Boot Configuration Data in Windows Vista

Feb. 4, 2008

Abstract
Microsoft has completely reengineered the boot environment for Windows Vista to address the increasing complexity and diversity of modern hardware and firmware. One aspect of this reengineering is a new firmware-independent data store that contains configuration data that influences the boot process. This paper provides an overview of this configuration data—called boot configuration data (BCD)—and describes how to use the related tools to manage boot options.

This information applies to the Windows Vista operating system.

The current version of this paper is maintained on the Web at:
http://www.microsoft.com/whdc/system/platform/firmware/bcd.mspx

References and resources discussed here are listed at the end of this paper.

Contents

Document History

Date	Change			
2/04/08	Removed reference to Longhorn			
5/18/2007	Created			

Introduction

When a computer is started or rebooted, it must load the operating system. The details of this process vary depending on the system's hardware and firmware and whether the system is booted from a disk drive, a network, or some other source.

For the common scenario of a PC/AT BIOS computer that is booting from its hard drive, the BIOS reads the master boot record (MBR) and transfers control to the MBR's code. MBR, in turn, transfers control to the code that loads the operating system. Historically, the primary application that is responsible for loading Windows® has been Ntldr.

The data that has determined how Ntldr loaded Windows has been contained in a text file that is named boot.ini and resides on the root folder of the boot drive. Boot.ini contains a separate *boot entry* for each version or configuration of Windows that is available to the user. If multiple configurations or versions of the operating system are available, Ntldr displays the list of boot entries to allow the user to specify which one should be loaded. It then proceeds to load the selected version of the operating system with a configuration that is based on the selected entry's *boot options*.

The boot process for computers that use Extensible Firmware Interface (EFI) firmware is completely different from that for PC/AT BIOS computers. EFI is the next-generation firmware model that serves as the interface between hardware platform and the operating system. It provides information about the platform that is necessary for the operating system to boot and is expected to replace the legacy BIOS in the coming decade.

The firmware on a computer that uses EFI contains a boot manager that loads an operating system EFI application that is based on variables that are stored in non-volatile RAM (NVRAM). The Windows EFI operating system loader does not use boot.ini at all. For further information on EFI, see the white paper titled “EFI and Windows Vista.”

Windows Vista® introduces boot configuration data (BCD). This new data store serves essentially the same purpose as boot.ini. However, BCD abstracts the underlying firmware and provides a common programming interface to manipulate the boot environment for all Windows-supported computer platforms. BCD currently supports PC/AT BIOS and EFI systems. However, its programming interface is extensible and portable and has the ability to support other types of firmware in addition to the two discussed here.

Windows Vista introduces several new boot applications, including:

- Bootmgr: A system-wide application that controls boot flow. With a multiboot system, the boot manager displays an operating system selection menu.
- Winload.exe: The Windows Vista operating system loader. Each version of Windows Vista and Windows Server® 2008 that is installed on a computer has its own instance of winload.exe. The operating system loader creates the execution environment for the operating system and also loads the Windows Vista kernel, hardware abstraction layer (HAL), and boot drivers into memory.
- Winresume.exe: The Windows Vista *resume loader*. Each version of Windows Vista and Windows Server 2008 that is installed on a computer has its own instance of winresume.exe. The resume loader restores Windows to its running state when a computer resumes from hibernation.

NTLDR can still be used on PC/AT BIOS systems to dual boot a Windows version earlier than Windows Vista.

This white paper includes:

- The architecture of BCD.
- How to manage the boot environment and BCD with system tools.
- How to manage the boot environment and BCD programmatically through the BCD Windows Management Instrumentation (WMI) provider.
- A “cookbook” that shows how to do a number of common operations with BCDEdit and the BCD WMI interface.

BCD Overview

BCD provides a firmware-independent mechanism for manipulating boot environment data for any type of Windows system. Windows Vista and later versions of Windows will use it to load the operating system or to run boot applications such as memory diagnostics. Some key characteristics include:

- BCD abstracts the underlying firmware. BCD currently supports both PC/AT BIOS and EFI systems. BCD interfaces perform all necessary interaction with firmware. For example, on EFI systems, BCD creates and maintains EFI NVRAM entries.
- BCD provides clean and intuitive structured storage for boot settings.
- BCD interfaces abstract the underlying data store.
- BCD is available at run time and during the boot process.
- BCD manipulation requires elevated permissions.
- BCD is designed to handle systems with multiple versions and configurations of Windows, including versions earlier than Windows Vista. It can also handle non-Windows operating systems.
- BCD is the only boot data store that is required for Windows Vista and later versions of Windows. BCD can describe NTLDR and the boot process for loading of earlier versions of Windows, but these operating systems are ultimately loaded by Ntldr and must still store their boot options in a boot.ini file.

Note: If a system includes earlier versions of Windows along with Windows Vista, the earlier versions should be installed first.

There are two approaches to modifying the settings that are contained in BCD:

- Users can interact with BCD through several tools. The details of what can be modified depend on the particular tool.
- Developers can programmatically manipulate a BCD store through the BCD WMI provider. The WMI provider supports a unified programming interface that can be used for both local and remote management of BCD stores. The interface is independent of the underlying firmware, so developers can write one application that works on any type of system.

Note: BCD’s data store is a registry hive, but that hive should not be accessed with the registry API. Interaction with the underlying firmware occurs in the supported BCD interfaces. For this reason, BCD stores should be accessed only through the associated tools or WMI API.

BCD Architecture

The BCD architecture is a hierarchy composed of three basic components: stores, objects, and elements.

- A *BCD store* is the top-level component in the hierarchy. It serves as a namespace container for the BCD objects and elements that make up the contents of the store.
- A *BCD object* is a container of BCD elements. The most common type of BCD object describes a boot environment application, such as an instance of the Windows boot loader. However, BCD objects are also used for other purposes.
- A *BCD element* is a singular item of data such as a debugger setting, a boot application name, or an operating system device.

Figure 1 is a schematic illustration of the BCD hierarchy.

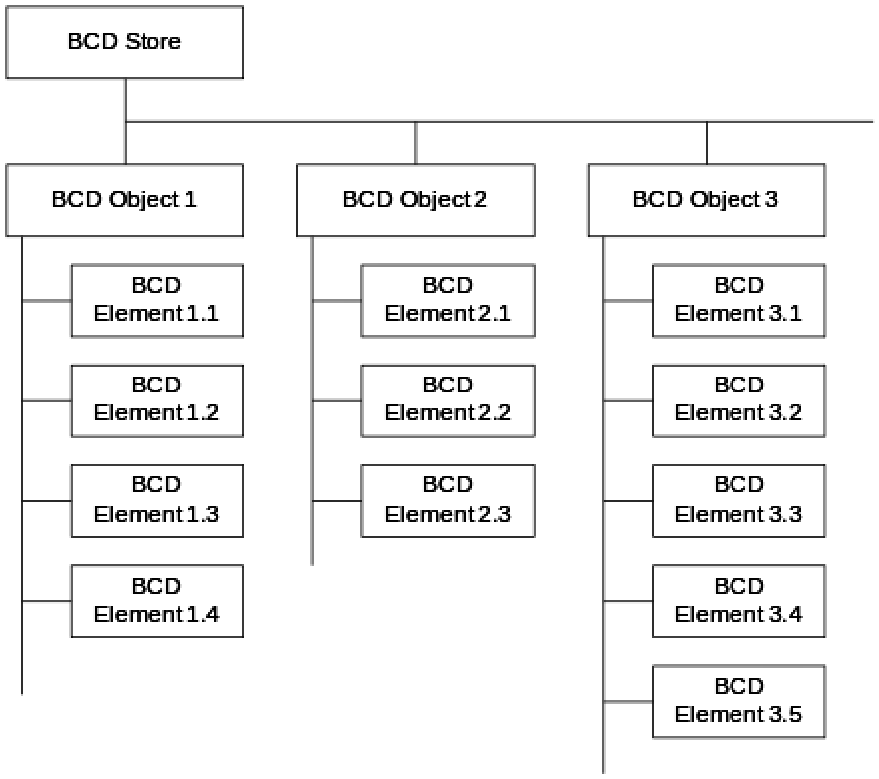


Figure 1. The BCD hierarchy

BCD Stores

A BCD store is a namespace container for BCD objects and elements that holds the information that is required to load Windows or run other boot applications. Physically, a BCD store is a binary file in the registry hive format. A computer has a system BCD store that describes all installed Windows Vista operating systems and installed Windows boot applications. A computer can optionally have many non-system BCD stores. The characteristics of BCD stores include:

- The *system store* is a registry hive whose file is named BCD. On PC/AT BIOS systems, the file resides in the active partition's \boot folder. On EFI systems, the file is located on the EFI system partition (ESP) under \EFI\Microsoft\Boot.

- The *system store* is used by the Windows boot manager to control boot flow. With a multiboot system, it presents a selection menu to the user.
- BCD has two interfaces: the BCD WMI provider and BCDedit.exe. Both interfaces abstract the location of the system store. BCDedit.exe operates on the system store unless a specific store is specified. With the BCD WMI API, the system store is specified by an empty string ("").
- Administrators or support professionals can create additional BCD stores with BCDEdit or programmatically with the BCD WMI API. Additional stores can be useful for recovery, repair, and imaging.
- Administrators or support professionals can use BCDEdit or the WMI API to import a non-system store as the system store.

Figure 2 shows an example of how the BCD hierarchy is implemented in a typical BCD store.

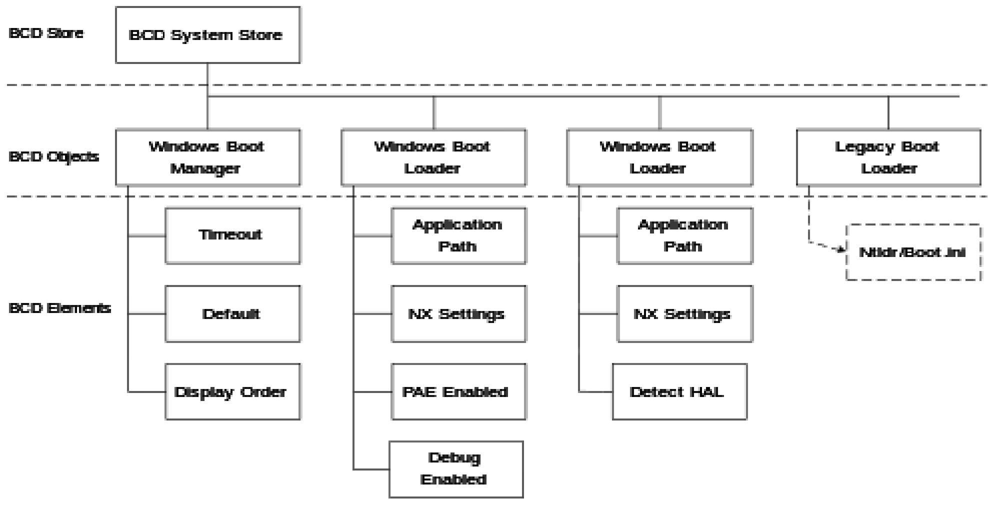


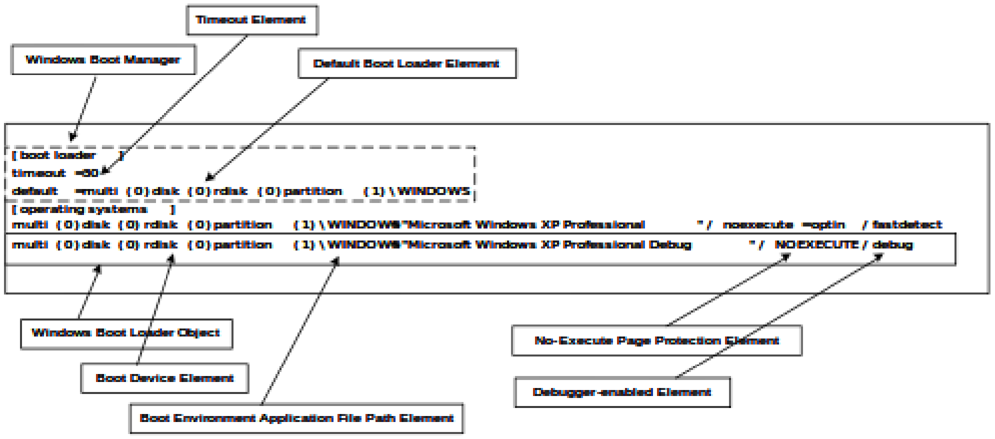
Figure 2. A typical BCD store

A BCD store normally has at least two and optionally many BCD objects.

- A *Windows boot manager* object. This object contains BCD elements that pertain to the Windows boot manager, such as the entries to display in an operating system selection menu, boot tool selection menu, and timeout for the selection menus. The Windows boot manager object and its associated elements serve essentially the same purpose as the [boot loader] section of a boot.ini file. A store can optionally have multiple instances of the Windows boot manager. However, only one of them can be represented by the Windows boot manager's well-known globally unique identifier (GUID). The GUID's alias, {bootmgr} can be used to manipulate a store with BCDEdit.
- At least one and optionally several *Windows boot loader* objects. Stores contain one instance of this object for each version or configuration of Windows Vista or Windows Server 2008 that is installed on the system. These objects contain BCD elements that are used when loading Windows or during Windows initialization such as no-execute (NX) page protection policy, physical-address extensions (PAEs) policy, kernel debugger settings, and so on. Each object and its associated elements serve essentially the same purpose as one of the lines in the [operating systems] section of boot.ini. When a computer is booted into Windows Vista, the associated boot loader object is represented by the alias {current}. When manipulating a store with BCDEdit, the default boot loader object has the alias {default}.

- An optional *Windows Ntldr object*. The Ntldr object describes the location of Ntldr, which can be executed to boot earlier versions of Windows. It is required only if the system includes versions of Windows that are earlier than Windows Vista. It is possible to have multiple instances of objects that describe Ntldr. However, as with the Windows boot manager, only one instance can be represented by Ntldr's well-known GUID. The GUID's alias, {ntldr} can be used to manipulate a store with BCDEdit.
- Optional boot applications. Stores can optionally have BCD objects that perform other boot-related operations. One example is the *Windows Memory Tester*, which runs memory diagnostics.

For comparison, Figure 3 shows the contents of a typical boot.ini file and how the boot.ini entries correspond to BCD objects and elements.


Timeout Element
Windows Boot Manager
Default Boot Loader Element
Windows Boot Loader Object
Boot Device Element
Boot Environment Application File Path Element
No-Execute Page Protection Element
Debugger-enabled Element

```
[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" /noexecute=optin /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional Debug" /NOEXECUTE /debug
```

Figure 3. The relationship between boot.ini and BCD

Note: Figure 3 uses descriptive labels for BCD objects and elements. In practice, they are represented by names that depend on the particular tool. Some commonly used names are given later.

BCD Objects

There are three general categories of BCD objects: *application objects*, *inheritable objects*, and *device objects*. The most common type of BCD object is an application object, which describes a boot environment application such as the Windows boot manager or Windows boot loader. Each object is represented by a 128-bit unique GUID and contains a 32-bit description that describes the type of object.

The following table shows the object codes and associated object code values for the three object categories.

Object Types

Description	Value
Application	0x1
Inheritable	0x2
Device	0x3

Figure 4 shows the layout of the type. The details of how data is packed into bits 0 through 27 depends on the category.

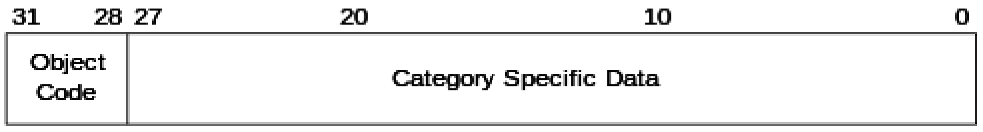


Figure 4. Layout of the BCD object type.

BCD Application Objects

A BCD application object represents a boot environment executable such as the Windows boot loader. Standard application objects include:

- The Windows boot manager object, which controls boot flow. In a dual-boot system, the Windows boot manager displays a boot selection menu to the user.
- The Windows boot loader object, which loads a particular version or configuration of Windows Vista or later versions of Windows.
- The Windows Ntldr object, which loads versions of Windows earlier than Windows Vista.
- The Windows resume loader object, which restores Windows to its running state when a computer resumes from hibernation.
- The Windows Memory Tester *o*bject, which runs a set of memory diagnostics.

BCD application objects have two defining characteristics: *image type* and *application type.* Image type specifies how the executable is loaded. For example, an executable can be loaded through the firmware or by the Windows boot manager (as a boot application). The following table lists the valid image types, along with the associated numerical value:

Image Types

Description	Value
Firmware application	0x1
Boot application	0x2
Ntldr-based loader object	0x3
Real-mode application	0x4

Application type specifies what the application does. Each valid type has an associated code. The following table lists the standard application types, along with their numerical codes:

Application Types

Description	Value	Description
Firmware boot manager	0x1	Applies only to EFI systems.
Windows boot manager	0x2	Controls boot flow. In a dual-boot system, displays a boot selection menu to the user.
Windows boot loader	0x3	Loads a particular version or configuration of Windows.
Windows resume application	0x4	Restores Windows to its running state when a computer resumes from hibernation.
Windows memory tester	0x5	A memory diagnostics application.
Ntldr	0x6	Applies only to PC/AT BIOS systems. Loads versions of Windows earlier than Windows Vista.
Boot sector	0x8	A 16-bit real-mode application. Applies only to PC/AT BIOS systems. Can be used to restart the boot process and load a non-Windows operating

Description	Value	Description
		system.

Figure 5 shows how the layout of an object type.

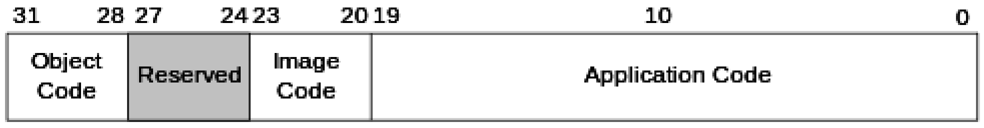


Figure 5. Layout of the BCD application object type

The following table gives the BCDEdit identifier and GUID for some commonly used application objects:

Standard Application Objects

Description	BCDEdit ID	GUID
Windows Boot Manager	{bootmgr}	9dea862c-5cdd-4e70-acc1-f32b344d4795
Firmware Boot Manager	{fwbootmgr}	a5a30fa2-3d06-4e9f-b5f4-a01df9d1fcba
Windows Memory Tester	{memdiag}	b2721d73-1db4-4c62-bf78-c548a880142d
Windows Resume Application	No alias	147aa509-0358-4473-b83b-d950dda00615
Legacy Windows Loader	{ntldr}	466f5a88-0af2-4f76-9038-095b170dc21c
Current boot entry	{current}	fa926493-6f1c-4193-a414-58f0b2456d1e
Default boot entry	{default}	

BCD Inheritable Objects

Some BCD elements can be applied to more than one BCD application object, and a few are global to the entire BCD store. It is possible to associate these elements separately with each instance of an object that uses them. A more efficient approach is to create an inheritable object. This is a container for elements that are shared across multiple BCD object instances. For example, there are inheritable objects to specify whether the kernel debugger uses a COM, USB, or 1394 connection. A BCD object includes the *inheritable object* that contains the element instead of having a separate instance of the element itself.

As described in the BCD element section, BCD element namespace is divided so that elements that are used by two different application classes can share the same type code. This means that an inheritable application class object can be associated only with a particular class of BCD application objects. Alternatively, an inheritable BCD object can contain only BCD elements that apply to all boot environment applications. Such objects can be included by any BCD application class. The two types of inheritable object are distinguished by their class. The numerical class code is given in parentheses:

- Library class (0x1). Inheritable objects of this type can be inherited by any BCD object and can contain only library class BCD elements.
- Application class (0x2). Inheritable objects of this type can be inherited only by the specified BCD application class.

Inheritable objects that have a class code set to Application class must also include the type of application objects that can include the inheritable object. Possible values include any of the BCD application object types, such as Windows boot loaders or Windows boot manager.

The following table gives the BCDEdit identifier and GUID for some commonly used inheritable objects:

Standard Inheritable Objects

BCDEdit ID	GUID	Description
{badmemory}	5189b25c-5558-4bf2-bca4-289b11bd29e2	Global RAM defect list that can be inherited by any boot application.
{bootloadersettings}	6efb52bf-1766-41db-a6b3-0ee5eff72bd7	Global settings that should be inherited by all Windows boot loader applications.
{dbgsettings}	4636856e-540f-4170-a130-a84776f4c654	Global debugger settings that can be inherited by any boot application.
{emssettings}	0ce4991b-e6b3-4b16-b23c-5e0d9250e5d9	Global Emergency Management Services settings that can be inherited by any boot application.
{globalsettings}	7ea2e1ac-2e61-4728-aaa3-896d9d0a9f0e	Global settings that should be inherited by all boot applications.
{resumeloadersettings}	1afa9c49-16ab-4a5c-901b-212802da9460	Global settings that should be inherited by all resume applications.

The inherited type has the following layout. The contents of the value field depend on the class code:

- Library. Value is not used.
- Application. Value specifies the type of application that can inherit from the object. It should be set to one of the application codes that were listed earlier.

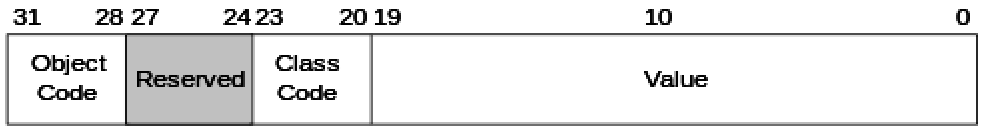


Figure 6. Layout of the BCD inherited object type

BCD Device Objects

Most devices, such as a partition on a hard disk, can be described by a single BCD element. However, more complicated devices could exist that require more than one element to describe. A BCD device object is a container for BCD elements for a complex device.

For example, a BCD device object is used when booting from a RAM disk that was created from a Windows image (WIM) file. The device object contains the location of the WIM file and, when booted over the network, the network port information. The BCDEdit identifier for this object is {ramdiskoptions}. The associated GUID is {ae5534e0-a924-466c-b836-758539a3ee3a}). It is also possible to create custom device objects.

The device type has the following layout.

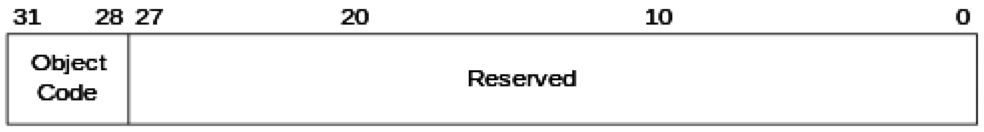


Figure 7. Layout of the BCD device object type

BCD Elements

An element is an item of configuration data for a boot environment application or part of the Windows boot process. With boot.ini, these properties and their values were specified as boot options. With BCD, each item of input data is contained in a separate BCD element.

BCD elements are contained within a BCD object. A boot environment application has a BCD application object that contains BCD elements to specify the configuration properties for the application. A BCD application object can also include inheritable objects that contain additional configuration data.

Some elements can be associated only with certain objects, whereas others can be applied to any type of boot environment application. To manage the different types of elements, the element namespace is divided into three classes. The following table shows the classes and their associated numerical code.

Element Classes

Description	Value	Description
Library	0001	Elements can be applied to all boot environment applications.
Application	0010	Elements can be applied to only a particular class of boot environment applications, such as boot loaders.
Device	0011	Elements can be included only by device objects.

BCD elements are structured data. The following table shows the formats and their associated numerical code.

Element Formats

Description	Value	Description
String	0010	For example, kernel debugger on or off.
Object	0011	For example, NX policy.
Integer	0101	For example, the boot menu default value.
Boolean	0110	For example, the path to the operating system loader.

The techniques for adding, deleting, or modifying BCD elements depend on the particular tool.

BCDEdit and the BCD WMI API provide well-known names for the standard elements. However, for custom element types, Figure 8 shows the layout.

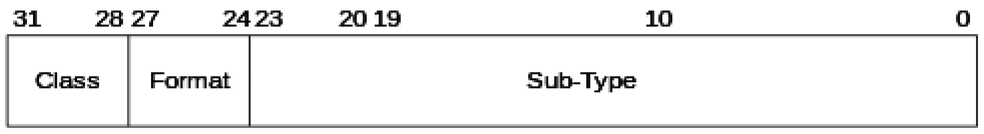


Figure 8. Layout of the BCD element type

Tools for Managing the BCD

The BCD store is in a binary format and can be modified only by tools that are designed for that purpose. The first two discussed here are designed for regular users and provide limited access to BCD. The BCDEdit tool is designed for developers and support professionals. It provides fairly complete access to BCD, including the ability to create BCD stores. The most flexible and powerful approach to managing BCD stores is programmatically, with the BCD WMI API. This API is discussed later in this paper.

The Shell

The **System** Control Panel application allows the user to specify values for two global BCD elements:

- The default operating system and configuration
- The boot manager's timeout setting

MSConfig

MSConfig.exe is used primarily by product support service (PSS) when it assists users with boot settings. It supports BCD and allows the user to enumerate the loader-type objects in the display order along with selected elements. MSConfig can also be used to modify selected elements, including the debug and safe-mode settings.

BCDEdit

BCDedit is a command-line tool that support professionals and developers can use to manage BCD stores. It can be used for a variety of purposes, including creating new stores, modifying existing stores, adding boot menu options, and so on. BCDEdit serves essentially the same purpose as Bootcfg.exe on earlier versions of Windows, but with two major improvements:

- It exposes a wider range of boot options than Bootcfg.exe.
- It has improved scripting support.

Note: Administrative privileges are required to use BCDEdit to modify BCD.

BCDEdit is the only boot configuration editor that can be used to edit the boot configuration for Windows Vista and later versions of Windows. It is included with the Windows Vista distribution in the %WINDIR%\System32 folder. However, BCDEdit can also be used on earlier versions of Windows. Bootcfg.exe is also included with Windows Vista, but it can be used only to edit the boot configuration for earlier versions of Windows that might also be installed on the computer.

BCDEdit is limited to the standard data types and is designed primarily to perform single common changes to BCD. For more complex operations or nonstandard data types, consider using the BCD WMI API to create more powerful and flexible custom tools. The command syntax for BCDEdit is straightforward:

```
BCDEdit /Command [Argument1] [Argument2] ...
```

The following list gives the BCDEdit commands plus a brief description. Unless otherwise specified, BCDEdit operates on the system store by default. For examples of how to use BCDEdit for common tasks, see “BCD Cookbook,” later in this paper. For complete details, see the online reference.

General Commands

- /?. Displays a list of BCDEdit commands. Running this command without an argument displays a summary of the available commands. To display detailed help for a particular command, run "bcdedit /? *command*", where *command* is the name of the desired command without the leading forward slash (/). For example, "bcdedit /? createstore" displays detailed help for the createstore command.

Commands that Operate on a Store

- /createstore. Creates a new empty boot configuration data store. The created store is not a system store.
- /export. Exports the contents of the system store into a file. This file can be used later to restore the state of the system store. This command is valid only for the system store.
- /import. Restores the state of the system store by using a backup data file previously generated by using the /export command. This command deletes any existing entries in the system store before the import takes place. This command is valid only for the system store.
- /store. This option can be used with most BCDedit commands to specify the store to be used. If this option is not specified, then BCDEdit operates on the system store. Running `bcdedit /store` by itself is equivalent to running `bcdedit /enum ACTIVE.`

Commands that Operate on Entries in a Store

- /copy. Makes a copy of a specified boot entry in the same system store.
- /create. Creates a new entry in the boot configuration data store. If a well-known identifier is specified, then the /application, /inherit, and /device options cannot be specified. If an identifier is not specified or not well known, a /application, /inherit, or /device option must be specified.
- /delete. Deletes an element from a specified entry.

Commands that Operate on Entry Options

- /deletevalue. Deletes a specified element from a boot entry.
- /set. Sets an entry option value.

Commands that Control Output

- /enum. Lists entries in a store. The /enum command is the default value for BCDEdit, so running "bcdedit" without parameters is equivalent to running `bcdedit /enum ACTIVE`.
- /v. Verbose mode. Usually, any well-known entry identifiers are represented by their friendly shorthand form. Specifying /v as a command-line option displays all identifiers in full. Running "bcdedit /v" by itself is equivalent to running `bcdedit /enum ACTIVE /v`.

Commands that Control the Boot Manager

- /bootsequence. Specifies a one-time display order to be used for the next boot. This command is similar to /displayorder, except that it is used only the next time the system is booted. After that has occurred, the system reverts to the original display order.
- /default. Specifies the default entry that the boot manager selects when the timeout expires.
- /displayorder. Specifies the display order that the boot manager uses when displaying boot options to a user.
- /timeout. Specifies the time to wait, in seconds, before the boot manager selects the default entry.
- /toolsdisplayorder. Specifies the display order for the boot manager to use when displaying the tools menu.

Commands that Control EMS

- /bootems. Enables or disables Emergency Management Services for the specified entry.
- /ems. Enables or disables Emergency Management Services for the specified operating system boot entry.
- /emssettings. Sets the global Emergency Management Services settings for the system. Emssettings does not enable or disable Emergency Management Services for any particular boot entry.

Commands that Control Debugging

- /bootdebug. Enables or disables the boot debugger for a specified boot entry. Although this command works for any boot entry, it is effective only for boot applications.
- /dbgsettings. Specifies or displays the global debugger settings for the system. This command does not enable or disable the kernel debugger; use /debug for that purpose. To set an individual global debugger setting, use `bcdedit /set {dbgsettings}` *`type value`*.
- /debug. Enables or disables the kernel debugger for a specified boot entry.

How to Manage BCD Programmatically with WMI

The BCD WMI API gives developers essentially complete control over the contents of a store. It allows developers to create applications that use custom boot data or make complex changes to a store that are difficult or impossible with BCDEdit. Applications based on the WMI API can be run locally or remotely.

The BCD WMI provider consists of a scriptable set of classes that support programmatic access to BCD stores. The classes are exposed as COM objects, which allows applications to also be implemented in C++. Although the BCD WMI provider was written primarily for Windows Vista and later versions of Windows, it can also be used with Microsoft Windows XP, Windows Server 2003, and recovery environments that support WMI.

This section provides a brief summary of the capabilities of the BCD WMI provider. For detailed information, see the documentation in the MSDN Library. For some examples of how to use the WMI API for specific tasks, see “BCD Cookbook,” later in this paper.

The BCDStore Class

The **BCDStore** class represents a BCD store. It allows developers to do such tasks as create stores, add or delete objects, and import the contents of a non-system store into system store. The object has one property, **StoreFilePath**, which contains the fully-qualified path of the object's BCD store. For convenience, the system store is represented by an empty string (""). The following table lists the object's methods.

BCDStore Methods

Method	Description
CopyObject	Copies the specified object from another store.
CopyObjects	Copies the objects of the specified type from another store.
CreateObject	Creates the specified object.
CreateStore	Creates a new store.
DeleteObject	Deletes the specified object.
DeleteSystemStore	Deletes the system store.

Method	Description
EnumerateObjects	Enumerates the objects of the specified type.
GetSystemDisk	Gets the system disk.
GetSystemPartition	Gets the system partition.
ExportStore	Exports a store to a specified file.
ImportStore	Marks the specified store as the system store.
OpenObject	Opens the specified object.
OpenStore	Opens a store.

The BCDObject Class

The **BCDObject** class represents a BCD object. It allows developers to do such tasks as add or delete elements or modify the values of existing elements. Objects are identified by a GUID and also contain a **Type** property that specifies the object's purpose. The following table lists the object's properties:

BCDObject Properties

Property	Description
StoreFilePath	The fully-qualified path of the BCD store. The system store is represented by an empty string ("").
Id	The object's GUID, in string format.
Type	The object's type.

The following table lists the object's methods.

BCDObject Methods

Method	Description
DeleteElement	Deletes the specified element.
EnumerateElementTypes	Enumerates the types of elements in the object.
EnumerateElements	Enumerates the elements in the object.
GetElement	Gets the specified element.
SetBooleanElement	Sets the specified Boolean element.
SetDeviceElement	Sets the specified device element.
SetFileDeviceElement	Sets the specified file device element.
SetIntegerElement	Sets the specified integer element.
SetObjectElement	Sets the specified object element.
SetObjectListElement	Sets the specified object list element.
SetPartitionDeviceElement	Sets the specified partition device element.
SetStringElement	Sets the specified string element.

BCDElement Classes

A number of classes represent elements, one for each supported data type. The base class is **BCDElement**, which has no methods. It supports the same three properties that are supported by **BCDObject.** However, with **BCDElement** the **ID** property is named **ObjectID**. The following table lists the classes that are derived from **BCDElement**.

BCDElement Types and Subclasses

Method	Description
BcdBooleanElement	Contains a Boolean value.
BcdIntegerElement	Contains an integer element.
BcdObjectElement	Contains an object ID.
BcdObjectListElement	Contains a list of object IDs.
BcdStringElement	Contains a string.

BCD Cookbook

This section contains a number of brief examples of how to use BCDEdit for common tasks. It also shows some examples of how to use the BCD WMI API to do the same tasks programmatically.

Kernel Debugging

This section shows how to use BCDEdit to enable kernel debugging and specify debug settings.

Enable Kernel Debugging

Use the following BCDEdit command to enable or disable kernel debugging for a specified boot entry.

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

ID is the GUID that is associated with a boot entry. If it is omitted, BCDEdit modifies the current boot entry by default. To specify a particular boot entry, set ID to the string form of the associated GUID. The following example enables kernel debugging for the specified entry.

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

Specify Global Debug Settings

To specify debug settings globally, use the following command.

```
bcdedit /dbgsettings type settings
```

The following examples show how to specify global debug settings for serial, 1394, and USB connections.

```
bcdedit /dbgsettings serial debugport:1 baudrate:115200
bcdedit /dbgsettings 1394 channel:32
bcdedit /dbgsettings USB targetname:U1
```

Specify Debug Settings for a specified Boot Entry

To specify debug settings for a specific boot entry, use BCDEdit to set the appropriate elements. The particular settings depend on the connection type.

```
bcdedit /set ID debugsetting1
bcdedit /set ID debugsetting2
...
```

The following example shows how to specify debug settings for a serial connection for a specified entry.

```
bcdedit /set {cbd971bf-b7b8-4885-951a-fa03044f5d71} debugtype serial
bcdedit /set {cbd971bf-b7b8-4885-951a-fa03044f5d71} debugport 2
bcdedit /set {cbd971bf-b7b8-4885-951a-fa03044f5d71} baudrate 115200
```

Specify the Default Operating System

To specify the default operating system, use:

```
bcdedit /default ID
```

ID is the GUID for the Windows boot loader boot entry that is associated with the desired operating system. For example:

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

To change the default boot entry to the legacy loader, set ID to {ntldr}, which is BCDEdit 's well-known name for the GUID that is associated with Ntldr.

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

The following Microsoft® Visual Basic® Script sample shows how to use the BCD WMI API to specify the default operating system. It takes a single argument, the GUID that is associated with the boot entry for the new default operating system.

```
set Locator = CreateObject("WbemScripting.SWbemLocator")
set Services = Locator.ConnectServer(".", "root\wmi")
Services.Security_.ImpersonationLevel = 3

DefaultOsIdentifier = WScript.Arguments(0)

'These hardcoded values will be replaced with official constants
' whenavailable.

const BootMgrId = "{9dea862c-5cdd-4e70-acc1-f32b344d4795}"
const DefaultType = &h23000003
'
'Open up a connection to WMI BcdStore class, allowing for
'impersonation. We need to request that Backup and Restore
'privileges be granted as well.

set BcdStoreClass = GetObject("winmgmts:{impersonationlevel=impersonate,
(Backup,Restore)}!" & MachineName & "root/wmi:BcdStore")

if not BcdStoreClass.OpenStore("", BcdStore) then
    WScript.Echo "Couldn't open the system store!"
    WScript.Quit
end if
'
' Open the "boot manager" object.
'
if not BcdStore.OpenObject(BootMgrId, BootMgr) then
    WScript.Echo "Couldn't open the boot manager object!"
    WScript.Quit
end if
'
' Set the boot manager's default OS object to the specified OS.
' Note that objects must be passed as strings.
'
if not BootMgr.SetObjectElement(DefaultType, DefaultOSIdentifier) then
    WScript.Echo "Couldn't set the default OS value!"
    WScript.Quit
end if

WScript.Echo "Successfully set the boot manager's default OS value."
```

Specify the Boot Manager's Timeout Value

To specify the boot manager's timeout value, use:

```
bcdedit /timeout Timeout
```

Timeout specifies the value in seconds. For example, to specify a 15-second timeout value:

```
bcdedit /timeout 15
```

Manage Boot Entries

This section shows how to manage the boot entries in BCD.

Change a Boot Entry's Description

The description is the text that appears in the list of boot entries that is displayed to the user at boot time. Use the following command to change a boot entry's description. *ID* is the GUID that is associated with the desired boot entry.

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

For example:

```
bcdedit /set {802d5e32-0784-11da-bd33-000476eba25f} description "My
Favorite OS"
```

Control How Boot Entries Appear to the User

To specify the order in which boot entries appear to the user, run the following command. *ID1*, *ID2*, and so on are the GUIDs that are associated with the boot entries. Any boot entries that are not included in the list do not appear. If only one entry is specified, the Windows boot manager simply selects that entry without displaying a list.

```
bcdedit /displayorder ID1 [ID2] [ID3] [...]
```

The following command specifies three boot entries: two identified by their GUIDS and Ntldr by its well-known name.

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

The following command adds a boot entry to the beginning or end of the current list, or removes an entry from the list.

```
bcdedit /displayorder ID [/addlast] [/addfirst] [/remove]
```

The following example adds an Ntldr entry to the end of the display order.

```
bcdedit /displayorder {ntldr} /addlast
```

It is also possible to specify a display order that applies only to the next reboot. After that, BCD reverts to the original display order. Use the following command, where the IDs are the GUIDs that are associated with the boot entries.

```
bcdedit /bootsequence ID1 [ID2] [ID3] ...
```

Create a New Windows Vista Boot Entry

The following procedure creates an additional Windows Vista boot entry. This allows a user, for example, to have separate normal and debug configurations for the same version of the operating system.

1. Make a copy of an existing Windows Vista boot entry, as shown in the following example. *ID* is the GUID that is associated with the boot entry to be copied. BCDEdit creates a GUID for the new boot entry.

   ```
   Bcdedit /copy ID /d "New entry description"
   ```

2. The preceding command returns the GUID that is associated with the new boot entry. Use that GUID to modify the partition information, as shown in the following example. *NewID* is the GUID of the new boot entry, and this example sets the partition to "d:".

   ```
   Bcdedit /set NewID device partition=d:
   Bcdedit /set NewID osdevice partition=d:
   ```

3. Add the new boot entry to the display order. The following example adds it to the end of the list.

   ```
   Bcdedit /displayorder NewID –addlast
   ```

4. Make any additional configuration changes that are required, such as enabling the kernel debugger.

Delete a Boot Entry

The following command deletes a boot entry from BCD. *ID* is the GUID that is associated with the boot entry.

```
bcdedit /delete ID
```

The following Visual Basic Script example shows how to use the BCD WMI API to delete a boot entry. It takes a single argument, the GUID that is associated with the boot entry to be deleted.

```
set Locator = CreateObject("WbemScripting.SWbemLocator")
set Services = Locator.ConnectServer(".", "root\wmi")
Services.Security_.ImpersonationLevel = 3

if WScript.Arguments.Count < 1 then
   WScript.Echo "Usage: " & WScript.FullName & " " & WScript.ScriptName & "
<GUID of OS to delete>"
   WScript.Quit
end if

TargetOS = WScript.Arguments(0)

'
' These hardcoded values will be replaced with official constants when
' available.
'
const BootMgrId = "{9dea862c-5cdd-4e70-acc1-f32b344d4795}"
const DefaultType = &h23000003
const DisplayOrderType = &h24000001
```

```
set BcdStoreClass = GetObject("winmgmts:{impersonationlevel=impersonate,
(Backup,Restore)}!" & "root/wmi:BcdStore")
if not BcdStoreClass.OpenStore("",BcdStore) then
    WScript.Echo "Couldn't open the system store!"
    WScript.Quit
end if
'
' Open the "boot manager" object.
'
if not BcdStore.OpenObject(BootMgrId, BootMgr) then
    WScript.Echo "Couldn't open the boot manager object!"
    WScript.Quit
end if
'
' Get the boot manager's display order list.
'
if not BootMgr.GetElement(DisplayOrderType, BootOrderList) then
    WScript.Echo "Couldn't get the display order list!"

else
    '
    ' remove the target os from the boot order list.
    '
    dim NewBootOrderList()
    i = 0
    for each OSIdentifier in BootOrderList.Ids
        if not TargetOS = OSIdentifier then
            redim preserve NewBootOrderList(i)
            NewBootOrderList(i) = OSIdentifier
            i =i + 1
        end if
    next

    if not BootMgr.SetObjectListElement(DisplayOrderType, NewBootOrderList)
then
        WScript.Echo "Couldn't set the new display order list!"
        WScript.Quit
    end if
end if
'
' Finally, delete the OS object
'
if not BcdStore.DeleteObject(TargetOS) then
    WScript.Echo "Couldn't delete target OS: " & TargetOS
    WScript.Quit
end if

WScript.Echo "Successfully deleted target OS: " & TargetOS
```

Enable PAE

The following command enables PAE for a specified boot entry. *ID* is the GUID that is associated with the desired boot entry. If no ID is specified, BCDEdit modifies the setting for the currently active operating system.

```
Bcdedit /set ID PAE ForceEnable
```

For example:

```
bcdedit /set {802d5e32-0784-11da-bd33-000476eba25f} PAE ForceEnable
```

Create a Boot Entry to Boot a WIM from a Hard Disk

This section shows how to create a boot entry to boot a WIM from a hard disk. It assumes that the boot drive is "c:/". The WIM is contained in boot.wim, which is a normal WIM with Winload.exe in the System32 folder.

1 Use the following set of commands to create a ramdiskoptions object in the BCD store. The string "{ramdiskoptions}" is the well-known name for the object's GUID.

```
bcdedit /create {ramdiskoptions} /d "Ramdisk options"
bcdedit /set {ramdiskoptions} ramdisksdidevice partition=c:
bcdedit /set {ramdiskoptions} ramdisksdipath \boot\boot.sdi
```

2. Create a new boot entry.

```
bcdedit -create /d "Display Text" /application OSLOADER
```

3. Step 2 returns the GUID that is associated with the newly created boot entry. It is referred to as NewGUID in the remaining examples. Run the following set of commands to configure the boot entry.

```
bcdedit /set {NewGUID} device ramdisk=[c:]\sources\boot.wim,{ramdiskoptions}
bcdedit /set {NewGUID} path \windows\system32\winload.exe
bcdedit /set {NewGUID} osdevice ramdisk=[c:]\sources\boot.wim,{ramdiskoptions}
bcdedit /set {NewGUID} systemroot \windows
```

Make a Non-system Store into the System Store

Systems can have any number of BCD stores. However, there can be only one system store, and it controls the boot process. The /import command replaces the contents of the system store with the contents of a specified non-system store. To preserve the contents of the current system store, run the /export command to create a backup copy of the system store. To restore the original system store, import the backup copy.

The following commands save a backup copy of the system store and import a non-system store into the system store. *NewStoreName* is the fully-qualified name of the file that contains the non-system store, and *BackupStoreName* is the fully-qualified name of the file that contains the backup store.

```
bcdedit /export BackupStoreName
bcdedit /import NewStoreName
```

The following Visual Basic Script example shows how to import a specified non-system store into the system store. It takes one parameter—the fully-qualified path for the BCD store that is to become the new system store.

```
set Locator = CreateObject("WbemScripting.SWbemLocator")
set Services = Locator.ConnectServer(".", "root\wmi")
Services.Security_.ImpersonationLevel = 3

FilePath = WScript.Arguments(0)
'
' Retrieve the BcdStore class object and call the static
' ImportStore method on it.
'
set BcdStoreClass = Services.Get("BcdStore")
if not BcdStoreClass.ImportStore(FilePath) then
   WScript.Echo "Couldn't import the system store!"
   WScript.Quit
end if

WScript.Echo "Successfully imported the system store from " &
FilePath & "."
```

Resources

The following links provide further information about BCD and the Windows boot process.

MSDN:

BCD WMI Reference
http://msdn2.microsoft.com/en-us/library/aa362677.aspx

Boot Configuration Data (BCD)
http://msdn2.microsoft.com/en-us/library/aa362692.aspx

Introduction to Boot Options
http://msdn2.microsoft.com/en-us/library/ms791478.aspx

White Paper:

EFI and Windows Vista
http://www.microsoft.com/whdc/system/platform/firmware/efibrief.mspx