ситуация, когда необходимо будет изменить размер буфера в связи с взаимодействием пользователя или запросом, пришедшим извне от другого устройства отображения/взаимодействия.

Это действие можно выполнить с помощью функции [ResizePseudoConsole](#), которая определяет высоту и ширину буфера в количестве символов.

C

```
// Theoretical event handler function with theoretical
// event that has associated display properties
// on Source property.
void OnWindowResize(Event e)
{
    // Retrieve width and height dimensions of display in
    // characters using theoretical height/width functions
    // that can retrieve the properties from the display
    // attached to the event.
    COORD size;
    size.X = GetViewWidth(e.Source);
    size.Y = GetViewHeight(e.Source);

    // Call pseudoconsole API to inform buffer dimension update
    ResizePseudoConsole(m_hpc, size);
}
```

## Завершение сеанса псевдоконсоли

Чтобы завершить сеанс, вызовите функцию [ClosePseudoConsole](#), воспользовавшись дескриптором, использованным при начальном создании псевдоконсоли. Все подключенные клиентские приложения с поддержкой символьного режима, например приложение из вызова [CreateProcess](#), прервут свою работу после завершения сеанса. Если исходное дочернее приложение было

оболочкой и создавало другие процессы, также будет прервана работа всех связанных присоединенных процессов в дереве.

### Предупреждение

Если использовать псевдоконсоль в однопотоковом синхронном режиме, закрытие сеанса создает несколько побочных эффектов, способных привести к ситуации взаимоблокировки. Завершение сеанса псевдоконсоли может вызвать конечное обновление группы данных для `hOutput`, которое следует удалить из буфера канала связи. Кроме того, если при создании псевдоконсоли выбрать параметр `PSEUDOCONSOLE_INHERIT_CURSOR`, то при попытке закрыть псевдоконсоль, игнорируя сообщение о запросе на наследование курсора (присыляемое на `hOutput` и получающее ответ через `hInput`), может возникнуть еще одна ситуация взаимоблокировки. Каналы связи для псевдоконсоли рекомендуется обслуживать в отдельных потоках, а также очищать и обрабатывать до их ожидаемого прекращения работы, вызванного прекращением работы клиентского приложения или завершением действий по переналадке во время вызова функции `ClosePseudoConsole`.

# ФУНКЦИИ КОНСОЛИ

Статья • 23.10.2023

Для доступа к консоли используются следующие функции.

Function	Description
<a href="#">AddConsoleAlias</a>	Определяет псевдоним консоли для указанного исполняемого файла.
<a href="#">AllocConsole</a>	Выделяет новую консоль для вызывающего процесса.
<a href="#">AttachConsole</a>	Присоединяет вызывающий процесс к консоли указанного процесса.
<a href="#">ClosePseudoConsole</a>	Закрывает псевдоконсоль из заданного дескриптора.
<a href="#">CreatePseudoConsole</a>	Выделяет новый псевдоконсоль для вызывающего процесса.
<a href="#">CreateConsoleScreenBuffer</a>	Создает буфер экрана консоли.
<a href="#">FillConsoleOutputAttribute</a>	Задает атрибуты цвета текста и фона для указанного количества ячеек символов.
<a href="#">FillConsoleOutputCharacter</a>	Записывает символ в буфер экрана консоли в заданное количество раз.
<a href="#">FlushConsoleInputBuffer</a>	Очищает входной буфер консоли.
<a href="#">FreeConsole</a>	Отсоединяет вызывающий процесс от консоли.
<a href="#">GenerateConsoleCtrlEvent</a>	Отправляет указанный сигнал в группу процессов консоли, которая предоставляет общий доступ к консоли, связанной с вызывающим процессом.
<a href="#">GetConsoleAlias</a>	Извлекает указанный псевдоним для указанного исполняемого файла.
<a href="#">GetConsoleAliases</a>	Извлекает все определенные псевдонимы консоли для указанного исполняемого файла.
<a href="#">GetConsoleAliasesLength</a>	Возвращает размер буфера в байтах, необходимый для хранения всех псевдонимов консоли для указанного исполняемого файла.
<a href="#">GetConsoleAliasExes</a>	Извлекает имена всех исполняемых файлов с определенными псевдонимами консоли.

<b>Function</b>	<b>Description</b>
<a href="#">GetConsoleAliasExesLength</a>	Возвращает размер буфера в байтах, необходимый для хранения имен всех исполняемых файлов с определенными псевдонимами консоли.
<a href="#">GetConsoleCP</a>	Извлекает входную кодовую страницу, используемую консолью, связанной с вызывающим процессом.
<a href="#">GetConsoleCursorInfo</a>	Извлекает сведения о размере и видимости курсора для указанного буфера экрана консоли.
<a href="#">GetConsoleDisplayMode</a>	Извлекает режим отображения текущей консоли.
<a href="#">GetConsoleFontSize</a>	Извлекает размер шрифта, используемого указанным буфером экрана консоли.
<a href="#">GetConsoleHistoryInfo</a>	Извлекает параметры журнала для консоли вызывающего процесса.
<a href="#">GetConsoleMode</a>	Извлекает текущий входной режим входного буфера консоли или текущий выходной режим буфера экрана консоли.
<a href="#">GetConsoleOriginalTitle</a>	Извлекает исходное название текущего окна консоли.
<a href="#">GetConsoleOutputCP</a>	Извлекает выходную кодовую страницу, используемую консолью, связанной с вызывающим процессом.
<a href="#">GetConsoleProcessList</a>	Извлекает список процессов, подключенных к текущей консоли.
<a href="#">GetConsoleScreenBufferInfo</a>	Извлекает сведения о указанном буфере экрана консоли.
<a href="#">GetConsoleScreenBufferInfoEx</a>	Извлекает расширенные сведения о указанном буфере экрана консоли.
<a href="#">GetConsoleSelectionInfo</a>	Извлекает сведения о текущем выборе консоли.
<a href="#">GetConsoleTitle</a>	Извлекает заголовок текущего окна консоли.
<a href="#">GetConsoleWindow</a>	Извлекает дескриптор окна, используемый консолью, связанной с вызывающим процессом.
<a href="#">GetCurrentConsoleFont</a>	Извлекает сведения о текущем шрифте консоли.
<a href="#">GetCurrentConsoleFontEx</a>	Извлекает расширенные сведения о текущем шрифте консоли.

Function	Description
<a href="#">GetLargestConsoleWindowSize</a>	Извлекает размер максимального возможного окна консоли.
<a href="#">GetNumberOfConsoleInputEvents</a>	Извлекает количество непрочитанных входных записей в входном буфере консоли.
<a href="#">GetNumberOfConsoleMouseButtons</a>	Извлекает количество кнопок мыши, используемых текущей консолью.
<a href="#">GetStdHandle</a>	Извлекает дескриптор стандартного входного, стандартного выходного или стандартного устройства ошибок.
<a href="#">HandlerRoutine</a>	Определяемая приложением функция, используемая с функцией <a href="#">SetConsoleCtrlHandler</a> .
<a href="#">PeekConsoleInput</a>	Считывает данные из указанного входного буфера консоли, не удаляя его из буфера.
<a href="#">ReadConsole</a>	Считывает входные данные символов из буфера ввода консоли и удаляет его из буфера.
<a href="#">ReadConsoleInput</a>	Считывает данные из входного буфера консоли и удаляет его из буфера.
<a href="#">ReadConsoleInputEx</a>	Считывает данные из входного буфера консоли и удаляет его из буфера с настраиваемым поведением.
<a href="#">ReadConsoleOutput</a>	Считывает данные символов и атрибутов цвета из прямоугольного блока ячеек символов в буфере экрана консоли.
<a href="#">ReadConsoleOutputAttribute</a>	Копирует указанное количество атрибутов переднего плана и фона из последовательных ячеек буфера экрана консоли.
<a href="#">ReadConsoleOutputCharacter</a>	Копирует несколько символов из последовательных ячеек буфера экрана консоли.
<a href="#">ResizePseudoConsole</a>	Изменяет размер внутренних буферов для псевдоконсоли до заданного размера.
<a href="#">ScrollConsoleScreenBuffer</a>	Перемещает блок данных в буфере экрана.
<a href="#">SetConsoleActiveScreenBuffer</a>	Задает указанный буфер экрана для текущего отображаемого буфера экрана консоли.
<a href="#">SetConsoleCP</a>	Задает входную кодовую страницу, используемую консолью, связанной с вызывающим процессом.

Function	Description
<a href="#">SetConsoleCtrlHandler</a>	Добавляет или удаляет определяемый <a href="#">приложением ОбработчикRoutine</a> из списка функций обработчика для вызывающего процесса.
<a href="#">SetConsoleCursorInfo</a>	Задает размер и видимость курсора для указанного буфера экрана консоли.
<a href="#">SetConsoleCursorPosition</a>	Задает позицию курсора в указанном буфере экрана консоли.
<a href="#">SetConsoleDisplayMode</a>	Задает режим отображения указанного буфера экрана консоли.
<a href="#">SetConsoleHistoryInfo</a>	Задает параметры журнала для консоли вызывающего процесса.
<a href="#">SetConsoleMode</a>	Задает режим ввода для входного буфера консоли или режим вывода для буфера экрана консоли.
<a href="#">SetConsoleOutputCP</a>	Задает выходную кодовую страницу, используемую консолью, связанной с вызывающим процессом.
<a href="#">SetConsoleScreenBufferInfoEx</a>	Задает расширенные сведения о указанном буфере экрана консоли.
<a href="#">SetConsoleScreenBufferSize</a>	Изменяет размер указанного буфера экрана консоли.
<a href="#">SetConsoleTextAttribute</a>	Задает атрибуты переднего плана (текста) и фона символов, записанных в буфер экрана консоли.
<a href="#">SetConsoleTitle</a>	Задает заголовок текущего окна консоли.
<a href="#">SetConsoleWindowInfo</a>	Задает текущий размер и положение окна буфера экрана консоли.
<a href="#">SetCurrentConsoleFontEx</a>	Задает расширенные сведения о текущем шрифте консоли.
<a href="#">SetStdHandle</a>	Задает дескриптор стандартного входного, стандартного выходного или стандартного устройства ошибок.
<a href="#">WriteConsole</a>	Записывает строку символов в буфер экрана консоли, начиная с текущего положения курсора.
<a href="#">WriteConsoleInput</a>	Записывает данные непосредственно в буфер входных данных консоли.
<a href="#">WriteConsoleOutput</a>	Записывает данные символов и атрибутов цвета в указанный прямоугольный блок ячеек символов в

Function	Description
	буфере экрана консоли.
<a href="#">WriteConsoleOutputAttribute</a>	Копирует ряд атрибутов переднего плана и фона в последовательные ячейки буфера экрана консоли.
<a href="#">WriteConsoleOutputCharacter</a>	Копирует ряд символов в последовательные ячейки буфера экрана консоли.

# Функция AddConsoleAlias

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Определяет псевдоним консоли для указанного исполняемого файла.

## Синтаксис

C

```
BOOL WINAPI AddConsoleAlias(
    _In_ LPCTSTR Source,
    _In_ LPCTSTR Target,
    _In_ LPCTSTR ExeName
);
```

## Параметры

*Источник* [in]

Псевдоним консоли, сопоставленный с текстом , указанным *target*.

*Целевой объект* [in]

Текст, заменяющийся источником. Если этот параметр имеет значение **NULL**, то псевдоним консоли удаляется.

*ExeName* [in]

Имя исполняемого файла, для которого необходимо определить псевдоним консоли.

# Возвращаемое значение

Если функция выполнена успешно, возвращаемое значение равно **TRUE**.

Если функция завершается ошибкой, возвращаемое значение равно **FALSE**.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Примеры

Пример см. в разделе "[Псевдонимы консоли](#)".

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)

Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>AddConsoleAliasW</b> (Юникод) и <b>AddConsoleAliasA</b> (ANSI)

## См. также

[Псевдонимы в консоли](#)

[Функции консоли](#)

[\*\*GetConsoleAlias\*\*](#)

[\*\*GetConsoleAliases\*\*](#)

[\*\*GetConsoleAliasExes\*\*](#)

# Функция AllocConsole

Статья • 12.07.2023

Выделяет новую консоль для вызывающего процесса.

## Синтаксис

C

```
BOOL WINAPI AllocConsole(void);
```

## Параметры

У этой функции нет параметров.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

Процесс можно связать только с одной консолью, поэтому функция **AllocConsole** завершается сбоем, если у вызывающего процесса уже есть консоль. Процесс может использовать функцию [FreeConsole](#), чтобы отключиться от текущей консоли, а затем вызвать **AllocConsole**, чтобы создать новую консоль или [AttachConsole](#), чтобы подключиться к другой консоли.

Если вызывающий процесс создает дочерний процесс, то этот дочерний процесс наследует новую консоль.

**AllocConsole** инициализирует стандартный ввод, стандартный вывод и стандартные дескрипторы ошибок для новой консоли. Стандартный дескриптор ввода представляет собой дескриптор для входного буфера консоли, а стандартный выходной дескриптор и стандартный дескриптор ошибок являются дескрипторами для буфера экрана консоли. Чтобы получить эти дескрипторы, используйте функцию [GetStdHandle](#).

Эта функция в основном используется приложением графического пользовательского интерфейса (GUI) для создания окна консоли. Приложения GUI инициализируются без консоли. Консольные приложения инициализируются с помощью консоли, если они не созданы как отключенные процессы (путем вызова функции [CreateProcess](#) с флагом `DETACHED_PROCESS`).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll

## См. также

[Функции консоли](#)

[Консоли](#)

[AttachConsole](#)

[CreateProcess](#)

[FreeConsole](#)

[GetStdHandle](#)

# Функция AttachConsole

Статья • 23.10.2023

Присоединяет вызывающий процесс к консоли указанного процесса в качестве клиентского приложения.

## Синтаксис

C

```
BOOL WINAPI AttachConsole(
    _In_ DWORD dwProcessId
);
```

## Параметры

*dwProcessId* [in]

Идентификатор процесса, консоль которого должна использоваться. Этот параметр может принимать одно из указанных ниже значений.

Значение	Значение
<code>pid</code>	Используйте консоль указанного процесса.
<code>ATTACH_PARENT_PROCESS</code> (DWORD) -1	Используйте консоль родительского элемента текущего процесса.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Процесс может быть присоединен к одной консоли. Если вызывающий процесс уже подключен к консоли, возвращается код ошибки `ERROR_ACCESS_DENIED`. Если указанный процесс не имеет консоли, возвращается код ошибки

**ERROR\_INVALID\_HANDLE**. Если указанный процесс не существует, возвращается код ошибки **ERROR\_INVALID\_PARAMETER**.

Процесс может использовать [функцию FreeConsole](#) для отсоединения от консоли. Если другие процессы совместно используют консоль, консоль не уничтожается, но процесс, который называется **FreeConsole**, не может ссылаться на него. Консоль закрывается, когда последний процесс, подключенный к нему, завершается или вызывает **FreeConsole**. После вызова **FreeConsole** процесс может вызвать [функцию AllocConsole](#), чтобы создать новую консоль или **AttachConsole**, чтобы подключиться к другой консоли.

Эта функция в первую очередь полезна для приложений, связанных с **/SUBSYSTEM:WINDOWS**, что подразумевает, что консоль не требуется перед вводом основного метода программы. В этом случае стандартные дескрипторы, полученные с помощью **GetStdHandle**, скорее всего, будут недопустимыми при запуске до вызова **AttachConsole**. Исключение заключается в том, что приложение запускается с наследованием по родительскому процессу.

Чтобы скомпилировать приложение, использующее эту функцию, определите **\_WIN32\_WINNT** как **0x0501** или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

## Требования

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Консоли](#)

[AllocConsole](#)

**FreeConsole**

**GetConsoleProcessList**

# Функция ClosePseudoConsole

Статья • 23.10.2023

Закрывает псевдоконсоль из заданного дескриптора.

## Синтаксис

```
С

void WINAPI ClosePseudoConsole(
    _In_ HPCON hPC
);
```

## Параметры

*hPC* [in]

Дескриптор активного псевдоконсоля, открытый [CreatePseudoConsole](#).

## Возвращаемое значение

*None*

## Замечания

После закрытия псевдоконсоля клиентские приложения, подключенные к сеансу, также будут прекращены.

Окончательный окрашенный кадр может прибыть на *hOutput* дескриптор, первоначально предоставленный [CreatePseudoConsole](#) при вызове этого API. Ожидается, что вызывающий объект будет удалять эти сведения из буфера канала коммуникации и либо представить его, либо отсутствовать карта. Сбой очистки буфера может привести к бесконечному ожиданию закрытия вызова до тех пор, пока он не будет сбоем или каналы связи не будут нарушены другим способом.

## Требования

Минимальная версия клиента	обновление Windows 10 за октябрь 2018 г. (версия 1809) [только классические приложения]
Минимальная версия сервера	Windows Server 2019 [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Псевдоконсоли](#)

[CreatePseudoConsole](#)

[ResizePseudoConsole](#)

# Функция CreateConsoleScreenBuffer

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Создает буфер экрана консоли.

## Синтаксис

C

```
HANDLE WINAPI CreateConsoleScreenBuffer(
    _In_          DWORD      dwDesiredAccess,
    _In_          DWORD      dwShareMode,
    _In_opt_      const SECURITY_ATTRIBUTES *lpSecurityAttributes,
    _In_          DWORD      dwFlags,
    _Reserved_   LPVOID     lpScreenBufferData
);
```

## Параметры

*dwDesiredAccess* [in]

Доступ к буферу экрана консоли. Список прав доступа см. в разделе "[Безопасность буфера консоли](#)" и "[Права доступа](#)".

*dwShareMode* [in]

Этот параметр может быть равен нулю, указывая, что буфер не может быть общим или может быть одним или несколькими из следующих значений.

Значение	Значение
FILE_SHARE_READ 0x00000001	Другие открытые операции можно выполнять в буфере экрана консоли для доступа на чтение.
FILE_SHARE_WRITE 0x00000002	Другие открытые операции можно выполнять в буфере экрана консоли для доступа на запись.

### *lpSecurityAttributes* [in, необязательный]

Указатель на структуру **SECURITY\_ATTRIBUTES**, которая определяет, может ли возвращаемый дескриптор наследоваться дочерними процессами. Если значение *lpSecurityAttributes* равно **NULL**, дескриптор не может быть унаследован. Элемент **lpSecurityDescriptor** структуры задает дескриптор безопасности для нового буфера экрана консоли. Если значение *lpSecurityAttributes* равно **NULL**, буфер экрана консоли получает дескриптор безопасности по умолчанию. Списки управления доступом в дескрипторе безопасности по умолчанию для буфера экрана консоли приходят из основного или олицетворения маркера создателя.

### *dwFlags* [in]

Тип создаваемого буфера экрана консоли. Единственным поддерживаемым типом буфера экрана является **CONSOLE\_TEXTMODE\_BUFFER**.

### *lpScreenBufferData*

Защищены; должно иметь значение **NULL**.

## Возвращаемое значение

Если функция выполнена успешно, возвращаемое значение является дескриптором нового буфера экрана консоли.

Если функция завершается неудачно, возвращается значение **INVALID\_HANDLE\_VALUE**. Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Консоль может иметь несколько буферов экрана, но только один активный буфер экрана. К неактивным буферам экрана можно обращаться для чтения и записи, но отображается только активный буфер экрана. Чтобы создать буфер экрана активным буфером экрана, используйте [функцию SetConsoleActiveScreenBuffer](#).

Созданный буфер экрана будет копировать некоторые свойства из активного буфера экрана во время вызова этой функции. Поведение выглядит следующим

образом:

- `Font` — скопировано из активного буфера экрана
- `Display Window Size` — скопировано из активного буфера экрана
- `Buffer Size` — сопоставлено `Display Window Size` с (`HE` скопировано)
- `Default Attributes` (цвета) — скопировано из активного буфера экрана
- `Default Popup Attributes` (цвета) — скопировано из активного буфера экрана

Вызывающий процесс может использовать возвращенный дескриптор в любой функции, требующей дескриптора в буфер экрана консоли, при условии ограничений доступа, указанных *параметром dwDesiredAccess*.

Вызывающий процесс может использовать [функцию `DuplicateHandle`](#) для создания повторяющегося дескриптора буфера экрана, имеющего другой доступ или наследование от исходного дескриптора. Однако для создания дубликата, **допустимого для другого процесса (за исключением наследования)**, нельзя использовать **Дубликат**.

Чтобы закрыть дескриптор **буфера экрана консоли**, используйте функцию [`CloseHandle`](#).

### 💡 Совет

Этот API не рекомендуется, но он имеет приблизительный **виртуальный терминал** эквивалент в альтернативной **последовательности буфера** экрана. *Настройка альтернативного буфера* экрана может предоставить приложению отдельное изолированное пространство для рисования в течение среды выполнения сеанса при сохранении содержимого, отображаемого вызывающим объектом приложения. Это сохраняет, что рисование сведений для простого восстановления при выходе процесса.

## Примеры

Пример см. в разделе "[Чтение и запись блоков символов и атрибутов](#)".

## Требования

Минимальная версия клиента Windows 2000 Professional [только классические приложения]

Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

[CloseHandle](#)

[ДубликатHandle](#)

[GetConsoleScreenBufferInfo](#)

[SECURITY\\_ATTRIBUTES](#)

[SetConsoleActiveScreenBuffer](#)

[SetConsoleScreenBufferSize](#)

# Функция CreatePseudoConsole

Статья • 23.10.2023

Создает новый псевдоконсоль для вызывающего процесса.

## Синтаксис

C

```
HRESULT WINAPI CreatePseudoConsole(
    _In_ COORD size,
    _In_ HANDLE hInput,
    _In_ HANDLE hOutput,
    _In_ DWORD dwFlags,
    _Out_ HPCON* phPC
);
```

## Параметры

*размер* [in]

Измерения окна или буфера в количестве символов, которые будут использоваться при первоначальном создании псевдоконсоли. Это можно изменить позже с помощью [ResizePseudoConsole](#).

*hInput* [in]

Открытый дескриптор потока данных, который представляет входные данные пользователя на устройство. В настоящее время это ограничено [синхронным вводом-выводом](#).

*hOutput* [in]

Открытый дескриптор потока данных, представляющий выходные данные приложения с устройства. В настоящее время это ограничено [синхронным вводом-выводом](#).

*dwFlags* [in]

Может иметь следующие значения:

Значение	Значение
0	Выполните стандартное создание псевдоконсоли.
PSEUDOCONSOLE_INHERIT_CURSOR (DWORD)1	Созданный сеанс псевдоконсоли попытается наследовать положение курсора родительской

Значение	Значение
----------	----------

*phPC* [out]

Указатель на расположение, которое получит дескриптор нового псевдоконсоля устройства.

## Возвращаемое значение

Тип: **HRESULT**

Если этот метод выполнен успешно, он возвращает **S\_OK**. В противном случае возвращается код ошибки **HRESULT**.

## Замечания

Эта функция в основном используется приложениями, пытающимися быть окном терминала для приложения пользовательского интерфейса командной строки (CUI). Вызывающие пользователи отвечают за представление информации о выходном потоке и сборе входных данных пользователя и сериализации его в входном потоке.

Входные и выходные потоки, закодированные как UTF-8, содержат обычный текст, чередуемый с [виртуальными последовательностями терминалов](#).

В выходном потоке [последовательности виртуальных терминалов](#) можно декодировать вызывающим приложением для макета и представить обычный текст в окне отображения.

В входном потоке обычный текст представляет стандартные входные клавиши клавиатуры пользователем. Более сложные операции представлены клавишами управления кодировкой и движениями мыши в виде [последовательностей виртуальных терминалов, внедренных](#) в этот поток.

При завершении операций дескриптор, созданный этой функцией, должен быть закрыт с [помощью ClosePseudoConsole](#).

При использовании `PSEUDOCONSOLE_INHERIT_CURSOR` вызывающее приложение должно быть готово реагировать на запрос состояния курсора в асинхронном режиме в фоновом потоке, переадресовав или интерпретируя запрос на сведения курсора, которые будут получены `hOutput` и отвечать на `hInput` них. Сбой этого может привести к зависанию вызывающего приложения при выполнении другого запроса псевдоконсоли системы.

# Примеры

Полное пошаговое руководство по использованию этой функции для создания сеанса псевдоконсоли см. в статье "[Создание сеанса Псевдоконсоли](#)".

## Требования

Минимальная версия клиента	обновление Windows 10 за октябрь 2018 г. (версия 1809) [только классические приложения]
Минимальная версия сервера	Windows Server 2019 [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Псевдоконсоли](#)

[Создание сеанса псевдоконсоли](#)

[ResizePseudoConsole](#)

[ClosePseudoConsole](#)

# Функция ConsoleControl

Статья • 13.12.2023

Выполняет специальные операции ядра для консольных приложений. Это включает повторную обработку окна консоли, что позволяет консоли передавать права переднего плана на запущенные приложения подсистемы консоли и завершая присоединенные процессы.

**Обратите внимание, что эта функция не связана с библиотекой импорта. Эта функция доступна в качестве ресурса с именем `ConsoleControl` в `User32.dll`. Для динамической связи с `User32.dll` необходимо использовать функции `LoadLibrary` и `GetProcAddress`.**

## Синтаксис

C

```
typedef enum _CONSOLECONTROL
{
    Reserved1,
    ConsoleNotifyConsoleApplication,
    Reserved2,
    ConsoleSetCaretInfo,
    Reserved3,
    ConsoleSetForeground,
    ConsoleSetWindowOwner,
    ConsoleEndTask,
} CONSOLECONTROL;

typedef struct _CONSOLEENDTASK
{
    HANDLE ProcessId;
    HWND hwnd;
    ULONG ConsoleEventCode;
    ULONG ConsoleFlags;
} CONSOLEENDTASK, *PCONSOLEENDTASK;

typedef struct _CONSOLEWINDOWOWNER
{
    HWND hwnd;
    ULONG ProcessId;
    ULONG ThreadId;
} CONSOLEWINDOWOWNER, *PCONSOLEWINDOWOWNER;

typedef struct _CONSOLESETFOREGROUND
```

```
{  
    HANDLE hProcess;  
    BOOL bForeground;  
} CONSOLESETFOREGROUND, *PCONSOLESETFOREGROUND;  
  
typedef struct _CONSOLE_PROCESS_INFO  
{  
    IN DWORD dwProcessID;  
    IN DWORD dwFlags;  
} CONSOLE_PROCESS_INFO, *PCONSOLE_PROCESS_INFO;  
  
typedef struct _CONSOLE_CARET_INFO  
{  
    IN HWND hwnd;  
    IN RECT rc;  
} CONSOLE_CARET_INFO, *PCONSOLE_CARET_INFO;  
  
NTSTATUS ConsoleControl(  
    _In_ CONSOLECONTROL Command,  
    _In_reads_bytes_(ConsoleInformationLength) PVOID ConsoleInformation,  
    _In_ DWORD ConsoleInformationLength  
);
```

## Параметры

*Команда* [in]

Одно из значений `CONSOLECONTROL`, указывающее, какая функция управления консолью должна выполняться.

*ConsoleInformation* [in]

Указатель на одну из `CONSOLEENDTASK` `CONSOLEWINDOWOWNER` структур или `CONSOLESETFOREGROUND` структур, указывающих дополнительные данные для запрошенной функции управления консолью.

*ConsoleInformationLength* [in]

Размер структуры, на которую указывает параметр *ConsoleInformation*.

## Возвращаемое значение

Если функция выполнена успешно, возвращается `STATUS_SUCCESS` значение.

Если функция завершается ошибкой `NTSTATUS`, возвращаемое значение является причиной сбоя.

# Замечания

Эта функция не определена в заголовке пакета SDK и должна быть объявлена вызывающим оператором. Эта функция экспортируется из user32.dll.

Каждая команда ожидает другую структуру для `ConsoleInformation` параметра.

- `ConsoleNotifyConsoleApplication`: ожидает указатель на объект `CONSOLE_PROCESS_INFO`
- `ConsoleSetCaretInfo`: ожидает указатель на объект `CONSOLE_CARET_INFO`
- `ConsoleSetForeground`: ожидает указатель на объект `CONSOLESETFOREGROUND`
- `ConsoleSetWindowOwner`: ожидает указатель на объект `CONSOLEWINDOWOWNER`
- `ConsoleEndTask`: ожидает указатель на объект `CONSOLEENDTASK`

# Requirements

 Развернуть таблицу

Минимальная версия клиента	Windows 7 [только классические приложения]
Минимальная версия сервера	Windows Server 2008 R2 [только классические приложения]
Верхний колонтитул	нет, см. примечания
Библиотека	нет, см. примечания
DLL-библиотеки	User32.dll

# Функция FillConsoleOutputAttribute

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает атрибуты символов для указанного числа ячеек символов, начиная с указанных координат в буфере экрана.

## Синтаксис

C

```
BOOL WINAPI FillConsoleOutputAttribute(
    _In_    HANDLE   hConsoleOutput,
    _In_    WORD     wAttribute,
    _In_    DWORD    nLength,
    _In_    COORD    dwWriteCoord,
    _Out_   LPDWORD  lpNumberOfAttrsWritten
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*wAttribute* [in]

Атрибуты, используемые при записи в буфер экрана консоли. Дополнительные сведения см. в разделе ["Атрибуты символов"](#).

*nLength* [in]

Число символьных ячеек, заданных указанными атрибутами цвета.

*dwWriteCoord* [in]

[Структура COORD](#), указывающая координаты символов первой ячейки, атрибуты которой необходимо задать.

*lpNumberOfAttrsWritten* [out]

Указатель на переменную, которая получает число ячеек символов, атрибуты которых фактически заданы.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Если число символьных ячеек, атрибуты которых должны быть заданы, выходит за пределы указанной строки в буфере экрана консоли, будут заданы ячейки следующей строки. Если число ячеек для записи выходит за пределы буфера экрана консоли, ячейки записываются до конца буфера экрана консоли.

Значения символов на позициях, записанных в них, не изменяются.

### 💡 Совет

Этот API не рекомендуется и не имеет определенного [эквивалента виртуального терминала](#). Заполнение региона за пределами окна просмотра не поддерживается и зарезервировано для пространства журнала терминала. Заполнение видимой области новым текстом или цветом выполняется путем [перемещения курсора, задания новых атрибутов](#), а затем написания нужного текста для этого региона, повторяющихся символов при необходимости для длины выполнения заливки. Для заполнения прямоугольной области может потребоваться дополнительное перемещение курсора. Ожидается, что клиентское приложение хранит собственную память о том, что находится на экране и не может запрашивать удаленное состояние. Дополнительные сведения см. в [классической консоли и документации по виртуальному терминалу](#).

# Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[COORD](#)

[FillConsoleOutputCharacter](#)

[Функции вывода консоли низкого уровня](#)

[SetConsoleTextAttribute](#)

[WriteConsoleOutputAttribute](#)

# Функция FillConsoleOutputCharacter

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Записывает символ в буфер экрана консоли указанное количество раз, начиная с указанной координаты.

## Синтаксис

C

```
BOOL WINAPI FillConsoleOutputCharacter(
    _In_     HANDLE   hConsoleOutput,
    _In_     TCHAR    cCharacter,
    _In_     DWORD    nLength,
    _In_     COORD    dwWriteCoord,
    _Out_    LPDWORD  lpNumberOfCharsWritten
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*cCharacter* [in]

Символ, записываемый в буфер экрана консоли.

*nLength* [in]

Число ячеек символов, в которые должен быть записан символ.

*dwWriteCoord* [in]

[Структура COORD](#), указывающая координаты символов первой ячейки, в которую необходимо записать символ.

*lpNumberOfCharsWritten* [out]

Указатель на переменную, которая получает количество символов, фактически записанных в буфер экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Если число символов для записи выходит за пределы указанной строки в буфере экрана консоли, символы записываются в следующую строку. Если число символов для записи в буфер экрана консоли выходит за пределы буфера экрана консоли, символы записываются до конца буфера экрана консоли.

Значения атрибутов в записываемых позициях не изменяются.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

### 💡 Совет

Этот API не рекомендуется и не имеет определенного **эквивалента виртуального терминала**. Заполнение региона за пределами окна просмотра не поддерживается и зарезервировано для пространства журнала терминала. Заполнение видимой области новым текстом или цветом выполняется путем **перемещения курсора, задания новых атрибутов**, а затем написания нужного

текста для этого региона, повторяющихся символов при необходимости для длины выполнения заливки. Для заполнения прямоугольной области может потребоваться дополнительное перемещение курсора. Ожидается, что клиентское приложение хранит собственную память о том, что находится на экране и не может запрашивать удаленное состояние. Дополнительные сведения см. в [классической консоли и документации по виртуальному терминалу](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>FillConsoleOutputCharacterW</code> (Юникод) и <code>FillConsoleOutputCharacterA</code> (ANSI)

## См. также

[Функции консоли](#)

[COORD](#)

[FillConsoleOutputAttribute](#)

[Функции вывода консоли низкого уровня](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[WriteConsoleOutputCharacter](#)

# Функция FlushConsoleInputBuffer

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Очищает входной буфер консоли. Все входные записи в данный момент в входном буфере не карта.

## Синтаксис

C

```
BOOL WINAPI FlushConsoleInputBuffer(
    _In_ HANDLE hConsoleInput
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#).

Попытка очистить входную очередь одновременно может уничтожить состояние в очереди неожиданно.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Низкоуровневые функции ввода консоли](#)

[GetNumberOfConsoleInputEvents](#)

[PeekConsoleInput](#)

[ReadConsoleInput](#)

[WriteConsoleInput](#)

# Функция FreeConsole

Статья • 23.10.2023

Отсоединяет вызывающий процесс от консоли.

## Синтаксис

C

```
BOOL WINAPI FreeConsole(void);
```

## Параметры

У этой функции нет параметров.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Процесс может быть присоединен к одной консоли. Процесс может использовать функцию [FreeConsole](#) для отсоединения от консоли. Если другие процессы совместно используют консоль, консоль не уничтожается, но процесс, который называется [FreeConsole](#), не может ссылаться на него. Консоль закрывается, когда последний процесс, подключенный к нему, завершается или вызывает [FreeConsole](#). После вызова [FreeConsole](#) процесс может вызвать функцию [AllocConsole](#), чтобы создать новую консоль или [AttachConsole](#), чтобы подключиться к другой консоли. Если вызывающий процесс еще не подключен к консоли, запрос [FreeConsole](#) по-прежнему завершается успешно.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[AllocConsole](#)

[AttachConsole](#)

[Функции консоли](#)

[Консоли](#)

# Функция GenerateConsoleCtrlEvent

Статья • 23.10.2023

Отправляет указанный сигнал в группу процессов консоли, которая предоставляет общий доступ к консоли, связанной с вызывающим процессом.

## Синтаксис

C

```
BOOL WINAPI GenerateConsoleCtrlEvent(
    _In_ DWORD dwCtrlEvent,
    _In_ DWORD dwProcessGroupId
);
```

## Параметры

*dwCtrlEvent* [in]

Тип создаваемого сигнала. Этот параметр может принимать одно из указанных ниже значений.

Значение	Значение
CTRL_C_EVENT 0	Создает сигнал CTRL+C. Этот сигнал не может быть ограничен определенной группой процессов. Если <i>dwProcessGroupId</i> не является ненулевой, эта функция будет выполнена успешно, но сигнал CTRL+C не будет получен процессами в указанной группе процессов.
CTRL_BREAK_EVENT 1	Создает сигнал CTRL+BREAK.

*dwProcessGroupId* [in]

Идентификатор группы процессов для получения сигнала. Группа процессов создается при указании флага CREATE\_NEW\_PROCESS\_GROUP в вызове [функции CreateProcess](#). Идентификатор процесса нового процесса также является идентификатором группы процессов новой группы процессов. Группа процессов включает все процессы, которые являются потомками корневого процесса. Только те процессы в группе, которые используют ту же консоль, что и вызывающий процесс, получают сигнал. Другими словами, если процесс в группе создает новую консоль, этот процесс не получает сигнал, а потомки не получают.

Если этот параметр равен нулю, сигнал создается во всех процессах, которые совместно используют консоль вызывающего процесса.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

`GenerateConsoleCtrlEvent` приводит к вызову функций обработчика элементов управления в целевой группе. Все процессы консоли имеют функцию обработчика по умолчанию, которая вызывает [функцию ExitProcess](#). Консольный процесс может использовать [функцию SetConsoleCtrlHandler](#) для установки или удаления других функций обработчика.

`SetConsoleCtrlHandler` также может включить наследуемый атрибут, который вызывает процесс пропуска сигналов CTRL+C. Если `GenerateConsoleCtrlEvent` отправляет сигнал CTRL+C в процесс, для которого включен этот атрибут, функции обработчика для этого процесса не вызываются. Сигналы CTRL+BREAK всегда вызывают функции обработчика.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

Обработчики команд управления в консоли

Функции консоли

[Createprocess](#)

[ExitProcess](#)

[SetConsoleCtrlHandler](#)

# Функция GetConsoleAlias

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает текст для указанного псевдонима консоли и исполняемого файла.

## Синтаксис

C

```
DWORD WINAPI GetConsoleAlias(
    _In_    LPTSTR lpSource,
    _Out_   LPTSTR lpTargetBuffer,
    _In_    DWORD   TargetBufferLength,
    _In_    LPTSTR lpExeName
);
```

## Параметры

*lpSource* [in]

Псевдоним консоли, текст которого требуется извлечь.

*lpTargetBuffer* [out]

Указатель на буфер, который получает текст, связанный с псевдонимом консоли.

*TargetBufferLength* [in]

Размер буфера, на который указывает *lpTargetBuffer*, в байтах.

*lpExeName* [in]

Имя исполняемого файла.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib

DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>GetConsoleAliasW</b> (Юникод) и <b>GetConsoleAliasA</b> (ANSI)

## См. также

[Псевдонимы в консоли](#)

[Функции консоли](#)

[AddConsoleAlias](#)

[GetConsoleAliases](#)

[GetConsoleAliasExes](#)

# Функция GetConsoleAliases

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает все определенные псевдонимы консоли для указанного исполняемого файла.

## Синтаксис

C

```
DWORD WINAPI GetConsoleAliases(
    _Out_ LPTSTR lpAliasBuffer,
    _In_  DWORD   AliasBufferLength,
    _In_  LPTSTR lpExeName
);
```

## Параметры

*lpAliasBuffer* [out]

Указатель на буфер, который получает псевдонимы.

Формат данных выглядит следующим образом: *Source1 Target1=\0Source2 Target2=\0... SourceN TargetN=\0*, где *N* — это число определенных псевдонимов консоли.

*AliasBufferLength* [in]

Размер буфера, на который указывает *lpAliasBuffer*, в байтах.

*lpExeName* [in]

Исполняемый файл, псевдоним которого требуется извлечь.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы определить требуемый размер буфера *lpExeName*, используйте [функцию GetConsoleAliasesLength](#).

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]

Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>GetConsoleAliasesW</b> (Юникод) и <b>GetConsoleAliasesA</b> (ANSI)

## См. также

[AddConsoleAlias](#)

[Псевдонимы в консоли](#)

[Функции консоли](#)

[GetConsoleAlias](#)

[GetConsoleAliasesLength](#)

[GetConsoleAliasExes](#)

# Функция GetConsoleAliasesLength

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает необходимый размер буфера, используемого [функцией](#) `GetConsoleAliases`.

## Синтаксис

C

```
DWORD WINAPI GetConsoleAliasesLength(
    _In_ LPTSTR lpExeName
);
```

## Параметры

*lpExeName* [in]

Имя исполняемого файла, псевдонимы консоли которого необходимо извлечь.

## Возвращаемое значение

Размер буфера, необходимого для хранения всех псевдонимов консоли, определенных для этого исполняемого файла в байтах.

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>GetConsoleAliasesLengthW</code> (Юникод) и <code>GetConsoleAliasesLengthA</code> (ANSI)

## См. также

[AddConsoleAlias](#)

[Псевдонимы в консоли](#)

[Функции консоли](#)

[\*\*GetConsoleAlias\*\*](#)

[\*\*GetConsoleAliases\*\*](#)

[\*\*GetConsoleAliasExes\*\*](#)

# Функция GetConsoleAliasExes

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает имена всех исполняемых файлов с определенными псевдонимами консоли.

## Синтаксис

C

```
DWORD WINAPI GetConsoleAliasExes(
    _Out_ LPTSTR lpExeNameBuffer,
    _In_  DWORD   ExeNameBufferLength
);
```

## Параметры

*lpExeNameBuffer* [out]

Указатель на буфер, получающий имена исполняемых файлов.

*ExeNameBufferLength* [in]

Размер буфера, на который указывает *lpExeNameBuffer*, в байтах.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы определить требуемый размер буфера lpExeNameBuffer, используйте [функцию GetConsoleAliasExesLength](#).

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<a href="#">GetConsoleAliasExesW</a> (Юникод) и <a href="#">GetConsoleAliasExesA</a> (ANSI)

## См. также

[AddConsoleAlias](#)

[Псевдонимы в консоли](#)

[Функции консоли](#)

[GetConsoleAlias](#)

[GetConsoleAliasExesLength](#)

[GetConsoleAliases](#)

# Функция GetConsoleAliasExesLength

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает необходимый размер буфера, используемого [функцией GetConsoleAliasExes](#).

## Синтаксис

C

```
DWORD WINAPI GetConsoleAliasExesLength(void);
```

## Параметры

У этой функции нет параметров.

## Возвращаемое значение

Размер буфера, необходимый для хранения имен всех исполняемых файлов с определенными псевдонимами консоли в байтах.

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения

см. в разделе "Использование заголовков Windows".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	GetConsoleAliasExesLengthW (Юникод) и GetConsoleAliasExesLengthA (ANSI)

## См. также

[AddConsoleAlias](#)

[Псевдонимы в консоли](#)

[Функции консоли](#)

[GetConsoleAlias](#)

[GetConsoleAliases](#)

## GetConsoleAliasExes

# Функция GetConsoleCP

Статья • 13.12.2023

Извлекает входную кодовую страницу, используемую консолью, связанной с вызывающим процессом. Консоль использует свою кодовую страницу ввода для перевода ввода клавиатуры в соответствующее значение символа.

## Синтаксис

C

```
UINT WINAPI GetConsoleCP(void);
```

## Параметры

У этой функции нет параметров.

## Возвращаемое значение

Возвращаемое значение — это код, определяющий кодовую страницу. Список идентификаторов см. в разделе "[Идентификаторы кодовой страницы](#)".

Если возвращаемое значение равно нулю, функция завершилась ошибкой. Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Кодовая страница сопоставляет 256 кодов символов с отдельными символами. Разные кодовые страницы включают разные специальные символы, как правило, настроенные для языка или группы языков. Дополнительные сведения о кодовой странице, включая имя, см. в [функции GetCPIInfoEx](#).

Чтобы задать входную кодовую страницу консоли, используйте [функцию SetConsoleCP](#). Чтобы задать и запросить выходную кодовую страницу консоли, используйте функции [SetConsoleOutputCP](#) и [GetConsoleOutputCP](#).

## Requirements

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Кодовые страницы консоли](#)

[Функции консоли](#)

[GetConsoleOutputCP](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

# Функция GetConsoleCursorInfo

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает сведения о размере и видимости курсора для указанного буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI GetConsoleCursorInfo(
    _In_     HANDLE             hConsoleOutput,
    _Out_    PCONSOLE_CURSOR_INFO lpConsoleCursorInfo
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpConsoleCursorInfo* [out]

Указатель на структуру **CONSOLE\_CURSOR\_INFO**, которая получает сведения о курсоре консоли.

# Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

[CONSOLE\\_CURSOR\\_INFO](#)

## **SetConsoleCursorInfo**

# Функция GetConsoleDisplayMode

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает режим отображения текущей консоли.

## Синтаксис

```
C  
BOOL WINAPI GetConsoleDisplayMode(  
    _Out_ LPDWORD lpModeFlags  
) ;
```

## Параметры

*lpModeFlags* [out]

Режим отображения консоли. Этот параметр может быть одним или несколькими из следующих значений.

 [Развернуть таблицу](#)

Значение	Значение
CONSOLE_FULLSCREEN 1	Полноэкранная консоль. Консоль находится в этом режиме, как только окно будет развернуто. На этом этапе переход на полноэкранный режим по-прежнему может завершиться ошибкой.

Значение	Значение
CONSOLE_FULLSCREEN_HARDWARE 2	Полноэкранная консоль напрямую взаимодействует с видео оборудованием. Этот режим устанавливается после того, как консоль находится в <b>режиме CONSOLE_FULLSCREEN</b> , чтобы указать, что переход на полноэкранный режим завершен.

### ⓘ Примечание

Переход на 100% полноэкранный режим оборудования видео был удален в Windows Vista с переформированием графического стека на **WDDM**. В более поздних версиях Windows максимальное итоговое состояние **CONSOLE\_FULLSCREEN** представляет окно без фреймов, которое отображается в полноэкранном режиме, но не находится под монопольным контролем оборудования.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите **\_WIN32\_WINNT** как **0x0500** или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

# Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Консольные режимы](#)

[SetConsoleDisplayMode](#)

# Функция GetConsoleFontSize

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает размер шрифта, используемого указанным буфером экрана консоли.

## Синтаксис

```
C  
  
COORD WINAPI GetConsoleFontSize(  
    _In_ HANDLE hConsoleOutput,  
    _In_ DWORD nFont  
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*nFont* [in]

Индекс шрифта, размер которого требуется извлечь. Этот индекс получается путем [вызыва функции GetCurrentConsoleFont](#).

## Возвращаемое значение

Если функция выполнена успешно, возвращаемое значение представляет собой **структуру COORD**, содержащую ширину и высоту каждого символа в шрифте в логических единицах. Элемент **X** содержит ширину, а элемент **Y** содержит высоту.

Если функция завершается ошибкой, ширина и высота равны нулю.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите **\_WIN32\_WINNT** как **0x0500** или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

**COORD**

[GetCurrentConsoleFont](#)

# Функция GetConsoleHistoryInfo

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает параметры журнала для консоли вызывающего процесса.

## Синтаксис

C

```
BOOL WINAPI GetConsoleHistoryInfo(
    _Out_ PCONSOLE_HISTORY_INFO lpConsoleHistoryInfo
);
```

## Параметры

*lpConsoleHistoryInfo* [out]

Указатель на [CONSOLE\\_HISTORY\\_INFO](#) структуру, которая получает параметры журнала для консоли вызывающего процесса.

## Возвращаемое значение

Если функция завершается успешно, возвращаемое значение ненулевое.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Если вызывающий процесс не является консольным процессом, функция завершается ошибкой и задает последнюю ошибку **ERROR\_ACCESS\_DENIED**.

## 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

# Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[CONSOLE\\_HISTORY\\_INFO](#)

[SetConsoleHistoryInfo](#)

# Функция GetConsoleMode

Статья • 12.07.2023

Извлекает текущий режим ввода входного буфера консоли или текущий режим вывода буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI GetConsoleMode(
    _In_    HANDLE  hConsoleHandle,
    _Out_   LPDWORD lpMode
);
```

## Параметры

*hConsoleHandle* [ввод]

Дескриптор входного буфера консоли или буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpMode* [out]

Указатель на переменную, которая получает текущий режим указанного буфера.

Если в качестве входного дескриптора используется параметр *hConsoleHandle*, режим может быть одним из следующих. При создании консоли все режимы ввода, кроме **ENABLE\_WINDOW\_INPUT** и **ENABLE\_VIRTUAL\_TERMINAL\_INPUT**, включены по умолчанию.

Значение	Значение
<b>ENABLE_ECHO_INPUT</b> 0x0004	Символы, считанные функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , записываются в активный буфер экрана по мере ввода в консоли. Этот режим можно использовать только в том случае, если также включен режим <b>ENABLE_LINE_INPUT</b> .
<b>ENABLE_INSERT_MODE</b> 0x0020	Если этот режим включен, вводимый в окне консоли текст будет вставлен по текущему расположению курсора, а весь последующий текст не будет

Значение	Значение
	перезаписан. Если этот режим отключен, весь последующий текст будет перезаписан.
ENABLE_LINE_INPUT 0x0002	Функция <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> возвращает значение только в том случае, если считан символ возврата каретки. Если этот режим отключен, функции возвращают значение при доступности одного или нескольких символов.
ENABLE_MOUSE_INPUT 0x0010	Если указатель мыши находится в границах окна консоли и окно находится в фокусе для ввода текста с клавиатуры, события мыши, связанные с ее перемещением и нажатиями кнопок, помещаются во входной буфер. Такие события удаляются функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , даже если этот режим включен. Функцию <a href="#">ReadConsoleInput</a> можно использовать для считывания входных записей <a href="#">MOUSE_EVENT</a> из входного буфера.
ENABLE_PROCESSED_INPUT 0x0001	Нажатие клавиш CTRL+C обрабатывается системой и не помещается во входной буфер. Если входной буфер считывается функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , нажатия других управляющих клавиш обрабатываются системой и не возвращаются в буфере <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> . Если также включен режим <a href="#">ENABLE_LINE_INPUT</a> , символы стирания назад, возврата каретки и перевода строки обрабатываются системой.
ENABLE_QUICK_EDIT_MODE 0x0040	Этот флаг позволяет пользователю использовать мышь для выбора и редактирования текста. Чтобы включить этот режим, используйте <code>ENABLE_QUICK_EDIT_MODE   ENABLE_EXTENDED_FLAGS</code> . Чтобы отключить этот режим, используйте <a href="#">ENABLE_EXTENDED_FLAGS</a> без этого флага.
ENABLE_WINDOW_INPUT 0x0008	Действия пользователей, которые приводят к изменению буфера экрана консоли, регистрируются во входном буфере консоли. Сведения о таких событиях могут быть считаны приложениями из входного буфера с помощью функции <a href="#">ReadConsoleInput</a> , но не могут быть считаны приложениями, использующими <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> .
ENABLE_VIRTUAL_TERMINAL_INPUT 0x0200	Этот флаг указывает модулю обработки виртуального терминала преобразовать ввод пользователя, полученный в окне консоли, в <a href="#">последовательности виртуального терминала консоли</a> , которые

Значение	Значение
	<p>вспомогательное приложение может получить с помощью функций <a href="#">WriteFile</a> или <a href="#">WriteConsole</a>.</p> <p>Обычно этот флаг рекомендуется использовать вместе с флагом <code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code> для выходного дескриптора, чтобы обеспечить подключение к приложению, которое обменивается данными исключительно через последовательности виртуального терминала.</p>

Если в качестве дескриптора буфера экрана используется параметр `hConsoleHandle`, режим может быть одним из следующих. При создании буфера экрана оба режима вывода включены по умолчанию.

Значение	Значение
<code>ENABLE_PROCESSED_OUTPUT</code> 0x0001	<p>Символы, записываемые функцией <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> либо выводимые функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a>, анализируются для управляющих последовательностей ASCII, после чего выполняется нужное действие.</p> <p>Обрабатываются символы стирания назад, возврата каретки и перевода строки. Этот режим следует включить, если используются управляющие последовательности или задано значение <code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code>.</p>
<code>ENABLE_WRAP_AT_EOL_OUTPUT</code> 0x0002	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> либо выводе с помощью функции <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> курсор перемещается в начало следующей строки, если он достиг конца текущей строки. Это приводит к тому, что строки, отображаемые в окне консоли, автоматически прокручиваются вверх, если курсор переходит далее с последней строки в окне. Кроме того, содержимое буфера экрана консоли также прокручивается вверх (с удалением верхней строки в буфере экрана консоли), если курсор переходит далее с последней строки в буфере экрана консоли.</p> <p>Если этот режим отключен, последний символ в строке будет перезаписан последующими символами.</p>
<code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code> 0x0004	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> символы обрабатываются для</p>

Значение	Значение
	<p>VT100 и аналогичных управляющих символьных последовательностей, которые управляют перемещением курсора, цветом и режимом шрифта, а также другими операциями, доступными для выполнения через существующие API-интерфейсы консоли. Дополнительные сведения см. в статье <a href="#">Последовательности виртуального терминала консоли</a>.</p> <p>При использовании этого флага убедитесь, что установлен флаг <code>ENABLE_PROCESSED_OUTPUT</code>.</p>
<code>DISABLE_NEWLINE_AUTO_RETURN</code> 0x0008	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> к переносу в конце строки добавляется дополнительное состояние, которое задерживает перемещение курсора и помещает операции прокрутки в буфер.</p> <p>Как правило, если флаг <code>ENABLE_WRAP_AT_EOL_OUTPUT</code> установлен и текст достигает конца строки, курсор немедленно переходит на следующую строку, а содержимое буфера прокручивается на одну строку. Но если задан этот флаг, курсор не переходит на следующую строку, а операция прокрутки не выполняется. Записанный символ будет выведен в последней позиции строки, а курсор будет располагаться над этим символом (как в случае с отключенным флагом <code>ENABLE_WRAP_AT_EOL_OUTPUT</code>). Но следующий печатаемый символ будет выведен таким образом, как если бы флаг <code>ENABLE_WRAP_AT_EOL_OUTPUT</code> был включен. Перезапись при этом не выполняется. В частности, курсор быстро переходит на следующую строку, при необходимости выполняется прокрутка, символ выводится, а курсор передвигается еще на одну позицию.</p> <p>Обычно этот флаг рекомендуется использовать вместе с флагом <code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code>, что позволяет оптимально сымитировать эмулятор терминала, в котором запись последнего символа на экране (в правом нижнем углу) без немедленной прокрутки является желаемым поведением.</p>

Значение	Значение
ENABLE_LVB_GRID_WORLDWIDE 0x0010	API-интерфейсы для записи атрибутов символов, в том числе <a href="#">WriteConsoleOutput</a> и <a href="#">WriteConsoleOutputAttribute</a> , позволяют использовать флаги <a href="#">атрибутов символов</a> для изменения цвета переднего плана и фона текста. Кроме того, некоторые флаги DBCS указываются с префиксом COMMON_LVB. Исторически эти флаги работали только в кодовых страницах DBCS для китайского, японского и корейского языков.
	За исключением флагов начальных и конечных байтов оставшиеся флаги, описывающие отрисовку строки и обратный видеовывод (смена местами цветов переднего плана и фона), можно использовать и с другими языками для выделения определенных частей выходных данных.
	Установка этого флага режима консоли позволяет использовать эти атрибуты для каждой кодовой страницы любого языка.
	По умолчанию он отключен для сохранения совместимости с известными приложениями, которые исторически игнорируют такие флаги на компьютерах без поддержки китайского, японского и корейского языков для хранения битов в таких полях (случайно или с собственными целями).
	Обратите внимание, что использование режима ENABLE_VIRTUAL_TERMINAL_PROCESSING может привести к установке флагов сетки LVB и обратного видеовывода, хотя этот флаг не включается, если подключенное приложение запрашивает видеовывод с подчеркиванием или инвертированием через <a href="#">последовательности виртуального терминала консоли</a> .

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

Консоль состоит из входного буфера и одного или нескольких буферов экрана.

Режим буфера консоли определяет функционирование консоли во время операций ввода-вывода. Один набор констант с флагом используется с дескрипторами ввода, а другой — с дескрипторами буфера экрана (вывод).

Задание режимов вывода одного буфера экрана не влияет на режимы вывода других буферов экрана.

Режимы `ENABLE_LINE_INPUT` и `ENABLE_ECHO_INPUT` влияют только на процессы, в которых используются [ReadFile](#) или [ReadConsole](#) для чтения данных из входного буфера консоли. Аналогичным образом режим `ENABLE_PROCESSED_INPUT` в первую очередь влияет на пользователей [ReadFile](#) и [ReadConsole](#), кроме того, что он также определяет, передается ли ввод данных с помощью `CTRL+C` во входной буфер (для чтения функцией [ReadConsoleInput](#)) или в функцию, определенную приложением.

Режимы `ENABLE_WINDOW_INPUT` и `ENABLE_MOUSE_INPUT` определяют, будут ли передаваться во входной буфер данные изменения размера окна и действий мыши. Эти события могут считываться с помощью [ReadConsoleInput](#), но они всегда фильтруются с помощью [ReadFile](#) и [ReadConsole](#).

Режимы `ENABLE_PROCESSED_OUTPUT` и `ENABLE_WRAP_AT_EOL_OUTPUT` влияют только на процессы с использованием [ReadFile](#) или [ReadConsole](#) и [WriteFile](#) или [WriteConsole](#).

Чтобы изменить режимы ввода-вывода консоли, вызовите функцию [SetConsoleMode](#).

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента	Windows 2000 Профессиональная [классические приложения   Приложения UWP]
Минимальная версия сервера	Windows 2000 Server [классические приложения   Приложения UWP]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll

## См. также

[Функции консоли](#)

[Консольные режимы](#)

[ReadConsole](#)

[ReadConsoleInput](#)

[ReadFile](#)

[SetConsoleMode](#)

[WriteConsole](#)

[WriteFile](#)

# Функция GetConsoleOriginalTitle

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает исходное название текущего окна консоли.

## Синтаксис

C

```
DWORD WINAPI GetConsoleOriginalTitle(
    _Out_ LPTSTR lpConsoleTitle,
    _In_  DWORD   nSize
);
```

## Параметры

*lpConsoleTitle* [out]

Указатель на буфер, получающий строку, завершающую значение NULL, содержащую исходное название.

*nSize* [in]

Размер буфера *lpConsoleTitle* в символах.

## Возвращаемое значение

Если значение *nSize* равно нулю, возвращаемое значение равно нулю.

Если функция выполнена успешно, возвращаемое значение имеет длину исходного заголовка консоли в символах.

Если функция завершается ошибкой, возвращаемое значение равно нулю, и [GetLastError](#) возвращает код ошибки.

## Замечания

Чтобы задать заголовок окна консоли, используйте [функцию SetConsoleTitle](#).

Чтобы получить текущую строку заголовка, используйте [функцию GetConsoleTitle](#).

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0600` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#).

Это решение намеренно сопоставляет платформу Windows с другими операционными системами. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<a href="#">GetConsoleOriginalTitleW</a> (Юникод) и <a href="#">GetConsoleOriginalTitleA</a> (ANSI)

## **См. также**

[Функции консоли](#)

[GetConsoleTitle](#)

[SetConsoleTitle](#)

# Функция GetConsoleOutputCP

Статья • 23.10.2023

Извлекает выходную кодовую страницу, используемую консолью, связанной с вызывающим процессом. Консоль использует свою выходную кодовую страницу для перевода значений символов, написанных различными выходными функциями в изображения, отображаемые в окне консоли.

## Синтаксис

C

```
UINT WINAPI GetConsoleOutputCP(void);
```

## Параметры

У этой функции нет параметров.

## Возвращаемое значение

Возвращаемое значение — это код, определяющий кодовую страницу. Список идентификаторов см. в разделе "[Идентификаторы кодовой страницы](#)".

Если возвращаемое значение равно нулю, функция завершилась ошибкой. Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Кодовая страница сопоставляет 256 кодов символов с отдельными символами. Разные кодовые страницы включают разные специальные символы, как правило, настроенные для языка или группы языков. Дополнительные сведения о кодовой странице, включая имя, см. в [функции GetCPIInfoEx](#).

Чтобы задать выходную страницу кода консоли, используйте [функцию SetConsoleOutputCP](#). Чтобы задать и запросить входную кодовую страницу консоли, используйте функции [SetConsoleCP](#) и [GetConsoleCP](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Кодовые страницы консоли](#)

[Функции консоли](#)

[GetConsoleCP](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

# Функция GetConsoleProcessList

Статья • 23.10.2023

Извлекает список процессов, подключенных к текущей консоли.

## Синтаксис

С

```
DWORD WINAPI GetConsoleProcessList(
    _Out_ LPDWORD lpdwProcessList,
    _In_  DWORD    dwProcessCount
);
```

## Параметры

*lpdwProcessList* [out]

Указатель на буфер, получающий массив идентификаторов процесса при успешном выполнении. Это должен быть допустимый буфер и не может быть `NULL`. Буфер должен иметь пространство для получения не менее 1 возвращаемого идентификатора процесса.

*dwProcessCount* [in]

Максимальное количество идентификаторов процесса, которые могут храниться в буфере *lpdwProcessList*. Значение должно быть больше 0.

## Возвращаемое значение

Если функция выполнена успешно, возвращаемое значение меньше или равно *dwProcessCount* и представляет количество идентификаторов процессов, хранящихся в буфере *lpdwProcessList*.

Если буфер слишком мал для хранения всех допустимых идентификаторов процесса, возвращаемое значение является обязательным числом элементов массива. Функция не будет хранить идентификаторы в буфере. В этой ситуации используйте возвращаемое значение для выделения буфера, достаточно большого размера, чтобы сохранить весь список и снова вызвать функцию.

Если возвращаемое значение равно нулю, функция завершилась ошибкой, так как каждая консоль имеет по крайней мере один процесс, связанный с ним.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

**NULL** Если был указан список процессов или число процессов было 0, вызов вернется 0 и `GetLastError` вернет `ERROR_INVALID_PARAMETER`. Укажите буфер по крайней мере одного элемента для вызова этой функции. Выделите больший буфер и снова вызовите, если возвращаемый код больше длины предоставленного буфера.

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0501` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно сопоставляет платформу Windows с другими операционными системами. Это состояние относится только к локальному пользователю, сеансу и контексту привилегий. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[AttachConsole](#)

## Функции консоли

# Функция GetConsoleScreenBufferInfo

Статья • 13.12.2023

Извлекает сведения о указанном буфере экрана консоли.

## Синтаксис

C

```
BOOL WINAPI GetConsoleScreenBufferInfo(
    _In_     HANDLE                 hConsoleOutput,
    _Out_    PCONSOLE_SCREEN_BUFFER_INFO lpConsoleScreenBufferInfo
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpConsoleScreenBufferInfo* [out]

Указатель на структуру **CONSOLE\_SCREEN\_BUFFER\_INFO**, которая получает сведения о буфере экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Прямоугольник, возвращаемый в элементе [srWindow структуры CONSOLE\\_SCREEN\\_BUFFER\\_INFO](#), можно изменить, а затем передать функцию [SetConsoleWindowInfo](#) для прокрутки буфера экрана консоли в окне, чтобы изменить размер окна или оба.

Все координаты, возвращаемые в структуре `CONSOLE_SCREEN_BUFFER_INFO`, находятся в координатах ячейки символов, где источник (0, 0) находится в левом верхнем углу буфера экрана консоли.

### 💡 Совет

Этот API не имеет эквивалента виртуального [терминала](#). Его использование может по-прежнему потребоваться для приложений, которые пытаются нарисовать столбцы, сетки или заполнить отображение для получения размера окна. Это состояние окна управляется TTY/PTY/Pseudoconsole за пределами обычного потока потока и обычно считается привилегией пользователя, не настраиваемой клиентским приложением. Обновления можно получить на [ReadConsoleInput](#).

## Примеры

Пример см. в разделе "[Прокрутка окна](#) буфера экрана".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[CONSOLE\\_SCREEN\\_BUFFER\\_INFO](#)

[\*\*GetLargestConsoleWindowSize\*\*](#)

[\*\*SetConsoleCursorPosition\*\*](#)

[\*\*SetConsoleScreenBufferSize\*\*](#)

[\*\*SetConsoleWindowInfo\*\*](#)

Размер буфера окна и экрана

# Функция GetConsoleScreenBufferInfoEx

Статья • 23.10.2023

Извлекает расширенные сведения о указанном буфере экрана консоли.

## Синтаксис

C

```
BOOL WINAPI GetConsoleScreenBufferInfoEx(
    _In_     HANDLE                 hConsoleOutput,
    _Out_    PCONSOLE_SCREEN_BUFFER_INFOEX lpConsoleScreenBufferInfoEx
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpConsoleScreenBufferInfoEx* [out]

Структура **CONSOLE\_SCREEN\_BUFFER\_INFOEX**, которая получает запрошенные сведения о буфере экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Прямоугольник, возвращенный в элементе [srWindow структуры CONSOLE\\_SCREEN\\_BUFFER\\_INFOEX](#), можно изменить, а затем передать в функцию [SetConsoleWindowInfo](#) для прокрутки буфера экрана консоли в окне, чтобы изменить размер окна или оба.

Все координаты, возвращаемые в структуре CONSOLE\_SCREEN\_BUFFER\_INFOEX, находятся в координатах ячейки символов, где источник (0, 0) находится в левом верхнем углу буфера экрана консоли.

### 💡 Совет

Этот API не имеет эквивалента виртуального [терминала](#). Его использование может по-прежнему потребоваться для приложений, которые пытаются нарисовать столбцы, сетки или заполнить отображение для получения размера окна. Это состояние окна управляется TTY/PTY/Pseudoconsole за пределами обычного потока потока и обычно считается привилегией пользователя, не настраиваемой клиентским приложением. Обновления можно получить на [ReadConsoleInput](#).

## Требования

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[CONSOLE\\_SCREEN\\_BUFFER\\_INFOEX](#)

[SetConsoleScreenBufferSizeEx](#)

# Функция GetConsoleSelectionInfo

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает сведения о текущем выборе консоли.

## Синтаксис

C

```
BOOL WINAPI GetConsoleSelectionInfo(
    _Out_ PCONSOLE_SELECTION_INFO lpConsoleSelectionInfo
);
```

## Параметры

*lpConsoleSelectionInfo* [out]

Указатель на структуру [CONSOLE\\_SELECTION\\_INFO](#), которая получает сведения о выборе.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите \_WIN32\_WINNT как 0x0500 или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

## 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

# Requirements

[] [Развернуть таблицу](#)

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

# См. также

[Функции консоли](#)

[Выбор консоли](#)

[CONSOLE\\_SELECTION\\_INFO](#)

# Функция GetConsoleTitle

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает заголовок текущего окна консоли.

## Синтаксис

C

```
DWORD WINAPI GetConsoleTitle(
    _Out_ LPTSTR lpConsoleTitle,
    _In_  DWORD   nSize
);
```

## Параметры

*lpConsoleTitle* [out]

Указатель на буфер, получающий строку, завершающую значение NULL, содержащую заголовок. Если буфер слишком мал, чтобы сохранить заголовок, функция сохраняет столько символов заголовка, сколько будет соответствовать буферу, заканчивая концом null.

*nSize* [in]

Размер буфера, на который указывает параметр *lpConsoleTitle*, в символах.

## Возвращаемое значение

Если функция выполнена успешно, возвращаемое значение является длиной заголовка окна консоли в символах.

Если функция завершается ошибкой, возвращаемое значение равно нулю, и [GetLastError](#) возвращает код ошибки.

## Замечания

Чтобы задать заголовок окна консоли, используйте [функцию SetConsoleTitle](#).

Чтобы получить исходную строку заголовка, используйте [функцию GetConsoleOriginalTitle](#).

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно сопоставляет платформу Windows с другими операционными системами. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Примеры

Пример см. в разделе [SetConsoleTitle](#).

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]

Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>GetConsoleTitleW</b> (Юникод) и <b>GetConsoleTitleA</b> (ANSI)

## См. также

[ФУНКЦИИ КОНСОЛИ](#)

[GetConsoleOriginalTitle](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[SetConsoleTitle](#)

# Функция GetConsoleWindow

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает дескриптор окна, используемый консолью, связанной с вызывающим процессом.

## Синтаксис

C

```
HWND WINAPI GetConsoleWindow(void);
```

## Параметры

У этой функции нет параметров.

## Возвращаемое значение

Возвращаемое значение — это дескриптор окна, используемого консолью, связанной с вызывающим процессом или **NULL**, если такой связанной консоли нет.

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0500` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно сопоставляет платформу Windows с другими операционными системами. Это состояние относится только к локальному пользователю, сеансу и контексту привилегий. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

Для приложения, размещенного в сеансе **псевдоконсоля**, эта функция возвращает дескриптор окна только для целей очереди сообщений. Связанное окно не отображается локально, так как псевдоконсол серIALIZУЕТ все действия в поток для презентации в другом окне терминала в другом окне терминала в другом месте.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

# Функция GetCurrentConsoleFont

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает сведения о текущем шрифте консоли.

## Синтаксис

C

```
BOOL WINAPI GetCurrentConsoleFont(
    _In_     HANDLE          hConsoleOutput,
    _In_     BOOL            bMaximumWindow,
    _Out_    PCONSOLE_FONT_INFO lpConsoleCurrentFont
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*bMaximumWindow* [in]

Если этот параметр имеет значение **TRUE**, сведения о шрифте извлекаются для максимального размера окна. Если этот параметр имеет значение **FALSE**, сведения о шрифте извлекаются для текущего размера окна.

*lpConsoleCurrentFont* [out]

Указатель на структуру [CONSOLE\\_FONT\\_INFO](#), которая получает запрошенные сведения о шрифте.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите `_WIN32_WINNT` как `0x0500` или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Requirements

[+] [Развернуть таблицу](#)

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

[CONSOLE\\_FONT\\_INFO](#)

[GetConsoleFontSize](#)

# Функция GetCurrentConsoleFontEx

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Извлекает расширенные сведения о текущем шрифте консоли.

## Синтаксис

C

```
BOOL WINAPI GetCurrentConsoleFontEx(
    _In_     HANDLE          hConsoleOutput,
    _In_     BOOL            bMaximumWindow,
    _Out_    PCONSOLE_FONT_INFOEX lpConsoleCurrentFontEx
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*bMaximumWindow* [in]

Если этот параметр имеет значение **TRUE**, сведения о шрифте извлекаются для максимального размера окна. Если этот параметр имеет значение **FALSE**, сведения о шрифте извлекаются для текущего размера окна.

*lpConsoleCurrentFontEx* [out]

Указатель на структуру CONSOLE\_FONT\_INFOEX, которая получает запрошенные сведения о шрифте.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[CONSOLE\\_FONT\\_INFOEX](#)

# Функция GetLargestConsoleWindowSize

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использований для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли](#) и в [документе виртуального терминала](#).

Извлекает размер максимального возможного окна консоли на основе текущего шрифта и размера дисплея.

## Синтаксис

```
С

COORD WINAPI GetLargestConsoleWindowSize(
    _In_ HANDLE hConsoleOutput
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли.

## Возвращаемое значение

Если функция выполнена успешно, возвращаемое значение представляет собой [структуру COORD](#), указывающую количество столбцов ячейки символов (член X) и

строк (член `Y`) в максимально возможном окне консоли. В противном случае элементы структуры равны нулю.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Функция не учитывает размер буфера экрана консоли, что означает, что возвращаемый размер окна может быть больше размера буфера экрана консоли.

Функцию [GetConsoleScreenBufferInfo](#) можно использовать для определения максимального размера окна консоли, учитывая текущий размер буфера экрана, текущий шрифт и размер дисплея.

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

**COORD**

**GetConsoleScreenBufferInfo**

**SetConsoleWindowInfo**

Размер буфера окна и экрана

# Функция GetNumberOfConsoleInputEvents

Статья • 12.07.2023

Извлекает количество непрочитанных входных записей во входном буфере консоли.

## Синтаксис

C

```
BOOL WINAPI GetNumberOfConsoleInputEvents(
    _In_    HANDLE  hConsoleInput,
    _Out_   LPDWORD lpcNumberOfEvents
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpcNumberOfEvents* [out]

Указатель на переменную, которая получает количество непрочитанных входных записей во входном буфере консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

Функция **GetNumberOfConsoleInputEvents** сообщает общее количество непрочитанных входных записей во входном буфере, включая клавиатуру, мышь и входные записи с изменением размера окна. Процессы, использующие функцию

[ReadFile](#) или [ReadConsole](#) , могут считывать только ввод с клавиатуры. Процессы, использующие функцию [ReadConsoleInput](#) , могут считывать все типы входных записей.

Процесс может указать дескриптор буфера ввода консоли в одной из [функций ожидания](#) , чтобы определить, когда есть непрочитанные входные данные консоли. Если входной буфер не пуст, сигнализирует о состоянии дескриптора входного буфера консоли.

Чтобы считывать входные записи из входного буфера консоли, не влияя на количество непрочитанных записей, используйте функцию [PeekConsoleInput](#) .

Чтобы удалить все непрочитанные записи во входном буфере консоли, используйте функцию [FlushConsoleInputBuffer](#) .

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll

## См. также

[Функции консоли](#)

[FlushConsoleInputBuffer](#)

[Низкоуровневые функции ввода консоли](#)

[PeekConsoleInput](#)

[ReadConsole](#)

[ReadConsoleInput](#)

[ReadFile](#)

# Функция GetNumberOfConsoleMouseButtons

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использований для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли](#) и в [документе виртуального терминала](#).

Извлекает количество кнопок мыши, используемых текущей консолью.

## Синтаксис

C

```
BOOL WINAPI GetNumberOfConsoleMouseButtons(
    _Out_ LPDWORD lpNumberOfMouseButtons
);
```

## Параметры

*lpNumberOfMouseButtons* [out]

Указатель на переменную, которая получает количество кнопок мыши.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Когда консоль получает входные данные мыши, [INPUT\\_RECORD структура, содержащая структуру MOUSE\\_EVENT\\_RECORD](#), помещается в входной буфер консоли. Элемент dwButtonState [MOUSE\\_EVENT\\_RECORD](#) имеет бит, указывающий состояние каждой кнопки мыши. Бит равен 1, если кнопка вниз и 0, если кнопка находится вверх. Чтобы определить количество битов, которые являются значительными, используйте [GetNumberOfConsoleMouseButtons](#).

## 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно сопоставляет платформу Windows с другими операционными системами. Это состояние относится только к локальному пользователю, сеансу и контексту привилегий. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

# Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

# См. также

[Функции консоли](#)

[Входной буфер консоли](#)

[ReadConsoleInput](#)

**INPUT\_RECORD**

**MOUSE\_EVENT\_RECORD**

# Функция GetStdHandle

Статья • 12.07.2023

Извлекает дескриптор для указанного стандартного устройства (стандартный ввод, стандартный вывод или стандартная ошибка).

## Синтаксис

C

```
HANDLE WINAPI GetStdHandle(  
    _In_ DWORD nStdHandle  
) ;
```

## Параметры

*nStdHandle* [ввод]

Стандартное устройство. Этот параметр может принимать одно из указанных ниже значений.

Значение	Значение
STD_INPUT_HANDLE ((DWORD)-10)	Стандартное устройство ввода. Изначально это входной буфер консоли, CONIN\$.
STD_OUTPUT_HANDLE ((DWORD)-11)	Стандартное выходное устройство. Изначально это активный буфер экрана консоли, CONOUT\$.
STD_ERROR_HANDLE ((DWORD)-12)	Устройство стандартных ошибок. Изначально это активный буфер экрана консоли, CONOUT\$.

### ⓘ Примечание

Значения этих констант являются числами без знака, но определяются в файлах заголовков как приведение из числа со знаком и используют компилятор C для их преобразования к 32-разрядному значению, которое ниже максимального. При взаимодействии с этими дескрипторами на языке, который не анализирует заголовки и переопределяет константы, следует учитывать это ограничение. В качестве примера ((DWORD)-10) — это число 4294967286 без знака.

# Возвращаемое значение

Если функция выполняется успешно, возвращается дескриптор для указанного устройства или перенаправленный дескриптор, заданный предыдущим вызовом метода [SetStdHandle](#). Дескриптор имеет права доступа **GENERIC\_READ** и **GENERIC\_WRITE**, если только с помощью метода [SetStdHandle](#) в приложении не был задан стандартный дескриптор с более ограниченным доступом.

## 💡 Совет

По завершении работы удалять этот дескриптор с помощью [CloseHandle](#) необязательно. Дополнительные сведения см. в [примечаниях](#).

Если функция завершается неудачно, возвращается значение **INVALID\_HANDLE\_VALUE**. Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

Если приложение не имеет связанных стандартных дескрипторов, например службы, работающей на интерактивном рабочем столе, и не перенаправляет их, возвращается значение **NULL**.

## Комментарии

Дескрипторы, возвращаемые методом [GetStdHandle](#), могут использоваться приложениями, которые должны выполнять чтение или запись данных в консоли. При создании консоли стандартный дескриптор ввода представляет собой дескриптор для входного буфера консоли, а стандартный дескриптор вывода и стандартный дескриптор ошибок являются дескрипторами активного буфера экрана консоли. Эти дескрипторы могут использоваться функциями [ReadFile](#) и [WriteFile](#) или любыми консольными функциями, которые обращаются к входному буферу консоли или буферу экрана (например, функции [ReadConsoleInput](#), [WriteConsole](#) или [GetConsoleScreenBufferInfo](#)).

Стандартные дескрипторы процесса могут перенаправляться с помощью вызова [SetStdHandle](#). В этом случае [GetStdHandle](#) возвращает перенаправленный дескриптор. Если стандартные дескрипторы были перенаправлены, можно указать значение `CONIN$` в вызове функции [CreateFile](#), чтобы получить дескриптор для входного буфера консоли. Аналогичным образом можно указать значение `CONOUT$`, чтобы получить дескриптор для активного буфера экрана консоли.

Стандартные дескрипторы процесса вначале метода `main` определяются конфигурацией флага `/SUBSYSTEM`, переданного компоновщику при создании приложения. Если указать `/SUBSYSTEM:CONSOLE`, операционная система получит запрос на заполнение дескрипторов данными сеанса консоли при запуске, если родительский объект еще не заполнил стандартную таблицу дескрипторов путем наследования. `/SUBSYSTEM:WINDOWS`, в свою очередь, означает, что приложению не нужна консоль и, скорее всего, не будут использоваться стандартные дескрипторы. Дополнительные сведения о наследовании дескриптора можно найти в документации по [STARTF\\_USESTDHANDLES](#).

Некоторые приложения работают за пределами границ объявленной подсистемы. Например, приложение `/SUBSYSTEM:WINDOWS` может проверять или использовать стандартные дескрипторы для ведения журнала или отладки, но нормально работать с графическим пользовательским интерфейсом. Эти приложения должны тщательно проверять состояние стандартных дескрипторов при запуске и использовать [AttachConsole](#), [AllocConsole](#) и [FreeConsole](#), чтобы добавить или удалить консоль при необходимости.

Некоторые приложения могут также изменять свое поведение в зависимости от типа унаследованного дескриптора. Устранить неоднозначности типа консоли, конвейера, файла и других элементов можно с помощью [GetFileType](#).

## Освобождение дескриптора

По завершении работы с дескриптором, полученным от [GetStdHandle](#), использовать [CloseHandle](#) необязательно. Возвращаемое значение — это просто копия значения, хранимого в таблице процессов. Сам процесс, как правило, считается владельцем этих дескрипторов и их времени существования. Каждый дескриптор помещается в таблицу при создании в зависимости от наследования. При этом выполняется определенный вызов [CreateProcess](#), и дескриптор будет освобожден при уничтожении процесса.

Изменение времени существования этих дескрипторов вручную может быть желательным для приложения, которое намеренно пытается заменить или заблокировать их использование другими частями процесса. Так как `HANDLE` можно кэшировать путем выполнения кода, этот код не обязательно будет получать изменения, внесенные через [SetStdHandle](#). Закрытие дескриптора явным образом с помощью [CloseHandle](#) закроет его на уровне процесса, и при следующем использовании любой кэшированной ссылки возникнет ошибка.

Действия по замене стандартного дескриптора в таблице процессов заключаются в получении существующего экземпляра `HANDLE` из таблицы с помощью

`GetStdHandle`. Используйте `SetStdHandle` для размещения нового экземпляра `HANDLE` в экземпляре, открытый с помощью `CreateFile` (или аналогичной функции), а затем закрытия полученного дескриптора.

Проверка значений, хранимых в виде дескрипторов в таблице процессов, функциями `GetStdHandle` или `SetStdHandle` не выполняется. Проверка выполняется во время фактической операции чтения или записи, такой как `ReadFile` или `WriteFile`.

## Поведение при подключении и отключении

При подключении к новой консоли стандартные дескрипторы всегда заменяются дескрипторами консоли, если только во время создания процесса не указано `STARTF_USESTDHANDLES`.

Если имеющееся значение стандартного дескриптора равно `NULL` или выглядит как псевдодескриптор консоли, этот дескриптор заменяется дескриптором консоли.

Если родительский элемент для создания консольного процесса использует `CREATE_NEW_CONSOLE` и `STARTF_USESTDHANDLES`, стандартные дескрипторы не будут заменены, если только стандартный дескриптор не имеет значение `NULL` или имеющееся значение не является псевдодескриптором.

### ⓘ Примечание

Консольные процессы должны начинаться с заполнения стандартных дескрипторов, иначе они будут автоматически заполнены соответствующими дескрипторами для новой консоли. Приложения с графическим пользовательским интерфейсом могут запускаться без стандартных дескрипторов и их автоматического заполнения.

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента Windows 2000 Professional [только классические

	приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ProcessEnv.h (через Winbase.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll

## См. также

[Функции консоли](#)

[Дескрипторы консоли](#)

[CreateFile](#)

[GetConsoleScreenBufferInfo](#)

[ReadConsoleInput](#)

[ReadFile](#)

[SetStdHandle](#)

[WriteConsole](#)

[WriteFile](#)

# Функция обратного вызова HandlerRoutine

Статья • 12.07.2023

Определяемая приложением функция, используемая с функцией [SetConsoleCtrlHandler](#). Консольный процесс использует эту функцию для обработки сигналов управления, полученных процессом. При получении сигнала система создает новый поток в процессе для выполнения функции.

Тип **PHANDLER\_ROUTINE** определяет указатель на эту функцию обратного вызова. **HandlerRoutine** — это заполнитель для имени функции, определяемой приложением.

## Синтаксис

C

```
BOOL WINAPI HandlerRoutine(
    _In_ DWORD dwCtrlType
);
```

## Параметры

*dwCtrlType* [in]

Тип сигнала управления, полученного обработчиком. Этот параметр может принимать одно из указанных ниже значений.

Значение	Значение
CTRL_C_EVENT 0	Сигнал <b>CTRL</b> + <b>C</b> был получен из ввода с клавиатуры или из сигнала, созданного функцией <a href="#">GenerateConsoleCtrlEvent</a> .
CTRL_BREAK_EVENT 1	Сигнал <b>CTRL</b> + <b>BREAK</b> был получен из ввода с клавиатуры или сигнала, созданного <a href="#">GenerateConsoleCtrlEvent</a> .
CTRL_CLOSE_EVENT 2	Сигнал, который система отправляет всем процессам, подключенным к консоли, когда пользователь закрывает консоль (нажав кнопку <b>Закрыть</b> в меню окна консоли или нажав кнопку <b>Завершить задачу</b> в диспетчере задач).
CTRL_LOGOFF_EVENT 5	Сигнал, который система отправляет всем процессам консоли при выходе пользователя из системы. Этот сигнал не указывает,

Значение	Значение
	какой пользователь выполняет выход, поэтому нельзя сделать никаких предположений.
CTRL_SHUTDOWN_EVENT 6	<p>Обратите внимание, что этот сигнал получается только службами. Интерактивные приложения завершаются при выходе из системы, поэтому они отсутствуют, когда система отправляет этот сигнал.</p> <p>Сигнал, который система отправляет при завершении работы системы. Интерактивные приложения отсутствуют к тому времени, когда система отправляет этот сигнал, поэтому в этой ситуации его можно получить только в службах. Службы также имеют собственный механизм уведомления о событиях завершения работы. Дополнительные сведения см. в разделе <a href="#">Обработчик</a>.</p> <p>Этот сигнал также может быть создан приложением с помощью <a href="#">GenerateConsoleCtrlEvent</a>.</p>

## Возвращаемое значение

Если функция обрабатывает сигнал управления, она должна возвращать значение **TRUE**. Если возвращается значение **FALSE**, используется следующая функция обработчика в списке обработчиков для этого процесса.

## Комментарии

Поскольку система создает новый поток в процессе для выполнения функции обработчика, возможно, функция обработчика будет завершена другим потоком в процессе. Не забудьте синхронизировать потоки в процессе с потоком для функции обработчика.

Каждый процесс консоли имеет собственный список функций **HandlerRoutine**. Изначально этот список содержит только функцию обработчика по умолчанию, которая вызывает [ExitProcess](#). Консольный процесс добавляет или удаляет дополнительные функции обработчика путем вызова функции [SetConsoleCtrlHandler](#), которая не влияет на список функций обработчика для других процессов. Когда консольный процесс получает любой из управляющих сигналов, его функции обработчика вызываются по последней регистрации, сначала вызываемой основе, пока один из обработчиков не вернет **TRUE**. Если ни один из обработчиков не возвращает значение **TRUE**, вызывается обработчик по умолчанию.

Сигналы `CTRL_CLOSE_EVENT`, `CTRL_LOGOFF_EVENT` и `CTRL_SHUTDOWN_EVENT` дают возможность очистить процесс до завершения. `HandlerRoutine` может выполнить любую необходимую очистку, а затем выполнить одно из следующих действий:

- Вызовите функцию `ExitProcess`, чтобы завершить процесс.
- Возвращает значение `FALSE`. Если ни одна из зарегистрированных функций обработчика не возвращает значение `TRUE`, обработчик по умолчанию завершает процесс.
- Возвращает значение `TRUE`. В этом случае другие функции обработчика не вызываются, и система завершает процесс.

Процесс может использовать функцию `SetProcessShutdownParameters`, чтобы предотвратить отображение системой диалогового окна для пользователя во время выхода из системы или завершения работы. В этом случае система завершает процесс, когда `HandlerRoutine` возвращает значение `TRUE` или истекает время ожидания.

Когда консольное приложение запускается как служба, оно получает измененный обработчик управления консоли по умолчанию. Этот измененный обработчик не вызывает `ExitProcess` при обработке сигналов `CTRL_LOGOFF_EVENT` и `CTRL_SHUTDOWN_EVENT`. Это позволяет службе продолжать работу после выхода пользователя из системы. Если служба устанавливает собственный обработчик управления консоли, этот обработчик вызывается перед обработчиком по умолчанию. Если установленный обработчик вызывает `ExitProcess` при обработке сигнала `CTRL_LOGOFF_EVENT`, служба завершает работу при выходе пользователя из системы.

Обратите внимание, что сторонняя библиотека или библиотека DLL могут установить обработчик элементов управления консоли для вашего приложения. Если это так, этот обработчик переопределяет обработчик по умолчанию и может привести к выходу приложения при выходе пользователя.

## Истекшее время ожидания

Событие	Обстоятельства	Время ожидания
<code>CTRL_CLOSE_EVENT</code>	Любой	системный параметр <code>SPI_GETHUNGAPPTIMEOUT</code> , 5000 мс
<code>CTRL_LOGOFF_EVENT</code>	<code>quick[1]</code>	раздел <code>CriticalAppShutdownTimeout</code> реестра или 500 мс

<b>Событие</b>	<b>Обстоятельства</b>	<b>Время ожидания</b>
<code>CTRL_LOGOFF_EVENT</code>	ни один из перечисленных выше	системный параметр <code>SPI_GETWAITTOKILLTIMEOUT</code> , 5000 мс
<code>CTRL_SHUTDOWN_EVENT</code>	Процесс обслуживания	параметр <code>SPI_GETWAITTOKILLSERVICETIMEOUT</code> system , 20000ms
<code>CTRL_SHUTDOWN_EVENT</code>	<i>quick</i> [1]	раздел <code>CriticalAppShutdownTimeout</code> реестра или 500 мс
<code>CTRL_SHUTDOWN_EVENT</code>	ни один из перечисленных выше	системный параметр <code>SPI_GETWAITTOKILLTIMEOUT</code> , 5000 мс
<code>CTRL_C</code> , <code>CTRL_BREAK</code>	Любой	нет времени ожидания

[1]: "быстрые" события никогда не используются, но есть еще код для их поддержки.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)

## См. также

[Обработчики команд управления в консоли](#)

[Функции консоли](#)

[ExitProcess](#)

[GenerateConsoleCtrlEvent](#)

[GetProcessShutdownParameters](#)

[SetConsoleCtrlHandler](#)

[SetProcessShutdownParameters](#)

# Функция PeekConsoleInput

Статья • 13.12.2023

Считывает данные из указанного входного буфера консоли, не удаляя его из буфера.

## Синтаксис

C

```
BOOL WINAPI PeekConsoleInput(
    _In_     HANDLE      hConsoleInput,
    _Out_    PINPUT_RECORD lpBuffer,
    _In_     DWORD       nLength,
    _Out_    LPDWORD     lpNumberOfEventsRead
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [out]

Указатель на массив INPUT\_RECORD структур, получающих входные данные буфера.

*nLength* [in]

Размер массива, на который указывает параметр *lpBuffer*, в элементах массива.

*lpNumberOfEventsRead* [out]

Указатель на переменную, которая получает количество операций чтения входных записей.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Если количество запрошенных записей превышает количество записей, доступных в буфере, число доступно для чтения. Если данные недоступны, функция возвращается немедленно.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>PeekConsoleInputW</code> (Юникод) и <code>PeekConsoleInputA</code> (ANSI)

## См. также

[Функции консоли](#)

[ReadConsoleInput](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[WriteConsoleInput](#)

## INPUT\_RECORD

# Функция ReadConsole

Статья • 12.07.2023

Считывает входные символы из входного буфера консоли и удаляет их из буфера.

## Синтаксис

C

```
BOOL WINAPI ReadConsole(
    _In_      HANDLE hConsoleInput,
    _Out_     LPVOID lpBuffer,
    _In_      DWORD nNumberOfCharsToRead,
    _Out_     LPDWORD lpNumberOfCharsRead,
    _In_opt_  LPVOID pInputControl
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [out]

Указатель на буфер, который получает данные, считываемые из входного буфера консоли.

*nNumberOfCharsToRead* [in]

Число считываемых символов. Размер буфера, на который указывает параметр *lpBuffer*, должен быть не менее *nNumberOfCharsToRead \* sizeof(TCHAR)* байтов.

*lpNumberOfCharsRead* [out]

Указатель на переменную, которая получает количество фактически прочитанных символов.

*pInputControl* [in, необязательный параметр]

Указатель на [структуре CONSOLE\\_READCONSOLE\\_CONTROL](#), указывающую управляющий символ, сигналив об окончании операции чтения. Этот параметр может принимать значение **NULL**.

Для этого параметра по умолчанию требуются входные данные в Юникоде. Для режима ANSI задайте для этого параметра значение **NULL**.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

`ReadConsole` считывает ввод с клавиатуры из входного буфера консоли. Она ведет себя как функция [ReadFile](#), за исключением того, что она может считывать данные в юникоде (расширенные символы) или в режиме ANSI. Чтобы приложения, поддерживающие единый набор источников, совместимые с обоими режимами, используйте `ReadConsole`, а не `ReadFile`. Хотя `ReadConsole` можно использовать только с дескриптором входного буфера консоли, `ReadFile` можно использовать с другими дескрипторами (например, с файлами или каналами). `ReadConsole` завершается сбоем, если используется со стандартным дескриптором, перенаправленным в нечто, отличное от дескриптора консоли.

Все режимы ввода, влияющие на поведение [ReadFile](#), одинаково влияют на `ReadConsole`. Чтобы получить и задать режимы ввода входного буфера консоли, используйте функции [GetConsoleMode](#) и [SetConsoleMode](#).

Если входной буфер содержит события ввода, отличные от событий клавиатуры (например, события мыши или изменения размера окна), они удаляются. Эти события можно считывать только с помощью функции [ReadConsoleInput](#).

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

Параметр *pInputControl* можно использовать для включения промежуточных пробуждений из операции чтения в ответ на управляющий символ завершения файла, указанный в [CONSOLE\\_READCONSOLE\\_CONTROL](#) структуре. Эта функция

требует включения расширений команд, стандартного дескриптора вывода — дескриптора вывода консоли, а входных данных — Юникода.

**Windows Server 2003 и Windows XP/2000:** Промежуточная функция чтения не поддерживается.

**Режим приготовления** — это когда **ENABLE\_LINE\_INPUT** задано с помощью **SetConsoleMode** на дескрипторе ввода консоли. В готовом режиме узел консоли предоставит строку редактирования от имени приложения командной строки, и вызовы **ReadFile** или **ReadConsole** не будут возвращаться до нажатия клавиши ВВОД.

**Промежуточное чтение** — это дополнение к поведению при вызове **ReadConsole** в режиме чтения. Установка флага в **dwCtrlWakeupMask** в структуре **CONSOLE\_READCONSOLE\_CONTROL** и его передача в *pinputControl* при вызове **ReadConsole** приведет к тому, что при чтении не обязательно будет ждаться новая строка, но возвращается и указанный символ.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll
Имя в кодировке Юникод и ANSI	<b>ReadConsoleW</b> (Юникод) и <b>ReadConsoleA</b> (ANSI)

## См. также

[Функции консоли](#)

[CONSOLE\\_READCONSOLE\\_CONTROL](#)

[GetConsoleMode](#)

[Методы ввода и вывода](#)

[ReadConsoleInput](#)

[ReadFile](#)

[SetConsoleCP](#)

[SetConsoleMode](#)

[SetConsoleOutputCP](#)

[WriteConsole](#)

# Функция ReadConsoleInput

Статья • 23.10.2023

Считывает данные из входного буфера консоли и удаляет его из буфера.

## Синтаксис

C

```
BOOL WINAPI ReadConsoleInput(
    _In_     HANDLE      hConsoleInput,
    _Out_    PINPUT_RECORD lpBuffer,
    _In_     DWORD       nLength,
    _Out_    LPDWORD     lpNumberOfEventsRead
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [out]

Указатель на массив INPUT\_RECORD структур, получающих входные данные буфера.

*nLength* [in]

Размер массива, на который указывает параметр *lpBuffer*, в элементах массива.

*lpNumberOfEventsRead* [out]

Указатель на переменную, которая получает количество операций чтения входных записей.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Если число записей, запрошенных в параметре `nLength`, превышает количество записей, доступных в буфере, число доступно для чтения. Функция не возвращается, пока не будет прочитана по крайней мере одна входная запись.

Процесс может указать дескриптор входного буфера консоли в одной из [функций ожидания](#), чтобы определить, когда есть непрочитанные входные данные консоли. Если входной буфер не пуст, состояние дескриптора входного буфера консоли сигнализируется.

Чтобы определить количество непрочитанных входных записей в входном буфере консоли, используйте функцию [GetNumberOfConsoleInputEvents](#). Чтобы считывать входные записи из буфера ввода консоли, не влияя на количество непрочитанных записей, используйте [функцию PeekConsoleInput](#). Чтобы отменить карта все непрочитанные записи во входном буфере консоли, используйте [функцию FlushConsoleInputBuffer](#).

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib

DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>ReadConsoleInputW</code> (Юникод) и <code>ReadConsoleInputA</code> (ANSI)

## См. также

[Функции консоли](#)

[FlushConsoleInputBuffer](#)

[GetNumberOfConsoleInputEvents](#)

[INPUT\\_RECORD](#)

[Низкоуровневые функции ввода консоли](#)

[PeekConsoleInput](#)

[ReadConsole](#)

[ReadConsoleInputEx](#)

[ReadFile](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[WriteConsoleInput](#)

# Функция ReadConsoleInputEx

Статья • 23.10.2023

Считывает данные из входного буфера консоли и удаляет его из буфера с настраиваемым поведением.

Обратите внимание, что эта функция не связана с библиотекой импорта. Эта функция доступна в виде ресурсов с именем `ReadConsoleInputExA` и `ReadConsoleInputExW` в `Kernel32.dll`. Для динамической связи с `Kernel32.dll` необходимо использовать функции `LoadLibrary` и `GetProcAddress`.

## Синтаксис

C

```
BOOL WINAPI ReadConsoleInputEx(
    _In_     HANDLE      hConsoleInput,
    _Out_    PINPUT_RECORD lpBuffer,
    _In_     DWORD       nLength,
    _Out_    LPDWORD     lpNumberOfEventsRead,
    _In_     USHORT      wFlags
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [out]

Указатель на массив `INPUT_RECORD` структур, получающих входные данные буфера.

*nLength* [in]

Размер массива, на который указывает параметр *lpBuffer*, в элементах массива.

*lpNumberOfEventsRead* [out]

Указатель на переменную, которая получает количество операций чтения входных записей.

*wFlags* [in]

Набор флагов (ORed вместе), указывающий поведение чтения консоли.

Значение	Значение
CONSOLE_READ_NOREMOVE 0x0001	Оставьте события в входном буфере (как и в <a href="#">PeekConsoleInput</a> )
CONSOLE_READ_NOWAIT 0x0002	Возвращается немедленно, даже если в входном буфере отсутствуют события.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Эта функция является настраиваемой версией `ReadConsoleInput`. Дополнительные сведения о работе см. в примечаниях [ReadConsoleInput](#).

Вызов `ReadConsoleInputEx` с флагами `CONSOLE_READ_NOMOVE` | `CONSOLE_READ_NOWAIT` эквивалентен вызову [PeekConsoleInput](#).

Эта функция не существует в заголовках консоли Windows. Чтобы получить доступ к нему из приложения С или С++, включите следующие объявления и динамически свяжите файл kernel32.dll, как описано выше.

```
C

#ifndef CONSOLE_READ_NOREMOVE
#define CONSOLE_READ_NOREMOVE    0x0001
#endif

#ifndef CONSOLE_READ_NOWAIT
#define CONSOLE_READ_NOWAIT     0x0002
#endif

BOOL
WINAPI
ReadConsoleInputExA(
    _In_ HANDLE hConsoleInput,
    _Out_writes_(nLength) PINPUT_RECORD lpBuffer,
    _In_ DWORD nLength,
    _Out_ LPDWORD lpNumberOfEventsRead,
    _In_ USHORT wFlags);
```

```
BOOL  
WINAPI  
ReadConsoleInputExW(  
    _In_ HANDLE hConsoleInput,  
    _Out_writes_(nLength) PINPUT_RECORD lpBuffer,  
    _In_ DWORD nLength,  
    _Out_ LPDWORD lpNumberOfEventsRead,  
    _In_ USHORT wFlags);
```

## Требования

Минимальная версия клиента	Windows 7 [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Заголовок	нет, см. примечания
Библиотека	нет, см. примечания
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	ReadConsoleInputExW (Юникод) и ReadConsoleInputExA (ANSI)

## См. также

[Функции консоли](#)

[FlushConsoleInputBuffer](#)

[GetNumberOfConsoleInputEvents](#)

[INPUT\\_RECORD](#)

[Низкоуровневые функции ввода консоли](#)

[ReadConsoleInput](#)

[PeekConsoleInput](#)

[ReadConsole](#)

[ReadFile](#)

[\*\*SetConsoleCP\*\*](#)

[\*\*SetConsoleOutputCP\*\*](#)

[\*\*WriteConsoleInput\*\*](#)

# Функция ReadConsoleOutput

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Считывает данные символов и атрибутов цвета из прямоугольного блока ячеек символов в буфере экрана консоли, а функция записывает данные в прямоугольный блок в указанном расположении в целевом буфере.

## Синтаксис

C

```
BOOL WINAPI ReadConsoleOutput(
    _In_      HANDLE      hConsoleOutput,
    _Out_     PCHAR_INFO  lpBuffer,
    _In_      COORD       dwBufferSize,
    _In_      COORD       dwBufferCoord,
    _Inout_   PSMALL_RECT lpReadRegion
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [out]

Указатель на целевой буфер, который получает данные, считываемые из буфера

экрана консоли. Этот указатель рассматривается как источник двухмерного массива CHAR\_INFO структур, размер которого задается *параметром dwBufferSize*.

#### *dwBufferSize [in]*

Размер *параметра lpBuffer* в ячейках символов. Элемент **X структуры COORD** — это количество столбцов; элемент **Y** — это число строк.

#### *dwBufferCoord [in]*

Координаты левой верхней ячейки в параметре *lpBuffer*, получающем данные из буфера экрана консоли. Элемент **X структуры COORD** — это столбец, а элемент **Y** — строка.

#### *lpReadRegion [in, out]*

Указатель на структуру **SMALL\_RECT**. Во входных данных члены структуры указывают координаты прямоугольника буфера экрана консоли в левом верхнем и нижнем углу, из которого требуется считывать функцию. В выходных данных члены структуры указывают фактический прямоугольник, который использовался.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

`ReadConsoleOutput` обрабатывает буфер экрана консоли и целевой буфер как двухмерные массивы (столбцы и строки символьных ячеек). Прямоугольник, на который указывает *параметр lpReadRegion*, указывает размер и расположение блока для чтения из буфера экрана консоли. Прямоугольник назначения с тем же размером расположен с левой верхней ячейкой в координатах *параметра dwBufferCoord* в массиве *lpBuffer*. Данные, считываемые из ячеек в исходном прямоугольнике буфера экрана консоли, копируются в соответствующие ячейки в целевом буфере. Если соответствующая ячейка находится за пределами прямоугольника буфера назначения (измерения которых указаны *параметром dwBufferSize*), данные не копируются.

Ячейки в целевом буфере, соответствующие координатам, которые не находятся в границах буфера экрана консоли, остаются неизменными. Другими словами, это ячейки, для которых данные буфера экрана недоступны для чтения.

Перед возвратом `ReadConsoleOutput` он задает элементы структуры, на которые указывает параметр `lpReadRegion`, на фактический прямоугольник буфера экрана, ячейки которого были скопированы в целевой буфер. Этот прямоугольник отражает ячейки в исходном прямоугольнике, для которого существовала соответствующая ячейка в целевом буфере, так как `ReadConsoleOutput` обрезает размеры исходного прямоугольника, чтобы соответствовать границам буфера экрана консоли.

Если прямоугольник, указанный `lpReadRegion`, находится полностью за пределами буфера экрана консоли или если соответствующий прямоугольник находится полностью за пределами целевого буфера, данные не копируются. В этом случае функция возвращает элементы структуры, на которые указывает набор `lpReadRegion`, таким образом, что правый элемент меньше левого, или нижний элемент меньше верхнего. Чтобы определить размер буфера экрана консоли, используйте [функцию `GetConsoleScreenBufferInfo`](#).

Функция `ReadConsoleOutput` не влияет на положение курсора буфера экрана консоли. Содержимое буфера экрана консоли не изменяется функцией.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [`SetConsoleCP`](#) или [`SetConsoleOutputCP`](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, как ожидается, запоминает собственное состояние рисования для дальнейшего управления. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Примеры

Пример см. в разделе "[Чтение и запись блоков символов и атрибутов](#)".

# Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>ReadConsoleOutputW</b> (Юникод) и <b>ReadConsoleOutputA</b> (ANSI)

## См. также

[Функции консоли](#)

[Функции вывода консоли низкого уровня](#)

[ReadConsoleOutputAttribute](#)

[ReadConsoleOutputCharacter](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[SMALL\\_RECT](#)

[WriteConsoleOutput](#)

[CHAR\\_INFO](#)

[COORD](#)

# Функция ReadConsoleOutputAttribute

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Копирует указанное количество атрибутов символов из последовательных ячеек буфера экрана консоли, начиная с указанного расположения.

## Синтаксис

C

```
BOOL WINAPI ReadConsoleOutputAttribute(
    _In_    HANDLE   hConsoleOutput,
    _Out_   LPWORD   lpAttribute,
    _In_    DWORD    nLength,
    _In_    COORD    dwReadCoord,
    _Out_   LPDWORD  lpNumberOfAttrsRead
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpAttribute* [out]

Указатель на буфер, который получает атрибуты, используемые буфером экрана консоли.

Дополнительные сведения см. в разделе "Атрибуты символов".

#### *nLength* [in]

Количество ячеек буфера экрана, из которых следует читать. Размер буфера, на который указывает параметр *lpAttribute*, должен быть `nLength * sizeof(WORD)`.

#### *dwReadCoord* [in]

Координаты первой ячейки в буфере экрана консоли, из которой следует читать символы. Элемент X [структуре COORD](#) — это столбец, а элемент Y — строка.

#### *lpNumberOfAttrsRead* [out]

Указатель на переменную, которая получает количество фактически считываемых атрибутов.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Если число атрибутов, считываемых из строки буфера экрана, выходит за пределы указанной строки буфера экрана, атрибутычитываются из следующей строки.

Если число атрибутов, считываемых из буфера экрана консоли, выходит за пределы буфера экрана консоли, считаются атрибуты до конца буфера экрана консоли.

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, как ожидается, запоминает собственное состояние рисования для дальнейшего управления. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[COORD](#)

[Функции вывода консоли низкого уровня](#)

[ReadConsoleOutput](#)

[ReadConsoleOutputCharacter](#)

[WriteConsoleOutput](#)

[WriteConsoleOutputAttribute](#)

[WriteConsoleOutputCharacter](#)

# Функция ReadConsoleOutputCharacter

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Копирует ряд символов из последовательных ячеек буфера экрана консоли, начиная с указанного расположения.

## Синтаксис

C

```
BOOL WINAPI ReadConsoleOutputCharacter(
    _In_    HANDLE   hConsoleOutput,
    _Out_   LPTSTR   lpCharacter,
    _In_    DWORD    nLength,
    _In_    COORD    dwReadCoord,
    _Out_   LPDWORD  lpNumberOfCharsRead
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpCharacter* [out]

Указатель на буфер, который получает символы, считываемые из буфера экрана консоли.

*nLength* [in]

Количество ячеек буфера экрана, из которых следует читать. Размер буфера, на который указывает параметр *lpCharacter*, должен быть `nLength * sizeof(TCHAR)`.

*dwReadCoord* [in]

Координаты первой ячейки в буфере экрана консоли, из которой следует читать символы. Элемент X [структуре COORD](#) — это столбец, а элемент Y — строка.

*lpNumberOfCharsRead* [out]

Указатель на переменную, получающую количество символов, фактически считываемых.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Если число символов, считываемых из строки буфера экрана, выходит за пределы указанной строки буфера экрана, символычитываются из следующей строки. Если число символов, считываемых из буфера экрана консоли, выходит за пределы буфера экрана консоли, считаются символы до конца буфера экрана консоли.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#).

Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, как ожидается, запоминает собственное состояние рисования для дальнейшего управления. Удаленное взаимодействие приложений с помощью

межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	ReadConsoleOutputCharacterW (Юникод) и ReadConsoleOutputCharacterA (ANSI)

## См. также

[Функции консоли](#)

[COORD](#)

[Функции вывода консоли низкого уровня](#)

[ReadConsoleOutput](#)

[ReadConsoleOutputAttribute](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[WriteConsoleOutput](#)

[WriteConsoleOutputAttribute](#)

[WriteConsoleOutputCharacter](#)

# Функция ResizePseudoConsole

Статья • 13.12.2023

Изменяет размер внутренних буферов для псевдоконсоля до заданного размера.

## Синтаксис

```
C

HRESULT WINAPI ResizePseudoConsole(
    _In_ HPCON hPC ,
    _In_ COORD size
);
```

## Параметры

*hPC* [in]

Дескриптор активного псевдоконсоля, открытый [CreatePseudoConsole](#).

*размер* [in]

Размеры окна или буфера в количестве символов, которые будут использоваться для внутреннего буфера этого псевдоконсоля.

## Возвращаемое значение

Тип: **HRESULT**

Если этот метод выполнен успешно, он возвращает **S\_OK**. В противном случае возвращается код ошибки **HRESULT**.

## Замечания

Эта функция может изменить размер внутренних буферов в сеансе псевдоконсоля, чтобы он соответствовал размеру окна или буфера, используемому для отображения в конце терминала. Это гарантирует, что присоединенные приложения интерфейса командной строки (CUI) с помощью [консольных функций](#) для обмена данными будут иметь правильные измерения, возвращаемые в их вызовах.

# Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	обновление Windows 10 за октябрь 2018 г. (версия 1809) [только классические приложения]
Минимальная версия сервера	Windows Server 2019 [только классические приложения]
Верхний колонтидул	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Псевдоконсоли](#)

[CreatePseudoConsole](#)

[ClosePseudoConsole](#)

# Функция ScrollConsoleScreenBuffer

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Перемещает блок данных в буфере экрана. Эффекты перемещения могут быть ограничены путем указания вырезки прямоугольника, поэтому содержимое буфера экрана консоли за пределами прямоугольника вырезки не изменяется.

## Синтаксис

C

```
BOOL WINAPI ScrollConsoleScreenBuffer(
    _In_          HANDLE      hConsoleOutput,
    _In_          const SMALL_RECT *lpScrollRectangle,
    _In_opt_       const SMALL_RECT *lpClipRectangle,
    _In_          COORD       dwDestinationOrigin,
    _In_          const CHAR_INFO *lpFill
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpScrollRectangle* [in]

Указатель на [структуре SMALL\\_RECT](#), члены которой указывают координаты

прямоугольника буфера экрана консоли в верхнем левом и правом нижнем углу.

*lpClipRectangle* [in, необязательный]

Указатель на [структуре SMALL\\_RECT](#), члены которой указывают координаты левого верхнего и нижнего левого угла прямоугольника буфера экрана консоли, на который влияет прокрутка. Этот указатель может иметь значение **NULL**.

*dwDestinationOrigin* [in]

[Структура COORD](#), указывающая левый верхний угол нового расположения содержимого *lpScrollRectangle* в символах.

*lpFill* [in]

Указатель на [структуре CHAR\\_INFO](#), указывающую атрибуты символа и цвета, которые будут использоваться в заполнении ячеек в пересечении *lpScrollRectangle* и *lpClipRectangle*, которые были оставлены пустыми в результате перемещения.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

`ScrollConsoleScreenBuffer` копирует содержимое прямоугольной области буфера экрана, указанного *параметром lpScrollRectangle*, в другую область буфера экрана консоли. Целевой прямоугольник имеет те же измерения, что и *прямоугольник lpScrollRectangle* с верхним левым углом в координатах, указанных параметром *dwDestinationOrigin*. Эти части *lpScrollRectangle*, которые не перекрываются с целевым прямоугольником, заполняются атрибутами символа и цвета, заданными параметром *lpFill*.

Прямоугольник обрезки применяется к изменениям, внесенным как в прямоугольник *lpScrollRectangle*, так и к целевому прямоугольнику. Например, если прямоугольник вырезки не включает область, заполненную содержимым *lpFill*, *исходное содержимое* региона остается неизменным.

Если области прокрутки или целевые области выходят за рамки буфера экрана консоли, они обрезаются. Например, если *lpScrollRectangle* — это регион, содержащийся (0,0) и (19,19) и *dwDestinationOrigin* (10,15), целевой прямоугольник — это регион, содержащийся (10 15) и (29 34). Тем не менее, если буфер экрана

консоли имеет 50 символов ширины и 30 символов высокого уровня, целевой прямоугольник обрезается (10 15) и (29 29 29). Изменения буфера экрана консоли также обрезаются в соответствии с *lpClipRectangle*, если параметр задает **SMALL\_RECT** структуру. Если прямоугольник вырезки указан как (0,0) и (49 19), вносятся только изменения, происходящие в этом регионе буфера экрана консоли.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции **SetConsoleCP** или **SetConsoleOutputCP**. Пользователи прежних версий могут также использовать команды **chcp** или **mode con cp select=** (но это не рекомендуется для новой разработки).

#### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. С помощью полей прокрутки можно определить **область экрана, положение курсора**, чтобы задать активную позицию за пределами региона, а также новые строки для принудительного перемещения текста. Оставшееся пространство можно заполнить, переместив курсор, **задав графические атрибуты** и написав обычный текст.

## Примеры

Пример см. в разделе "[Прокрутка содержимого буфера экрана](#)".

## Requirements

[\[+\] Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib

DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>ScrollConsoleScreenBufferW</code> (Юникод) и <code>ScrollConsoleScreenBufferA</code> (ANSI)

## См. также

[CHAR\\_INFO](#)

[Функции консоли](#)

[COORD](#)

[Прокрутка буфера экрана](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[SetConsoleWindowInfo](#)

[SMALL\\_RECT](#)

# Функция SetConsoleActiveScreenBuffer

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает указанный буфер экрана для текущего отображаемого буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleActiveScreenBuffer(
    _In_ HANDLE hConsoleOutput
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Консоль может иметь несколько буферов экрана. `SetConsoleActiveScreenBuffer` определяет, какой из них отображается. Вы можете записать в неактивный буфер экрана, а затем использовать `SetConsoleActiveScreenBuffer` для отображения содержимого буфера.

## 💡 Совет

Этот API не рекомендуется, но он имеет приблизительный **виртуальный терминал** эквивалент в альтернативной **последовательности буфера** экрана. *Настройка альтернативного буфера* экрана может предоставить приложению отдельное изолированное пространство для рисования в течение среды выполнения сеанса при сохранении содержимого, отображаемого вызывающим объектом приложения. Это сохраняет, что рисование сведений для простого восстановления при выходе процесса.

# Примеры

Пример см. в разделе "[Чтение и запись блоков символов и атрибутов](#)".

# Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

# См. также

ФУНКЦИИ КОНСОЛИ

Буферы экрана консоли

[CreateConsoleScreenBuffer](#)

# Функция SetConsoleCP

Статья • 13.12.2023

Задает входную кодовую страницу, используемую консолью, связанной с вызывающим процессом. Консоль использует свою кодовую страницу ввода для перевода ввода клавиатуры в соответствующее значение символа.

## Синтаксис

C

```
BOOL WINAPI SetConsoleCP(  
    _In_ UINT wCodePageID  
) ;
```

## Параметры

*wCodePageID* [in]

Идентификатор заданной кодовой страницы. Дополнительные сведения см. в подразделе "Примечания".

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Кодовая страница сопоставляет 256 кодов символов с отдельными символами.

Разные кодовые страницы включают разные специальные символы, как правило, настроенные для языка или группы языков.

Чтобы найти кодовые страницы, установленные или поддерживаемые операционной системой, используйте [функцию EnumSystemCodePages](#).

Идентификаторы кодовых страниц, доступных на локальном компьютере, также хранятся в реестре в следующем разделе:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage`

Однако лучше использовать [EnumSystemCodePages](#) для перечисления кодовых страниц, так как реестр может отличаться в разных версиях Windows.

Чтобы определить, является ли определенная кодовая страница допустимой, используйте [функцию IsValidCodePage](#). Чтобы получить дополнительные сведения о кодовой странице, включая его имя, используйте [функцию GetCPIInfoEx](#). Список доступных идентификаторов кодовой страницы см. в разделе "[Идентификаторы кодовой страницы](#)".

Чтобы определить текущую кодовую страницу консоли, используйте [функцию GetConsoleCP](#). Чтобы задать и получить выходную кодовую страницу консоли, используйте функции SetConsoleOutputCP и [GetConsoleOutputCP](#).

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Кодовые страницы консоли](#)

[Функции консоли](#)

[GetConsoleCP](#)

[GetConsoleOutputCP](#)

[SetConsoleOutputCP](#)

# Функция SetConsoleCtrlHandler

Статья • 12.07.2023

Добавляет определяемую приложением функцию [HandlerRoutine](#) в список функций обработчика для вызывающего процесса или удаляет ее из такого списка.

Если функция обработчика не указана, функция задает наследуемый атрибут, который определяет, игнорирует ли вызывающий процесс сигналы при нажатии клавиш **CTRL**+**C**.

## Синтаксис

```
С

BOOL WINAPI SetConsoleCtrlHandler(
    _In_opt_ PHANDLER_ROUTINE HandlerRoutine,
    _In_      BOOL           Add
);
```

## Параметры

*HandlerRoutine* [ввод, необязательный]

Указатель на определяемую приложением функцию [HandlerRoutine](#), которая будет добавлена или удалена. Этот параметр может принимать значение **NULL**.

*Add* [ввод]

Если этот параметр имеет значение **TRUE**, обработчик добавляется; если **FALSE** — обработчик удаляется.

Если параметр *HandlerRoutine* имеет значение **NULL**, значение **TRUE** приводит к тому, что вызывающий процесс игнорирует нажатие клавиш **CTRL**+**C**, а значение **FALSE** восстанавливает нормальную обработку нажатия **CTRL**+**C**. Этот атрибут для игнорирования обработки **CTRL**+**C** также наследуется дочерними процессами.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Комментарии

Эта функция предоставляет аналогичное уведомление для консольного приложения и служб, которое доступно в [WM\\_QUERYENDSESSION](#) для графических приложений с конвейером сообщений. Вы также можете использовать эту функцию из графического приложения, но при этом не гарантируется ее доставка перед получением уведомления от [WM\\_QUERYENDSESSION](#).

Каждый процесс консоли имеет собственный список определяемых приложением функций [HandlerRoutine](#), которые обрабатывают сигналы при нажатии клавиш `CTRL + C` и `CTRL + BREAK`. Функции обработчика также обрабатывают сигналы, генерируемые системой, когда пользователь закрывает консоль, выходит из системы или завершает ее работу. Изначально список обработчиков для каждого процесса содержит только стандартную функцию обработчика, которая вызывает функцию [ExitProcess](#). Процесс консоли добавляет или удаляет дополнительные функции обработчика, вызывая функцию [SetConsoleCtrlHandler](#), которая не влияет на список функций обработчика для других процессов. Если процесс консоли получает любые управляющие сигналы, его функции обработчика вызываются по принципу "зарегистрирован последним — вызван первым", пока один из обработчиков не вернет значение `TRUE`. Если ни один из обработчиков не возвращает значение `TRUE`, вызывается обработчик по умолчанию.

Вызов [AttachConsole](#), [AllocConsole](#) или [FreeConsole](#) приведет к сбросу таблицы обработчиков элементов управления в клиентском процессе до исходного состояния. Обработчики необходимо повторно зарегистрировать при изменении сеанса присоединенной консоли.

Для процессов консоли ввод `CTRL + C` и `CTRL + BREAK` обычно обрабатывается как сигналы ([CTRL\\_C\\_EVENT](#) и [CTRL\\_BREAK\\_EVENT](#)). Если окно консоли с фокусом для ввода текста с клавиатуры получает нажатие клавиш `CTRL + C` или `CTRL + BREAK`, сигнал обычно передается всем процессам, использующим эту консоль.

Нажатие клавиш `CTRL + BREAK` всегда обрабатывается как сигнал, но типичное поведение `CTRL + C` можно изменить тремя способами, чтобы предотвратить вызов функций обработчика:

- Функция [SetConsoleMode](#) может отключить режим [ENABLE\\_PROCESSED\\_INPUT](#) для входного буфера консоли, поэтому `CTRL + C` регистрируется как ввод с клавиатуры, а не как сигнал.
- Вызов функции [SetConsoleCtrlHandler](#) с аргументами `NULL` и `TRUE` приводит к тому, что вызывающий процесс игнорирует сигналы нажатия `CTRL + C`. Этот

атрибут наследуется дочерними процессами, но его можно включить или отключить в любом процессе, не затрагивая имеющиеся процессы.

- Если выполняется отладка процесса консоли и сигналы ввода **CTRL**+**C** не отключены, система выдаст исключение **DBG\_CONTROL\_C**. Это исключение отображается только для того, чтобы упростить отладку — приложению не нужно использовать обработчик исключений для его обработки. Если отладчик обрабатывает исключение, приложение проигнорирует нажатие клавиш **CTRL**+**C** (с одним исключением: ожидания с возможностью оповещения будут завершены). Если отладчик передает исключение необработанным, нажатие клавиш **CTRL**+**C** передается в процесс консоли и обрабатывается как сигнал, как было описано ранее.

Процесс консоли может использовать функцию [GenerateConsoleCtrlEvent](#) для отправки сигнала нажатия **CTRL**+**C** или **CTRL**+**BREAK** в группу процессов консоли.

Система выводит сигналы **CTRL\_CLOSE\_EVENT**, **CTRL\_LOGOFF\_EVENT** и **CTRL\_SHUTDOWN\_EVENT**, если пользователь закрывает консоль, выходит из системы или завершает работу системы таким образом, чтобы процесс мог предварительно очистить данные в памяти. Функции консоли или любые функции среды выполнения С, которые вызывают функции консоли, могут работать нестабильно в процессе обработки любого из упомянутых выше трех сигналов. Причина заключается в том, что некоторые из внутренних подпрограмм очистки консоли могут вызываться перед выполнением обработчика сигналов процесса.

В Windows 7, Windows 8, Windows 8.1 и Windows 10:

Если консольное приложение загружает библиотеку gdi32.dll или user32.dll, функция [HandlerRoutine](#), указанная вами при вызове [SetConsoleCtrlHandler](#), не вызывается для событий **CTRL\_LOGOFF\_EVENT** и **CTRL\_SHUTDOWN\_EVENT**.

Операционная система обрабатывает процессы, которые загружают библиотеку gdi32.dll или user32.dll, как приложения Windows, а не как консольные приложения. Такое поведение также характерно для консольных приложений, которые не вызывают функции непосредственно в gdi32.dll или user32.dll, но вызывают, например, [функции оболочки](#), которые в свою очередь вызывают функции в gdi32.dll или user32.dll.

Чтобы получать события, когда пользователь выходит из системы или устройство завершает работу в таких условиях, создайте скрытое окно в консольном приложении, а затем обеспечьте обработку сообщений окна **WM\_QUERYENDSESSION** и **WM\_ENDSESSION**, которые получает скрытое окно. Вы можете создать скрытое окно, вызвав метод [CreateWindowEx](#) с параметром

*dwExStyle*, для которого задано значение 0. См. пример базового дескриптора ниже.

## Примеры

Пример см. в статье [Регистрация функции обработчика команд управления](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll
Имя в кодировке Юникод и ANSI	

## См. также

[Обработчики команд управления в консоли](#)

[Функции консоли](#)

[ExitProcess](#)

[GenerateConsoleCtrlEvent](#)

[GetConsoleMode](#)

[HandlerRoutine](#)

[SetConsoleMode](#)

# Функция SetConsoleCursorInfo

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает размер и видимость курсора для указанного буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleCursorInfo(
    _In_          HANDLE      hConsoleOutput,
    _In_ const CONSOLE_CURSOR_INFO *lpConsoleCursorInfo
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpConsoleCursorInfo* [in]

Указатель на **CONSOLE\_CURSOR\_INFO** структуру, которая предоставляет новые спецификации для курсора буфера экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Когда курсор буфера экрана отображается, его внешний вид может отличаться, начиная от полного заполнения ячейки символом до горизонтальной линии в нижней части ячейки. Элемент `dwSize` структуры [CONSOLE\\_CURSOR\\_INFO](#) указывает процент символьной ячейки, заполненной курсором. Если этот элемент меньше 1 или больше 100, `SetConsoleCursorInfo` завершается ошибкой.

### 💡 Совет

Этот API имеет виртуальный терминал, эквивалентный разделу [видимости](#) курсора, и `^[[?25h ^[[?25l` последовательности.

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтидул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

**CONSOLE\_CURSOR\_INFO**

[GetConsoleCursorInfo](#)

[SetConsoleCursorPosition](#)

# Функция SetConsoleCursorPosition

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает позицию курсора в указанном буфере экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleCursorPosition(
    _In_ HANDLE hConsoleOutput,
    _In_ COORD dwCursorPosition
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*dwCursorPosition* [in]

[Структура COORD](#), указывающая новую позицию курсора в символах. Координаты — это столбец и строка ячейки буфера экрана. Координаты должны находиться в границах буфера экрана консоли.

# Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Позиция курсора определяет, где отображаются символы, написанные функцией `WriteFile` или `WriteConsole`, или отложенные функцией `ReadFile` или `ReadConsole`.

Чтобы определить текущую позицию курсора, используйте функцию [GetConsoleScreenBufferInfo](#).

Если новая позиция курсора не находится в границах окна буфера экрана консоли, источник окна изменяется, чтобы сделать курсор видимым.

### 💡 Совет

Этот API имеет эквивалент виртуального [терминала в простых разделах размещения](#) курсора и [размещения](#) курсоров. Использование новой линии, возврата каретки, заднего пространства и последовательностей элементов управления табуляции также может помочь в расположении курсора.

## Примеры

Пример см. в разделе "[Использование высокоуровневых входных и выходных функций](#)".

## Requirements

↔ [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]

Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

[GetConsoleCursorInfo](#)

[GetConsoleScreenBufferInfo](#)

[ReadConsole](#)

[ReadFile](#)

[SetConsoleCursorInfo](#)

[WriteConsole](#)

[WriteFile](#)

# Функция SetConsoleDisplayMode

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает режим отображения указанного буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleDisplayMode(
    _In_      HANDLE hConsoleOutput,
    _In_      DWORD  dwFlags,
    _Out_opt_ PCOORD lpNewScreenBufferDimensions
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли.

*dwFlags* [in]

Режим отображения консоли. Этот параметр может быть одним или несколькими из следующих значений.

 Развернуть таблицу

Значение	Значение
CONSOLE_FULLSCREEN_MODE 1	Текст отображается в полноэкранном режиме.
CONSOLE_WINDOWED_MODE 2	Текст отображается в окне консоли.

*lpNewScreenBufferDimensions* [out, необязательный]

Указатель на [структуру COORD](#), которая получает новые измерения буфера экрана в символах.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib

DLL-библиотеки

Kernel32.dll

## См. также

[Функции консоли](#)

[Консольные режимы](#)

[GetConsoleDisplayMode](#)

# Функция SetConsoleHistoryInfo

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает параметры журнала для консоли вызывающего процесса.

## Синтаксис

C

```
BOOL WINAPI SetConsoleHistoryInfo(
    _In_ PCONSOLE_HISTORY_INFO lpConsoleHistoryInfo
);
```

## Параметры

*lpConsoleHistoryInfo* [in]

Указатель на [структуру CONSOLE\\_HISTORY\\_INFO](#), содержащую параметры журнала для консоли процесса.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

Если вызывающий процесс не является консольным процессом, функция завершается ошибкой и задает для ERROR\_ACCESS\_DENIED последний код ошибки.

## 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, выступающее в качестве оболочки или интерпретатора, должно поддерживать собственные функции удобства пользователя, такие как поведение чтения строк и манипуляций, включая псевдонимы и журнал команд. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

# Requirements

[\[+\] Развернуть таблицу](#)

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[CONSOLE\\_HISTORY\\_INFO](#)

[GetConsoleHistoryInfo](#)

# Функция SetConsoleMode

Статья • 12.07.2023

Задает режим ввода для входного буфера консоли или режим вывода для буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleMode(
    _In_ HANDLE hConsoleHandle,
    _In_ DWORD   dwMode
);
```

## Параметры

*hConsoleHandle* [ввод]

Дескриптор входного буфера консоли или буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*dwMode* [ввод]

Устанавливаемый режим ввода или вывода.

Если в качестве входного дескриптора используется параметр *hConsoleHandle*, режим может быть одним из следующих. При создании консоли все режимы ввода, кроме **ENABLE\_WINDOW\_INPUT** и **ENABLE\_VIRTUAL\_TERMINAL\_INPUT**, включены по умолчанию.

Значение	Значение
<b>ENABLE_ECHO_INPUT</b> 0x0004	Символы, считанные функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , записываются в активный буфер экрана по мере ввода в консоли. Этот режим можно использовать только в том случае, если также включен режим <b>ENABLE_LINE_INPUT</b> .
<b>ENABLE_INSERT_MODE</b> 0x0020	Если этот режим включен, вводимый в окне консоли текст будет вставлен по текущему расположению курсора, а весь последующий текст не будет

Значение	Значение
	перезаписан. Если этот режим отключен, весь последующий текст будет перезаписан.
ENABLE_LINE_INPUT 0x0002	Функция <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> возвращает значение только в том случае, если считан символ возврата каретки. Если этот режим отключен, функции возвращают значение при доступности одного или нескольких символов.
ENABLE_MOUSE_INPUT 0x0010	Если указатель мыши находится в границах окна консоли и окно находится в фокусе для ввода текста с клавиатуры, события мыши, связанные с ее перемещением и нажатиями кнопок, помещаются во входной буфер. Такие события удаляются функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , даже если этот режим включен. Функцию <a href="#">ReadConsoleInput</a> можно использовать для считывания входных записей <a href="#">MOUSE_EVENT</a> из входного буфера.
ENABLE_PROCESSED_INPUT 0x0001	Нажатие клавиш CTRL+C обрабатывается системой и не помещается во входной буфер. Если входной буфер считывается функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> , нажатия других управляющих клавиш обрабатываются системой и не возвращаются в буфере <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> . Если также включен режим <a href="#">ENABLE_LINE_INPUT</a> , символы стирания назад, возврата каретки и перевода строки обрабатываются системой.
ENABLE_QUICK_EDIT_MODE 0x0040	Этот флаг позволяет пользователю использовать мышь для выбора и редактирования текста. Чтобы включить этот режим, используйте <code>ENABLE_QUICK_EDIT_MODE   ENABLE_EXTENDED_FLAGS</code> . Чтобы отключить этот режим, используйте <a href="#">ENABLE_EXTENDED_FLAGS</a> без этого флага.
ENABLE_WINDOW_INPUT 0x0008	Действия пользователей, которые приводят к изменению буфера экрана консоли, регистрируются во входном буфере консоли. Сведения о таких событиях могут быть считаны приложениями из входного буфера с помощью функции <a href="#">ReadConsoleInput</a> , но не могут быть считаны приложениями, использующими <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> .
ENABLE_VIRTUAL_TERMINAL_INPUT 0x0200	Этот флаг указывает модулю обработки виртуального терминала преобразовать ввод пользователя, полученный в окне консоли, в <a href="#">последовательности виртуального терминала консоли</a> , которые

Значение	Значение
	<p>вспомогательное приложение может получить с помощью функций <a href="#">WriteFile</a> или <a href="#">WriteConsole</a>.</p> <p>Обычно этот флаг рекомендуется использовать вместе с флагом <code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code> для выходного дескриптора, чтобы обеспечить подключение к приложению, которое обменивается данными исключительно через последовательности виртуального терминала.</p>

Если в качестве дескриптора буфера экрана используется параметр `hConsoleHandle`, режим может быть одним из следующих. При создании буфера экрана оба режима вывода включены по умолчанию.

Значение	Значение
<code>ENABLE_PROCESSED_OUTPUT</code> 0x0001	<p>Символы, записываемые функцией <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> либо выводимые функцией <a href="#">ReadFile</a> или <a href="#">ReadConsole</a>, анализируются для управляющих последовательностей ASCII, после чего выполняется нужное действие.</p> <p>Обрабатываются символы стирания назад, возврата каретки и перевода строки. Этот режим следует включить, если используются управляющие последовательности или задано значение <code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code>.</p>
<code>ENABLE_WRAP_AT_EOL_OUTPUT</code> 0x0002	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> либо выводе с помощью функции <a href="#">ReadFile</a> или <a href="#">ReadConsole</a> курсор перемещается в начало следующей строки, если он достиг конца текущей строки. Это приводит к тому, что строки, отображаемые в окне консоли, автоматически прокручиваются вверх, если курсор переходит далее с последней строки в окне. Кроме того, содержимое буфера экрана консоли также прокручивается вверх (с удалением верхней строки в буфере экрана консоли), если курсор переходит далее с последней строки в буфере экрана консоли.</p> <p>Если этот режим отключен, последний символ в строке будет перезаписан последующими символами.</p>
<code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code> 0x0004	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> символы обрабатываются для</p>

Значение	Значение
	<p>VT100 и аналогичных управляющих символьных последовательностей, которые управляют перемещением курсора, цветом и режимом шрифта, а также другими операциями, доступными для выполнения через существующие API-интерфейсы консоли. Дополнительные сведения см. в статье <a href="#">Последовательности виртуального терминала консоли</a>.</p> <p>При использовании этого флага убедитесь, что установлен флаг <code>ENABLE_PROCESSED_OUTPUT</code>.</p>
<code>DISABLE_NEWLINE_AUTO_RETURN</code> 0x0008	<p>При записи с помощью функции <a href="#">WriteFile</a> или <a href="#">WriteConsole</a> к переносу в конце строки добавляется дополнительное состояние, которое задерживает перемещение курсора и помещает операции прокрутки в буфер.</p> <p>Как правило, если флаг <code>ENABLE_WRAP_AT_EOL_OUTPUT</code> установлен и текст достигает конца строки, курсор немедленно переходит на следующую строку, а содержимое буфера прокручивается на одну строку. Но если задан этот флаг, курсор не переходит на следующую строку, а операция прокрутки не выполняется. Записанный символ будет выведен в последней позиции строки, а курсор будет располагаться над этим символом (как в случае с отключенным флагом <code>ENABLE_WRAP_AT_EOL_OUTPUT</code>). Но следующий печатаемый символ будет выведен таким образом, как если бы флаг <code>ENABLE_WRAP_AT_EOL_OUTPUT</code> был включен. Перезапись при этом не выполняется. В частности, курсор быстро переходит на следующую строку, при необходимости выполняется прокрутка, символ выводится, а курсор передвигается еще на одну позицию.</p> <p>Обычно этот флаг рекомендуется использовать вместе с флагом <code>ENABLE_VIRTUAL_TERMINAL_PROCESSING</code>, что позволяет оптимально сымитировать эмулятор терминала, в котором запись последнего символа на экране (в правом нижнем углу) без немедленной прокрутки является желаемым поведением.</p>

Значение	Значение
ENABLE_LVB_GRID_WORLDWIDE 0x0010	API-интерфейсы для записи атрибутов символов, в том числе <a href="#">WriteConsoleOutput</a> и <a href="#">WriteConsoleOutputAttribute</a> , позволяют использовать флаги <a href="#">атрибутов символов</a> для изменения цвета переднего плана и фона текста. Кроме того, некоторые флаги DBCS указываются с префиксом COMMON_LVB. Исторически эти флаги работали только в кодовых страницах DBCS для китайского, японского и корейского языков.
	За исключением флагов начальных и конечных байтов оставшиеся флаги, описывающие отрисовку строки и обратный видеовывод (смена местами цветов переднего плана и фона), можно использовать и с другими языками для выделения определенных частей выходных данных.
	Установка этого флага режима консоли позволяет использовать эти атрибуты для каждой кодовой страницы любого языка.
	По умолчанию он отключен для сохранения совместимости с известными приложениями, которые исторически игнорируют такие флаги на компьютерах без поддержки китайского, японского и корейского языков для хранения битов в таких полях (случайно или с собственными целями).
	Обратите внимание, что использование режима ENABLE_VIRTUAL_TERMINAL_PROCESSING может привести к установке флагов сетки LVB и обратного видеовывода, хотя этот флаг не включается, если подключенное приложение запрашивает видеовывод с подчеркиванием или инвертированием через <a href="#">последовательности виртуального терминала консоли</a> .

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

Консоль состоит из входного буфера и одного или нескольких буферов экрана.

Режим буфера консоли определяет функционирование консоли во время операций ввода-вывода. Один набор констант с флагом используется с дескрипторами ввода, а другой — с дескрипторами буфера экрана (вывод).

Задание режимов вывода одного буфера экрана не влияет на режимы вывода других буферов экрана.

Режимы `ENABLE_LINE_INPUT` и `ENABLE_ECHO_INPUT` влияют только на процессы, в которых используются [ReadFile](#) или [ReadConsole](#) для чтения данных из входного буфера консоли. Аналогичным образом режим `ENABLE_PROCESSED_INPUT` в первую очередь влияет на пользователей [ReadFile](#) и [ReadConsole](#), кроме того, что он также определяет, передается ли ввод данных с помощью `CTRL+C` во входной буфер (для чтения функцией [ReadConsoleInput](#)) или в функцию, определенную приложением.

Режимы `ENABLE_WINDOW_INPUT` и `ENABLE_MOUSE_INPUT` определяют, будут ли передаваться во входной буфер данные изменения размера окна и действий мыши. Эти события могут считываться с помощью [ReadConsoleInput](#), но они всегда фильтруются с помощью [ReadFile](#) и [ReadConsole](#).

Режимы `ENABLE_PROCESSED_OUTPUT` и `ENABLE_WRAP_AT_EOL_OUTPUT` влияют только на процессы с использованием [ReadFile](#) или [ReadConsole](#) и [WriteFile](#) или [WriteConsole](#).

Чтобы определить текущий режим входного буфера консоли или буфера экрана, используйте функцию [GetConsoleMode](#).

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll

## См. также

[Функции консоли](#)

[Консольные режимы](#)

[GetConsoleMode](#)

[HandlerRoutine](#)

[ReadConsole](#)

[ReadConsoleInput](#)

[ReadFile](#)

[WriteConsole](#)

[WriteFile](#)

# Функция SetConsoleOutputCP

Статья • 13.12.2023

Задает выходную кодовую страницу, используемую консолью, связанной с вызывающим процессом. Консоль использует свою выходную кодовую страницу для перевода значений символов, написанных различными выходными функциями в изображения, отображаемые в окне консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleOutputCP(
    _In_ UINT wCodePageID
);
```

## Параметры

*wCodePageID* [in]

Идентификатор заданной кодовой страницы. Дополнительные сведения см. в подразделе "Примечания".

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Кодовая страница сопоставляет 256 кодов символов с отдельными символами.

Разные кодовые страницы включают разные специальные символы, как правило, настроенные для языка или группы языков.

Если текущий шрифт является шрифтом Юникода фиксированного поля, `SetConsoleOutputCP` изменяет сопоставление значений символов в набор глифов шрифта, а не загружает отдельный шрифт при каждом вызове. Это влияет на то, как расширенные символы (значение ASCII больше 127) отображаются в окне консоли.

Однако если текущий шрифт является растровым шрифтом, **SetConsoleOutputCP** не влияет на отображение расширенных символов.

Чтобы найти кодовые страницы, установленные или поддерживаемые операционной системой, используйте [функцию \*\*EnumSystemCodePages\*\*](#).

Идентификаторы кодовых страниц, доступных на локальном компьютере, также хранятся в реестре в следующем разделе:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage`

Однако лучше использовать [EnumSystemCodePages](#) для перечисления кодовых страниц, так как реестр может отличаться в разных версиях Windows. Чтобы определить, является ли определенная кодовая страница допустимой, используйте [функцию \*\*IsValidCodePage\*\*](#). Чтобы получить дополнительные сведения о кодовой странице, включая его имя, используйте [функцию \*\*GetCPIInfoEx\*\*](#). Список доступных идентификаторов кодовой страницы см. в разделе "[Идентификаторы кодовой страницы](#)".

Чтобы определить текущую кодовую страницу консоли, используйте [функцию \*\*GetConsoleOutputCP\*\*](#). Чтобы задать и получить входную кодовую страницу консоли, используйте функции **SetConsoleCP** и [GetConsoleCP](#).

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Кодовые страницы консоли](#)

## ФУНКЦИИ КОНСОЛИ

[GetConsoleCP](#)

[GetConsoleOutputCP](#)

[SetConsoleCP](#)

# Функция SetConsoleScreenBufferInfoEx

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает расширенные сведения о указанном буфере экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleScreenBufferInfoEx(
    _In_     HANDLE                 hConsoleOutput,
    _In_     PCONSOLE_SCREEN_BUFFER_INFOEX lpConsoleScreenBufferInfoEx
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpConsoleScreenBufferInfoEx* [in]

Структура [CONSOLE\\_SCREEN\\_BUFFER\\_INFOEX](#), содержащая сведения о буфере экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

### 💡 Совет

Этот API имеет частичный эквивалент виртуального терминала . Буфер размещения курсора и текстовые атрибуты имеют определенные эквиваленты последовательности. Таблица цветов не настраивается, но расширенные цвета доступны за пределами обычно доступных через функции консоли. Атрибуты всплывающих окон не имеют эквивалента, так как всплывающие меню отвечают за клиентское приложение командной строки в мире виртуального терминала . Наконец, размер окна и состояние полноэкранного экрана считаются привилегиями, принадлежащими пользователю в мире виртуального терминала и не имеют эквивалентной последовательности.

## Requirements

[+] [Развернуть таблицу](#)

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[CONSOLE\\_SCREEN\\_BUFFER\\_INFOEX](#)

## GetConsoleScreenBufferInfoEx

# Функция SetConsoleScreenBufferSize

Статья • 12.07.2023

## ⓘ Важно!

В этом документе описаны функциональные возможности платформы консоли, которые больше не являются частью [нашей стратегии развития экосистемы](#). Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие варианты использования в неопределенном будущем. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию см. в нашем [документе о классической консоли и виртуальном терминале](#).

Изменяет размер указанного буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleScreenBufferSize(
    _In_ HANDLE hConsoleOutput,
    _In_ COORD dwSize
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*dwSize* [in]

Структура **COORD**, указывающая новый размер буфера экрана консоли в символьных строках и столбцах. Указанная ширина и высота не могут быть меньше ширины и высоты окна буфера экрана консоли. Указанные размеры также не могут быть меньше минимального размера, допустимого системой. Это минимальное

значение зависит от текущего размера шрифта консоли (выбранного пользователем), а также значений **SM\_CXMIN** и **SM\_CYMIN**, возвращаемых функцией [GetSystemMetrics](#).

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента [виртуального терминала](#). Это решение намеренно согласовывает платформу Windows с другими операционными системами, в которых пользователю предоставляется полный контроль над этим вариантом презентации. Приложения, работающие с помощью кроссплатформенных служебных программ и транспортных средств, таких как SSH, могут не работать должным образом при использовании этого API.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll

## См. также

## ФУНКЦИИ КОНСОЛИ

Входной буфер консоли

**COORD**

**GetConsoleScreenBufferInfo**

**SetConsoleWindowInfo**

# Функция SetConsoleTextAttribute

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает атрибуты символов, записанных в буфер экрана консоли с помощью функции [WriteFile](#) или [WriteConsole](#), или от имени функции [ReadFile](#) или [ReadConsole](#). Эта функция влияет на текст, написанный после вызова функции.

## Синтаксис

C

```
BOOL WINAPI SetConsoleTextAttribute(
    _In_ HANDLE hConsoleOutput,
    _In_ WORD   wAttributes
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*wAttributes* [in]

Атрибуты символов.

# Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы определить текущие атрибуты цвета буфера экрана, вызовите [функцию GetConsoleScreenBufferInfo](#).

### 💡 Совет

Этот API имеет эквивалент виртуального **терминала в последовательностях форматирования текста**. Для всех новых и текущих разработок рекомендуется использовать последовательности виртуальных терминалов .

## Примеры

Пример см. в разделе "[Использование высокоуровневых входных и выходных функций](#)".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Буферы экрана консоли](#)

[GetConsoleScreenBufferInfo](#)

[ReadConsole](#)

[ReadFile](#)

[WriteConsole](#)

[WriteFile](#)

# Функция SetConsoleTitle

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает заголовок текущего окна консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleTitle(
    _In_ LPCTSTR lpConsoleTitle
);
```

## Параметры

*lpConsoleTitle* [in]

Строка, отображаемая в строке заголовка окна консоли. Общий размер должен быть меньше 64 КБ.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

# Замечания

После завершения процесса система восстанавливает исходное название консоли.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

## 💡 Совет

Этот API имеет эквивалент виртуального [терминала в последовательностях заголовков окна](#). Для всех новых и текущих разработок рекомендуется использовать последовательности виртуальных терминалов .

# Примеры

В следующем примере показано, как получить и изменить название консоли.

C

```
#include <windows.h>
#include <tchar.h>
#include <conio.h>
#include <strsafe.h>

int main( void )
{
    TCHAR szOldTitle[MAX_PATH];
    TCHAR szNewTitle[MAX_PATH];

    // Save current console title.

    if( GetConsoleTitle(szOldTitle, MAX_PATH) )
    {
        // Build new console title string.

        StringCchPrintf(szNewTitle, MAX_PATH, TEXT("TEST: %s"), szOldTitle);

        // Set console title to new title
        if( !SetConsoleTitle(szNewTitle) )
        {
            _tprintf(TEXT("SetConsoleTitle failed (%d)\n"), GetLastError());
        }
    }
}
```

```
        return 1;
    }
    else
    {
        _tprintf(TEXT("SetConsoleTitle succeeded.\n"));
    }
}
return 0;
}
```

## Requirements

 Развернуть таблицу

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>SetConsoleTitleW</b> (Юникод) и <b>SetConsoleTitleA</b> (ANSI)

## См. также

[Функции консоли](#)

[GetConsoleOriginalTitle](#)

[GetConsoleTitle](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

# Функция SetConsoleWindowInfo

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает текущий размер и положение окна буфера экрана консоли.

## Синтаксис

C

```
BOOL WINAPI SetConsoleWindowInfo(
    _In_         HANDLE     hConsoleOutput,
    _In_         BOOL       bAbsolute,
    _In_ const  SMALL_RECT *lpConsoleWindow
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. Этот дескриптор должен иметь право доступа **GENERIC\_READ**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*bAbsolute* [in]

Если этот параметр имеет значение **TRUE**, координаты указывают новые верхние и нижние правые углы окна. Если значение равно **FALSE**, координаты относительны к текущим координатам угла окна.

*lpConsoleWindow* [in]

Указатель на [структуру SMALL\\_RECT](#), указывающую новые верхние и правые нижние углы окна.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Функция завершается ошибкой, если указанный прямоугольник окна выходит за рамки буфера экрана консоли. Это означает, что **элементы прямоугольника *lpConsoleWindow*** (или вычисляемые верхние и левые координаты, если *bAbsolute* имеет значение FALSE), не может быть меньше нуля. Аналогичным образом **элементы нижнего и правого** (или вычисляемые нижние и правые координаты) не могут быть больше (высота буфера экрана – 1) и (ширина буфера экрана – 1), соответственно. Функция также завершается ошибкой, если правый элемент (или вычисляемая правая координата) меньше или равен левому элементу (или вычисляемой левой координате) или если **элемент Нижнего** (или вычисляемая нижняя координата) меньше или равен верхнему элементу (или вычисляемой верхней координате).

Для консоли с несколькими буферами экрана изменение расположения окна для одного буфера экрана не влияет на расположения окна других буферов экрана.

Чтобы определить текущий размер и положение окна буфера экрана, используйте [функцию GetConsoleScreenBufferInfo](#). Эта функция также возвращает максимальный размер окна, учитывая текущий размер буфера экрана, текущий размер шрифта и размер экрана. Функция [GetLargestConsoleWindowSize](#) возвращает максимальный размер окна с учетом текущих размеров шрифта и экрана, но не учитывает размер буфера экрана консоли.

[SetConsoleWindowInfo](#) можно использовать для прокрутки содержимого буфера экрана консоли путем смены положения прямоугольника окна без изменения его размера.

 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Примеры

Пример см. в разделе "[Прокрутка окна буфера экрана](#)".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[GetConsoleScreenBufferInfo](#)

[GetLargestConsoleWindowSize](#)

[ScrollConsoleScreenBuffer](#)

[Прокрутка буфера экрана](#)

[SMALL\\_RECT](#)

# Функция SetCurrentConsoleFontEx

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает расширенные сведения о текущем шрифте консоли.

## Синтаксис

C

```
BOOL WINAPI SetCurrentConsoleFontEx(
    _In_     HANDLE          hConsoleOutput,
    _In_     BOOL            bMaximumWindow,
    _In_     PCONSOLE_FONT_INFOEX lpConsoleCurrentFontEx
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*bMaximumWindow* [in]

Если этот параметр имеет значение **TRUE**, сведения о шрифте задаются для максимального размера окна. Если этот параметр имеет значение **FALSE**, сведения о шрифте задаются для текущего размера окна.

*lpConsoleCurrentFontEx* [in]

Указатель на структуру **CONSOLE\_FONT\_INFOEX**, содержащую сведения о шрифте.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Чтобы скомпилировать приложение, использующее эту функцию, определите **\_WIN32\_WINNT** как **0x0500** или более поздней версии. Дополнительные сведения см. в разделе "[Использование заголовков Windows](#)".

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального **терминала**. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где пользователь получает полный контроль над этим параметром презентации. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## **См. также**

[Функции консоли](#)

[CONSOLE\\_FONT\\_INFOEX](#)

# Функция SetStdHandle

Статья • 13.12.2023

Задает дескриптор указанного стандартного устройства (стандартная входная, стандартная выходная или стандартная ошибка).

## Синтаксис

C++

```
BOOL WINAPI SetStdHandle(
    _In_ DWORD nStdHandle,
    _In_ HANDLE hHandle
);
```

## Параметры

*nStdHandle* [ввод]

Стандартное устройство, для которого необходимо задать дескриптор. Этот параметр может принимать одно из указанных ниже значений.

 [Развернуть таблицу](#)

Значение	Значение
STD_INPUT_HANDLE ((DWORD)-10)	Стандартное устройство ввода. Изначально это входной буфер консоли, CONIN\$.
STD_OUTPUT_HANDLE ((DWORD)-11)	Стандартное выходное устройство. Изначально это активный буфер экрана консоли, CONOUT\$.
STD_ERROR_HANDLE ((DWORD)-12)	Устройство стандартных ошибок. Изначально это активный буфер экрана консоли, CONOUT\$.

### Примечание

Значения этих констант являются числами без знака, но определяются в файлах заголовков как приведение из числа со знаком и используют компилятор С для их преобразования к 32-разрядному значению, которое ниже максимального. При взаимодействии с этими дескрипторами на языке, который не анализирует заголовки и переопределяет константы, следует

учитывать это ограничение. В качестве примера ((DWORD)-10) — это число 4294967286 без знака.

*hHandle* [in]

Дескриптор стандартного устройства.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Стандартные дескрипторы процесса, возможно, были перенаправлены вызовом [SetStdHandle](#), в этом случае [GetStdHandle](#) вернет перенаправленный дескриптор. Если стандартные дескрипторы были перенаправлены, можно указать значение CONIN\$ в вызове [функции CreateFile](#), чтобы получить дескриптор входного буфера консоли. Аналогичным образом можно указать значение CONOUT\$, чтобы получить дескриптор активного буфера экрана консоли.

## Примеры

Пример см. в разделе "[Создание дочернего процесса с перенаправленными входными и выходными данными](#)".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ProcessEnv.h (через Winbase.h, включая Windows.h)

Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[Дескрипторы консоли](#)

[CreateFile](#)

[GetStdHandle](#)

# Функция WriteConsole

Статья • 12.07.2023

Записывает строку символов в буфер экрана консоли, начиная с текущего положения курсора.

## Синтаксис

C

```
BOOL WINAPI WriteConsole(
    _In_                 HANDLE hConsoleOutput,
    _In_     const VOID *lpBuffer,
    _In_             DWORD nNumberOfCharsToWrite,
    _Out_opt_ LPDWORD lpNumberOfCharsWritten,
    _Reserved_ LPVOID lpReserved
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [ввод]

Указатель на буфер, содержащий символы для записи в буфер экрана консоли.

Предполагается, что это массив `char` для `WriteConsoleA` или `wchar_t` для `WriteConsoleW`.

*nNumberOfCharsToWrite* [ввод]

Количество записываемых символов. Если общий размер указанного количества символов превышает доступную кучу, функция завершается ошибкой **ERROR\_NOT\_ENOUGH\_MEMORY**.

*lpNumberOfCharsWritten* [вывод, необязательный]

Указатель на переменную, которая получает количество фактически записанных символов.

*lpReserved* Зарезервировано; должно быть значение **NULL**.

# Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Комментарии

Функция [WriteConsole](#) записывает символы в буфер экрана консоли в текущем положении курсора. Курсор передвигается по мере написания символов. Функция [SetConsoleCursorPosition](#) задает текущее положение курсора.

Символы записываются с помощью атрибутов цвета переднего плана и фона, связанных с буфером экрана консоли. Функция [SetConsoleTextAttribute](#) изменяет эти цвета. Чтобы определить текущие атрибуты цвета и текущее положение курсора, используйте [GetConsoleScreenBufferInfo](#).

Все режимы ввода, влияющие на поведение функции [WriteFile](#), действуют так же на [WriteConsole](#). Чтобы получить и установить режимы вывода буфера экрана консоли, используйте функции [GetConsoleMode](#) и [SetConsoleMode](#).

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

[WriteConsole](#) завершается ошибкой, если используется со стандартным дескриптором, который перенаправляется в файл. Если приложение обрабатывает многоязычный вывод, который можно перенаправить, определите, является ли дескриптор вывода дескриптором консоли (один метод заключается в вызове функции [GetConsoleMode](#) и проверке успешности ее выполнения). Если дескриптор является дескриптором консоли, вызовите [WriteConsole](#). Если дескриптор не является дескриптором консоли, вывод перенаправляется и для выполнения операций ввода-вывода следует вызвать [WriteFile](#). Обязательно добавьте в обычный текстовый файл в Юникоде метку порядка следования байтов в качестве префикса. Дополнительные сведения см. в статье [Использование меток порядка следования байтов](#).

Несмотря на то что приложение может использовать **WriteConsole** в режиме ANSI для записи символов ANSI, консоли не поддерживают последовательности escape-кодов ANSI или виртуальных терминалов, если они не включены. Дополнительные сведения, а также информацию о применимости к версии операционной системы см. в статье [Последовательности виртуального терминала в консоли](#).

Если escape-последовательности виртуального терминала не включены, функции консоли могут обеспечить эквивалентную функциональность. Дополнительные сведения см. в статьях о функциях [SetCursorPos](#), [SetConsoleTextAttribute](#) и [GetConsoleCursorInfo](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL	Kernel32.dll
Имя в кодировке Юникод и ANSI	<b>WriteConsoleW</b> (Юникод) и <b>WriteConsoleA</b> (ANSI)

## См. также

[Функции консоли](#)

[GetConsoleCursorInfo](#)

[GetConsoleMode](#)

[GetConsoleScreenBufferInfo](#)

[Методы ввода и вывода](#)

[ReadConsole](#)

[SetConsoleCP](#)

[SetConsoleCursorPosition](#)

[\*\*SetConsoleMode\*\*](#)

[\*\*SetConsoleOutputCP\*\*](#)

[\*\*SetConsoleTextAttribute\*\*](#)

[\*\*SetCursorPos\*\*](#)

[\*\*WriteFile\*\*](#)

# Функция WriteConsoleInput

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Записывает данные непосредственно в буфер входных данных консоли.

## Синтаксис

C

```
BOOL WINAPI WriteConsoleInput(
    _In_          HANDLE      hConsoleInput,
    _In_  const INPUT_RECORD *lpBuffer,
    _In_          DWORD       nLength,
    _Out_         LPDWORD     lpNumberOfEventsWritten
);
```

## Параметры

*hConsoleInput* [in]

Дескриптор входного буфера консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpBuffer* [ввод]

Указатель на массив INPUT\_RECORD структур, содержащих данные, записываемые в входной буфер.

*nLength* [in]

Количество записей входных данных, которые необходимо записать.

*lpNumberOfEventsWritten* [out]

Указатель на переменную, которая получает количество записей ввода фактически записанных.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

`WriteConsoleInput` помещает входные записи в входной буфер за любыми ожидающих событий в буфере. Входной буфер динамически растет, если это необходимо, чтобы хранить столько событий, сколько записываются.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

### 💡 Совет

Этот API не рекомендуется и не имеет эквивалента виртуального [терминала](#). Это решение намеренно сопоставляет платформу Windows с другими операционными системами. Эта операция считается неправильной [командой](#) для этого буфера. Удаленное взаимодействие приложений с помощью межплатформенных служебных программ и транспорта, таких как SSH, может не работать должным образом, если используется этот API.

## Requirements

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>WriteConsoleInputW</code> (Юникод) и <code>WriteConsoleInputA</code> (ANSI)

## См. также

[Функции консоли](#)

[INPUT\\_RECORD](#)

[Низкоуровневые функции ввода консоли](#)

[MapVirtualKey](#)

[PeekConsoleInput](#)

[ReadConsoleInput](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[VkKeyScan](#)

# Функция WriteConsoleOutput

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Записывает данные символов и атрибутов цвета в указанный прямоугольный блок ячеек символов в буфере экрана консоли. Данные, записанные, взяты из соответствующего прямоугольного блока размера в указанном расположении в исходном буфере.

## Синтаксис

```
C

BOOL WINAPI WriteConsoleOutput(
    _In_           HANDLE      hConsoleOutput,
    _In_   const CHAR_INFO *lpBuffer,
    _In_           COORD      dwBufferSize,
    _In_           COORD      dwBufferCoord,
    _Inout_        PSMALL_RECT lpWriteRegion
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

#### *lpBuffer* [ввод]

Данные, записываемые в буфер экрана консоли. Этот указатель рассматривается как источник двухмерного массива CHAR\_INFO структур, размер которого задается параметром *dwBufferSize*.

#### *dwBufferSize* [in]

Размер буфера, на который указывает параметр *lpBuffer*, в ячейках символов.

Элемент **X** структуры COORD — это количество столбцов; элемент **Y** — это число строк.

#### *dwBufferCoord* [in]

Координаты левой верхней ячейки в буфере, на которую указывает параметр *lpBuffer*. Элемент **X** структуры COORD — это столбец, а элемент **Y** — строка.

#### *lpWriteRegion* [in, out]

Указатель на структуру **SMALL\_RECT**. Во входных данных члены структуры указывают координаты прямоугольника буфера экрана консоли в левом верхнем и нижнем углу для записи. В выходных данных члены структуры указывают фактический прямоугольник, который использовался.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

`WriteConsoleOutput` обрабатывает исходный буфер и буфер целевого экрана как двухмерные массивы (столбцы и строки символьных ячеек). Прямоугольник, на который указывает параметр *lpWriteRegion*, указывает размер и расположение блока, записываемого в буфер экрана консоли. Прямоугольник того же размера расположен с левой верхней ячейкой в координатах параметра *dwBufferCoord* в массиве *lpBufferCoord*. Данные из ячеек, которые находятся на пересечении этого прямоугольника и прямоугольника исходного буфера (измерения которых указаны параметром *dwBufferSize*) записываются в прямоугольник назначения.

Ячейки в прямоугольнике назначения, соответствующее исходное расположение которых находится вне границ исходного прямоугольника буфера, остаются не затронуты операцией записи. Другими словами, это ячейки, для которых данные не доступны для записи.

Перед возвратом `WriteConsoleOutput` он задает элементы *lpWriteRegion* фактическим прямоугольником буфера экрана, затронутым операцией записи. Этот прямоугольник отражает ячейки в целевом прямоугольнике, для которого существовала соответствующая ячейка в исходном буфере, так как `WriteConsoleOutput` обрезает размеры целевого прямоугольника к границам буфера экрана консоли.

Если прямоугольник, указанный *lpWriteRegion*, находится полностью за пределами буфера экрана консоли или если соответствующий прямоугольник находится полностью за пределами исходного буфера, данные не записываются. В этом случае функция возвращается с элементами структуры, на которую указывает параметр *lpWriteRegion*, таким образом, что правый элемент меньше левого или нижнего элемента меньше верхнего. Чтобы определить размер буфера экрана консоли, используйте [функцию GetConsoleScreenBufferInfo](#).

`WriteConsoleOutput` не влияет на положение курсора.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

#### 💡 Совет

Этот API имеет эквивалент виртуального [терминала в последовательностях форматирования текста и размещения курсоров](#). Переместите курсор в расположение для вставки, примените требуемое форматирование и запишите текст. Для всех новых и текущих разработок рекомендуется использовать последовательности виртуальных терминалов.

## Примеры

Пример см. в разделе "[Чтение и запись блоков символов и атрибутов](#)".

## Requirements

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<b>WriteConsoleOutputW</b> (Юникод) и <b>WriteConsoleOutputA</b> (ANSI)

## См. также

[Функции консоли](#)

[CHAR\\_INFO](#)

[COORD](#)

[GetConsoleScreenBufferInfo](#)

[Функции вывода консоли низкого уровня](#)

[ReadConsoleOutput](#)

[ReadConsoleOutputAttribute](#)

[ReadConsoleOutputCharacter](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[SMALL\\_RECT](#)

[WriteConsoleOutputAttribute](#)

[WriteConsoleOutputCharacter](#)

# Функция WriteConsoleOutputAttribute

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Копирует ряд атрибутов символов в последовательные ячейки буфера экрана консоли, начиная с указанного расположения.

## Синтаксис

C

```
BOOL WINAPI WriteConsoleOutputAttribute(
    _In_          HANDLE hConsoleOutput,
    _In_  const WORD  *lpAttribute,
    _In_          DWORD nLength,
    _In_          COORD dwWriteCoord,
    _Out_         LPDWORD lpNumberOfAttrsWritten
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpAttribute* [in]

Атрибуты, используемые при записи в буфер экрана консоли. Дополнительные сведения см. в разделе ["Атрибуты символов"](#).

*nLength* [in]

Количество ячеек буфера экрана, в которые будут скопированы атрибуты.

*dwWriteCoord* [in]

**Структура COORD**, указывающая координаты символов первой ячейки в буфере экрана консоли, в которую записываются атрибуты.

*lpNumberOfAttrsWritten* [out]

Указатель на переменную, которая получает количество атрибутов, фактически записанных в буфер экрана консоли.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Если число атрибутов, которые необходимо записать, выходит за пределы указанной строки в буфере экрана консоли, атрибуты записываются в следующую строку. Если число атрибутов, записываемых в буфер экрана консоли, выходит за пределы буфера экрана консоли, атрибуты записываются до конца буфера экрана консоли.

Значения символов на позициях, записанных в них, не изменяются.

### 💡 Совет

Этот API имеет эквивалент виртуального [терминала в последовательностях](#) форматирования [текста](#) и размещения курсоров. Переместите курсор в расположение для вставки, примените требуемое форматирование и запишите текст для заполнения. Нет эквивалента применению цвета к области без создания текста. Это решение намеренно выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, как ожидается, запоминает собственное состояние рисования для дальнейшего управления.

## Requirements

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll

## См. также

[Функции консоли](#)

[COORD](#)

[Функции вывода консоли низкого уровня](#)

[ReadConsoleOutput](#)

[ReadConsoleOutputAttribute](#)

[ReadConsoleOutputCharacter](#)

[WriteConsoleOutput](#)

[WriteConsoleOutputCharacter](#)

# Функция WriteConsoleOutputCharacter

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Копирует ряд символов в последовательные ячейки буфера экрана консоли, начиная с указанного расположения.

## Синтаксис

C

```
BOOL WINAPI WriteConsoleOutputCharacter(
    _In_     HANDLE   hConsoleOutput,
    _In_     LPCTSTR  lpCharacter,
    _In_     DWORD    nLength,
    _In_     COORD    dwWriteCoord,
    _Out_    LPDWORD  lpNumberOfCharsWritten
);
```

## Параметры

*hConsoleOutput* [ввод]

Дескриптор буфера экрана консоли. У дескриптора должно быть право на доступ **GENERIC\_WRITE**. Дополнительные сведения см. в статье [Безопасность и права доступа для буфера консоли](#).

*lpCharacter* [in]

Символы, записываемые в буфер экрана консоли.

*nLength* [in]

Количество записываемых символов.

*dwWriteCoord* [in]

[Структура COORD](#), указывающая координаты символов первой ячейки в буфере экрана консоли, в которую будут записываться символы.

*lpNumberOfCharsWritten* [out]

Указатель на переменную, которая получает количество фактически записанных символов.

## Возвращаемое значение

Если функция выполняется успешно, возвращается ненулевое значение.

Если функция выполняется неудачно, возвращается нулевое значение.

Дополнительные сведения об ошибке можно получить, вызвав [GetLastError](#).

## Замечания

Если число символов, записываемых в конец указанной строки в буфере экрана консоли, символы записываются в следующую строку. Если число символов, записываемых в буфер экрана консоли, выходит за пределы буфера экрана консоли, символы записываются до конца буфера экрана консоли.

Значения атрибутов на позициях, записанных в не изменяемые.

Эта функция использует либо символы Юникода, либо 8-разрядные символы из текущей кодовой страницы консоли. Кодовая страница консоли по умолчанию изначально соответствует кодовой странице OEM системы. Чтобы изменить кодовую страницу консоли, используйте функции [SetConsoleCP](#) или [SetConsoleOutputCP](#). Пользователи прежних версий могут также использовать команды `chcp` или `mode con cp select=` (но это не рекомендуется для новой разработки).

### 💡 Совет

Этот API имеет эквивалент виртуального [терминала в последовательностях](#) форматирования [текста](#) и размещения курсоров. Переместите курсор в расположение для вставки, примените требуемое форматирование и запишите текст для заполнения. Нет эквивалента отправки текста в область без применения активного форматирования цвета. Это решение намеренно

выравнивает платформу Windows с другими операционными системами, где отдельное клиентское приложение, как ожидается, запоминает собственное состояние рисования для дальнейшего управления.

# Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)
Библиотека	Kernel32.lib
DLL-библиотеки	Kernel32.dll
Имена Юникода и ANSI	<code>WriteConsoleOutputCharacterW</code> (Юникод) и <code>WriteConsoleOutputCharacterA</code> (ANSI)

## См. также

[Функции консоли](#)

[COORD](#)

[Функции вывода консоли низкого уровня](#)

[ReadConsoleOutput](#)

[ReadConsoleOutputAttribute](#)

[ReadConsoleOutputCharacter](#)

[SetConsoleCP](#)

[SetConsoleOutputCP](#)

[WriteConsoleOutput](#)

[WriteConsoleOutputAttribute](#)

# Структуры консоли

Статья • 12.07.2023

Для доступа к консоли используются следующие структуры.

- [CHAR\\_INFO](#)
- [CONSOLE\\_CURSOR\\_INFO](#)
- [CONSOLE\\_FONT\\_INFO](#)
- [CONSOLE\\_FONT\\_INFOEX](#)
- [CONSOLE\\_HISTORY\\_INFO](#)
- [CONSOLE\\_READCONSOLE\\_CONTROL](#)
- [CONSOLE\\_SCREEN\\_BUFFER\\_INFO](#)
- [CONSOLE\\_SCREEN\\_BUFFER\\_INFOEX](#)
- [CONSOLE\\_SELECTION\\_INFO](#)
- [COORD](#)
- [FOCUS\\_EVENT\\_RECORD](#)
- [INPUT\\_RECORD](#)
- [KEY\\_EVENT\\_RECORD](#)
- [MENU\\_EVENT\\_RECORD](#)
- [MOUSE\\_EVENT\\_RECORD](#)
- [SMALL\\_RECT](#)
- [WINDOW\\_BUFFER\\_SIZE\\_RECORD](#)

# Структура CONSOLE\_HISTORY\_INFO

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Содержит сведения о журнале консоли.

## Синтаксис

```
C

typedef struct {
    UINT    cbSize;
    UINT    HistoryBufferSize;
    UINT    NumberOfHistoryBuffers;
    DWORD   dwFlags;
} CONSOLE_HISTORY_INFO, *PCONSOLE_HISTORY_INFO;
```

## Участники

### cbSize

Размер структуры в байтах. Задайте для этого элемента `sizeof(CONSOLE_HISTORY_INFO)` значение.

### HistoryBufferSize

Количество команд, хранящихся в каждом буфере журнала.

### NumberOfHistoryBuffers

Количество буферов журнала, хранимых для этого процесса консоли.

## **dwFlags**

Этот параметр может быть равен нулю или следующему значению.

 [Развернуть таблицу](#)

Значение	Значение
HISTORY_NO_DUP_FLAG 0x1	Повторяющиеся записи не будут храниться в буфере журнала.

## **Requirements**

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Верхний колонтитул	ConsoleApi3.h (через WinCon.h, включая Windows.h)

## **См. также**

[GetConsoleHistoryInfo](#)

[SetConsoleHistoryInfo](#)

# Структура `CONSOLE_READCONSOLE_CONTROL`

Статья • 23.10.2023

Содержит сведения для операции чтения консоли.

## Синтаксис

C

```
typedef struct _CONSOLE_READCONSOLE_CONTROL {
    ULONG nLength;
    ULONG nInitialChars;
    ULONG dwCtrlWakeupMask;
    ULONG dwControlKeyState;
} CONSOLE_READCONSOLE_CONTROL, *PCONSOLE_READCONSOLE_CONTROL;
```

## Участники

### `nLength`

Размер структуры. Задайте для этого элемента

`sizeof(CONSOLE_READCONSOLE_CONTROL)` значение .

### `nInitialChars`

Количество символов, которые следует пропустить (и таким образом сохранить) перед записью только что прочитанных входных данных в буфере, переданном [функции ReadConsole](#). Это значение должно быть меньше *параметра nNumberOfCharsToRead* функции `ReadConsole`.

### `dwCtrlWakeupMask`

Маска, указывающая, какие символы элемента управления между `0x00` и `0x1F` должны использоваться для сигнала о завершении чтения. Каждый бит соответствует символу с наименьшим значительным битом, соответствующим `0x00` или `NUL` наиболее значимым битом, соответствующим `0x1F` или `us`. Можно указать несколько битов (символов управления).

### `dwControlKeyState`

Состояние ключей управления. Этот элемент может быть одним или несколькими из следующих значений.

Значение	Значение
CAPSLOCK_ON 0x0080	Индикатор CAPS LOCK включен.
ENHANCED_KEY 0x0100	Ключ улучшен. См . <a href="#">примечания</a> .
LEFT_ALT_PRESSED 0x0002	Нажата левая клавиша ALT.
LEFT_CTRL_PRESSED 0x0008	Нажата левая клавиша CTRL.
NUMLOCK_ON 0x0020	Индикатор NUM LOCK включен.
RIGHT_ALT_PRESSED 0x0001	Нажимается справа клавиша ALT.
RIGHT_CTRL_PRESSED 0x0004	Нажата правая клавиша CTRL.
SCROLLLOCK_ON 0x0040	Индикатор SCROLL LOCK включен.
SHIFT_PRESSED 0x0010	Клавиша SHIFT нажимается.

## Требования

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Заголовок	ConsoleApi.h (через WinCon.h, включая Windows.h)

## См. также

[ReadConsole](#)

# Структура CONSOLE\_SELECTION\_INFO

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Содержит сведения о выборе консоли.

## Синтаксис

C

```
typedef struct _CONSOLE_SELECTION_INFO {
    DWORD      dwFlags;
    COORD      dwSelectionAnchor;
    SMALL_RECT srSelection;
} CONSOLE_SELECTION_INFO, *PCONSOLE_SELECTION_INFO;
```

## Участники

### dwFlags

Индикатор выбора. Этот элемент может быть одним или несколькими из следующих значений.

Значение	Значение
CONSOLE_MOUSE_DOWN 0x0008	Мышь вниз. Пользователь активно настраивает прямоугольник выбора с помощью мыши.
CONSOLE_MOUSE_SELECTION 0x0004	Выбор с помощью мыши. Если этот параметр отключен, пользователь выбирает <code>conhost.exe</code> режим

Значение	Значение
	разметки с помощью клавиатуры.
CONSOLE_NO_SELECTION 0x0000	Нет выбора.
CONSOLE_SELECTION_IN_PROGRESS 0x0001	Выбор начался. Если выбрана мышь, это обычно не происходит без флага <code>CONSOLE_SELECTION_NOT_EMPTY</code> . Если выбрана клавиатура, это может произойти при вводе режима разметки, но пользователь по-прежнему перемещается на начальную позицию.
CONSOLE_SELECTION_NOT_EMPTY 0x0002	Прямоугольник выделения не пуст. Полезные данные <code>dwSelectionAnchor</code> и <code>srSelection</code> допустимы.

### **dwSelectionAnchor**

[Структура COORD](#), указывающая привязку выделения в символах.

### **srSelection**

Структура [SMALL\\_RECT](#), указывающая прямоугольник выбора.

## Требования

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Заголовок	ConsoleApi3.h (через WinCon.h, включая Windows.h)

## См. также

[COORD](#)

[GetConsoleSelectionInfo](#)

[SMALL\\_RECT](#)

# Структура CONSOLE\_CURSOR\_INFO

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Содержит сведения о курсоре консоли.

## Синтаксис

C

```
typedef struct _CONSOLE_CURSOR_INFO {
    DWORD dwSize;
    BOOL bVisible;
} CONSOLE_CURSOR_INFO, *P_CONSOLE_CURSOR_INFO;
```

## Участники

### dwSize

Процент ячейки символа, заполненной курсором. Это значение составляет от 1 до 100. Внешний вид курсора зависит от полного заполнения ячейки до отображения в виде горизонтальной линии в нижней части ячейки.

## ⓘ Примечание

Хотя значение dwSize обычно составляет от 1 до 100, при некоторых обстоятельствах может быть возвращено значение за пределами этого

диапазона. Например, если CursorSize имеет значение 0 в реестре, возвращаемое значение dwSize будет равно 0.

### bVisible

Видимость курсора. Если курсор виден, этот элемент имеет значение TRUE.

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	WinCon.h (включая Windows.h)

## См. также

[GetConsoleCursorInfo](#)

[SetConsoleCursorInfo](#)

# Структура CONSOLE\_FONT\_INFO

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Содержит сведения о шрифте консоли.

## Синтаксис

```
C

typedef struct _CONSOLE_FONT_INFO {
    DWORD nFont;
    COORD dwFontSize;
} CONSOLE_FONT_INFO, *PCONSOLE_FONT_INFO;
```

## Участники

### nFont

Индекс шрифта в таблице шрифтов консоли системы.

### dwFontSize

[Структура COORD](#), содержащая ширину и высоту каждого символа в шрифте в логических единицах. Элемент X содержит ширину, а элемент Y содержит высоту.

## Замечания

Чтобы получить размер шрифта, передайте индекс [шрифта](#) функции [GetConsoleFontSize](#).

# Требования

Минимальная версия клиента	Windows XP [только классические приложения]
Минимальная версия сервера	Windows Server 2003 [только классические приложения]
Заголовок	WinCon.h (включая Windows.h)

## См. также

[COORD](#)

[GetCurrentConsoleFont](#)

# Структура CONSOLE\_FONT\_INFOEX

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Содержит расширенные сведения для шрифта консоли.

## Синтаксис

```
C

typedef struct _CONSOLE_FONT_INFOEX {
    ULONG cbSize;
    DWORD nFont;
    COORD dwFontSize;
    UINT  FontFamily;
    UINT  FontWeight;
    WCHAR FaceName[LF_FACESIZE];
} CONSOLE_FONT_INFOEX, *PCONSOLE_FONT_INFOEX;
```

## Участники

### cbSize

Размер этой структуры в байтах. Этот элемент должен быть задан перед `sizeof(CONSOLE_FONT_INFOEX)` вызовом [GetCurrentConsoleFontEx](#) или завершится сбоем.

### nFont

Индекс шрифта в таблице шрифтов консоли системы.

## **dwFontSize**

Структура **COORD**, содержащая ширину и высоту каждого символа в шрифте в логических единицах. Элемент **X** содержит ширину, а элемент **Y** содержит высоту.

## **FontFamily**

Шаг шрифта и семья. Сведения о возможных значениях этого элемента см. в описании элемента **tmPitchAndFamily** структуры **TEXTMETRIC**.

## **FontWeight**

Вес шрифта. Вес может варьироваться от 100 до 1000, в нескольких 100. Например, нормальный вес составляет 400, а 700 полужирным шрифтом.

## **FaceName**

Имя шрифта (например, Courier или Arial).

## **Замечания**

Чтобы получить размер шрифта, передайте индекс [шрифта](#) функции [GetConsoleFontSize](#).

## **Requirements**

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Верхний колонтитул	WinCon.h (включая Windows.h)

## **См. также**

[GetCurrentConsoleFontEx](#)

# Структура **CONSOLE\_SCREEN\_BUFFER\_INFO**

Статья • 13.12.2023

Содержит сведения о буфере экрана консоли.

## Синтаксис

```
C

typedef struct _CONSOLE_SCREEN_BUFFER_INFO {
    COORD      dwSize;
    COORD      dwCursorPosition;
    WORD       wAttributes;
    SMALL_RECT srWindow;
    COORD      dwMaximumWindowSize;
} CONSOLE_SCREEN_BUFFER_INFO;
```

## Участники

### **dwSize**

[Структура COORD](#), содержащая размер буфера экрана консоли в символьных столбцах и строках.

### **dwCursorPosition**

[Структура COORD](#), содержащая координаты столбца и строки курсора в буфере экрана консоли.

### **wAttributes**

Атрибуты символов, записанных в буфер экрана функциями [WriteFile](#) и [WriteConsole](#), или эхо в буфер экрана функциями [ReadFile](#) и [ReadConsole](#).  
Дополнительные сведения см. в разделе "[Атрибуты символов](#)".

### **srWindow**

Структура [SMALL\\_RECT](#), содержащая координаты буфера экрана консоли в левом верхнем и правом нижнем углу окна отображения.

### **dwMaximumWindowSize**

[Структура COORD](#), содержащая максимальный размер окна консоли в символьных столбцах и строках, учитывая текущий размер буфера экрана и шрифт и размер экрана.

# Примеры

Пример см. в разделе "Прокрутка содержимого буфера экрана".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	ConsoleApi2.h (через WinCon.h, включая Windows.h)

## См. также

[COORD](#)

[GetConsoleScreenBufferInfo](#)

[ReadConsole](#)

[ReadFile](#)

[SMALL\\_RECT](#)

[WriteConsole](#)

[WriteFile](#)

# Структура **CONSOLE\_SCREEN\_BUFFER\_INFOEX**

Статья • 23.10.2023

Содержит расширенные сведения о буфере экрана консоли.

## Синтаксис

C

```
typedef struct _CONSOLE_SCREEN_BUFFER_INFOEX {
    ULONG      cbSize;
    COORD      dwSize;
    COORD      dwCursorPosition;
    WORD       wAttributes;
    SMALL_RECT srWindow;
    COORD      dwMaximumWindowSize;
    WORD       wPopupAttributes;
    BOOL       bFullscreenSupported;
    COLORREF   ColorTable[16];
} CONSOLE_SCREEN_BUFFER_INFOEX, *PCONSOLE_SCREEN_BUFFER_INFOEX;
```

## Участники

### **cbSize**

Размер этой структуры в байтах.

### **dwSize**

[Структура COORD](#), содержащая размер буфера экрана консоли в символьных столбцах и строках.

### **dwCursorPosition**

[Структура COORD](#), содержащая координаты столбца и строки курсора в буфере экрана консоли.

### **wAttributes**

Атрибуты символов, записанных в буфер экрана функциями [WriteFile](#) и [WriteConsole](#), или эхо в буфер экрана функциями [ReadFile](#) и [ReadConsole](#).

Дополнительные сведения см. в разделе "Атрибуты символов".

### **srWindow**

Структура [SMALL\\_RECT](#), содержащая координаты буфера экрана консоли в левом

верхнем и правом нижнем углу окна отображения.

#### **dwMaximumWindowSize**

**Структура COORD**, содержащая максимальный размер окна консоли в символьных столбцах и строках, учитывая текущий размер буфера экрана и шрифт и размер экрана.

#### **wPopupAttributes**

Атрибут заполнения всплывающих окон консоли.

#### **bFullscreenSupported**

Если этот член является `TRUE`, поддерживается полноэкранный режим. В противном случае это не так. Это всегда будет для `FALSE` систем после Windows Vista с **моделью** драйвера WDDM, так как прямой доступ VGA к монитору больше недоступен.

#### **ColorTable**

Массив значений **COLORREF**, описывающих параметры цвета консоли.

## Требования

Минимальная версия клиента	Windows Vista [только классические приложения]
Минимальная версия сервера	Windows Server 2008 [только классические приложения]
Заголовок	ConsoleApi2.h (через WinCon.h, включая Windows.h)

## См. также

[COORD](#)

[GetConsoleScreenBufferInfoEx](#)

[SetConsoleScreenBufferInfoEx](#)

[SMALL\\_RECT](#)

# Структура CHAR\_INFO

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Задает символ Юникода или ANSI и его атрибуты. Эта структура используется функциями консоли для чтения и записи в буфер экрана консоли.

## Синтаксис

C

```
typedef struct _CHAR_INFO {
    union {
        WCHAR UnicodeChar;
        CHAR AsciiChar;
    } Char;
    WORD Attributes;
} CHAR_INFO, *PCHAR_INFO;
```

## Участники

### Char

Объединение следующих членов.

### ЮникодChar

Символ Юникода ячейки символов буфера экрана.

### AsciiChar

Символ ANSI ячейки буфера экрана.

## Атрибуты

Атрибуты символов. Этот элемент может быть нулевым или любым сочетанием следующих значений.

 [Развернуть таблицу](#)

Значение	Значение
FOREGROUND_BLUE 0x0001	Текст содержит синий цвет.
FOREGROUND_GREEN 0x0002	Текст содержит зеленый цвет.
FOREGROUND_RED 0x0004	Текст содержит красный цвет.
BACKGROUND_INTENSITY 0x0008	Для цвета текста изменена интенсивность.
BACKGROUND_BLUE 0x0010	Фон содержит синий цвет.
BACKGROUND_GREEN 0x0020	Фон содержит зеленый цвет.
BACKGROUND_RED 0x0040	Фон содержит красный цвет.
BACKGROUND_INTENSITY 0x0080	Для цвета фона изменена интенсивность.
COMMON_LVB.LEADING_BYTE 0x0100	Начальный байт.
COMMON_LVB.TRAILING_BYTE 0x0200	Конечный байт.
COMMON_LVB_GRID_HORIZONTAL 0x0400	Верхний горизонтальный.
COMMON_LVB_GRID_LVERTICAL 0x0800	Левый вертикальный.
COMMON_LVB_GRID_RVERTICAL 0x1000	Правый вертикальный.
COMMON_LVB_REVERSE_VIDEO 0x4000	Обратный передний план и фоновый атрибут.
COMMON_LVB_UNDERSCORE 0x8000	Знак подчеркивания.

## Примеры

Пример см. в разделе "[Прокрутка содержимого буфера экрана](#)".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	WinCon.h (включая Windows.h)

## См. также

[ReadConsoleOutput](#)

[ScrollConsoleScreenBuffer](#)

[WriteConsoleOutput](#)

# Структура COORD

Статья • 13.12.2023

Определяет координаты символьной ячейки в буфере экрана консоли. Источник системы координат (0,0) находится в верхней левой ячейке буфера.

## Синтаксис

C

```
typedef struct _COORD {  
    SHORT X;  
    SHORT Y;  
} COORD, *PCOORD;
```

## Участники

X

Значение горизонтальной координаты или столбца. Единицы зависят от вызова функции.

Y

Вертикальная координата или значение строки. Единицы зависят от вызова функции.

## Примеры

Пример см. в разделе "[Прокрутка содержимого буфера экрана](#)".

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]

## См. также

[CONSOLE\\_FONT\\_INFO](#)

[CONSOLE\\_SCREEN\\_BUFFER\\_INFO](#)

[CONSOLE\\_SELECTION\\_INFO](#)

[FillConsoleOutputAttribute](#)

[FillConsoleOutputCharacter](#)

[GetConsoleFontSize](#)

[GetLargestConsoleWindowSize](#)

[MOUSE\\_EVENT\\_RECORD](#)

[ReadConsoleOutput](#)

[ReadConsoleOutputAttribute](#)

[ReadConsoleOutputCharacter](#)

[ScrollConsoleScreenBuffer](#)

[SetConsoleCursorPosition](#)

[SetConsoleDisplayMode](#)

[SetConsoleScreenBufferSize](#)

[WINDOW\\_BUFFER\\_SIZE\\_RECORD](#)

[WriteConsoleOutput](#)

[WriteConsoleOutputAttribute](#)

[WriteConsoleOutputCharacter](#)

# Структура SMALL\_RECT

Статья • 13.12.2023

Определяет координаты верхнего левого и нижнего правых углов прямоугольника.

## Синтаксис

C

```
typedef struct _SMALL_RECT {
    SHORT Left;
    SHORT Top;
    SHORT Right;
    SHORT Bottom;
} SMALL_RECT;
```

## Участники

### Left

Координата x левого верхнего угла прямоугольника.

### Top

Координата y левого верхнего угла прямоугольника.

### Right

Координата x правого нижнего угла прямоугольника.

### Снизу

Координата y правого нижнего угла прямоугольника.

## Замечания

Эта структура используется функциями консоли для указания прямоугольных областей буферов экрана консоли, где координаты указывают строки и столбцы ячеек символов буфера экрана.

## Примеры

Пример см. в разделе "[Прокрутка содержимого буфера экрана](#)".

# Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтидул	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[RECT](#) ↗

[RECTL](#) ↗

# Структура INPUT\_RECORD

Статья • 23.10.2023

Описывает входное событие в буфере входных данных консоли. Эти записи можно считывать из входного буфера с помощью [функции ReadConsoleInput](#) или [PeekConsoleInput](#) или записи в входной буфер с помощью [функции WriteConsoleInput](#).

## Синтаксис

C

```
typedef struct _INPUT_RECORD {
    WORD EventType;
    union {
        KEY_EVENT_RECORD KeyEvent;
        MOUSE_EVENT_RECORD MouseEvent;
        WINDOW_BUFFER_SIZE_RECORD WindowBufferSizeEvent;
        MENU_EVENT_RECORD MenuEvent;
        FOCUS_EVENT_RECORD FocusEvent;
    } Event;
} INPUT_RECORD;
```

## Участники

### EventType

Дескриптор типа входного события и записи события, хранящейся в элементе **события**.

Этот элемент может быть одним из следующих значений.

Значение	Значение
FOCUS_EVENT 0x0010	Элемент <b>события</b> содержит структуру <a href="#">FOCUS_EVENT_RECORD</a> . Эти события используются внутренне и должны игнорироваться.
KEY_EVENT 0x0001	Элемент <b>события</b> содержит структуру <a href="#">KEY_EVENT_RECORD</a> с информацией о событии клавиатуры.
MENU_EVENT 0x0008	Элемент <b>события</b> содержит структуру <a href="#">MENU_EVENT_RECORD</a> . Эти события используются внутренне и должны игнорироваться.

Значение	Значение
MOUSE_EVENT 0x0002	Элемент <b>события</b> содержит структуру <b>MOUSE_EVENT_RECORD</b> со сведениями о перемещении мыши или нажатии кнопки.
WINDOW_BUFFER_SIZE_EVENT 0x0004	Элемент <b>события</b> содержит структуру <b>WINDOW_BUFFER_SIZE_RECORD</b> с информацией о новом размере буфера экрана консоли.

## Событие

Сведения о событии. Формат этого элемента зависит от типа события, указанного членом **EventType**.

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[FOCUS\\_EVENT\\_RECORD](#)

[KEY\\_EVENT\\_RECORD](#)

[MENU\\_EVENT\\_RECORD](#)

[MOUSE\\_EVENT\\_RECORD](#)

[PeekConsoleInput](#)

[ReadConsoleInput](#)

[WriteConsoleInput](#)

# Структура KEY\_EVENT\_RECORD

Статья • 23.10.2023

Описывает событие ввода клавиатуры в структуре консоли [INPUT\\_RECORD](#).

## Синтаксис

C

```
typedef struct _KEY_EVENT_RECORD {
    BOOL bKeyDown;
    WORD wRepeatCount;
    WORD wVirtualKeyCode;
    WORD wVirtualScanCode;
    union {
        WCHAR UnicodeChar;
        CHAR AsciiChar;
    } uChar;
    DWORD dwControlKeyState;
} KEY_EVENT_RECORD;
```

## Участники

### bKeyDown

Если клавиша нажимается, этот элемент имеет значение **TRUE**. В противном случае этот элемент имеет значение **FALSE** (ключ освобождается).

### wRepeatCount

Число повторов, указывающее, что ключ удерживается. Например, если ключ удерживается, вы можете получить пять событий с этим членом, равным 1, одному событию с этим членом равно 5 или нескольким событиям с этим элементом больше или равно 1.

### wVirtualKeyCode

Код виртуального ключа, определяющий заданный ключ независимо от устройства.

### wVirtualScanCode

Код виртуальной проверки заданного ключа, представляющий значение, зависящее от устройства, созданное оборудованием клавиатуры.

### Uchar

Объединение следующих членов.

## ЮникодChar

Переведенный символ Юникода.

## AsciiChar

Переведенный символ ASCII.

## dwControlKeyState

Состояние ключей управления. Этот элемент может быть одним или несколькими из следующих значений.

Значение	Значение
CAPSLOCK_ON 0x0080	Индикатор CAPS LOCK включен.
ENHANCED_KEY 0x0100	Ключ улучшен. См . <a href="#">примечания</a> .
LEFT_ALT_PRESSED 0x0002	Нажата левая клавиша ALT.
LEFT_CTRL_PRESSED 0x0008	Нажата левая клавиша CTRL.
NUMLOCK_ON 0x0020	Индикатор NUM LOCK включен.
RIGHT_ALT_PRESSED 0x0001	Нажимается справа клавиша ALT.
RIGHT_CTRL_PRESSED 0x0004	Нажата правая клавиша CTRL.
SCROLLLOCK_ON 0x0040	Индикатор SCROLL LOCK включен.
SHIFT_PRESSED 0x0010	Клавиша SHIFT нажимается.

## Замечания

Расширенные ключи для клавиатуры IBM® 101 и 102—это INS, DEL, HOME, END, PAGE UP, PAGE DOWN и клавиши направления в кластерах слева от клавиатуры; и клавиши деления (/) и ВВОД на клавиатуре.

События ввода клавиатуры создаются при нажатии или освобождении любого ключа, включая клавиши управления. Однако клавиша ALT при нажатии и освобождении без объединения с другим символом имеет особое значение для системы и не передается в приложение. Кроме того, сочетание клавиш CTRL+C не передается, если входной дескриптор находится в обработанном режиме (ENABLE\_PROCESSED\_INPUT).

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[PeekConsoleInput](#)

[ReadConsoleInput](#)

[WriteConsoleInput](#)

[INPUT\\_RECORD](#)

# Структура MENU\_EVENT\_RECORD

Статья • 13.12.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Описывает событие меню в структуре консоли [INPUT\\_RECORD](#). Эти события используются внутренне и должны игнорироваться.

## Синтаксис

C

```
typedef struct _MENU_EVENT_RECORD {  
    UINT dwCommandId;  
} MENU_EVENT_RECORD, *PMENU_EVENT_RECORD;
```

## Участники

**dwCommandId**

Зарезервировано.

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента Windows 2000 Professional [только классические

	приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[INPUT\\_RECORD](#)

# Структура MOUSE\_EVENT\_RECORD

Статья • 23.10.2023

## ⓘ Важно!

В этом документе описаны функции платформы консоли, которые больше не являются частью стратегии развития экосистемы. Мы не рекомендуем использовать это содержимое в новых продуктах, но мы будем продолжать поддерживать существующие использования для неопределенного будущего. Наше предпочтительное современное решение ориентировано на [последовательности виртуальных терминалов](#) для обеспечения максимальной совместимости в кроссплатформенных сценариях. Дополнительные сведения об этом решении по проектированию можно найти в классической [консоли и в документе виртуального терминала](#).

Описывает событие ввода мыши в структуре консоли [INPUT\\_RECORD](#).

## Синтаксис

```
C

typedef struct _MOUSE_EVENT_RECORD {
    COORD dwMousePosition;
    DWORD dwButtonState;
    DWORD dwControlKeyState;
    DWORD dwEventFlags;
} MOUSE_EVENT_RECORD;
```

## Участники

### dwMousePosition

[Структура COORD](#), содержащая расположение курсора с точки зрения координат символьной ячейки буфера экрана консоли.

### dwButtonState

Состояние кнопок мыши. Наименьший значительный бит соответствует самой левой кнопке мыши. Следующий наименьший значимый бит соответствует самой правой кнопке мыши. Следующий бит указывает кнопку мыши рядом с левой кнопкой мыши. Затем биты соответствуют слева направо на кнопки мыши. Бит 1, если кнопка была нажата.

Для первых пяти кнопок мыши определены следующие константы.

Значение	Значение
FROM_LEFT_1ST_BUTTON_PRESSED 0x0001	Самая левая кнопка мыши.
FROM_LEFT_2ND_BUTTON_PRESSED 0x0004	Вторая кнопка слева.
FROM_LEFT_3RD_BUTTON_PRESSED 0x0008	Третья кнопка слева.
FROM_LEFT_4TH_BUTTON_PRESSED 0x0010	Четвертая кнопка слева.
RIGHTMOST_BUTTON_PRESSED 0x0002	Самая правая кнопка мыши.

### **dwControlKeyState**

Состояние ключей управления. Этот элемент может быть одним или несколькими из следующих значений.

Значение	Значение
CAPSLOCK_ON 0x0080	Индикатор CAPS LOCK включен.
ENHANCED_KEY 0x0100	Ключ улучшен. См . <a href="#">примечания</a> .
LEFT_ALT_PRESSED 0x0002	Нажата левая клавиша ALT.
LEFT_CTRL_PRESSED 0x0008	Нажата левая клавиша CTRL.
NUMLOCK_ON 0x0020	Индикатор NUM LOCK включен.
RIGHT_ALT_PRESSED 0x0001	Нажимается справа клавиша ALT.
RIGHT_CTRL_PRESSED 0x0004	Нажата правая клавиша CTRL.
SCROLLLOCK_ON 0x0040	Индикатор SCROLL LOCK включен.
SHIFT_PRESSED 0x0010	Клавиша SHIFT нажимается.

### **dwEventFlags**

Тип события мыши. Если это значение равно нулю, оно указывает на нажатие или освобождение кнопки мыши. В противном случае этот элемент является одним из следующих значений.

Значение	Значение
DOUBLE_CLICK 0x0002	Второй щелчок (нажатие кнопки) при двойном щелчке. Первый щелчок возвращается как обычное событие нажатия кнопки.
MOUSE_HWHEELED 0x0008	Горизонтальное колесо мыши было перемещено.

<b>Значение</b>	<b>Значение</b> Если высокое слово элемента <b>dwButtonState</b> содержит положительное значение, колесо было повернуто справа. В противном случае колесо было повернуто влево.
MOUSE_MOVED 0x0001 0x0001	Произошло изменение положения мыши.
MOUSE_WHEELED 0x0004	Вертикальное колесо мыши было перемещено.  Если высокое слово элемента <b>dwButtonState</b> содержит положительное значение, колесо было повернуто вперед, от пользователя. В противном случае колесо было повернуто назад, в сторону пользователя.

## Замечания

События мыши помещаются в входной буфер, когда консоль находится в режиме мыши (**ENABLE\_MOUSE\_INPUT**).

События мыши создаются всякий раз, когда пользователь перемещает мышь, или нажимает или освобождает одну из кнопок мыши. События мыши помещаются в входной буфер консоли, только если группа консоли имеет фокус клавиатуры, а курсор находится в границах окна консоли.

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Требования

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Заголовок	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[COORD](#)

**INPUT\_RECORD**

[PeekConsoleInput](#)

[ReadConsoleInput](#)

[WriteConsoleInput](#)

# Структура FOCUS\_EVENT\_RECORD

Статья • 13.12.2023

Описывает событие фокуса в структуре консоли [INPUT\\_RECORD](#). Эти события используются внутренне и должны игнорироваться.

## Синтаксис

C

```
typedef struct _FOCUS_EVENT_RECORD {
    BOOL bSetFocus;
} FOCUS_EVENT_RECORD;
```

## Участники

### bSetFocus

Зарезервировано.

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[INPUT\\_RECORD](#)

# Структура `WINDOW_BUFFER_SIZE_RECORD`

Статья • 13.12.2023

Описывает изменение размера буфера экрана консоли.

## Синтаксис

```
C

typedef struct _WINDOW_BUFFER_SIZE_RECORD {
    COORD dwSize;
} WINDOW_BUFFER_SIZE_RECORD;
```

## Участники

### `dwSize`

[Структура `COORD`](#), содержащая размер буфера экрана консоли в столцах и строках ячейки символов.

## Замечания

События размера буфера помещаются в входной буфер, когда консоль находится в режиме с поддержкой окна ([ENABLE\\_WINDOW\\_INPUT](#)).

## Примеры

Пример см. в статье о [чтении событий входного буфера](#).

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
----------------------------	--

Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	WinConTypes.h (через WinCon.h, включите Windows.h)

## См. также

[COORD](#)

[INPUT\\_RECORD](#)

[ReadConsoleInput](#)

# Консоль WinEvents

Статья • 13.12.2023

## ⓘ Важно!

WinEvents являются частью устаревшей **платформы специальных возможностей Microsoft Active**. Разработка с помощью этих событий настоятельно не рекомендуется использовать **платформу Microsoft модель автоматизации пользовательского интерфейса**, которая предоставляет более надежный и полный набор интерфейсов для приложений специальных возможностей и автоматизации для взаимодействия с консолью.

## ⚠ Предупреждение

Регистрация этих событий — это глобальное действие, которое значительно влияет на производительность всех приложений командной строки, работающих в системе одновременно, включая службы и фоновые служебные программы. Платформа **Microsoft модель автоматизации пользовательского интерфейса** является сеансом консоли и преодолевает это ограничение.

Следующие константы событий используются в параметре *события функции обратного вызова WinEventProc*. Дополнительные сведения см. в статье [WinEvents](#).

 [Развернуть таблицу](#)

Константа/значение	Description
EVENT_CONSOLE_CARET 0x4001	Элемент управления консолью перемещен. Параметр <i>idObject</i> является одним или несколькими из следующих значений: <b>CONSOLE_CARET_SELECTION</b> или <b>CONSOLE_CARET_VISIBLE</b> . Параметр <i>idChild</i> — это <b>структура COORD</b> , указывающая текущую позицию курсора.
EVENT_CONSOLE_END_APPLICATION 0x4007	Процесс консоли завершился. Параметр <i>idObject</i> содержит идентификатор процесса завершенного процесса.
EVENT_CONSOLE_LAYOUT 0x4005	Макет консоли изменился.

Константа/значение	Description
EVENT_CONSOLE_START_APPLICATION 0x4006	Начался новый процесс консоли. Параметр <i>idObject</i> содержит идентификатор процесса только что созданного процесса. Если приложение является 16-разрядным приложением, параметр <i>idChild</i> CONSOLE_APPLICATION_16BIT и <i>idObject</i> является идентификатором процесса сеанса NTVDM, связанного с консолью.
EVENT_CONSOLE_UPDATE_REGION 0x4002	Несколько символов изменились. Параметр <i>idObject</i> — это <a href="#">структура COORD</a> , указывающая начало измененного региона. Параметр <i>idChild</i> — это <a href="#">структура COORD</a> , указывающая конец измененного региона.
EVENT_CONSOLE_UPDATE_SCROLL 0x4004	Консоль прокрутила. Параметр <i>idObject</i> — это <i>горизонтальное</i> расстояние, которое прокрутила консоль. Параметр <i>idChild</i> — это <i>вертикальное</i> расстояние, которое прокрутила консоль.
EVENT_CONSOLE_UPDATE_SIMPLE 0x4003	Один символ изменился. Параметр <i>idObject</i> — это <a href="#">структура COORD</a> , указывающая измененный символ. Параметр <i>idChild</i> указывает символ в нижнем слове и <a href="#">атрибуты символов</a> в высоком слове.

## Requirements

 [Развернуть таблицу](#)

Минимальная версия клиента	Windows 2000 Professional [только классические приложения]
Минимальная версия сервера	Windows 2000 Server [только классические приложения]
Верхний колонтитул	Winuser.h