DELL Technologies

[Technical Whitepaper]

Dell Agentless Client Manageability Technical Whitepaper

BIOS Development

Abstract

How to manage Dell client devices by leveraging their direct WMI capabilities

July 2020

Revisions

Date	Description
July 2020	Initial release

Acknowledgements

Author: Girish Prakash

Table of contents

Executive summary

Managing Dell client devices without additional agentry is an important capability for many customers. This paper explains how to use the WMI interface to leverage the default namespaces that are available to manage Dell client devices without the need for additional agents or tools.

1 Introduction

This whitepaper provides technical information on how customers can leverage “zero-touch” or agentless management aspects of Dell© commercial client platforms. The interface described in this whitepaper is included on Dell commercial client systems released to market after calendar year 2018.

1.1 Background

Configuring Dell client systems without first installing a system management agent such as Dell Command Suite agents can be challenging. These agents equip the system with proprietary interfaces, services, and console UI- and CLI-based tools, but as soon as you introduce agent software into the equation, you also need to develop and maintain update and redeployment plans. There is, however, another way to accomplish a seamless, out-of-box experience without the need to install and maintain agents.

1.2 Windows Management Instrumentation & PowerShell

Windows Management Instrumentation (WMI) is the infrastructure for management data & operations on Windows based operating systems. For more details on WMI see https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page.

PowerShell is a cross-platform task automation and configuration management framework consisting of command-line shell and scripting language. For more details see https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7

Most Dell commercial client systems are Windows-based, and IT administrators are not new to WMI and PowerShell. These two technologies are built on industry standards, are simple. and enable scripting which allows IT professionals to easily integrate them with their existing infrastructure or develop custom scripts around these technologies. Microsoft has done a great job enhancing PowerShell capabilities to integrate and manage WMI infrastructure in a seamless manner. This is why we chose this path to expose BIOS configurable entities through WMI, and this technical whitepaper demonstrates sample scripts to configure the same.

It is also worth mentioning that using this method to configure Dell business client systems is a uniform common interface across multiple brands, including Latitude, Optiplex, Precision, and XPS laptops. It also provides stable ways to enhance hardware management features and does not alter across various Windows operating system versions.

2 Platform Level Data Modeling

On Dell's commercial client systems, most of the configurable items available on the BIOS Setup (F2) screen are based on the Distributed Management Task Force (DMTF) PLDM for BIOS Control and Configuration Specification. You can find more information on this using the below link.
https://www.dmtf.org/sites/default/files/standards/documents/DSP0247_1.0.0.pdf

Each configurable entity according to the PLDM specification is called as an attribute and there are multiple types of attributes, each described in the following sections.

2.1 Enumeration attribute

Represents a BIOS attribute that can have a value from a set of possible or predefined values.

Example: NumLock (Enabled, Disabled)

Number Lock attribute which can take values Enabled or Disabled.

2.2 Integer attribute

Represents a BIOS integer. Each BIOS integer has a lower bound and an upper bound on the values that it can take.

Example: AutoOnHr

Auto On Hour value which can range from 0 to 23.

2.3 String attribute

Represents a string that the BIOS uses. Each BIOS string is characterized by the minimum length of the string, the maximum length of the string, and the type of the string.

Example: SvcTag

Service Tag of the system.

3 WMI namespace, classes and instances

All objects in WMI are exposed within a unique namespace, and so are BIOS attributes. The attributes mentioned above are modeled as individual classes under the namespace

`root/dcim/sysman/biosattributes`

Attributes are exposed under the following class names:

3.1 EnumerationAttribute

```
class EnumerationAttribute
{
    [key, read] string InstanceName;
    [read]  boolean ReadOnly;
    [WmiDataId(1),read] string AttributeName;
    [WmiDataId(2),read] string DisplayNameLangCode;
    [WmiDataId(3),read] string DisplayName;
    [WmiDataId(4),read] string DefaultValue;
    [WmiDataId(5),read] string CurrentValue;
    [WmiDataId(6),read] string Modifiers;
    [WmiDataId(7),read] uint32 ValueModifierCount;
    [WmiDataId(8),read, WmiSizeIs ("ValueModifierCount")] string
ValueModifiers[];
    [WmiDataId(9),read] uint32 PossibleValueCount;
    [WmiDataId(10),read, WmiSizeIs ("PossibleValueCount")] string
PossibleValue[];
};
```

3.2 IntegerAttribute

```
class IntegerAttribute
{
    [key, read] string InstanceName;
    [read] boolean ReadOnly;
    [WmiDataId(1),read] string AttributeName;
    [WmiDataId(2),read] string DisplayNameLangCode;
    [WmiDataId(3),read] string DisplayName;
    [WmiDataId(4),read] uint32 DefaultValue;
    [WmiDataId(5),read] uint32 CurrentValue;
    [WmiDataId(6),read] string Modifiers;
    [WmiDataId(7),read] uint32 LowerBound;
    [WmiDataId(8),read] uint32 UpperBound;
    [WmiDataId(9),read] uint32 ScalarIncrement;
};
```

3.3 StringAttribute

```
class StringAttribute
```

```
{
    [key, read] string  InstanceName;
    [read] boolean ReadOnly;
    [WmiDataId(1),read] string AttributeName;
    [WmiDataId(2),read] string DisplayNameLangCode;
    [WmiDataId(3),read] string DisplayName;
    [WmiDataId(4),read] string DefaultValue;
    [WmiDataId(5),read] string CurrentValue;
    [WmiDataId(6),read] string Modifiers;
    [WmiDataId(7),read]uint32 MinLength;
    [WmiDataId(8),read] uint32 MaxLength;
};
```

To enumerate the attributes supported on any given system, the `Get-WmiObject` PowerShell command-let can be used.

```
Get-WmiObject -Namespace root/dcim/sysman/biosattributes -Class StringAttribute
```

The above command lists all the instances of StringAttribute supported on the system. Similarly, replacing StringAttribute with the EnumerationAttribute or IntegerAttribute class will yield their respective instances.

Thus far we have discussed how to obtain all the configurable items and attributes available in the system. Next, we review how to modify the value for any of the above attributes.

4 Modifying Attributes & BIOS Defaults.

BIOSAttributeInterface is the name of the WMI class which exposes various Set methods to set the attribute value to a required state. It also exposes methods to restore entire BIOS configurations to previously registered default values. In addition to various Set methods, BIOSAttributeInterface class also exposes a method to get the Help string for any given attribute. The following is the entire class definition:

```
class BIOSAttributeInterface
{

      [key, read] string InstanceName;
      [read] boolean Active;
      [WmiMethodId(1), Implemented, Description("Set default value of
      BiosAttribute")]
      void SetDefaultValue(
      [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
      "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
      sint32 Status,
      [ValueMap{0,1 }, Values{"NONE","PlainText"}, IN] uint32 SecType,
      [IN] uint32 SecHndCount,
      [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
      [IN] string AttributeName);

      [WmiMethodId(2), Implemented, Description("Set default value of
      BiosAttributes")]
      void SetDefaultValues(
      [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
      "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
      sint32 Status,
      [ValueMap{0,1 }, Values{"NONE","PlainText"}, IN] uint32 SecType,
      [IN] uint32 SecHndCount,
      [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
      [IN] uint32 AttributeCount,
      [IN, WmiSizeIs("AttributeCount"):Toinstance] string AttributeNames[]);

      [WmiMethodId(3), Implemented, Description("Reset BIOS")]
      void SetBIOSDefaults(
      [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
      "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
      sint32 Status,
      [ValueMap{0,1 }, Values{"NONE","PlainText"}, IN] uint32 SecType,
      [IN] uint32 SecHndCount,
      [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
      [ValueMap{0,1,2,3,4}, Values{"BuiltInSafeDefaults","LastKnownGood",
      "Factory", "UserConf1", "UserConf2"}, IN] uint8 DefaultType);

      [WmiMethodId(4), Implemented, Description("Set value for the given
      attribute")]
      void SetAttribute(
```

```
        [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
        "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
        sint32 Status,
        [ValueMap{0,1 }, Values{"NONE","PlainText"}, IN] uint32 SecType,
        [IN] uint32 SecHndCount,
        [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
        [IN] string AttributeName,
        [IN] string AttributeValue);

        [WmiMethodId(5), Implemented, Description("Set value for the given
        attributes")]
        void SetAttributes(
        [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
        "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
        sint32 Status,
        [ValueMap{0,1,2,3}, Values{"NONE","PlainText"}, IN] uint32 SecType,
        [IN] uint32 SecHndCount,
        [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
        [IN] uint32 AttributeCount,
        [IN, WmiSizeIs("AttributeCount"):Toinstance] string AttributeNames[],
        [IN, WmiSizeIs("AttributeCount"):Toinstance] string
        AttributeValueNames[]);

        [WmiMethodId(6), Implemented, Description("Get Help string for the given
        attribute")]
        void GetHelpString(
        [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
        "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
        sint32 Status,
        [IN] string AttributeName,
        [OUT] string LangCode,
        [OUT] uint32 HelpStrCount,
        [OUT, WmiSizeIs("HelpStrCount"):Toinstance] uint8 HelpString[]);

};
```

To use this WMI class in PowerShell, first you need to enumerate an instance. To do so, execute the following PowerShell command with Administrator privilege.

```
$BAI = Get-WmiObject -Namespace root/dcim/sysman/biosattributes -Class
BIOSAttributeInterface
```

Note that you are storing the instance in a PowerShell environment variable called $BAI. This will allow you to use the same instance for future configuration settings. Once you have the class instance, executing a Set operation becomes simple.

```
$BAI.SetAttribute(0,0,0,"UefiNwStack","Disabled")
```

Note that the first three arguments are set to zero if BIOS administrator password is not set on the system.

But if the password is set to PASSWORD, for example, then the commands look like

```
$pwd = ”PASSWORD”
$encoder = New-Object System.Text.UTF8Encoding
$bytes = $encoder.GetBytes($pwd)
$BAI.SetAttribute(1,$bytes.Length,$bytes,"UefiNwStack","Disabled")
```

The first three commands above encode the plaintext password to bytes, and the SetAttribute() method’s first argument is 1(Plain Text) instead of 0 (None). The second argument is the length of the byte array, and the third parameter is the byte array itself which contains the encoded BIOS administrator password.

This is how you can use the rest of the methods to Set, Reset to default value, or Reset to BIOS defaults for all attributes. For the SetBIOSDefaults method to succeed on UserConf1 or UserConf2, these configurations must be already saved or created in Setup.

5 Boot order enumeration and configuration

Like attributes, bootorder is another class exposed in WMI on Dell client systems in the same namespace (`root/dcim/sysman/biosattributes`).

```
class BootOrder
{
    [key, read] string InstanceName;
    [WmiDataId(1),read] string BootListType;
    [WmiDataId(2),read] uint32 IsActive;
    [WmiDataId(3),read] uint32 BOCount;
    [WmiDataId(4),read, WmiSizeIs ("BOCount")] string BootOrder[];
};

class SetBootOrder
{

      [key, read] string InstanceName;
      [read] boolean Active;
      [WmiMethodId(1), Implemented, Description("Set Boot order for the given
      bootlist")]
      void Set(
      [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
      "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
      sint32 Status,
      [ValueMap{0,1 }, Values{"NONE","PlainText"}, IN] uint32 SecType,
      [IN] uint32 SecHndCount,
         [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
      [IN] string BootListType,
      [IN] uint32 BOCount,
      [IN ,WmiSizeIs ("BOCount"):Toinstance] string BootOrder[]);

};
```

You can enumerate boot lists and boot devices using the following PowerShell command:

```
Get-WmiObject -Namespace root/dcim/sysman/biosattributes -Class BootOrder
```

A typical output would look like the following:

```
__GENUS          : 2
__CLASS          : BootOrder
__SUPERCLASS     :
__DYNASTY        : BootOrder
__RELPATH        : BootOrder.InstanceName="ACPI\\PNP0C14\\WBAT_0"
__PROPERTY_COUNT : 5
__DERIVATION     : {}
__SERVER         : COMPNAME
__NAMESPACE      : root\dcim\sysman\biosattributes
```

```
__PATH           :
\\COMPNAME\root\dcim\sysman\biosattributes:BootOrder.InstanceName="ACPI\\PNP0C14
\\WBAT_0"
BOCount          : 0
BootListType     : LEGACY
BootOrder        :
InstanceName     : ACPI\PNP0C14\WBAT_0
IsActive         : 0
PSComputerName   : COMPNAME

__GENUS          : 2
__CLASS          : BootOrder
__SUPERCLASS     :
__DYNASTY        : BootOrder
__RELPATH        : BootOrder.InstanceName="ACPI\\PNP0C14\\WBAT_1"
__PROPERTY_COUNT : 5
__DERIVATION     : {}
__SERVER         : COMPNAME
__NAMESPACE      : root\dcim\sysman\biosattributes
__PATH           :
\\COMPNAME\root\dcim\sysman\biosattributes:BootOrder.InstanceName="ACPI\\PNP0C14
\\WBAT_1"
BOCount          : 2
BootListType     : UEFI
BootOrder        : {Windows Boot Manager, UEFI Hard Drive}
InstanceName     : ACPI\PNP0C14\WBAT_1
IsActive         : 1
PSComputerName   : COMPNAME
```

If the BOCount member is greater than one, then you can use the following commands to change the boot order.

```
$SBO = Get-WmiObject -Namespace root/dcim/sysman/biosattributes -Class
SetBootOrder
$NewBO = “UEFI Hard Drive”, “Windows Boot Manager”
$SBO.Set(0,0,0, ”UEFI”, $NewBO.Count, $NewBO)
```

Remember to use the password parameters as demonstrated in the previous section if a password is already set.

6 Managing passwords

The BIOS administrator password and system password can be set, reset, or cleared using the WMI classes from namespace:

```
root/dcim/sysman/wmisecurity
```

There are two classes which are exposed from this namespace: PasswordObject and SecurityInterface. PasswordObject instances indicate properties of the password whereas SecurityInterface can be used to perform operations on the password.

```
class PasswordObject
{
  [key, read] string InstanceName;
  [read] boolean Active;
  [WmiDataId(1),read] string NameId;
  [WmiDataId(2),read] uint32 IsPasswordSet;
  [WmiDataId(3),read] uint32 MinimumPasswordLength;
  [WmiDataId(4),read] uint32 MaximumPasswordLength;
};

class SecurityInterface
{

      [key, read] string InstanceName;
      [read] boolean Active;
      [WmiMethodId(1), Implemented, Description("Set New Password")] void
      SetNewPassword(
      [ValueMap{0,1,2,3,4,5,6}, Values{"Success", "Failed", "Invalid Parameter",
      "Access Denied", "Not Supported", "Memory Error", "Protocol Error"}, OUT]
      sint32 Status,
      [ValueMap{0,1}, Values{"NONE", "PlainText"}, IN] uint32 SecType,
      [IN] uint32 SecHndCount,
      [IN ,WmiSizeIs ("SecHndCount"):Toinstance] uint8 SecHandle[],
      [IN] string NameId,
      [IN] string OldPassword,
      [IN] string NewPassword);

};
```

If passwords are currently not set on the system then you can use following PowerShell command to set it.

```
$SI = Get-WmiObject -Namespace root/dcim/sysman/wmisecurity -Class
SecurityInterface
$SI.SetnewPassword(0, 0, 0, "Admin", "", "PASSWORD")
```

To reset the password just use empty string "" for the NewPassword parameter.

Conclusion

Now you can manage the BIOS configurations on your Dell commercial client systems directly from WMI, without using additional agents or applications.