Tools for Changing Boot Options for Driver Testing and Debugging

8/23/2022 • 2 minutes to read • Edit Online

To test and debug drivers on a Microsoft Windows operating system, you must enable and configure features that are established when the operating system loads. The settings for these features are included in the *boot options*--values that determine how the boot loader loads and configures the operating system and other bootable programs and devices.

This section explains how to add, delete, and change boot options to create new load configurations for an operating system and how to use the boot entry parameters to customize a load configuration for driver testing and debugging.

By editing boot options, you can:

- Enable and configure debugging
- Load a particular kernel or hardware abstraction layer (HAL) file
- Limit the physical memory available to Windows
- Enable, disable, and configure Physical Address Extension (PAE) on 32-bit versions of Windows
- Reapportion virtual address space between user-mode and kernel-mode components (3GB) to test a driver in a very small kernel-mode address space
- Enable and configure Data Execution Prevention (/noexecute)
- Designate ports for Emergency Management Services (EMS) console redirection on headless servers
- Display the names of drivers as they load

This section includes:

- Overview of Boot Options in Windows.
- Boot Options Identifiers
- Editing Boot Options
- BCD Boot Options Reference
- Using Boot Parameters
- BCD Boot Options Reference
- Boot Options in Previous Versions of Windows

Overview of Boot Options in Windows

8/23/2022 • 2 minutes to read • Edit Online

The Windows boot loader architecture includes a firmware-independent boot configuration and storage system called *Boot Configuration Data* (BCD) and a boot option editing tool, BCDEdit (BCDEdit.exe). During development, you can use BCDEdit to configure boot options for debugging, testing, and troubleshooting your driver on computers running Windows 11, Windows 10, Windows 8, Windows Server 2012, Windows 7, and Windows Server 2008.

Caution

Administrative privileges are required to use BCDEdit to modify BCD. Changing some boot entry options using BCDEdit could render your computer inoperable. As an alternative, use the System Configuration utility (MSConfig.exe) to change boot settings.

Boot Loading Architecture

Windows includes boot loader components that are designed to load Windows quickly and securely. It uses three components:

- Windows Boot Manager
- Windows operating system loader
- Windows resume loader

In this configuration, the Windows Boot Manager is generic and unaware of the specific requirements for each operating system while the system-specific boot loaders are optimized for the system that they load.

When a computer with multiple boot entries includes at least one entry for Windows, the Windows Boot Manager, starts the system and interacts with the user. It displays the boot menu, loads the selected system-specific boot loader, and passes the boot parameters to the boot loader.

The boot loaders reside in each Windows partition. Once selected, the boot loaders take over the boot process and load the operating system in accordance with the selected boot parameters.

For additional detail on the Windows startup process refer to *Windows Internals*, published by Microsoft Press.

Boot Configuration Data

Windows boot options are stored in the Boot Configuration Data (BCD) store on BIOS-based and EFI-based computers.

BCD provides a common, firmware-independent boot option interface. It is more secure than previous boot option storage configurations, and lets Administrators assign rights for managing boot options. BCD is available at run time and during all phases of system setup.

You can manage BCD remotely and manage BCD when the system boots from media other than the media on which the BCD store resides. This feature is can be used for debugging and troubleshooting, especially when a BCD store must be restored while running Startup Repair, from USB-based storage media, or even remotely.

The BCD store, with its object-and-element architecture, uses GUIDs and names such as "Default" to identify boot-related applications.

BCD includes its own set of boot options. For more information about these boot options, see BCD Boot Options

Reference.

Editing Boot Options

To edit boot options in Windows, one option is to use BCDEdit (BCDEdit.exe), a tool included in Windows.

To use BCDEdit, you must be a member of the Administrators group on the computer.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

You can also use the System Configuration utility (MSConfig.exe) to change boot settings. In addition, many options can be set using the Advanced Startup settings UI.

To change boot options programmatically in Windows, use the Windows Management Instrument (WMI) interface to boot options. This BCD WMI interface is the best method to programmatically change the boot options. For information about the BCD WMI interface, see Boot Configuration Data WMI Provider in the Windows SDK documentation.

Related topics

- BCD Edit Options Reference
- Editing Boot Options
- Using Boot Parameters

Boot Options Identifiers

8/23/2022 • 3 minutes to read • Edit Online

Many of the bcdedit commands require identifiers. An identifier uniquely identifies entries contained in the boot setting store.

Use bcdedit /enum to display the identifers.

```
C:\>bcdedit /enum

Windows Boot Manager
--------------------
identifier              {bootmgr}

...

Windows Boot Loader
-------------------
identifier              {current}
```

Several entries can be identified by well-known identifiers. If an entry has a well-known identifier, bcdedit displays it in output unless the /v command-line switch is used. For more information, run "bcdedit /? /v".

The common well-known identifiers are often used:

IDENTIFIER	DESCRIPTION
{default}	Specifies a virtual identifier that corresponds to the boot manager default application entry.
{current}	Specifies a virtual identifier that corresponds to the operating system boot application entry for the operating system that is currently running.
{bootmgr}	Specifies the Windows boot manager application entry.

These common well-known identifiers can be inherited by any boot application entry:

IDENTIFIER	DESCRIPTION
{globalsettings}	Contains the collection of global settings that should be inherited by all boot application entries.
{bootloadersettings}	Contains the collection of global settings that should be inherited by all boot loader application entries.

These well-known identifiers are also available for use:

IDENTIFIER	DESCRIPTION
{dbgsettings}	Contains the global debugger settings that can be inherited by any boot application entry.

IDENTIFIER	DESCRIPTION
{hypervisorsettings}	Contains the hypervisor settings that can be inherited by any OS loader entry.
{emssettings}	Contains the global Emergency Management Services settings that can be inherited by any boot application entry.
{resumeloadersettings}	Contains the collection of global settings that should be inherited by all Windows resume from hibernation application entries.
{badmemory}	Contains the global RAM defect list that can be inherited by any boot application entry.
{memdiag}	Specifies the memory diagnostic application entry.
{ramdiskoptions}	Contains the additional options required by the boot manager for RAM disk devices.

These well-known identifiers are used with earlier versions of Windows:

IDENTIFIER	DESCRIPTION
{ntldr}	Specifies a OS loader (Ntldr) that can be used to start operating systems earlier than Windows Vista.
{fwbootmgr}	Specifies the firmware boot manager entry, specifically on systems that implement the Extensible Firmware Interface (EFI) specification.

Boot Option Inheritance

Some boot settings can be inherited. This allows for groups of settings to be used in different boot scenarios, for example when resuming from hibernation.

Use the bcdedit command /enum option to display information about any identifier.

In the example below, displaying information on the {current} identifier shows that it inherits the {bootloadersettings}

```
C:\>bcdedit /enum {current}

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \WINDOWS\system32\winload.exe
description             Windows 10
locale                  en-US
inherit                 {bootloadersettings}
...
```

Use the bcdedit /enum command to see which settings are inherited.

In the example below, {globalsettings}, inherits whatever is set in {dbgsettings}, {emssettings} and {badmemory}.

```
C:\>bcdedit /enum {globalsettings}

Global Settings
---------------
identifier              {globalsettings}
inherit                 {dbgsettings}
                        {emssettings}
                        {badmemory}
```

Use the inherit option with bcdedit /enum to display information about inheritance.

In the example below, the {bootloadersettings} inherits {globalsettings} and the {hypervisorsettings} and the {resumeloadersettings} inherit the {globalsettings}.

```
C:\>bcdedit /enum inherit

...

Boot Loader Settings
--------------------
identifier              {bootloadersettings}
inherit                 {globalsettings}
                        {hypervisorsettings}


Resume Loader Settings
----------------------
identifier              {resumeloadersettings}
inherit                 {globalsettings}

...
```

Use the bcdedit /enum all command to see all of the settings.

```
C:\>bcdedit /enum all

Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=\Device\HarddiskVolume1
description             Windows Boot Manager

...
```

GUIDs and Identifiers

An identifier uses a globally unique identifier, or GUID. A GUID has the following format, where each "x" represents a hexadecimal digit. Because working with GUIDs is error prone, it is recommended to use the english identifier name, such as {current} to work with the current boot information configured for Windows.

```
{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}
```

For example:

```
{d2b69192-8f14-11da-a31f-ea816ab185e9}
```

The position of the dashes (-) and the braces at the beginning and end of the GUID are required.

Use bcdedit /enum /v to display GUIDs associated with identifiers.

```
C:\>bcdedit /enum /v

Windows Boot Manager
--------------------
identifier              {9dea862c-5cdd-4e70-acc1-f32b344d4795}
device                  partition=\Device\HarddiskVolume1
description             Windows Boot Manager
locale                  en-US
inherit                 {7ea2e1ac-2e61-4728-aaa3-896d9d0a9f0e}
```

BCDEdit Options Reference

8/23/2022 • 2 minutes to read • Edit Online

Boot entry parameters, or *boot parameters*, are optional, system-specific settings that represent configuration options. You can add boot parameters to a boot entry for an operating system. They are stored in a boot configuration data (BCD) store.

This section describes the boot options for supported versions of Windows that are related to developing, testing, and debugging drivers on computers with x86-based and x64-based processors. You can add these parameters to the boot entries for Windows operating systems.

Caution

Administrative privileges are required to use BCDEdit to modify BCD. Changing some boot entry options using the **BCDEdit /set** command could render your computer inoperable. As an alternative, use the System Configuration utility (MSConfig.exe) to change boot settings. For more information, see How to open MSConfig in Windows 10.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

In this section

TOPIC	DESCRIPTION
BCDEdit /bootdebug	The /bootdebug boot option enables or disables boot debugging of the current or specified Windows operating system boot entry.
BCDEdit /bootsequence	Sets the one-time boot sequence for the boot manager.
BCDEdit /dbgsettings	The /dbgsettings option sets or displays the current global debugger settings for the computer. To enable or disable the kernel debugger, use the BCDEdit /debug option.
BCDEdit /debug	The /debug boot option enables or disables kernel debugging of the Windows operating system associated with the specified boot entry or the current boot entry.
BCDEdit /default	Sets the default entry that the boot manager will use.
BCDEdit /deletevalue	The /deletevalue option deletes or removes a boot entry option (and its value) from the Windows boot configuration data store (BCD). Use the BCDEdit /deletevalue command to remove options that were added using BCDEdit /set command. You might need to remove boot entry options when you are testing and debugging your driver.
BCDEdit /displayorder	Sets the order in which the boot manager displays the multiboot menu.

TOPIC	DESCRIPTION
BCDEdit /ems	The /ems option enables or disables Emergency Management Services (EMS) for the specified operating system boot entry.
BCDEdit /emssettings	The /emssettings option sets the global Emergency Management Services (EMS) settings for the computer. To enable or disable EMS, use the /ems option. The /emssettings option does not enable or disable EMS for any boot entry.
BCDEdit /enum	The /enum command lists entries in boot configuration data (BCD) store.
BCDEdit /event	The /event command enables or disables the remote event logging for the specified boot entry.
BCDEdit /hypervisorsettings	The /hypervisorsettings option sets or displays the hypervisor debugger settings for the system.
BCDEdit /set	The BCDEdit /set command sets a boot entry option value in the Windows boot configuration data store (BCD). Use the BCDEdit /set command to configure specific boot entry elements, such as kernel debugger settings, memory options, or options that enable test-signed kernel-mode code or load alternate hardware abstraction layer (HAL) and kernel files. To remove a boot entry option, use the BCDEdit /deletevalue command.
BCDEdit /timeout	Sets the boot manager time-out value.
BCDEdit /tooldisplayorder	Sets the order in which the boot manager displays the tools menu.

See also

Adding Boot Entries

BCDEdit /bootsequence

8/23/2022 • 2 minutes to read • Edit Online

The **/bootsequence** command sets the one-time boot sequence to be used by the boot manager.

```
bcdedit /bootsequence <id> [...] [ /addfirst | /addlast | /remove ]
```

Parameters

<id> [...]

Specifies a list of data store identifiers that make up the boot sequence. You must specify at least one identifier and must separate identifiers by spaces. For more information about identifiers, run "bcdedit /? ID".

/addfirst

Adds the specified entry identifier to the top of the boot sequence. If this switch is specified, only a single identifier may be specified. If the identifier is already in the list, it is moved to the top of the list.

/addlast

Adds the specified entry identifier to the end of the boot sequence. If this switch is specified, only a single identifier may be specified. If the identifier is already in the list, it is moved to the end of the list.

/remove

Removes the specified entry identifier from the boot sequence. If this switch is specified, only a single entry identifier may be specified. If the identifier is not in the list then the operation has no effect. If the last entry is being removed, then the boot sequence value is deleted from the boot manager entry.

Examples

The following command sets two OS entries and the NTLDR based OS loader in the boot manager one-time boot sequence:

```
bcdedit /bootsequence {802d5e32-0784-11da-bd33-000476eba25f} {cbd971bf-b7b8-4885-951a-fa03044f5d71} {ntldr}
```

The following command adds the specified OS entry to the end of the boot manager one-time boot sequence:

```
bcdedit /bootsequence {802d5e32-0784-11da-bd33-000476eba25f} /addlast
```

BCDEdit /bootdebug

8/23/2022 • 2 minutes to read • Edit Online

The **/bootdebug** boot option enables or disables boot debugging of the current or specified Windows operating system boot entry.

```
bcdedit /bootdebug [{ID}] { on | off }
```

Parameters

{*ID*}

The **{*ID*}** is the ID that is associated with the boot entry, such as {DEFAULT} for the default OS boot entry. If you do not specify an **{*ID*}**, the command modifies the operating system that is currently active. For more information about working with boot entry identifiers, see Boot Options Identifiers.

on

Enables boot debugging of the specified boot entry. If a boot entry is not specified, boot debugging is enabled for the current operating system.

off

Disables boot debugging of the specified boot entry. If a boot entry is not specified, boot debugging is disabled for the current operating system.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Comments

The **/bootdebug** boot option enables boot debugging for a specific boot entry. Use the **/dbgsettings** option to configure the type of debugging connection (*debugtype*) to use and the connection parameters.

The default values for the dbgsettings are shown in the following table.

DBGSETTING PARAMETER	DEFAULT VALUE
debugtype	Local
debugstart	Active
noumex	Yes

The following command enables boot debugging of the Windows boot loader for the current operating system. The Windows boot loader (Winload.exe) controls the load UI and loads the kernel boot drivers.

```
bcdedit /bootdebug on
```

The following command disables boot debugging of Windows Boot Manager (Bootmgr.exe). Windows Boot

Manager selects which operating system will start, and then loads the Windows boot loader.

```
bcdedit /bootdebug {bootmgr} off
```

In the following example, the commands enable debugging of Windows Boot Manager, the boot loader, and then kernel debugging of the operating system. This combination allows debugging at every stage of startup. If this combination is used, the target computer will break into the debugger three times: when Windows Boot Manager loads, when the boot loader loads, and when the operating system starts up.

```
bcdedit /bootdebug {bootmgr} on
bcdedit /bootdebug on
bcdedit /debug on
```

For general information about Windows debugging tools, see Windows Debugging.

BCDEdit /dbgsettings

8/23/2022 • 7 minutes to read • Edit Online

The **/dbgsettings** option sets or displays the current global debugger settings for the computer. To enable or disable the kernel debugger, use the **BCDEdit /debug** option.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

```
bcdedit /dbgsettings NET HOSTIP:ip PORT:port [KEY:key] [nodhcp] [newkey] [/start startpolicy] [/noumex]

bcdedit /dbgsettings LOCAL [/start startpolicy] [/noumex]

bcdedit /dbgsettings SERIAL [DEBUGPORT:port] [BAUDRATE:baud] [/start startpolicy] [/noumex]

bcdedit /dbgsettings USB [TARGETNAME:targetname] [/start startpolicy] [/noumex]

bcdedit /dbgsettings 1394 [CHANNEL:channel] [/start startpolicy] [/noumex] NOTE: The 1394 TRANSPORT IS
DEPRECATED
```

Parameters

NET

Specifies that the target machine and the host machine will use an Ethernet network connection for debugging. When this option is used, the **HOSTIP** and **PORT** parameters must be included as well. The target computer must have a network adapter that is supported by Debugging Tools for Windows.

HOSTIP:*ip*
For network debugging, specifies the IP address of the host debugger.

KEY:*key*
For network debugging, specifies the key with which to encrypt the connection. [0-9] and [a-z] allowed only. Do not specify this parameter if you have specified the **newkey** parameter.

PORT:*port*
For network debugging, specifies the port to communicate with on the host debugger. Should be 49152 or higher.

newkey
For network debugging specifies that a new encryption key should be generated for the connection. Do not specify this parameter if you have specified a **KEY** parameter.

nodhcp

Setting *nodhcp* prevents use of DHCP to obtain the target IP address. This option is rarely required as even small routers provide support for DHCP. The *nodhcp* option should only be used if you know that there are no DHCP servers on the network. In most situations, the KDNET transport works best when this option is not set, and DHCP is enabled.

busparams = *Bus.Device.Function* Specifies the target controller. *Bus* specifies the bus number, *Device* specifies

the device number, and *Function* specifies the function number.

To specify the bus parameters, Open Device Manager, and locate the network adapter that you want to use for debugging. Open the property page for the network adapter, and make a note of the bus number, device number, and function number. These values are displayed in Device Manager under *Location* on the *General* tab. In an elevated Command Prompt Window, enter the following command, where b, d, and f are the bus, device and function numbers in decimal format:

```
bcdedit /set "{dbgsettings}" busparams b.d.f
```

If you are manually configuring a debugger connection, you must specify the bus parameters. For more information, see Setting Up KDNET Network Kernel Debugging Manually and Setting Up Kernel-Mode Debugging over a USB 3.0 Cable Manually.

Examples

The following command configures the target computer to use an Ethernet connection for debugging and specifies the IP address of the host computer. The command also specifies a port number that the host computer can use to connect to the target computer.

```
bcdedit /dbgsettings net hostip:10.125.5.10 port:50000
```

The following command sets the global debugger settings to network debugging using IPv6 with a debugger host at 2001:48:d8:2f:5e:c0:42:28:4f5b communicating on port 50000:

```
bcdedit /dbgsettings NET HOSTIPV6:2001:48:d8:2f:5e:c0:42:28:4f5b PORT:50000
```

> **IMPORTANT**
> Setting up a network debugging manually is a complex and error prone process. To set up network debugging automatically, see Setting Up KDNET Network Kernel Debugging Automatically. Using the KDNET utility is **strongly** recommended for all debugger users.

For more information on manual setup, see Setting Up Kernel-Mode Debugging over a Network Cable Manually.

LOCAL

The **LOCAL** option sets the global debugging option to local debugging. This is kernel-mode debugging on a single computer. In other words, the debugger runs on the same computer that is being debugged. With local debugging you can examine state, but not break into kernel mode processes that would cause the OS to stop running.

Example

The following command sets the global debugger settings to local debugging.

```
bcdedit /dbgsettings LOCAL
```

The LOCAL option is available in Windows 8.0 and Windows Server 2012 and later.

For information on setting up local kernel mode debugging manually, see Setting Up Local Kernel Debugging of

a Single Computer Manually.

SERIAL

Specifies that the target machine and the host machine will use a serial connection for debugging. When this option is used, the **DEBUGPORT** and **BAUDRATE** parameters should be specified.

BAUDRATE:*baud*
Specifies the baud rate to use. This parameter is optional. Valid values for *baud* are 9600, 19200, 38400, 57600, and 115200. The default baud rate is 115200 bps.

DEBUGPORT:*port*
Specifies the serial port to use as the debugging port. This is an optional setting. The default port is **1** (COM 1).

Example

The following command configures the target computer to use a serial connection for debugging. The command also specifies that the debugging connection will use COM1 and a baud rate of 115,200.

```
bcdedit /dbgsettings serial debugport:1 baudrate:115200
```

For more information, see Setting Up Kernel-Mode Debugging over a Serial Cable Manually.

USB

Specifies that the target machine and the host machine will use a USB 2.0 or USB 3.0 connection for debugging. When this option is used, the **TARGETNAME** parameter must be included as well.

TARGETNAME: *targetname*
Specifies a string value to use for the target name. Note that TargetName does not have to be the official name of the target computer; it can be any string that you create as long as it meets these restrictions:

- The string must not contain “debug” anywhere in the TargetName in any combination of upper or lower case. For example if you use “DeBuG” or "DEBUG" anywhere in your targetname, debugging will not work correctly.
- The only characters in the string are the hyphen (-), the underscore(_), the digits 0 through 9, and the letters A through Z (upper or lower case).
- The maximum length of the string is 24 characters.

Example

The following command configures the target computer to use USB connection for debugging. The command also specifies a target name that the host computer can use to connect to the target computer.

```
bcdedit /dbgsettings usb targetname:myTarget
```

For more information, see:

- Setting Up Kernel-Mode Debugging over a USB 3.0 Cable Manually
- Setting Up Kernel-Mode Debugging over a USB 2.0 Cable Manually

1394

IMPORTANT

The 1394 transport is available for use in Windows 10, version 1607 and earlier. It is not available in later versions of Windows. You should transition your projects to other transports, such as KDNET using Ethernet. For more information about that transport, see Setting Up KDNET Network Kernel Debugging Automatically.

Specifies that the target machine and the host machine will use an IEEE 1394 (FireWire) connection for debugging. When this option is used, the **CHANNEL** parameter can be included as well.

CHANNEL:*channel*
(Only used when the connection type is **1394**.) Specifies the 1394 channel to use. The value for *channel* must be a decimal integer between 0 and 62, inclusive, and must match the channel number used by the host computer. The channel specified in this parameter does not depend on the physical 1394 port chosen on the adapter. The default value for *channel* is 0.

For more information, see Setting Up Kernel-Mode Debugging over a 1394 Cable Manually.

General Debugger Settings

/start *startpolicy*
This option specifies the debugger start policy. The following table shows the options for the *startpolicy*.

OPTION	DESCRIPTION
ACTIVE	Specifies that the kernel debugger is active.
AUTOENABLE	Specifies that the kernel debugger is enabled automatically when an exception or other critical event occurs. Until then, the debugger is active but is disabled.
DISABLE	Specifies that the kernel debugger is enabled when you type kdbgctrl to clear the enable block. Until then, the debugger is active but is disabled.

If a start policy is not specified, ACTIVE is the default.

/noumex
Specifies that the kernel debugger ignores user-mode exceptions. By default, the kernel debugger breaks for certain user-mode exceptions, such as STATUS_BREAKPOINT and STATUS_SINGLE_STEP. The **/noumex** parameter is effective only when there is no user-mode debugger attached to the process.

Comments

The **/dbgsettings** option configures the debugging settings, but does not enable debugging. You must use the **/debug** option to enable debugging for a specific boot entry. If there are no debugging settings specified for a particular boot entry, the default debug settings are used.

The default values for the dbgsettings are shown in the following table.

DBGSETTING PARAMETER	DEFAULT VALUE
debugtype	Local
debugstart	Active
noumex	Yes

DBGSETTING PARAMETER	DEFAULT VALUE

See also

For information about Windows debugging tools, see Windows Debugging.

For information about setting up and configuring a kernel-mode debugging session, see Setting Up Kernel-Mode Debugging Manually and Setting Up KDNET Network Kernel Debugging Automatically.

BCDEdit /debug

8/23/2022 • 2 minutes to read • Edit Online

The **/debug** boot option enables or disables kernel debugging of the Windows operating system associated with the specified boot entry or the current boot entry.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

```
bcdedit /debug [{ID}] { on | off }
```

Parameters

{*ID*}

The **{*ID*}** is the ID that is associated with the boot entry, such as {DEFAULT} for the default OS boot entry. If you do not specify an **{*ID*}**, the command modifies the operating system that is currently active. For more information about working with boot entry identifiers, see Boot Options Identifiers.

on
Enables kernel debugging of the specified boot entry. If a boot entry is not specified, kernel debugging is enabled for the current operating system.

off
Disables kernel debugger of the specified boot entry. If a boot entry is not specified, kernel debugging is disabled for the current operating system.

Comments

The **/debug** boot option enables kernel debugging for a specific boot entry. Use the **/dbgsettings** option to configure the type of debugging connection to use and the connection parameters. If no **/dbgsettings** are specified for the boot entry, the global debug settings are used. The default values for the global settings are shown in the following table.

DBGSETTING PARAMETER	DEFAULT VALUE
debugtype	Local
debugstart	Active
noumex	Yes

The following example enables kernel debugging of the default boot entry.

```
bcdedit /debug on
```

For information about Windows debugging tools, see Windows Debugging and Setting Up KDNET Network Kernel Debugging Automatically

BCDEdit /deletevalue

8/23/2022 • 2 minutes to read • Edit Online

The **BCDEdit /deletevalue** command deletes or removes a boot entry option (and its value) from the Windows boot configuration data store (BCD). Use the **BCDEdit /deletevalue** command to remove options that were added using the **BCDEdit /set** command.

```
bcdedit  /deletevalue [{ID}] datatype
```

> **NOTE**
> Before deleting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

To delete a boot option value that you have set, use the **BCDEdit /deletevalue** command. A common scenario for using the **BCDEdit /deletevalue** command is to remove boot entry options when you are testing and debugging a driver.

For example, if you use **BCDEdit /set** to change the **groupsize** processor group option to a new value for testing purposes, you can use **BCDEdit /deletevalue** to delete the new value and revert to the default value by typing the following command. Note that you must then restart the computer for the change to take effect.

```
bcdedit /deletevalue groupsize
```

Requirements

Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

BCDEdit /set

BCDEdit /displayorder

8/23/2022 • 2 minutes to read • Edit Online

The **/displayorder** command sets the display order to be used by the boot manager.

```
bcdedit /displayorder <id> [...] [ /addfirst | /addlast | /remove ]
```

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Parameters

<id> [...] Specifies a list of identifiers that make up the display order. At least one identifier must be specified and they must be separated by spaces. For more information about identifiers, run "bcdedit /? ID".

/addfirst Adds the specified entry identifier to the top of the display order. If this switch is specified, only a single entry identifier may be specified. If the specified identifier is already in the list, it will be moved to the top of the list.

/addlast Adds the specified entry identifier to the end of the display order. If this switch is specified, only a single entry identifier may be specified. If the specified identifier is already in the list, it is moved to the end of the list.

/remove

Removes the specified entry identifier from the display order. If this switch is specified, only a single entry identifier may be specified. If the identifier is not in the list then the operation has no effect. If the last entry is being removed, then the display order value is deleted from the boot manager entry.

Examples

The following command sets two OS entries and the NTLDR based OS loader in the boot manager display order:

```
bcdedit /displayorder {802d5e32-0784-11da-bd33-000476eba25f} {cbd971bf-b7b8-4885-951a-fa03044f5d71} {ntldr}
```

The following command adds the specified OS entry to the end of the boot manager display order:

```
bcdedit /displayorder {802d5e32-0784-11da-bd33-000476eba25f} /addlast
```

BCDEdit /ems

8/23/2022 • 2 minutes to read • Edit Online

The **BCDEdit /ems** option enables or disables Emergency Management Services (EMS) for the specified operating system boot entry.

```
bcdedit /ems [{ID}] { on | off }
```

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Parameters

{ID}

The {**ID**} is the GUID that is associated with the boot entry. If you do not specify an {**ID**}, the command modifies the current operating system boot entry. If a boot entry is specified, the GUID associated with the boot entry must be enclosed in braces { }.

Comments

Use **BCDEdit /emssettings** command and its parameters to establish EMS settings for all boot entries. Then, use the **BCDEdit /ems** command to enable EMS for a particular boot entry.

EMS allows users to control particular components of a server remotely, even when the server is not connected to the network or to other standard remote-administration tools.

Example

The following command enables EMS for a boot entry with the identifier of {49916baf-0e08-11db-9af4-000bdbd316a0}.

```
bcdedit /ems {49916baf-0e08-11db-9af4-000bdbd316a0} on
```

BCDEdit /emssettings

8/23/2022 • 2 minutes to read • Edit Online

The **/emssettings** option sets the global Emergency Management Services (EMS) settings for the computer. To enable or disable EMS, use the **/ems** option. The **/emssettings** option does not enable or disable EMS for any boot entry.

Syntax

```
bcdedit /emssettings [ BIOS ] | [ EMSPORT: port | [EMSBAUDRATE: baudrate] ]
```

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Parameters

BIOS
Specifies that the system will use BIOS settings for the EMS configuration. This works only on systems that have EMS support provided by the BIOS.

EMSPORT: *port*
Specifies the serial port to use as the EMS port. This parameter should not be specified with the **BIOS** option.

EMSBAUDRATE: *baudrate*
Specifies the serial baud rate to use for EMS. This command should not be specified with the BIOS. The *baudrate* is optional, and the default is 9,600 bps.

Comments

To properly enable EMS console redirection after Windows is installed, Windows needs to know the port and transmission rate that the computer uses for out-of-band communication. Windows uses these same settings for EMS console redirection.

On computers with BIOS firmware and an ACPI Serial Port Console Redirection (SPCR) table, Windows can find the out-of-band settings established in the BIOS by reading entries in the SPCR table. On these systems, you can use the **BIOS** parameter to direct Windows to look in the SPCR table for the port settings, or you can use the **emsport:***port* and **emsbaudrate:***baudrate* parameters to override the settings in the SPCR table.

On computers that have BIOS firmware, but do not have an SPCR table, use BCDEdit and the **/emssettings** command with the **emsport:***port* parameter to specify the port and with the **emsbaudrate:***baudrate* parameter to specify the transmission rate.

On all systems, use the **BCDEdit /ems** command and specify the boot entry to enable EMS console redirection on the operating system that the boot entry loads.

The boot parameters described in this section enable EMS console redirection after Windows is installed.

For a detailed example, see Boot Parameters to Enable EMS Redirection.

BCDEdit /enum

8/23/2022 • 2 minutes to read • Edit Online

The **BCDEdit /enum** command lists entries in Boot Configuration Data (BCD) store. The /enum command is the default, so running "bcdedit" without parameters is equivalent to running "bcdedit /enum ACTIVE".

```
bcdedit [/store <filename>] /enum [<type> | <id>] [/v]
```

Caution

Administrative privileges are required to use BCDEdit to view the BCD store. Changing some boot entry options using the BCDEdit /set command could render your computer inoperable. As an alternative, use the System Configuration utility (MSConfig.exe) to change boot settings.

Parameters

<filename>

Specifies the store to be used. If this option is not specified, the system store is used. For more information, run "bcdedit /? store".

<type>

Specifies the type of entries to be listed. Can be one of the following:

ACTIVE - All entries in the boot manager display order. This is the default.

FIRMWARE - All firmware applications.

BOOTAPP - All boot environment applications.

BOOTMGR - The boot manager.

OSLOADER - All operating system entries.

RESUME - All resume from hibernation entries.

INHERIT - All inherit entries.

ALL - All entries.

<id>

Specifies the identifier of the entry to be listed. If an identifier is provided, then only the specified object will be listed. For information about identifiers, run "bcdedit /? ID".

/v

Displays entry identifiers in full, rather than using names for well-known identifiers.

Examples

The following command lists all operating system loader boot entries.

```
bcdedit /enum OSLOADER
```

The following command lists all boot manager entries.

```
bcdedit /enum BOOTMGR
```

The following command lists only the default boot entry.

```
bcdedit /enum {default}
```

The following command enables enum for a boot entry with the identifier of {49916baf-0e08-11db-9af4-000bdbd316a0}.

```
bcdedit /enum {49916baf-0e08-11db-9af4-000bdbd316a0} on
```

BCDEdit /event

8/23/2022 • 2 minutes to read • Edit Online

The **/event** enables or disables the remote event logging for the specified boot entry.

```
bcdedit /event [{ID}] { on | off }
```

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Parameters

{ID}

The {**ID**} Specifies the identifier of the entry to be modified. Only Windows boot loader entries may be specified. If not specified, {current} is used. For more information about identifiers, run "bcdedit /? ID".

Example

The following command enables remote event logging for the current Windows operating system boot entry.

```
bcdedit /event ON
```

The following command disables remote event logging for the specified operating system entry:

```
bcdedit /event {cbd971bf-b7b8-4885-951a-fa03044f5d71} OFF
```

BCDEdit /hypervisorsettings

8/23/2022 • 3 minutes to read • Edit Online

The **/hypervisorsettings** command sets or displays the hypervisor debugger settings for the system.

To set an individual hypervisor debugger setting, use `bcdedit /set {hypervisorsettings} <type> <value>`. For more information on the set command, see BCDEdit /set.

```
bcdedit /hypervisorsettings [ <debugtype> [DEBUGPORT:<port>] [BAUDRATE:<baud>] [CHANNEL:<channel>] [HOSTIP:
<ip>] [PORT:<port>] [BUSPARAMS:<Bus.Device.Function>] ]
```

<debugtype> - Specifies the type of debugger. <debugtype> can be one of NET, SERIAL or 1394 as described below.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Network Debugging

<debugtype> *NET*
Specifies an Ethernet network connection for debugging. When this option is used, the **HOSTIP** option must be also be set by specifying the IPv4 address of the host debugger.

HOSTIP: *<ip>* The IP address is only used when the **hypervisordebugtype** is **Net**. For debugging hypervisor over a network connection, specifies the IPv4 address of the host debugger.

PORT: *<port>* For network debugging, specifies the port to communicate with on the host debugger. Should be 49152 or higher.

BUSPARAMS: *<Bus.Device.Function>* Defines the PCI bus, device, and function numbers of the debugging device. For example, 0.25.0 describes the debugging device on bus 0, device 25, function 0. These values are displayed in Device Manager under *Location* on the *General* tab.

Network Debugging Example

The following command sets the hypervisor debugger settings to network debugging with a debugger host at 192.168.1.2 communicating on port 50000:

```
C:\> bcdedit /hypervisorsettings NET HOSTIP:192.168.1.2 PORT:50000 BUSPARAMS:0.25.0
Key=2steg4fzbj2sz.23418vzkd4ko3.1g34ou07z4pev.1sp3yo9yz874p
```

Use the key that is returned to connect to the target.

These network debugging settings can be modified using the BCDEdit /set command.

hypervisorhostip *IP address* (Only used when the **hypervisordebugtype** is **Net**.) For debugging hypervisor over a network connection, specifies the IPv4 address of the host debugger. For information about debugging Hyper-V, see Create a Virtual Machine with Hyper-V.

hypervisorhostport [*port*]
(Only used when the **hypervisordebugtype** is **Net**.) For network debugging, specifies the port to

communicate with on the host debugger. Should be 49152 or higher.

hypervisorbusparams *Bus.Device.Function*
Defines the PCI bus, device, and function numbers of the debugging device. For example, 0.25.0 describes the debugging device on bus 0, device 25, function 0. These values are displayed in Device Manager under *Location* on the *General* tab.

hypervisorusekey *<key>* (Only used when the **hypervisordebugtype** is **Net**.) For network debugging specifies the key with which to encrypt the connection. [0-9] and [a-z] allowed only.

hypervisordhcp [**yes** | **no**]
Controls use of DHCP by the network debugger used with the hypervisor. Setting this to **no** forces the use of Automatic Private IP Addressing (APIPA) to obtain a local link IP address.

Serial Debugging

<debugtype> *Serial*
Specifies a serial connection for debugging. When the **Serial** option is specified, you also set the **hypervisordebugport** and **hypervisorbaudrate** options.

DEBUGPORT: *<port>* For SERIAL debugging, specifies the serial port to use as the debugging port.

BAUDRATE: *<baud>* For SERIAL debugging, specifies the baud rate to be used for debugging.

```
bcdedit /set hypervisordebugtype serial
bcdedit /set hypervisordebugport 1
bcdedit /set hypervisorbaudrate 115200
bcdedit /set hypervisordebug on
bcdedit /set hypervisorlaunchtype auto
```

Serial Debugging Example

The following command displays the current hypervisor settings.

```
C:\>bcdedit /hypervisorsettings
isolatedcontext         Yes
hypervisordebugtype     Serial
hypervisordebugport     1
hypervisorbaudrate      115200
The operation completed successfully.
```

The following command sets the hypervisor debugger settings to serial debugging over COM1 at 115,200 baud.

```
bcdedit /hypervisorsettings SERIAL DEBUGPORT:1 BAUDRATE:115200
```

1394 Debugging

> **IMPORTANT**
> The 1394 transport is available for use in Windows 10, version 1607 and earlier. It is not available in later versions of Windows. You should transition your projects to other transports, such as KDNET using Ethernet.

<debugtype> *1394*
Specifies an IEEE 1394 (FireWire) connection for debugging. When this option is used, the *channel* option should also be set.

CHANNEL: *<channel>*

For 1394 debugging, specifies the 1394 channel to be used for debugging.

The following related option should be set using the BCDEdit /set command.

hypervisorbusparams *Bus.Device.Function*
Defines the PCI bus, device, and function numbers of the debugging device. For example, 1.5.0 describes the debugging device on bus 1, device 5, function 0. These values are displayed in Device Manager under *Location* on the *General* tab.

Comments

This command does not enable or disable the hypervisor debugger for any particular OS loader entry. To enable the hypervisor debugger for a particular OS loader entry, use `bcdedit /set <identifier> HYPERVISORDEBUG ON`.

For information about identifiers, run "bcdedit /? ID".

See Also

BCDEdit /set command.

BCDEdit Options Reference

BCDEdit /set

8/23/2022 • 19 minutes to read • Edit Online

The **BCDEdit /set** command sets a boot entry option value in the Windows boot configuration data store (BCD). Use the **BCDEdit /set** command to configure specific boot entry elements, such as kernel debugger settings, memory options, or options that enable test-signed kernel-mode code or load alternate hardware abstraction layer (HAL) and kernel files. To remove a boot entry option, use the **BCDEdit /deletevalue** command.

Caution

Administrative privileges are required to use BCDEdit to modify BCD. Changing some boot entry options using the **BCDEdit /set** command could render your computer inoperable. As an alternative, use the Startup settings or the System Configuration utility (MSConfig.exe) to change boot settings.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Alternatives to BCDEdit

Settings startup options

> **TIP**
> To avoid the risk associated with using BCDEdit, consider using an alternative method to perform boot configuration discussed in this section.

Startup Settings

Some common boot options such as enabling debugging mode are available in the start up options. In Windows 10, the settings can be accessed in Settings, Update and Security, select Recovery. Under Advanced startup, select Restart Now. When the PC reboots, select Startup options. Then select Troubleshoot > Advanced options > Startup Settings , then select Restart button. When the PC restarts, you will be able to set the available startup options.

System Configuration Utility

Use the System Configuration Utility (MSConfig.exe) instead of BCDEdit when possible. For more information, see How to open MSConfig in Windows 10.

Syntax

```
bcdedit  /set [{ID}] datatype value
```

Parameters

[**{ID}**]
The **{ID}** is the GUID that is associated with the boot entry. If you do not specify an **{ID}**, the command modifies the current operating system boot entry. If a boot entry is specified, the GUID associated with the boot entry must be enclosed in braces **{ }**. To view the GUID identifiers for all of the active boot entries, use the **bcdedit**

/enum command. The identifier for the current boot entry is **{current}**. For more information about this option, use the following command: **bcdedit /? ID**

> **NOTE**
> If you are using Windows PowerShell, you must use quotes around the boot entry identifier, for example: **"{49916baf-0e08-11db-9af4-000bdbd316a0}"** or **"{current}"**.

datatype value

Use the command line help to view options

Use the command line help for BCDEdit to display information available for a specific version of Windows.

```
C:\> BCDEdit /?

BCDEDIT - Boot Configuration Data Store Editor

The Bcdedit.exe command-line tool modifies the boot configuration data store.
The boot configuration data store contains boot configuration parameters and
controls how the operating system is booted. These parameters were previously
in the Boot.ini file (in BIOS-based operating systems) or in the nonvolatile
RAM entries (in Extensible Firmware Interface-based operating systems). You can
use Bcdedit.exe to add, delete, edit, and append entries in the boot
configuration data store.

For detailed command and option information, type bcdedit.exe /? <command>. For
example, to display detailed information about the /createstore command, type:

 bcdedit.exe /? /createstore

For an alphabetical list of topics in this help file, run "bcdedit /? TOPICS".
```

The following sections describe some common *datatypes* and their associated *values*.

Boot Settings

bootlog [**yes** | **no**]
Enables the system initialization log. This log is stored in the Ntbtlog.txt file in the %WINDIR% directory. It includes a list of loaded and unloaded drivers in text format.

bootmenupolicy [**Legacy** | **Standard**]
Defines the type of boot menu the system will use. ForWindows 10, Windows 8.1, Windows 8 and Windows RT the default is **Standard**. For Windows Server 2012 R2, Windows Server 2012, the default is **Legacy**. When **Legacy** is selected, the Advanced options menu (**F8**) is available. When **Standard** is selected, the boot menu appears but only under certain conditions: for example, if there is a startup failure, if you are booting up from a repair disk or installation media, if you have configured multiple boot entries, or if you manually configured the computer to use Advanced startup. When **Standard** is selected, the **F8** key is ignored during boot. Windows 8 PCs start up quickly so there isn't enough time to press **F8**. For more information, see Windows Startup Settings (including safe mode).

> **NOTE**
> The option is available starting with Windows 8 and Windows Server 2012. You can also use the **onetimeadvancedoptions** to use the Advanced options (**F8**) menu (**Legacy**) one time on the next boot.

bootstatuspolicy *policy*

Controls the boot status policy. The boot status *policy* can be one of the following:

BOOT STATUS POLICY	DESCRIPTION
DisplayAllFailures	Displays all errors if there is a failed boot, failed shutdown, or failed checkpoint. The computer will fail over to the Windows recovery environment on reboot.
IgnoreAllFailures	Ignore errors if there is a failed boot, failed shutdown, or failed checkpoint. The computer will attempt to boot normally after an error occurs.
IgnoreShutdownFailures	Only ignore errors if there is a failed shutdown. If there is a failed shutdown, the computer does not automatically fail over to the Windows recovery environment on reboot. This is the default setting for Windows 8.
IgnoreBootFailures	Only ignore errors if there is a failed boot. If there is a failed boot, the computer does not automatically fail over to the Windows recovery environment on reboot.
IgnoreCheckpointFailures	Only ignore errors if there is a failed checkpoint. If there is a failed checkpoint, the computer does not automatically fail over to the Windows recovery environment on reboot. The option is available starting with Windows 8 and Windows Server 2012.
DisplayShutdownFailures	Displays errors if there is a failed shutdown. If there is a failed shutdown, the computer will fail over to the Windows recovery environment on reboot. Ignores boot failures and failed checkpoints. The option is available starting with Windows 8 and Windows Server 2012.
DisplayBootFailures	Displays errors if there is a failed boot. If there is a failed boot, the computer will fail over to the Windows recovery environment on reboot. Ignores shutdown failures and failed checkpoints. The option is available starting with Windows 8 and Windows Server 2012.
DisplayCheckpointFailures	Displays errors if there is a failed checkpoint. If there is a failed checkpoint, the computer will fail over to the Windows recovery environment on reboot. Ignores boot and shutdown failures. The option is available starting with Windows 8 and Windows Server 2012.

quietboot [**on** | **off**]
Controls the display of a high-resolution bitmap in place of the Windows boot screen display and animation.

> **NOTE**
> Do not use the **quietboot** option in Windows 8 as it will prevent the display of bug check data in addition to all boot graphics.

sos [**on** | **off**]
Controls the display of the names of the drivers as they load during the boot process. Use **sos on** to display the names. Use **sos off** to suppress the display.

lastknowngood [**on** | **off**]
Enables boot to last known good configuration.

nocrashautoreboot [**on** | **off**]
Disables automatic restart on crash.

resumeobject (id)
Defines the identifier of the resume object that is associated with this operating system object.

safebootalternateshell [**on** | **off**]
Uses the alternate shell when booted into Safe mode.

winpe [**on** | **off**]
Enables the computer to boot to Windows PE.

onetimeadvancedoptions [**on** | **off**]
Controls whether the system boots to the legacy menu (F8 menu) on the next boot.

```
bcdedit /set {current} onetimeadvancedoptions on
```

Display Settings

bootuxdisabled [**on** | **off**]
Disables boot graphics.

graphicsmodedisabled [**on** | **off**] Indicates whether graphics mode is disabled and boot applications must use text mode display.

graphicsresolution
Defines the graphics resolution, 1024x768, 800x600,1024x600, etc.

highestmode [**on** | **off**]
Enables boot applications to use the highest graphical mode exposed by the firmware.

Hardware Abstraction Layer (HAL) & KERNEL

hal *file*
Directs the operating system loader to load an alternate HAL file. The specified file must be located in the %SystemRoot%\system32 directory.

halbreakpoint [**yes** | **no**]
Enables the special hardware abstraction layer (HAL) breakpoint.

kernel *file*
Directs the operating system loader to load an alternate kernel. The specified file must be located in the %SystemRoot%\system32 directory.

useplatformclock [**yes** | **no**]
Forces the use of the platform clock as the system's performance counter.

> **NOTE**
> This option should only be used for debugging.

forcelegacyplatform [**yes** | **no**]
Forces the OS to assume the presence of legacy PC devices like CMOS and keyboard controllers.

> **NOTE**
> This option should only be used for debugging.

tscsyncpolicy [**Default** | **Legacy** | **Enhanced**]
Controls the times stamp counter synchronization policy. This option should only be used for debugging. Can be Default, Legacy or Enhanced.

Verification Settings

testsigning [**on** | **off**]
Controls whether Windows 10, Windows 8.1, Windows 8, Windows 7, Windows Server 2008, or Windows Vista will load any type of test-signed kernel-mode code. This option is not set by default, which means test-signed kernel-mode drivers on 64-bit versions of Windows 10, Windows 8.1, Windows 8, Windows 7, Windows Server 2008, and Windows Vista will not load by default. After you run the BCDEdit command, restart the computer so that the change takes effect. For more information, see Introduction to Test-Signing

nointegritychecks [**on** | **off**] Disables integrity checks. Cannot be set when secure boot is enabled. This value is ignored by Windows 7 and Windows 8.

disableelamdrivers [**yes** | **no**]
Controls the loading of Early Launch Antimalware (ELAM) drivers. The OS loader removes this entry for security reasons. This option can only be triggered by using the F8 menu. Someone must be physically present (at the computer) to trigger this option.

> **NOTE**
> This option should only be used for debugging.

nx [**Optin** |**OptOut** | **AlwaysOn** |**AlwaysOff**]
Enables, disables, and configures Data Execution Prevention (DEP), a set of hardware and software technologies designed to prevent harmful code from running in protected memory locations. For information about DEP settings, see Data Execution Prevention.

DEP OPTION	DESCRIPTION
Optin	Enables DEP only for operating system components, including the Windows kernel and drivers. Administrators can enable DEP on selected executable files by using the Application Compatibility Toolkit (ACT).
Optout	Enables DEP for the operating system and all processes, including the Windows kernel and drivers. However, administrators can disable DEP on selected executable files by using **System** in **Control Panel**.
AlwaysOn	Enables DEP for the operating system and all processes, including the Windows kernel and drivers. All attempts to disable DEP are ignored.
AlwaysOff	Disables DEP. Attempts to enable DEP selectively are ignored. On Windows Vista, this parameter also disables Physical Address Extension (PAE). This parameter does not disable PAE on Windows Server 2008.

Processor Settings

groupsize *maxsize*
Sets the maximum number of logical processors in a single processor group, where *maxsize* is any power of 2 between 1 and 64 inclusive. Must be an integer of power of 2. By default, processor groups have a maximum size of 64 logical processors. You can use this boot configuration setting to override the size and makeup of a computer's processor groups for testing purposes. Processor groups provide support for computers with greater than 64 logical processors. This boot option is available on 64-bit versions of Windows 7 and Windows Server 2008 R2 and later versions. This boot option has no effect on the 32-bit versions of Windows 7.

Use the **groupsize** option if you want to force multiple groups and the computer has 64 or fewer active logical processors. For more information about using this option, see Boot Parameters to Test Drivers for Multiple Processor Group Support.

groupaware [**on** | **off**]
Forces drivers to be aware of multiple groups in a multiple processor group environment. Use this option to help expose cross-group incompatibilities in drivers and components. Processor groups provide support for computers with greater than 64 logical processors. This boot option is available on 64-bit versions of Windows 7 and Windows Server 2008 R2 and later versions. This boot option has no effect on the 32-bit versions of Windows 7. You can use the **groupaware** option and the **groupsize** option to test driver compatibility to function with multiple groups when computer has 64 or fewer active logical processors.

The **groupaware on** setting ensures that processes are started in a group other than group 0. This increases the chances of cross-group interaction between drivers and components. The option also modifies the behavior of the legacy functions, **KeSetTargetProcessorDpc**, **KeSetSystemAffinityThreadEx**, and **KeRevertToUserAffinityThreadEx**, so that they always operate on the highest numbered group that contains active logical processors. Drivers that call any of these legacy functions should be changed to call their group-aware counterparts (**KeSetTargetProcessorDpcEx**, **KeSetSystemGroupAffinityThread**, and **KeRevertToUserGroupAffinityThread**).

For more information about using this option, see Boot Parameters to Test Drivers for Multiple Processor Group Support.

maxgroup [**on** | **off**]
Maximizes the number of groups created in a processor group configuration. The **maxgroup on** setting assigns NUMA nodes to groups in a manner that maximizes the number of groups for a particular computer. The number of groups created is either the number of NUMA nodes the computer has, or the maximum number of groups supported by this version of Windows, whichever is smaller. The default behavior (**maxgroup off**) is to pack the NUMA nodes tightly into as few groups as possible.

Use the maxgroup option if you want to use multiple groups, the computer has 64 or fewer active logical processors, and the computer already has multiple NUMA nodes. This option can also be used to alter the default group configuration of a computer that has more than 64 logical processors.

Processor groups provide support for computers with greater than 64 logical processors. This option is available on 64-bit versions of Windows 7 and Windows Server 2008 R2 and later versions. This boot option has no effect on the 32-bit versions of Windows 7.

For more information about using this option, see Boot Parameters to Test Drivers for Multiple Processor Group Support.

onecpu [**on** | **off**]
Forces only the boot CPU to be used in a computer that has more than one logical processor. For example, the following command configures the current operating system loader to use one processor.

```
bcdedit /set onecpu on
```

Memory Related Settings

increaseuserva *Megabytes*
Specifies the amount of memory, in megabytes, for user-mode virtual address space.

On 32-bit editions of Windows, applications have 4 gigabyte (GB) of virtual address space available. The virtual address space is divided so that 2 GB is available to the application and the other 2 GB is available only to the system.

The 4-gigabyte tuning feature, enabled with the **increaseuserva** option, allows you to increase the virtual address space that is available to the application up to 3 GB, which reduces the amount available to the system to between 1 and 2 GB. The **BCEdit /set increaseuserva** *Megabytes* command can specify any value between 2048 (2 GB) and 3072 (3 GB) megabytes in decimal notation. Windows uses the remaining address space (4 GB minus the specified amount) as its kernel-mode address space.

See 4-Gigabyte Tuning (Windows) for additional information about this feature.

nolowmem [**on** | **off**] Controls the use of low memory. When **nolowmem on** is specified, this option loads the operating system, device drivers, and all applications into addresses above the 4 GB boundary, and directs Windows to allocate all memory pools at addresses above the 4 GB boundary. Note that the **nolowmem** option is ignored in Windows 8, Windows Server 2012, and later versions of Windows.

pae [**Default** | **ForceEnable** | **ForceDisable**]
Enables or disables Physical Address Extension (PAE). When PAE is enabled, the system loads the PAE version of the Windows kernel.

The **pae** parameter is valid only on boot entries for 32-bit versions of Windows that run on computers with x86-based and x64-based processors. On 32-bit versions of Windows (prior to Windows 8) , PAE is disabled by default. However, Windows automatically enables PAE when the computer is configured for hot-add memory devices in memory ranges beyond the 4 GB region, as defined by the Static Resource Affinity Table (SRAT). *Hot-add memory* supports memory devices that you can add without rebooting or turning off the computer. In this case, because PAE must be enabled when the system starts, it is enabled automatically so that the system can immediately address extended memory that is added between restarts. Hot-add memory is supported only on Windows Server 2008, Datacenter Edition; Windows Server 2008 for Itanium-Based Systems; and on the datacenter and enterprise editions of all later versions of Windows Server. Moreover, for versions of Windows prior to Windows Server 2008, hot-add memory is supported only on computers with an ACPI BIOS, an x86 processor, and specialized hardware. For Windows Server 2008 and later versions of Windows Server, it is supported for all processor architectures.

On a computer that supports hardware-enabled Data Execution Prevention (DEP) and is running a 32-bit version of the Windows operating system that supports DEP, PAE is automatically enabled when DEP is enabled and, on all 32-bit versions of the Windows operating system, PAE is disabled when you disable DEP. To enable PAE when DEP is disabled, you must enable PAE explicitly, by using **/set nx AlwaysOff** and **/set pae ForceEnable**. For more information about DEP, see Boot Parameters to Configure DEP and PAE.

For more information about using the **pae** parameter and the other parameters that affect PAE configuration, see Boot Parameters to Configure DEP and PAE.

removememory *Megabytes*
Removes memory from the total available memory that the operating system can use.

For example, the following command removes 256 MB of memory from the total available to the operating system associated with the specified boot entry.

```
bcdedit /set {49916baf-0e08-11db-9af4-000bdbd316a0} removememory 256
```

truncatememory *address* Limits the amount of physical memory available to Windows. When you use this option, Windows ignores all memory at or above the specified physical address. Specify the *address* in bytes.

For example, the following command sets the physical address limit at 1 GB. You can specify the address in decimal (1073741824) or hexadecimal (0x40000000).

```
bcdedit /set {49916baf-0e08-11db-9af4-000bdbd316a0} truncatememory 0x40000000
```

VESA, PCI, VGA, and TPM

usefirmwarepcisettings [**yes** | **no**]
Enables or disables the use of BIOS-configured peripheral component interconnect (PCI) resources.

msi [**Default** | **ForceDisable**]
Can be Default or ForceDisable.

vga [**on** | **off**]
Forces the use of the VGA display driver.

novga [**on** | **off**]
Disables the use of VGA modes entirely.

tpmbootentropy [**default** | **ForceEnable** | **ForceDisable**]
Determines whether entropy is gathered from the trusted platform module (TPM) to help seed the random number generator in the operating system.

Processors and APICs

clustermodeaddressing [**integer**]
Defines the maximum number of processors to include in a single Advanced Programmable Interrupt Controller (APIC) cluster.

configflags [**integer**]
Specifies processor-specific configuration flags.

maxproc [**yes** | **no**]
Reports the maximum number of processors in the system.

numproc [**integer**]
Uses only the specified number of processors.

onecpu [**yes** | **no**]
Forces only the boot CPU to be used.

restrictapicluster [**integer**]
Defines the largest APIC cluster number to be used by the system.

usephysicaldestination [**yes** | **no**]
Forces the use of the physical APIC.

uselegacyapicmode [**yes** | **no**]
Forces legacy APIC mode, even if the processors and chipset support extended APIC mode.

x2apicpolicy [**enable** | **disable** | **default**]

Enables or disables the use of extended APIC mode, if supported. The system defaults to using extended APIC mode if it is available. Can be Enabled, Disabled or Default.

Additional Settings

disabledynamictick [**yes** | **no**]
Enables and disables dynamic timer tick feature.

> **NOTE**
> This option should only be used for debugging.

pciexpress [**default** | **forcedisable**]
Enables or disables PCI Express functionality. If the computer platform supports the PCI Express features and the ACPI _OSC method grants control of the features to the operating system, Windows enables the advanced features through the PCI Express Native Control feature (this is the default). Use the **forcedisable** option to override the advanced PCI Express features and use legacy PCI Express behavior. For more information, see Enabling PCI Express Native Control in Windows.

useplatformtick [**yes** | **no**]
Forces the clock to be backed by a platform source, no synthetic timers are allowed. The option is available starting in Windows 8 and Windows Server 2012.

> **NOTE**
> This option should only be used for debugging.

xsavedisable [**0** | **1**]
When set to a value other than zero (0), disables XSAVE processor functionality in the kernel.

Debugger Settings

To work with the debugger settings, use the following commands.

COMMAND	DESCRIPTION
BCDEdit /bootdebug	The /bootdebug boot option enables or disables boot debugging of the current or specified Windows operating system boot entry.
BCDEdit /dbgsettings	The /dbgsettings option sets or displays the current global debugger settings for the computer. To enable or disable the kernel debugger, use the BCDEdit /debug option.
BCDEdit /debug	The /debug boot option enables or disables kernel debugging of the Windows operating system associated with the specified boot entry or the current boot entry.

Hypervisor Debugger Settings

Use the **BCDEdit / hypervisorsettings** option to set or display the hypervisor debugger settings for the system. For more information, see BCDEdit /hypervisorsettings.

hypervisordebug [On | Off]

Controls whether the hypervisor debugger is enabled.

hypervisordebugtype [**SERIAL** | **1394** | **NET**] Can be SERIAL, 1394, or NET. For more infomation, see BCDEdit /hypervisorsettings.

Hypervisor Settings

hypervisorlaunchtype [**Off** | **Auto**]
Controls the hypervisor launch options. If you are setting up a debugger to debug Hyper-V on a target computer, set this option to **Auto** on the target computer. For more information, see Create a Virtual Machine with Hyper-V.

hypervisorloadoptions NOFORCESNOOP [**Yes** | **No**]
Specifies whether the hypervisor should enforce snoop control on system IOMMUs.

hypervisornumproc *number*
Specifies the total number of logical processors that can be started in the hypervisor.

hypervisorrootproc *number*
Specifies the maximum number of virtual processors in the root partition and limits the number of post-split Non-Uniform Memory Architecture (NUMA) nodes which can have logical processors started in the hypervisor.

hypervisorrootprocpernode *number*
Specifies the total number of virtual processors in the root partition that can be started within a pre-split Non-Uniform Memory Architecture (NUMA) node.

hypervisoruselargevtlb [**yes** | **no**]
Increases virtual Translation Lookaside Buffer (TLB) size.

hypervisoriommupolicy [**default** | **enable** | **disable**]
Controls whether the hypervisor uses an Input Output Memory Management Unit (IOMMU).

Drivers and System Root

driverloadfailurepolicy [**Fatal** | **UseErrorControl**]
Can be Fatal or UseErrorControl.

osdevice [**device**]
Defines the device that contains the system root.

systemroot [**string**]
Defines the path to the system root.

ems [**On** | **Off**]
Enables kernel Emergency Management Services. The BCDEdit /ems option enables or disables kernel Emergency Management Services (EMS) for the specified operating system boot entry. For more information, see BCDEdit /ems.

The BCDEdit /emssettings option sets the global Emergency Management Services (EMS) settings for the computer. For more information, see For more information, see BCDEdit /emssettings.

Virtual Secure Mode

vmslaunchtype [**Off** | **Auto**]
Controls the Virtual Secure Mode launch type. Can be Off or Auto. For more information, see Manage Windows Defender Credential Guard.

Event Logging

The BCDEdit /event command enables or disables the remote event logging for the specified boot entry. For more information, see BCDEdit /event.

Comments

For more information about specific BCD elements and boot options, you can use the commands **BCDEdit /? OSLOADER** and **BCDEdit /? TYPES OSLOADER**.

To view the current boot entries and their settings, use the **bcdedit /enum** command. This command displays the active boot entries and their associated globally unique identifiers (GUID). Use the identifiers with the **/set** command to configure options for a specific boot entry.

To delete a boot option value that you have set, use the **/deletevalue** option. The syntax for the command is as follows:

bcdedit /deletevalue [**{ID}**] *datatatype*

For example, if you change the processor group option, **groupsize**, to a new value for testing purposes, you can revert to the default value of 64 by typing the following command and then restarting the computer.

```
bcdedit /deletevalue groupsize
```

Any change to a boot option requires a restart to take effect. For information about commonly used BCDEdit commands, see Boot Configuration Data Editor Frequently Asked Questions.

DTrace

DTrace (DTrace.exe) is a command-line tool that displays system information and events. There is a bcedit option to enable dtrace. For information about the DTrace BCDEdit options available, see the installing section of DTrace on Windows.

Requirements

Minimum supported client: Windows Vista

Minimum supported server: Windows Server 2008

Related topics

- BCDEdit Options Reference
- BCDEdit /deletevalue

BCDEdit /timeout

8/23/2022 • 2 minutes to read • Edit Online

The **BCDEdit /timeout** command sets the time to wait, in seconds, before the boot manager selects a default entry. For information about setting the default entry, run "bcdedit /? default".

```
bcdedit /timeout <timeout>
```

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Parameters

<timeout> *seconds*

Specifies the time to wait, in seconds, before the boot manager selects a default entry.

Examples

The following command sets the boot manager <timeout> to 30 seconds:

```
bcdedit /timeout 30
```

BCDEdit /toolsdisplayorder

8/23/2022 • 2 minutes to read • Edit Online

The **BCDEdit /toolsdisplayorder** command sets the display order to be used by the boot manager when displaying the tools menu.

```
bcdedit /toolsdisplayorder <id> [...] [ /addfirst | /addlast | /remove ]
```

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

Parameters

<id> [...]

Specifies a list of identifiers that make up the tools display order. At least one identifier must be specified and they must be separated by spaces. For more information about identifiers, run "bcdedit /? ID".

/addfirst

Adds the specified entry identifier to the top of the tools display order. If this switch is specified, only a single entry identifier may be specified. If the specified identifier is already in the list, it is moved to the top of the list.

/addlast

Adds the specified entry identifier to the end of the tools display order. If this switch is specified, only a single entry identifier may be specified. If the specified identifier is already in the list, it is moved to the end of the list.

/remove

Removes the specified entry identifier from the tools display order. If this switch is specified, only a single entry identifier may be specified. If the identifier is not in the list, then the operation will have no effect. If the last entry is being removed, then the tools display order value is deleted from the boot manager entry.

Examples

The following command sets two tools entries and the memory diagnostic in the boot manager's tools display order:

```
bcdedit /toolsdisplayorder {802d5e32-0784-11da-bd33-000476eba25f} {cbd971bf-b7b8-4885-951a-fa03044f5d71}
{memdiag}
```

The following command adds the specified tool entry to the end of the boot manager's tools display order:

```
bcdedit /toolsdisplayorder {802d5e32-0784-11da-bd33-000476eba25f} /addlast
```

Editing Boot Options

8/23/2022 • 2 minutes to read • Edit Online

This section is a practical guide to editing the boot options on a computer running Windows Server 2008, Windows Server 2012, or Windows 7 or later. It suggests a step-by-step procedure for customizing the basic elements of boot options.

This section describes a method of using BCDEdit, a tool included with the operating system. For information about BCDEdit command syntax, type **bcdedit /?** or **bcdedit /? TOPICS** in a Command Prompt window. See BCDEdit Options Reference for more information.

> **NOTE**
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

For help on editing boot entry parameters to enable and disable Windows features, see Using Boot Parameters.

To configure operating system features in boot options:

- Add a new boot entry for the operating system by copying an existing boot entry from the same operating system.
- Change the friendly name of the newly created boot entry so that you can identify it in the boot menu.
- Add parameters to the boot entry that enable and configure Windows features.

Then, to make testing quicker and easier:

- Make the new boot entry the default entry.
- Change the boot menu time-out. You can shorten the boot menu time-out so that Windows boots quickly. Or, lengthen the boot menu time-out so that you have ample time to select the preferred boot entry.

Related Topics

BCDEdit Command-Line Options

Caution

Administrative privileges are required to use BCDEdit to modify BCD. Changing some boot entry options using the **BCDEdit /set** command could render your computer inoperable. As an alternative, use the System Configuration utility (MSConfig.exe) to change boot settings.

Changing Boot Parameters

8/23/2022 • 2 minutes to read • Edit Online

To enable and configure boot-related operating system features, such as debugging, you must add boot parameters to a boot entry for the operating system.

To change boot parameters on a system running Windows, you can use BCDEdit.

Using BCDEdit

To add a boot configuration parameter to a boot entry, use the BCDEdit boot entry options to change global settings, such as **/ems**, **/debug**, **/dbgsettings**, or set individual parameters using the **BCDEdit /set** options. For a complete list of BCDEdit options, at a command prompt, type **BCDEdit /?** or **BCDEdit /?** <command> to find help about a specific command.

For example, the following command enables PAE for a specified boot entry:

```
bcdedit /set {802d5e32-0784-11da-bd33-000476eba25f} pae forceenable
```

To turn the kernel debugger on or off, use the **/debug** option with the following syntax:

```
bcdedit /debug <ID> [on | off]
```

The <ID> is the GUID that is associated with the boot entry. If you do not specify an <ID>, the command modifies the operating system that is currently active. The following command turns on the kernel debugger for a boot entry, called DebugEntry:

```
bcdedit /debug {49916baf-0e08-11db-9af4-000bdbd316a0} on
```

To view the current boot entries, type **bcdedit** at the command prompt. The boot entry for DebugEntry shows that the kernel debugger is turned on.

```
## Windows Boot Loader
-------------------
identifier              {49916baf-0e08-11db-9af4-000bdbd316a0}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             DebugEntry
locale                  en-US
inherit                 {bootloadersettings}
osdevice                partition=C:
systemroot              \Windows
resumeobject            {3e3a9f69-024a-11db-b5fc-a50a1ad8a70e}
nx                      OptIn
pae                     ForceEnable
debug                   Yes
```

Adding Boot Entries

8/23/2022 • 4 minutes to read • Edit Online

One method to customize boot options in Windows is to add a new *boot entry* for an operating system. A *boot entry* is a set of options that define a load configuration for an operating system or bootable program.

You can have multiple boot entries for an operating system, each with a different set of boot parameters. The Windows Installer creates a standard boot entry when you install an operating system, and you can create additional, customized boot entries for an operating system by editing the boot options.

You can add, delete, and change the options in the boot entry that Windows Installer created. However, it is prudent to keep the standard entry and, instead, add a separate entry that you customize.

To add a boot entry, copy an existing boot entry, and then modify the copy.

This topic applies to Windows Vista and later, Windows Server 2008 and later, and the Windows Recovery Environment.

Adding a new boot entry

In Windows, you use BCDEdit to modify your boot options. To add a new boot entry, open a Command Prompt window with elevated privileges (select and hold (or right-click) **Command Prompt** and select **Run as administrator** from the shortcut menu).

Note Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

The easiest way to create a new boot entry is to copy an existing entry and then modify it as you need. To do this, use BCDEdit with the **/copy** option. For example, in the following command, BCDEdit copies the Microsoft Windows boot entry that was last used to boot Windows, identified as **{current}**, and creates a new boot entry. The **/d** description option specifies DebugEntry as the name of the new boot entry.

```
bcdedit /copy {current} /d "DebugEntry"
```

If the command succeeds, BCDEdit displays a message similar to the following:

```
The entry was successfully copied to {49916baf-0e08-11db-9af4-000bdbd316a0}.
```

When you copy a boot loader entry that appears on the boot menu, the copy is automatically added as the last item on the boot menu.

The GUID in the preceding message (which appears between braces ({})) is the identifier of the new boot entry. The **/copy** option creates a new GUID for the boot entry. You use the identifier to represent the entry in all subsequent BCDEdit commands.

If the command fails, be sure that you are running in a Command Prompt window with administrator privileges and that all of the command parameters are spelled correctly, including the braces around **{current}**.

You can also add a boot entry using the **/create** option. This method is more difficult because you need to provide additional information about the boot entry type. You also need to specify the **/application**, **/inherit**, or **/device** options. For example, the following creates a new operating system boot entry called "My Windows Vista":

```
bcdedit /create /d "My Windows Vista" /application osloader
```

When you use the **/create** option, the new boot loader entries are not added to the boot menu automatically. The **/create** option creates a new GUID for the boot entry. You must add the new boot entry to the boot menu by using the **/displayorder** option. You can place the boot loader entries in any order.

For information about the **/create** command parameters, type **bcdedit /? /create** in a Command Prompt window.

Editing the boot menu

In Windows, new boot loader entries are not added to the boot menu automatically. You can place the boot loader entries in any order.

You can use the **/displayorder** option to set the order in which the boot manager displays the boot entries on a multi-boot menu. The command has the following syntax:

```
bcdedit /displayorder {ID} {ID} ...
```

The ID is the GUID of the boot entry or a reserved identifier, such as **{current}**). Separate each identifier with a space. Be sure to include the braces ({}).

For example, to add the DebugEntry boot entry to the boot menu after the **{current}** entry, use the following command (remember to use `'{guid}'` in Windows PowerShell):

```
bcdedit /displayorder {current} {49916baf-0e08-11db-9af4-000bdbd316a0}
```

You can also use the options **/addlast, /addfirst**, and **/remove** to order and remove items from the menu. For example, the following command adds the DebugEntry boot entry as the last item on the menu:

```
bcdedit /displayorder {49916baf-0e08-11db-9af4-000bdbd316a0} /addlast
```

Removing and deleting a boot entry

The following command removes the {49916baf-0e08-11db-9af4-000bdbd316a0} boot entry item from the boot menu.

```
bcdedit /displayorder {49916baf-0e08-11db-9af4-000bdbd316a0} /remove
```

When you remove the specified boot entry using the **/displayorder** and **/remove** options, the boot entry is removed from the boot menu, but it is still in the BCD store. To completely remove a boot loader entry from the boot menu and from the store, use the **/delete** option.

```
bcdedit /delete {49916baf-0e08-11db-9af4-000bdbd316a0}
```

To verify that the display order is correct, use the following command:

```
bcdedit
```

When you type **bcdedit** without additional parameters, BCDEdit displays the boot manager entry and the boot

loader entries in the order that they will appear in the menu.

The Windows Boot Manager entry also includes the boot menu display order, as the following example shows.

```
## Windows Boot Manager
identifier              {bootmgr}
device                  partition=C:
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
isolatedcontext         Yes
default                 {current}
resumeobject            {18b123cd-2bf6-11db-bfae-00e018e2b8db}
displayorder            {current}
toolsdisplayorder       {memdiag}
timeout                 30

## Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             Microsoft Windows
locale                  en-US
inherit                 {bootloadersettings}
osdevice                partition=C:
systemroot              \Windows
resumeobject            {d7094401-2641-11db-baba-00e018e2b8db}
nx                      OptIn

## Windows Boot Loader
-------------------
identifier              {18b123cd-2bf6-11db-bfae-00e018e2b8db}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             Debugger Boot
locale                  en-US
inherit                 {bootloadersettings}
osdevice                partition=C:
systemroot              \Windows
resumeobject            {d7094401-2641-11db-baba-00e018e2b8db}
nx                      OptIn
debug                   Yes
```

See Also

Editing Boot Options

Changing the Default Boot Entry

8/23/2022 • 2 minutes to read • Edit Online

The default boot entry is the entry that the boot loader selects when the boot menu time-out expires. You can change the default boot entry to ensure that the operating system configuration that you prefer is loaded automatically.

For Windows, you can use BCDEdit to change the default boot entry.

Using BCDEdit

You can specify the default boot entry using the **/default** option. The syntax to specify the default operating system is as follows:

```
bcdedit /default <ID>
```

The *<ID>* is the GUID for the Windows boot loader boot entry that is associated with the operating system that you want to designate as the default. You must include the braces (**{ }**) around the GUID, for example:

```
bcdedit /default {cbd971bf-b7b8-4885-951a-fa03044f5d71}
```

To change the default boot entry to the earlier Windows operating system loader on a multiboot computer, set *<ID>* to **{ntldr}**, which is the reserved name for the GUID that is associated with Ntldr. This might present another menu depending on entries in Boot.ini file.

```
bcdedit /default {ntldr}
```

Changing the Boot Menu Time-out

8/23/2022 • 2 minutes to read • Edit Online

The boot menu time-out determines how long the boot menu is displayed before the default boot entry is loaded. It is calibrated in seconds.

If you want extra time to choose the operating system that loads on your computer, you can extend the time-out value. Or, you can shorten the time-out value so that the default operating system starts faster.

For Windows, you can use BCDEdit to change the default boot menu time-out value.

Using BCDEdit

To specify the boot menu time-out value, use the **/timeout** option:

```
bcdedit /timeout <timeout>
```

Use the **/timeout** option and specify the timeout value in seconds. For example, to specify a 15-second timeout value:

```
bcdedit /timeout 15
```

Changing the Friendly Name of a Boot Entry

8/23/2022 • 2 minutes to read • Edit Online

In Windows, the items that appear in the Windows Boot Manager are the descriptions of each boot entry.

Typically, after you copy a boot entry, you change the friendly name of the newly created entry to distinguish it from the original.

You can also change the friendly name to make it easier to recognize customized boot entries. A string that precisely describes the entry can save significant time and effort.

For example, the following friendly name strings add little value.

```
"Windows 10 Debug1"
"Windows 10 Debug2"
```

However, more precise strings, such as the ones that follow, make the boot choice much easier.

```
"Windows 10 kdnet"
"Windows 10 NullModem"
```

Note When a boot entry is configured for debugging (/debug /debugport) or for Emergency Management Services (EMS) (/redirect) on an x86- or an x64-based system, the boot loader appends a bracketed phrase ([debugger enabled] or [ems enabled]) to the friendly name that appears in the boot menu. However, the boot loader omits the bracketed phrase from the boot menu when the friendly name and the bracketed phrase together exceed 70 characters. To restore the bracketed phrase, shorten the friendly name.

To change the friendly name of a boot entry in a Boot.ini file, you can use Bootcfg or edit the Boot.ini file in Notepad. On systems that store boot options in EFI NVRAM, use Bootcfg.

To change the friendly name of a boot entry for Windows, use BCDEdit.

Caution

Administrative privileges are required to update the boot configuration. Changing some boot entry options could render your computer inoperable.

Using BCDEdit

To change the description of a boot entry as it appears on the boot menu, you can use the **/set** *IDdescription* option. The command uses the following syntax. The ID is the GUID that is associated with the boot entry (or one of the well-known identifiers, for example, {current}).

> **NOTE**
>
> If you are using Windows PowerShell, you must use quotes around the boot entry identifier, for example: **"{49916baf-0e08-11db-9af4-000bdbd316a0}"** or **"{current}"**.

```
bcdedit /set ID description "The new description"
```

For example:

```
bcdedit /set {802d5e32-0784-11da-bd33-000476eba25f} description "Windows 10 NullModem"
```

To change the description of the boot entry that corresponds to the operating system that is currently running, use the following example:

```
bcdedit /set {current} description "Windows 10 NullModem"
```

You can also change the description when you copy an existing boot entry using the **/d** option.

```
bcdedit /copy {current} /d "Windows 10 NullModem"
```

Using Bootcfg

With Bootcfg, you can change the friendly name of a boot entry only while copying the entry. Use the Bootcfg **/copy** switch to copy the entry and change its friendly name.

The following Bootcfg command copies the first boot entry to create a new entry. The **/ID** switch specifies the line number of the entry being copied. The **/d** (description) switch specifies the friendly name of the newly-created entry.

```
bootcfg /copy /ID 1 /d "Windows 10 Debug"
```

For complete instructions for using Bootcfg, see Help and Support Services. For examples, see Using Boot Parameters.

Editing the Boot.ini File

In the Boot.ini file, the friendly name of a boot entry appears in the boot entry in quotation marks.

For example, the following sample from a Boot.ini file has duplicate boot entries for Microsoft Windows 10 Professional.

```
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows 10 Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows 10 Professional" /fastdetect
```

To change the friendly name of a boot entry, type over the quoted string in the boot entry. In the following example, because the first entry will be customized for debugging, the name is changed to Windows 10 Debug.

```
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows 10 Debug" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows 10 Professional" /fastdetect
```

Using boot parameters

8/23/2022 • 2 minutes to read • Edit Online

Driver developers and testers often have to add, delete, and change the parameters of boot entries to test their drivers under variable conditions. This section describes a few common scenarios and suggests strategies for configuring boot parameters in the Boot.ini file and in NVRAM.

> **NOTE**
>
> Checked builds were available on older versions of Windows, before Windows 10 version 1803. Use tools such as Driver Verifier and GFlags to check driver code in later versions of Windows.

This section contains the following topics:

Boot Parameters to Enable Debugging

8/23/2022 • 2 minutes to read • Edit Online

When a kernel debugging connection is established, the system gives a kernel debugger control over its execution. Also, when a bug check occurs or a kernel-mode program communicates with a debugger, the computer waits for a response from a kernel debugger before it continues.

> **IMPORTANT**
>
> Setting up a network debugging manually is a complex and error prone process. To set up network debugging automatically, see **Setting Up KDNET Network Kernel Debugging Automatically**. Using the KDNET utility is **strongly** recommended for all debugger users.

For information on manual setup of a network connection, see Setting Up Kernel-Mode Debugging over a Network Cable Manually.

> **NOTE**
>
> Before setting BCDEdit options you might need to disable or suspend BitLocker and Secure Boot on the computer.

You can use the **bcdedit** command to view the current debugger boot entries and to change their settings. For more details, see **BCDEdit /debug** and **BCDEdit /dbgsettings**.

Boot Parameters to Debug the Boot Process in Windows

To enable boot debugging, use the **BCDEdit /bootdebug** command and specify the appropriate boot component. If you wish to perform kernel debugging after Windows starts, use the **BCDEdit /debug** command as well. You must also select a debugging connection, just as in normal kernel debugging. For more details, see **BCDEdit /bootdebug**.

See also

For information about Windows debugging tools, see Windows Debugging.

For information about setting up and configuring a kernel-mode debugging session, see Setting Up Kernel-Mode Debugging Manually and Setting Up KDNET Network Kernel Debugging Automatically.

Boot Parameters to Manipulate Memory

8/23/2022 • 2 minutes to read • Edit Online

You can simulate a low-memory environment for testing without changing the amount of physical memory on the computer. Instead, you can limit the memory available to the operating system by using **truncatememory** or **removememory** options with the **BCDedit /set** command.

The **/maxmem** parameter specifies the maximum amount of memory available to Windows. It is calibrated in megabytes (MB). Set the value to any amount less than the actual physical memory on the computer.

The **/maxmem** parameter actually determines the largest memory address available to Windows. Due to gaps in the mapping of physical memory, Windows might receive somewhat less memory than the value of **/maxmem**. For more precision, use **/burnmemory**.

The **truncatememory** or **removememory** options are available in Windows 7 and later. The **truncatememory** option disregards all memory at or above the specified physical address. The **removememory** option reduces memory available to Windows by the specified amount (measured in MB). Both options reduce memory, but the **removememory** option is better at restricting the operating system to use the specified memory while accounting for memory gaps.

Boot Parameters to Test in a Low-memory Environment in Windows

To simulate a low-memory environment, use the **BCDedit /set** command and the **removememory** option to modify a boot entry. Set the value of **removememory** to the amount of physical memory on the system minus the desired memory size for this test.

For example, to limit the memory of a computer with 2 GB of physical memory to a maximum of 512 MB of available memory, set the value of the **removememory** parameter to 1536 (2 GB (2048 MB) - 512 MB = 1536 MB).

The following example shows a BCDEdit command used to remove 1536 MB of memory from the total available to the system for the specified boot entry.

```
bcdedit /set {18b123cd-2bf6-11db-bfae-00e018e2b8db} removememory 1536
```

You can also use the **truncatememory** option with the **bcdedit /set** command to achieve the same result. When you use this option, Windows ignores all memory at or above the specified physical address. Specify the *address* in bytes. For example, the following command sets the physical address limit at 1 GB for the specified boot entry. You can specify the address in decimal (1073741824) or hexadecimal (0x40000000).

```
bcdedit /set {18b123cd-2bf6-11db-bfae-00e018e2b8db} truncatememory 0x40000000
```

Because the **removememory** option makes more efficient use of system memory, its use is recommended instead of **truncatememory**.

When you are finished testing, you can remove the **removememory** and **truncatememory** boot entry options using the **BCDEdit /deletevalue** command.

Boot Parameters to Load a Partial Checked Build

8/23/2022 • 2 minutes to read • Edit Online

A *partial checked build* contains checked build versions of the kernel and HAL and a free build of the remainder of the operating system. For details, see Installing Just the Checked Operating System and HAL (For Windows Vista and Later).

> **NOTE**
> Checked builds were available on older versions of Windows, before Windows 10 version 1803. Use tools such as Driver Verifier and GFlags to check driver code.

Configuring a Partial Checked Build in Windows

To configure a partial checked build use the **BCDedit /set** command and the **kernel** and **hal** options.

The following commands configure a boot entry to use the checked versions of the kernel and hardware abstraction layer (HAL).

```
bcdedit /set {18b123cd-2bf6-11db-bfae-00e018e2b8db} kernel ntoskrnl.chk
```

```
bcdedit /set {18b123cd-2bf6-11db-bfae-00e018e2b8db} hal halacpi.chk
```

To view the results of the commands, type **bcdedit /enum**. The **/enum** option lists all of the boot entries. The boot entry that has been modified to use the checked versions of the kernel and HAL has also been configured for kernel debugging over a serial connection.

```
## Windows Boot Loader
-------------------
identifier              {18b123cd-2bf6-11db-bfae-00e018e2b8db}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             PartialCheckedBuild
locale                  en-US
inherit                 {bootloadersettings}
debugtype               serial
debugport               1
baudrate                115200
osdevice                partition=C:
systemroot              \Windows
kernel                  ntoskrnl.chk
hal                     halacpi.chk
resumeobject            {d7094401-2641-11db-baba-00e018e2b8db}
nx                      OptIn
debug                   Yes
```

Boot Parameters to Test Drivers for Multiple Processor Group Support

8/23/2022 • 7 minutes to read • Edit Online

Windows 7 and Windows Server 2008 R2 provide support for computers with more than 64 processors. This support is made possible by introducing Processor groups. For testing purposes, you can configure any computer that has multiple logical processors to have multiple processor groups by limiting the group size. This means you can test drivers and components for multiple processor group compatibility on computers that have 64 or fewer logical processors.

Note The concept of *processor groups*, introduced with Windows 7, allows existing APIs and DDIs to continue to work on computers with more than 64 logical processors. Typically, a group's processors are represented by an affinity mask, which is 64 bits long. Any computer with more than 64 logical processors will necessarily have more than one group. When a process is created, the process is assigned to a specific group. By default, threads of the process can run on all logical processors of the same group, although the thread affinity can be explicitly changed. Calls to any API or DDI that takes an affinity mask or processor number as an argument, but not a group number, is limited to affecting or reporting on those processors in the calling thread's group. The same is true of APIs or DDIs that return an affinity mask or processor number, like **GetSystemInfo**.

Starting with Windows 7, an application or driver can make use of functions that extend the legacy APIs. These new group-aware functions accept a group number argument to unambiguously qualify a processor number or affinity mask, and therefore can manipulate processors outside of the calling thread's group. The interaction between drivers and components running in different groups within a computer introduces the potential for bugs when legacy APIs or DDIs are involved. You can use the legacy non-group-aware APIs on Windows 7 and Windows Server 2008 R2. However, driver requirements are more stringent. For functional correctness of drivers on computers that have more than one processor group, you must replace any DDI that either accepts a processor number or mask as a parameter without an accompanying processor group or returns a processor number or mask without an accompanying processor group. These legacy non-group-aware DDIs can perform erratically on a computer that has multiple process groups because the inferred group may be different than what the calling thread intended. Therefore, drivers that use these legacy DDIs and are targeted for Windows Server 2008 R2 must be updated to use the new extended versions of the interfaces. Drivers that do not call any functions that use processor affinity masks or processor numbers will operate correctly, regardless of the number of processors. Drivers that call the new DDIs can run on previous versions of Windows by including the procgrp.h header, calling **WdmlibProcgrpInitialize**, and linking against the Processor Group Compatibility Library (procgrp.lib).

For more information on the new group-aware APIs and DDIs, download the white paper Supporting Systems that Have More than 64 Logical Processors: Guidelines for Developers.

To help identify potential processor group-related problems in drivers and components, you can use the **BCDEdit /set** options. The two BCD boot configuration settings, **groupsize** and **maxgroup**, can configure any computer that has multiple logical processors to support multiple processor groups. The **groupaware** option modifies the behavior of certain DDIs and manipulates the group environment for testing purposes.

Create Multiple Processor Groups by Changing the Group Size

The **groupsize** option specifies the maximum number of logical processors in a group. By default, the **groupsize** option is not set, and any computer with 64 or fewer logical processors has one group, which is group 0.

Note A physical processor, or processor package, can have one or more cores, or processor units, each of

which can contain one or more logical processors. The operating system considers a logical processor as one logical computing engine.

To create multiple processor groups, run BCDEdit /set in an elevated Command Prompt window and specify a new *maxsize* value for **groupsize** that is less than the total number of logical processors. Note that the group size setting is for testing and you should not configure shipping systems with this setting. The *maxsize* value can be set to any power of 2 between 1 and 64 inclusive. The command uses the following syntax:

```
bcdedit.exe /set groupsize maxsize
```

For example, the following command sets the maximum number of processors in a group to 2.

```
bcdedit.exe /set groupsize 2
```

If a non-NUMA computer has 8 logical processors, setting the **groupsize** to 2 creates 4 processor groups with 2 logical processors each.

Group 0: 1 NUMA node containing 1 package of 2 logical processors

Group 1: 1 NUMA node containing 1 package of 2 logical processors

Group 2: 1 NUMA node containing 1 package of 2 logical processors

Group 3: 1 NUMA node containing 1 package of 2 logical processors

By design, a non-NUMA computer is considered to have one NUMA node. Because NUMA nodes cannot span groups, the system creates a node for each group after you restart the computer.

If **groupsize** is set to a value less than the number of logical processors in a physical processor package (socket), the system redefines its concept of a package upon restart such that the package does not span a group. This means that more packages than that are actually present are reported by processor topology APIs. This also means that the Windows (package-level) processor licensing limits might prevent some processor packages from starting when **groupsize** is set.

A processor package can span groups if it has multiple NUMA nodes defined within it and the system assigns these nodes to different groups.

Windows limits the number of groups supported. This number could change with new versions of Windows or in service pack releases. Drivers or components should not depend on the number of groups Windows supports as being constant. The limit on the number of groups could restrict the number of logical processors allowed to start when small values are used for the **groupsize** boot option.

To remove the **groupsize** setting you used for testing and return to the default setting of 64 logical processors per group, use the following BCDEdit command.

```
bcdedit.exe /deletevalue groupsize
```

This command is the equivalent of setting **groupsize** to 64.

Maximize the Number of Processor Groups

The **maxgroup** option is another way to create processor groups on a computer with multiple logical processors and NUMA nodes. The **maxgroup** boot option has no effect on non-NUMA computers.

To maximize the number of groups, run the BCDEdit /set command in an elevated Command Prompt window. The command uses the following syntax:

```
bcdedit.exe /set maxgroup on
```

For example, consider a computer that has 2 NUMA nodes, 1 processor package per node, and 4 processor cores per package, for a total of 8 logical processors.

The default group configuration is:

Group 0: 8 logical processors, 2 packages, 2 NUMA nodes

If you enter a **bcdedt.exe /set maxgroup on** command followed by a restart, the command yields the following group configuration:

Group 0: 4 logical processors, 1 package, 1 NUMA node

Group 1: 4 logical processors, 1 package, 1 NUMA node

Note that NUMA nodes are assigned to groups in a manner that maximizes the number of groups.

To change back to the default the setting, use the following **BCDEdit** command.

```
bcdedit.exe /set maxgroup off
```

Test Multiple-Group Compatibility by Setting the Group Aware Boot Option

Windows 7 and Windows Server 2008 R2 have introduced a new BCD option (**groupaware**) that forces drivers and components to be aware of multiple groups in a multiple processor group environment. The **groupaware** option changes the behavior of a set of device driver functions to help expose cross-group incompatibilities in drivers and components. You can use the **groupaware** boot option along with the **groupsize** and **maxgroup** options to test driver compatibility with multiple groups when a computer has 64 or fewer active logical processors.

When the **groupaware** boot option is set, the operating system ensures that processes are started in a group other than group 0. This increases the chances of cross-group interaction between drivers and components. The option also modifies the behavior of the legacy functions that are not group-aware, **KeSetTargetProcessorDpc**, **KeSetSystemAffinityThreadEx**, and **KeRevertToUserAffinityThreadEx**, so that they always operate on the highest numbered group that contains active logical processors. Drivers that call any of these legacy functions should be changed to call their group-aware counterparts (**KeSetTargetProcessorDpcEx**, **KeSetSystemGroupAffinityThread**, and **KeRevertToUserGroupAffinityThread**),

To test for compatibility, use the following **BCDEdit /set** command.

```
bcdedit.exe /set groupaware on
```

LEGACY NON-GROUP AWARE FUNCTIONS	WINDOWS 7 GROUP-AWARE REPLACEMENT
KeSetTargetProcessorDpc	KeSetTargetProcessorDpcEx
KeSetSystemAffinityThreadEx	KeSetSystemGroupAffinityThread
KeRevertToUserAffinityThreadEx	KeRevertToUserGroupAffinityThread

To reset the computer to the default setting, use the following **BCDEdit** command.

```
bcdedit.exe /set groupaware off
```

Boot Parameters to Configure DEP and PAE

8/23/2022 • 2 minutes to read • Edit Online

This topic explains how to use boot parameters to enable, disable, and configure Data Execution Prevention (DEP) and Physical Address Extension (PAE) on operating systems that support these features.

For information about the boot parameters for DEP and PAE see the **BCDEdit /set** command and the **nx** and **pae** options.

Important DEP is a highly effective security feature that should not be disabled unless you have no alternative. The default settings for DEP and PAE are optimal for most systems. Do not change the default settings unless they interfere with essential processing tasks. This section is included to show you how to configure these features, but it should not be interpreted as a recommendation to change the default settings.

DEP and PAE Boot Parameters

DEP and PAE are enabled at boot time and are configured by setting values for the **nx** and **pae** parameters using the **BCDEdit /set** command.

These boot parameters have conflicting effects. To configure DEP and PAE, use only the parameter combinations that are described in the documentation for each parameter and discussed in this topic. Do not experiment with conflicting parameters, especially on a production system.

The Interaction of DEP and PAE Boot Parameters

There are two types of DEP:

- *Hardware-enforced DEP* enables DEP for both kernel-mode and user-mode processes. It must be supported by the processor and the operating system.
- *Software-enforced DEP* enables DEP only on user-mode processes. It must be supported by the operating system.

On 32-bit versions of Windows, *hardware-enforced DEP* requires PAE, which is supported by all Windows operating systems that support DEP. When DEP is enabled on a computer with a processor that supports hardware-enforced DEP, Windows automatically enables PAE and ignores the boot parameter values that disable it.

The parameter combinations for each Windows operating system are summarized in the following section.

DEP and PAE Parameter Combinations

The following list describes the boot parameter combinations that can be used to configure DEP and PAE.

Note The optional {*ID*} is the GUID for the specific Windows boot loader boot entry that you want to configure. If you do not specify an {*ID*}, the command modifies the current operating system boot entry. For more information, see the **BCDEdit /set** command .

ACTION		WINDOWS VISTA AND LATER
To enable DEP (Select one parameter combination) When DEP is enabled on computers that support hardware-enforced DEP, these parameter combinations also enable PAE.		**/set [{/D}] nx AlwaysOn** **/set [{/D}] nx OptIn** **/set [{/D}] nx OptOut**
To enable DEP and PAE on systems with software-enforced DEP (Select one parameter combination) On computers that support hardware-enforced DEP, PAE is automatically enabled when you enable DEP.		**/set [{/D}] nx AlwaysOn** **/set [{/D}] pae default** **/set [{/D}] nx OptIn** **/set [{/D}] pae default** **/set [{/D}] nx OptOut** **/set [{/D}] pae default**
To disable DEP, but enable PAE		**/set [{/D}] nx AlwaysOff** **/set [{/D}] pae ForceEnable**
To disable DEP, but enable PAE		**/set [{/D}] nx AlwaysOff** **/set [{/D}] pae ForceEnable**
To disable both DEP and PAE		**/set [{/D}] nx AlwaysOff** **/set [{/D}] pae ForceDisable**
To disable both DEP and PAE		

Boot Parameters to Enable EMS Redirection

8/23/2022 • 7 minutes to read • Edit Online

Emergency Management Services (EMS) technology allows you to control the selected components of servers remotely, even when a server is not connected to the network or to other standard remote-administration tools. EMS is supported on all versions of Windows Server 2003 operating systems for x86-, x64-, and Itanium-based computers.

> **NOTE**
>
> This topic explains how to enable EMS on computers running Windows Server 2003. The boot parameters described in this section are not supported on Windows Vista or later versions of Windows. When a boot entry is configured for EMS on a computer with BIOS firmware, the boot loader appends a bracketed phrase, [ems enabled], to the friendly name that appears on the boot menu. However, the boot loader omits the bracketed phrase from the boot menu when the friendly name and the bracketed phrase together exceed 70 characters. To restore the bracketed phrase, shorten the friendly name.

To determine whether a computer has ACPI firmware, use Device Manager (devmgmt.msc). In Device Manager, expand the **Computer** node. On computers with ACPI firmware, the name of node under **Computer** includes the word, **ACPI**.

Enabling EMS on a computer without an ACPI SPCR table in operating systems prior to Windows Server 2008

To enable EMS console redirection on a computer that has BIOS firmware, but does not have an ACPI Serial Port Console Redirection (SPCR) table, add the **redirect=COM***x* and the **redirectbaudrate=** parameters to the [boot loader] section of the Boot.ini file. These parameters set the port and transmission rate for EMS console redirection. Use the same port and transmission rate that are established for out-of-band communication in the BIOS. Then, add the **/redirect** parameter to a boot entry.

The following Bootcfg command enables EMS console redirection on the first boot entry in the list. It sets the port for COM2 and sets the transmission rate to 115,200 kilobits per second (Kbps). These are the same port and baud rate settings that the administrator set in the BIOS for the out-of-band port.

```
bootcfg /ems ON /port COM2 /baud 115200 /id 1
```

The following Bootcfg display shows the result of the command. The newly added parameters are displayed in bold type.

```
## Boot Loader Settings
timeout:          3
default:          multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
redirect:         COM2
redirectbaudrate: 115200

Boot Entries
------------
Boot entry ID:   1
Friendly Name:   "Windows Server 2003, Standard with EMS"
Path:            multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
OS Load Options: /fastdetect /redirect
```

The following sample shows the result of the same command on a sample Boot.ini file.

```
[boot loader]
timeout=1
default=multi(0)disk(0)rdisk(0)partition(2)\WINDOWS
redirect=COM2
redirectbaudrate=115200
[operating systems]
multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="EMS boot" /fastdetect /redirect
multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="Windows Server 2003, Standard" /fastdetect
```

Enabling EMS on a Computer without an ACPI SPCR Table in Windows Server 2008

To enable EMS console redirection on a computer that has BIOS firmware, but does not have an ACPI Serial Port Console Redirection (SPCR) table, use the **BCDEdit /emssettings** command to set the COM port and baud rate.

These parameters set the global port and transmission rate for EMS console redirection. Use the same port and transmission rate that are established for out-of-band communication in the BIOS.

Then, use the **BCDEdit /ems** command to enable EMS for a boot entry.

The following commands set the global EMS redirection settings to use COM2 and a baud rate of 115200, and enable EMS for the specified boot entry.

```
bcdedit /emssettings EMSPORT:2 EMSBAUDRATE:115200
```

```
bcdedit /ems {18b123cd-2bf6-11db-bfae-00e018e2b8db} on
```

Enabling EMS on a computer with an SPCR table in operating systems prior to Windows Server 2008

To enable EMS on a computer with ACPI BIOS firmware and an ACPI SPCR table, you can either use the **redirect=USEBIOSSETTINGS** parameter or the **redirect=COM***x* and **redirectbaudrate=** parameters. Then, you can add the **/redirect** parameter to a boot entry.

The following example demonstrates use of the **redirect=USEBIOSSETTINGS** parameter. The following Bootcfg command enables EMS console redirection on the first boot entry in the list.

```
bootcfg /ems ON /port BIOSSET /id 1
```

The following Bootcfg display shows the result of the command. The newly added parameters are displayed in bold type.

```
## Boot Loader Settings
timeout: 1
default: multi(0)disk(0)rdisk(0)partition(2)\WINDOWS
redirect:USEBIOSSETTINGS

Boot Entries
------------
Boot entry ID:    1
OS Friendly Name: EMS boot
Path:             multi(0)disk(0)rdisk(0)partition(2)\WINDOWS
OS Load Options:  /fastdetect /redirect

Boot entry ID:    2
OS Friendly Name: Windows Server 2003, Standard
Path:             multi(0)disk(0)rdisk(0)partition(2)\WINDOWS
OS Load Options:  /fastdetect
```

The following sample shows the result of the same command on a sample Boot.ini file.

```
[boot loader]
timeout=1
default=multi(0)disk(0)rdisk(0)partition(2)\WINDOWS
redirect=USEBIOSSETTINGS
[operating systems]
multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="EMS boot" /fastdetect /redirect
multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="Windows Server 2003, Standard" /fastdetect
```

Enabling EMS on a Computer with an SPCR Table in Windows Server 2008

To enable EMS on a computer with ACPI BIOS firmware and an ACPI SPCR table, you can use the **BCDEdit /emssettings** and specify either the **BIOS** parameter or the **emsport** and **emsbaudrate** parameters. To enable EMS for a boot entry, use the **BCDEdit /ems** command.

The following example demonstrates how to use the **BIOS** parameter. The following BCDEdit command enables EMS console redirection on the current boot entry.

```
bcdedit /emssettings bios
bcdedit /ems on
```

Enabling EMS on a computer with EFI firmware in operating systems prior to Windows Server 2008

To enable EMS on a computer with EFI firmware, use Bootcfg to add the **/redirect** parameter to a boot entry. Windows finds the out-of-band port and its settings in the firmware by reading the SPCR table and uses the same port and rate for EMS console redirection.

The following Bootcfg command enables EMS redirection on an Itanium-based computer. It uses the Bootcfg **/ems** switch with the ON argument to add the **/redirect** parameter to the boot entry. The **/id** switch identifies the boot entry.

```
bootcfg /ems ON /id 1
```

The following Bootcfg display of boot options in EFI NVRAM shows the result of the Bootcfg command. The first boot entry is configured to load the operating system with EMS console redirection enabled.

```
Boot Options
------------
Timeout:             30
Default:             \Device\HarddiskVolume3\WINDOWS
CurrentBootEntryID:  1

Boot Entries
------------
Boot entry ID:    1
OS Friendly Name: Windows Server 2003, Enterprise with EMS
OsLoadOptions:     /fastdetect /redirect
BootFilePath:     \Device\HarddiskVolume1\EFI\Microsoft\WINNT50\ia64ldr.efi
OsFilePath:       \Device\HarddiskVolume3\WINDOWS
```

Enabling EMS on a Computer with EFI Firmware in Windows Server 2008

To enable EMS on a computer with EFI firmware, use the **BCDEdit /ems** command and specify a boot entry. Windows finds the out-of-band port and its settings in the firmware by reading the SPCR table and uses the same port and rate for EMS console redirection.

The following command enables EMS console redirection on the specified boot entry that has the identifier of {18b123cd-2bf6-11db-bfae-00e018e2b8db}.

```
bcdedit /ems {18b123cd-2bf6-11db-bfae-00e018e2b8db} on
```

Changing EMS Settings on a Computer with BIOS Firmware in Operating Systems prior to Windows Server 2008

When you configure EMS on a single boot entry, add the **redirect=** parameter to the [boot loader] section of the Boot.ini file. However, when you enable EMS on additional boot entries, you do not need to add the **redirect=** parameter again. Like all entries in the [boot loader] section, **redirect=** (and **redirectbaudrate=**) applies to all boot entries on the computer.

The following Bootcfg command enables EMS on the second boot entry. Because the port and baud rate are already set, there are no **/port** or **/baud** switches in the command.

```
bootcfg /ems ON /id 2
```

To change the port and baud rate settings, use the Bootcfg **/ems** switch with the EDIT argument. The following command changes the EMS port to COM1 and changes the baud rate to 57,600 Kbps.

```
bootcfg /ems EDIT /port COM1 /baud 57600
```

To disable EMS on a boot entry, use the Bootcfg **/ems** switch with the OFF argument. The following command disables EMS on the first boot entry.

```
bootcfg /ems OFF /id 1
```

If EMS is not enabled on any other boot entries, Bootcfg also deletes the EMS port and baud rate settings from the [boot loader] section of the Boot.ini file.

Changing EMS Settings on a Computer running Windows Server 2008

When you configure EMS on a boot entry on a computer that has ACPI BIOS firmware and an ACPI SPCR table, you can use the **BCDEdit /emssettings** command and specify either the **BIOS** option or the **emsport** and **emsbaudrate** options. If you use the **BIOS** option, do not set the **emsport** or **emsbaudrate** options.

When you configure EMS on a computer that has EFI firmware, or with ACPI BIOS firmware and without an ACPI SPCR table, you can use the **BCDEdit /emssettings** command and specify the **emsport** and **emsbaudrate** options.

The **emsport** and **emsbaudrate** options set the serial port and transmission rate for EMS console redirection. These settings apply to all boot entries on the computer. To use **emsbaudrate**, you must also set the **emsport** option. By default, the transmission rate is set to 9600 (9,600 Kbps).

For example, the following command changes the EMS port to COM2 and changes the baud to 57,600 Kbps.

```
bcdedit /emssettings EMSPORT:2 EMSBAUDRATE:57600
```

To enable or disable EMS on a boot entry, use the **BCDEdit /ems** command.

For example, the following command enables EMS on a specific boot entry that has an identifier of {173075c9-2cb2-11dc-b426-001558c41f5c}..

```
bcdedit /ems {173075c9-2cb2-11dc-b426-001558c41f5c} on
```

To disable EMS on the current boot entry, use the following command.

```
bcdedit /ems off
```

> **NOTE**
> Each boot entry uses a GUID as an identifier. If you do not specify an identifier, the **BCDEdit** command modifies the current operating system boot entry. If a boot entry is specified, the GUID associated with the boot entry must be enclosed in braces **{ }**. To view the GUID identifiers for all the active boot entries, use the **bcdedit /enum** command.

Boot Options in Previous Versions of Windows

8/23/2022 • 2 minutes to read • Edit Online

IMPORTANT

This topic describes the boot options supported in Windows XP and Windows Server 2003. If you are changing boot options for modern versions of Windows, see Boot Options in Windows Vista and Later.

This section includes:

- Boot Options in a Boot.ini File
- Boot Options in EFI NVRAM

Boot Options in a Boot.ini File

8/23/2022 • 2 minutes to read • Edit Online

IMPORTANT

This topic describes the boot options supported in Windows XP and Windows Server 2003. If you are changing boot options for Windows 8, Windows Server 2012, Windows 7, Windows Server 2008, or Windows Vista, see Boot Options in Windows Vista and Later.]

Boot.ini is a text file located at the root of the system partition, typically c:\Boot.ini. Boot.ini stores boot options for computers with BIOS firmware, traditionally, computers with IA-32-based and x64-based processors. On Windows Server 2003 and earlier versions of the Windows NT family of operating systems, when the computer starts, the Windows boot loader, called "ntldr", reads the Boot.ini file and displays the entries for each operating system in the boot menu. Then, ntldr loads the selected operating system in accordance with settings in the Boot.ini file.

By default, on NTFS drives, the **system**, **hidden**, **archived**, and **read-only** attributes are set to protect Boot.ini; however, members of the Administrators group can change these attributes. The file attributes do not affect the operation of the boot loader.

The following sections briefly describe Boot.ini and explain the aspects of boot options that are specific to computers with Personal Computer Advanced Technology (PC/AT)-type BIOS firmware.

This section includes:

- Overview of the Boot.ini File
- Editing the Boot.ini File
- Backing Up the Boot.ini File

This document describes aspects of Boot.ini that are of special interest to driver developers and testers. For a complete list of Boot.ini parameters, see Available Switch Options for the Windows XP and the Windows Server 2003 Boot.ini Files topic on the Microsoft Support website.

Mapping Boot.ini Options to BCDEdit Options and Elements

The following table provides a mapping from the boot options used in operating systems prior to Windows Vista (in Boot.ini), to the BCDEdit options and the BCD elements used in Windows. For information about the BCD boot elements with the context of WMI, see BCD WMI Provider Reference.

BOOT.INI	BCDEDIT OPTION	BCD ELEMENT TYPE
/3GB	**increaseuserva**	BcdOSLoaderInteger_IncreaseUserVa
/BASEVIDEO	**vga**	BcdOSLoaderBoolean_UseVgaDriver

BOOT.INI	BCDEDIT OPTION	BCD ELEMENT TYPE
/BOOTLOG	**bootlog**	BcdOSLoaderBoolean_BootLogInitialization
/BREAK	**halbreakpoint**	BcdOSLoaderBoolean_DebuggerHalBreakpoint
/CRASHDEBUG	**/dbgsettings /start**	
/DEBUG, BOOTDEBUG	**/debug** **/bootdebug**	BcdLibraryBoolean_DebuggerEnabled
/DEBUG	**/debug**	BcdOSLoaderBoolean_KernelDebuggerEnabled
/DEBUG, /DEBUGPORT=	**/dbgsettings**	BcdLibraryInteger_DebuggerType
/DEBUGPORT=	**/dbgsettings**	BcdLibraryInteger_SerialDebuggerPort BcdLibraryInteger_SerialDebuggerBaudRate BcdLibraryInteger_1394DebuggerChannel BcdLibraryString_UsbDebuggerTargetName BcdLibraryInteger_DebuggerNetHostIP BcdLibraryInteger_DebuggerNetPort BcdLibraryBoolean_DebuggerNetDhcp BcdLibraryString_DebuggerNetKey
/EXECUTE	**nx**	BcdOSLoaderInteger_NxPolicy
/FASTDETECT		
/HAL=	**hal**	BcdOSLoaderString_HalPath
/KERNEL=	**kernel**	BcdOSLoaderString_KernelPath
/MAXMEM=	**truncatememory**	BcdLibraryInteger_TruncatePhysicalMemory

BOOT.INI	BCDEDIT OPTION	BCD ELEMENT TYPE
/NODEBUG	**/debug**	
/NOEXECUTE	**nx** {	BcdOSLoaderInteger_NxPolicy
/NOGUIBOOT	**quietboot**	BcdOSLoaderBoolean_DisableBootDisplay
/NOLOWMEM	**nolowmem**	BcdOSLoaderBoolean_NoLowMemory
/NOPAE	**pae**	BcdOSLoaderInteger_PAEPolicy
/ONECPU	**onecpu**	BcdOSLoaderBoolean_UseBootProcessorOnly
/PAE	**pae**	BcdOSLoaderInteger_PAEPolicy
/PCILOCK	**usefirmwarepcisettings**	BcdOSLoaderInteger_UseFirmwarePciSettings
/REDIRECT	**/ems** **/emssettings** [**BIOS**] | [**EMSPORT:**{*port*} | [**EMSBAUDRATE:** {*baudrate*}]]	BcdOSLoaderBoolean_EmsEnabled
/SOS	**sos**	

Overview of the Boot.ini File

8/23/2022 • 2 minutes to read • Edit Online

IMPORTANT

This topic describes the boot options supported in Windows XP and Windows Server 2003. If you are changing boot options for modern versions of Windows, see Boot Options in Windows Vista and Later.

The Boot.ini file is a text file that contains the boot options for computers with BIOS firmware running NT-based operating system prior to Windows Vista. It is located at the root of the system partition, typically c:\Boot.ini. The following sample shows the content of a typical Boot.ini file.

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect
C:\CMDCONS\BOOTSECT.DAT="Microsoft Windows Recovery Console" /cmdcons
```

Boot.ini has two main sections:

- The **[boot loader]** section contains option settings that apply to all boot entries on the system. The options include **timeout**, the boot menu time-out value, and **default**, the location of the default operating system.

  The following sample shows the [boot loader] section of Boot.ini.

  ```
  [boot loader]
  timeout=30
  default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
  ```

- The **[operating systems]** section is comprised of one or more *boot entries* for each operating system or bootable program installed on the computer.

  A *boot entry* is a set of options that defines a load configuration for an operating system or bootable program. The boot entry specifies an operating system or bootable program and the location of its files. It can also include parameters that configure the operating system or program.

  The following sample shows the [operating systems] section of Boot.ini on a computer with two operating systems, Microsoft Windows XP and Microsoft Windows 2000. It has two boot entries, one for each operating system.

  ```
  [operating systems]
  multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect
  multi(0)disk(0)rdisk(0)partition(2)\WINNT="Microsoft Windows 2000 Professional" /fastdetect
  ```

Each boot entry includes the following elements:

- The location of the operating system. Boot.ini uses the Advanced RISC Computing (ARC) naming convention to display the path to the disk partition and directory where the operating system resides. For example:

```
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
```

- A friendly name for the boot entry. The friendly name represents the boot entry in the boot menu. The friendly name is surrounded by quotation marks and represents the boot entry in the boot menu. For example:

  ```
  "Microsoft Windows XP Professional"
  ```

- *Boot entry parameters*, also known as *boot parameters* or *load options* enable, disable, and configure operating system features. Boot parameters resemble command-line parameters, each beginning with a forward slash (/), such as **/debug**. You can have zero or more boot parameters on each boot entry.

  For a list of boot parameters that are relevant to driver testing and debugging, see Boot.ini Boot Parameter Reference.

You can have multiple boot entries for the same operating system, each with a different set of boot parameters. Windows creates a standard boot entry when you install the operating system, and you can create additional, customized entries for an operating system by editing Boot.ini.

Editing the Boot.ini File

8/23/2022 • 2 minutes to read • Edit Online

> **IMPORTANT**
> This topic describes the boot options supported in Windows XP and Windows Server 2003. If you are changing boot options for modern versions of Windows, see Boot Options in Windows Vista and Later.

Prior to Windows Vista, BIOS-based computers running Windows store boot options in a Boot.ini text file. You can edit Boot.ini using Bootcfg (`bootcfg.exe`), a tool included in Windows XP and Windows Server 2003, or using a text editor such as Notepad. Bootcfg is documented in Windows Help and Support. You can also view and change some boot options in Control Panel under System. In the System Properties dialog box, on the Advanced tab, select Settings under **Startup and Recovery**. Because this functionality is limited, it is not discussed in this section. For information about the **Startup and Recovery** dialog box, see Help and Support Center.

Bootcfg

Bootcfg is a command-line tool that can edit boot options on local and remote computers. Using the same Bootcfg commands and procedures, you can edit Boot.ini, as well as the boot options in Extensible Firmware Interface Non-Volatile Random Access Memory (EFI NVRAM). Bootcfg is included in the `%Systemroot%\\System32` directory in Windows XP and Windows Server 2003. (The Bootcfg display is slightly different on systems that store boot options in EFI NVRAM, but the commands are the same.)

You can use Bootcfg to add, delete, and change all boot entry parameters and boot options; however, you cannot use it to set an indefinite boot time-out value. You can also use Bootcfg commands in a script or batch file to set boot options or to reset them after you replace or upgrade an operating system.

Unlike manual editing, Bootcfg edits boot options without changing the protective attributes on Boot.ini. It also helps you avoid typing errors that might prevent the operating system from starting.

You must be a member of the Administrators group on the computer to use Bootcfg. For detailed instructions about using Bootcfg, see Help and Support Center.

Editing in Notepad

You can use a text editor, such as Notepad, to edit Boot.ini. However, because this method is prone to error, use it only when Bootcfg is not available.

Before editing Boot.ini, you must remove the file attributes that Windows uses to protect the file from inadvertent changes. When Boot.ini is on an NTFS volume, you must be a member of the Administrators group on the computer to change its attributes.

Use the following procedure to prepare Boot.ini for manual editing. This procedure removes the system, hidden, and read-only attributes of the file.

To configure Boot.ini attributes for editing

1. Open **Windows Command Prompt**.
2. Navigate to the root of the system volume.

3. Type the following text at the command line:

   ```
   attrib -s -h -r Boot.ini
   ```

   System, hidden, and read-only attributes are removed from the file.

4. Open the file in Notepad for editing. Since you are in Windows Command Prompt, the following command should do the trick quickly:

   ```
   notepad.exe Boot.ini
   ```

5. When your editing is complete, you can restore the file attributes to protect Boot.ini. However, Ntldr can use Boot.ini with any attribute set. To restore attributes, type the following in Windows Command Prompt:

   ```
   attrib +s +h +r Boot.ini
   ```

   This restores the attributes that protect the Boot.ini file.

Backing Up the Boot.ini File

8/23/2022 • 2 minutes to read • Edit Online

> **IMPORTANT**
> This topic describes the boot options supported in Windows XP and Windows Server 2003. If you are changing boot options for modern versions of Windows, see [Boot Options in Windows(boot-options-in-windows.md).

When you install or upgrade to Windows XP, Windows Server 2003 or one their predecessors, the Windows installer creates a new *Boot.ini* file for the computer. The new file retains some, but not all, of the changes you might have made to the file. Thus, to preserve an edited *Boot.ini* file, make a backup copy before upgrading or installing an operating system. After an update completes, you can replace the new file with your backup copy. If you have installed a new operating system, you can copy customized entries from your backup copy and then paste them into the new *Boot.ini* file.

An update or installation restores the default security attributes on Boot.ini, including the read-only attribute. To edit the file, use the Bootcfg command or change the file attributes. For more information, see Editing the Boot.ini File.

Boot Options in EFI NVRAM

8/23/2022 • 2 minutes to read • Edit Online

IMPORTANT

This topic describes the boot options supported in Windows XP and Windows Server 2003. If you are changing boot options for modern versions of Windows, see Boot Options in Windows.

Computers with Extensible Firmware Interface (EFI) firmware, such as Intel Itanium 2 processors, store boot options in NVRAM, a storage medium that can be edited, but retains its state even when you turn off the computer. EFI firmware serves the same purpose as BIOS firmware, but it overcomes many limitations of the traditional BIOS. The startup functions that are implemented in the BIOS and boot manager (NTLDR) on x86-based systems are handled by EFI components, namely the EFI BIOS and EFI Boot Manager.

To configure features related to driver debugging and testing on EFI-based systems running Windows XP, Windows Server 2003 and their predecessors, you must edit the boot options in NVRAM. The following sections briefly describe the boot options in EFI NVRAM and explain the aspects of boot options that are specific to systems that use this technology.

On Windows Vista and later versions of Windows, boot options on BIOS-based and EFI-based computers are stored in *Boot Configuration Data* (BCD), a firmware-independent configuration and storage system for boot options. For more information, see Boot Options in Windows Vista and Later.

For a detailed description of boot options on Itanium-based systems, see the Extensible Firmware Interface Specification. You can download a copy of the updated specification from the Intel Extensible Firmware Interface website.

This section includes:

- Overview of Boot Options in EFI
- Editing Boot Options in EFI
- Backing up Boot Options in EFI

Overview of Boot Options in EFI

8/23/2022 • 3 minutes to read • Edit Online

Like the boot options on a system with BIOS firmware, there are two types of boot options in EFI NVRAM:

- *Globally-defined variables* that apply to all bootable devices and bootable programs on the computer.
- *Boot option variables* that apply only to a particular load configuration of a bootable device or program, such as an operating system. The system-specific variables comprise a boot entry for each configuration of a bootable device or bootable program on the computer.

The Bootcfg tool discussed in Editing Boot Options in EFI allows you to view and edit the boot options in EFI NVRAM.

The following sample shows a Bootcfg display of a computer with an Itanium processor.

```
Boot Options
------------
Timeout:             30
Default:             \Device\HarddiskVolume3\WINDOWS
CurrentBootEntryID:  1

Boot Entries
------------

Boot entry ID:    1
OS Friendly Name: Windows Server 2003, Enterprise
OsLoadOptions: /debug /debugport=COM1 /baudrate=57600
BootFilePath:     \Device\HarddiskVolume1\EFI\Microsoft\WINNT50\ia64ldr.efi
OsFilePath:       \Device\HarddiskVolume3\WINDOWS

Boot entry ID:    2
OS Friendly Name: EFI Shell [Built-in]
```

The following table describes the elements of the Bootcfg display of boot data in EFI NVRAM.

FIELD	DESCRIPTION	EXAMPLE
Boot Options	Contains options that apply to all boot entries.	(Section heading)
Timeout	Determines how long the boot menu is displayed. When this value expires, the boot loader loads the default operating system.	`Timeout:  30`
Default	Specifies the location of the default operating system.	`\Device\HarddiskVolume3\WINDOWS`

FIELD	DESCRIPTION	EXAMPLE
CurrentBootEntryID	Identifies the boot entry that was used to start the current session of the operating system.	`CurrentBootEntryID:  1`
Boot Entries	Contains system-specific data. It is comprised of one or more *boot entries* for each operating system or bootable program installed on the computer. A *boot entry* is a set of options that defines a load configuration for an operating system or bootable program.	(Section heading)
Boot entry ID	Identifies the boot entry to Bootcfg. This value changes when you reorder the boot entries. This is not the *EFI boot entry ID*, which is a persistent identifier for the EFI components.	`Boot entry ID:    1`
OS Friendly Name	Represents the boot entry in the boot menu.	`Windows Server 2003, Enterprise`
OsLoadOptions	Specifies the *boot parameters* for the entry. *Boot parameters* are commands to enable, disable, and configure features of the operating system. The EFI Boot Manager passes these parameters to the bootable device or system to be interpreted and implemented. For a list of the boot parameters that are related to driver debugging and testing, see Boot Options in a Boot.ini File.	`OsLoadOptions: /debug /debugport=COM1 /baudrate=57600`

FIELD	DESCRIPTION	EXAMPLE
BootFilePath	Specifies the location of the EFI boot loader for the operating system. On EFI-based systems, each operating system or bootable device has its own copy of the boot loader on the EFI partition. In EFI NVRAM, the boot loader file path is stored as a binary EFI device path that uses a globally unique identifier (GUID) to identify the EFI partition . Bootcfg uses the NT device name of the partition in its path display.	`BootFilePath: \Device\HarddiskVolume1 \EFI\Microsoft\WINNT50\ia64ldr.efi`
OsFilePath	Specifies the location of the operating system. In NVRAM, this value is stored as an EFI device path that uses the GUID of the boot disk partition and the path to the directory that contains the operating system. Bootcfg uses the NT device name of the partition in its path display.	`OsFilePath: \Device\HarddiskVolume3 \WINDOWS`

In addition, there is an important element of an EFI boot entry that Bootcfg does not display, the *EFI boot entry ID*. The EFI boot entry is a unique identifier for an EFI boot entry. This identifier is assigned when the boot entry is created, and it does not change. It represents the boot entry in several lists, including the *BootOrder* array, and it is the name of the directory on disk in which the system stores data related to the boot entry, including backup copies of the boot entry. An EFI boot entry ID has the format, Boot*xxxx*, where *xxxx* is a hexadecimal number that reflects the order in which the boot entries are created.

> **NOTE**
>
> The **Boot entry ID** field in Bootcfg and the boot entry number in Nvrboot do not display the EFI boot entry ID. The Bootcfg and Nvrboot IDs are line numbers that represent the order of the boot entry in the **Boot Entries** section and change when the entries are reordered.

For a detailed description of boot options on Itanium-based systems, see the Extensible Firmware Interface Specification. You can download a copy of the specification from the Intel Extensible Firmware Interface website.

Editing Boot Options in EFI

8/23/2022 • 2 minutes to read • Edit Online

To edit boot options on computers with EFI NVRAM that are running Windows Server 2003 or earlier versions of NT-based Windows, use Bootcfg (bootcfg.exe), a tool that runs on Windows, or Nvrboot (nvrboot.efi), a tool that runs in the EFI environment. Both tools are included in the Windows XP 64-Bit Edition and the 64-bit version of Windows Server 2003.

You can also view and change some boot options in Control Panel under System. In the System Properties dialog box, on the Advanced tab, select Settings under **Startup and Recovery**. Because this functionality is limited, it is not discussed in this section. For information about the **Startup and Recovery** dialog box, see Help and Support Center.

Bootcfg

Bootcfg (bootcfg.exe) is a command-line tool that edits boot options on a local or remote computer Using the same Bootcfg commands and procedures, you can edit a Boot.ini file or the boot options in EFI NVRAM. Bootcfg is included in the %Systemroot%\System32 directory in Windows XP and Windows Server 2003. (The Bootcfg display is slightly different on systems that store boot options in EFI NVRAM, but the commands are the same.)

You can use Bootcfg to add, delete, and change the values of all valid boot options; however, you cannot set an indefinite time-out value. You can also use Bootcfg commands in a script or batch file to set boot options or to reset them after you replace or upgrade an operating system.

On systems that store boot options in EFI NVRAM, Bootcfg can also display the boot partition table, add boot entries for mirrored drives, and update the GUIDs for a system partition.

You must be a member of the Administrators group on the computer to use Bootcfg. For detailed instructions about using Bootcfg, see Help and Support Center.

Nvrboot

Nvrboot (nvrboot.efi) is an EFI-based boot entry editor included in Windows XP 64-Bit Edition and the 64-bit version of Windows Server 2003. Nvrboot runs in the EFI environment. You cannot run Nvrboot while an operating system is running.

Nvrboot edits only boot entries. You cannot use it to display or change the time-out value for the boot menu, although, you can use the **push** command (**nvrboot p**) to change the default boot entry.

Nvrboot also includes commands to export backup copies of boot entries and to import backup copies of boot entries into NVRAM. This procedure is discussed in the Backing up Boot Options in EFI section.

Nvrboot displays boot options in a user-friendly format. For example, it displays the operating system file path and the boot loader file path as a partition GUID followed by a Windows directory path.

The following procedure explains how to start Nvrboot from the EFI shell, a tool provided with many Itanium-based systems. Because the EFI shell tools vary among manufacturers, the description in this section might not accurately describe the EFI shell interface on a particular computer.

To run Nvrboot

1. Reboot the computer.
2. From the **boot** menu, choose **EFI Shell**.

3. At the shell prompt, type the drive letter or file system number of the system partition, such as C: or **FS**n, where n is the file system number of the system partition.
4. Type **cd msutil** to navigate to the Msutil directory where nvrboot.efi is located.
5. To start Nvrboot, type **nvrboot**.

To find instructions for Nvrboot, type **h**.

Backing up Boot Options in EFI

8/23/2022 • 2 minutes to read • Edit Online

When you install a 64-bit version of Windows, Setup automatically creates a boot entry for the installation and saves a backup copy of the boot entry to a binary file named Boot*xxxx*, where Boot*xxxx* is the NVRAM ID for the boot entry. Setup stores the boot entry backup copy in a new directory on the EFI partition, along with the EFI boot loader for the new installation.By default, the installation directory is in the \EFI\Microsoft\ subdirectory.

However, the system does not save backup copies of the boot entries that you create and it does not update backup copies of boot entries after you edit them.

To save backup copies of boot entries that you create and edit, and to save extra backup copies of the entries that Setup creates, use Nvrboot (nvrboot.efi). Nvrboot saves the entries in the binary format that Setup and the EFI components use. Then, if the boot entry for an installation is lost or corrupted, you can use the import command (**nvrboot i**) in Nvrboot to import the backup copy of the boot entry into NVRAM.

This section includes:

- Exporting and Importing Boot Entries in EFI
- Identifying Backup Files for Existing Boot Entries
- Identifying Backup Files for Deleted Boot Entries
- Recognizing Unusable Boot Entry Backup Files

Exporting and Importing Boot Entries in EFI

8/23/2022 • 2 minutes to read • Edit Online

Use the **nvrboot x** (export) command to create a backup copy of a boot entry. Design a naming and storage convention that will make it easy to find the backup copy files when you need them.

Use the **nvrboot i** (import) command to import boot entries from the backup copies that you exported or from the Boot*xxxx* boot entry backup files that Setup created.

Imported boot entries always receive new EFI boot entry IDs. For example, if you export a copy of Boot0003, and then import the copy into NVRAM, the imported boot entry receives a new boot entry ID, such as Boot000A.

Identifying Backup Files for Existing Boot Entries

8/23/2022 • 2 minutes to read • Edit Online

To search for the backup copy of a boot entry by its file name, you need the entry's EFI boot entry ID. However, neither Bootcfg nor Nvrboot display this ID.

Instead, you can find a boot entry backup copy by searching for a Boot*xxxx* file in the installation's directory on the EFI partition. To find the installation directory, locate the path to the boot loader file for the operating system installation. The boot entry backup file for the installation is stored in the same directory.

Use the **nvrboot d** (display) command or the **bootcfg** or **bootcfg query** commands to display the path to the directory in which the boot loader for the operating system is stored.

In the following example, the boot loader for a boot entry is stored on the EFI partition in the \Microsoft\WINNT50 subdirectory. The backup copy of the boot entry for this installation is a file named Boot*xxxx* in the same subdirectory.

> **NOTE**
> The **Boot entry ID** field in Bootcfg and the boot entry number in Nvrboot do not display the EFI boot entry ID. The Bootcfg and Nvrboot IDs are line numbers that represent the order of the boot entry in the **Boot Entries** section and change when the entries are reordered.

As shown in the following Bootcfg sample, the path to the boot loader file appears in the **BootFilePath** field.

Bootcfg displays the file location as the NT device name of the partition, followed by the file system path to the boot loader file.

```
Boot Entries
------------
Boot entry ID:    1
OS Friendly Name: Windows Server 2003, Enterprise
OsLoadOptions:     /debug /debugport=COM1 /baudrate=115200
BootFilePath:     \Device\HarddiskVolume1\EFI\Microsoft\WINNT50\ia64ldr.efi
OsFilePath:       \Device\HarddiskVolume3\WINDOWS
```

In the following sample of an Nvrboot display, the path to the boot loader file for an operating system installation appears in the **EFIOSLoaderFilePath** field.

Nvrboot displays the file location as a partition GUID followed by the path to the boot loader file.

```
1. Load identifier = Windows Server 2003, Enterprise
2. OsLoadOptions = /debug /debugport=COM1 /baudrate=115209
3. EFIOSLoaderFilePath = 006F0073-0066-0074-5C00-570049004E00  ::  \EFI\Microsoft\WINNT50\ia64ldr.efi
4. OSLoaderFilePath = 04000004-5D18-3F27-0000-0000205C273F  :: \Windows
```

In both cases, the boot loader file (and the Boot*xxxx* boot entry backup file) for the operating system are in the WINNT50 directory of the EFI system partition (EFI\Microsoft\WINNT50).

Identifying Backup Files for Deleted Boot Entries

8/23/2022 • 2 minutes to read • Edit Online

Typically, you need to locate a boot entry backup file when a boot entry is inadvertently deleted.

If a boot entry has been deleted from NVRAM, and the operating system is still installed, the boot loader file and the boot entry backup file for the installation still remain on the disk in the installation's directory on the EFI partition.

To find the boot entry backup file for a deleted entry, boot to the EFI shell, and search the EFI partition recursively for boot entry backup files using the command **dir boot* /s**. Exclude from your results boot entry backup files that are in directories associated with boot entries already in NVRAM. To display the directory for an existing boot entry, use the **nvrboot d** (display) command.

If there are multiple Boot*xxxx* files that are not associated with existing boot entries, use Nvrboot to import the entries from their backup files, and then delete the unwanted boot entries.