Intel DMI Provider User Documentation

Version 0.3
May 1, 1998
Intel Corporation

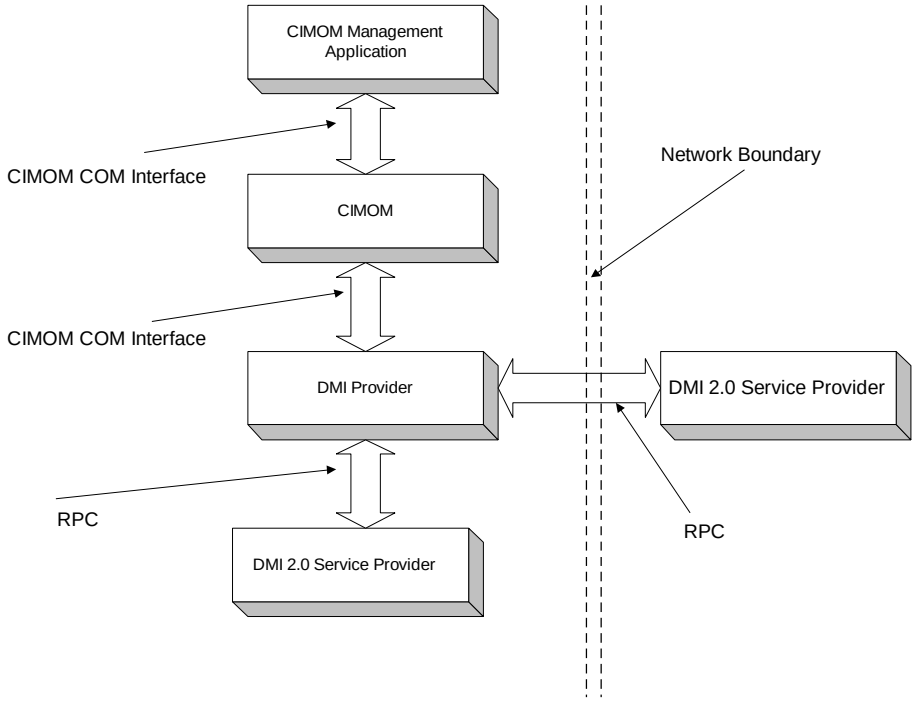
Table Of Contents

Introduction

The DMI provider gives (WBEM*) applications the ability to interact with the DMI 2.0 Service Providers running on the local or remote platforms. Wbem applications will be able to access and control the DMI enabled platforms. For additional information, see DMI provider.
For additional information on DMI, visit the DMTF website at http://www.dmtf.org.
For additional information on WBEM, visit the WBEM website at http://www.microsoft.com/management/wbem/default.htm.

Desktop Management Interface (DMI) Support

The DMI provider is located between CIMOM and the DMI service providers as illustrated in the following diagram.



DMI Provider

The WBEM DMI provider is capable of acting as:

- A dynamic class provider that enumerates all classes in the DmiNodes namespace and generates class definitions.
- An instance provider that enumerates all instances of a particular class in the DmiNodes namespace.
- An event provider that provides DMI events for the DmiNodes namespace.

A WBEM namespace must be created for each managed node. Each node's namespace resides in the \Default\DmiNodes namespace.
The DMI provider manipulates CIM class definitions. Therefore, the DMI Management Information Format (MIF) Components, Groups, and Attributes map to CIM classes, instances, and properties, respectively. For more information on how this mapping is done, see Mapping DMI to the CIM Schema.
With the DMI provider, WBEM applications will be able to perform the following operations:

- Access all DMI 2.0 Component, Group, and Attribute data.
- Add and Delete DMI Components.
- Add and Delete DMI Groups.
- Add and Delete the Rows of a DMI tablular group.
- Modify the writeable Attribute data.
- Set the language mapping of a DMI Component.
- Add or Delete Languages from a DMI Component.
- Handle DMI Events.

Location of the DMI Provider components

The MOF file and executables for the DMI Provider are in the WBEM directory.

The software components comprising the DMI Provider

·**WBEMDMIP.DLL,** the main DMI Provider module responsible for schema translation, communication with WBEM and DMI, and construction of dynamic classes and instances.
·**WBEMDMIP.MOF,** the sample MOF file shipped with the DMI Provider describing the DMI node namespace for the local node and registering the DMI Provider with WBEM.
·**MOTDMIENGINE.OCX,** an ActiveX Control used by the DMI Provider to establish communications and interactions with the DMI Service Providers.
·**WCDMI.DLL, WCDMIDCE.DLL, WDMIUTIL.DLL,** the DLL modules used by the MOTDMIENGINE.OCX to handle RPC connection and tear-down with DMI 2.0 Service Providers on the local and remote nodes.

Setting Up the DMI Provider

To set up the DMI provider create a MOF file and perform the following steps:

1. Create the \DmiNodes namespace. To create a namespace, create an instance of the **__NAMESPACE** class. The \DmiNodes namespace must reside in the \Root namespace:

```
#pragma namespace("\\\\.\\root")
instance of __Namespace
{
   Name = "DmiNodes";
};
```

2. Create a namespace for each managed node under DmiNodes. Each namespace representing a DMI node must include a class and an instance that describes the node, as well as instances of the required provider registration class.

```
#pragma namespace("\\\\.\\root\\DmiNodes")
instance of __NameSpace
{
    Name = "ManagedNode1";   // Logical name given to a
                             // remote node
};
```

3. Create the class DmiNode. This class must have the qualifier singleton and the string property called “NetworkAddress”:

```
[singleton]
class DmiNode
{
   string NetworkAddress;
};
```

4. Create an instance of DmiNode. Set the NetworkAddress property to the network name or address of the managed node:

```
instance of DmiNode
{
   // Network address for ManagedNode1 or its machine name
   NetworkAddress = "206.170.168.35”
};
```

5. Create an instance of **__Win32Provider** to register the DMI provider to handle class and instance operations for the node created above:

```
#pragma namespace("\\\\.\\root\\DmiNodes\\ManagedNode1")
instance of __Win32Provider As $Provider
{
   Name    = "WbemDmip" ;// Provider DLL nameClsId   =
   "{DE065A70-19B5-11D1-B30C-00609778D668}" ;
};
```

6. Create an instance of **__InstanceProviderRegistration** to tell the Common Information Model Object Manager (CIMOM*) that the provider supports instance operations:

```
{
    Provider = $Provider;    SupportsGet = TRUE;    SupportsPut =
TRUE;    SupportsDelete = TRUE;    SupportsEnumeration = TRUE;};
```

7. Create an instance of __MethodProviderRegistration to tell CIMOM that the provider handles methods:

```
instance of __MethodProviderRegistration{    Provider = $Provider;};
```

8. Create an instance of __ClassProviderRegistration to tell CIMOM that the provider supports class operations:

```
instance of __ClassProviderRegistration{  Provider = $Provider;
     SupportsGet = TRUE;
     SupportsPut = FALSE;
     SupportsDelete = TRUE;
     SupportsEnumeration = TRUE;
      QuerySupportLevels = NULL ;  ResultSetQueries = { "Select *
      From meta_class Where __this isa \"DmiComponent\"" ,"Select *
      From meta_class Where __this isa   \"DmiGroupRoot\"",
      "Select * From meta_class Where __this isa \"DmiBindingRoot\""
      ,
      "Select * From meta_class Where __this isa \"DmiNodeData\"" ,
      "Select * From meta_class Where __this isa \"DmiLanguage\"" ,
      "Select * From meta_class Where __this isa \"DmiEvent\"" ,
      "Select * From meta_class Where __this isa
      "DmiAddMethodParams\""
      "Select * From meta_class Where __this isa
      \"DmiGetEnumParams\"",
      "Select * From meta_class Where __this isa
      \"DmiLanguageMethodsParams\""
      } ;
};
```

9. Create an instance of __Win32Provider to tell CIMOM that the provider supports events:

```
instance of __Win32Provider as $EventProv{    Name      =
"WbemDmiEventp" ;    ClsId    =  "{B21FBFA0-1B39-11d1-B317-
00609778D668}" ;};
```

10. Create an instance of **__EventProviderRegistration** to tell CIMOM the types of events that the provider can handle:

```
Instance of __EventProviderRegistration
{
   Provider = "WbemDmiEventp";
   EventQueryList = {
      "select * from DmiEvent",
      "select * from __InstanceCreationEvent where TargetInstance
      is a \"DmiComponent\"",
      "select * from __InstanceDeletionEvent where TargetInstance
      is a \"DmiComponent\"",
      "select * from __InstanceCreationEvent where TargetInstance
      is a \"DmiLanguage\"",
      "select * from __InstanceDeletionEvent where TargetInstance
      is a \"DmiLanguage\""

   };
};
```

Alternately, you can use the WBEM Developer Studio to create namespaces and the required class and instance definitions. For instructions, see the WBEM SDK Applications Guide for instructions. See the sample MOF file, WBEMDMIP.MOF included with the WBEM SDK.

11. After you create a MOF for a managed node, you must submit the file to WBEM's MOF compiler.

```
mofcomp  <MOF-file>
```

Mapping DMI To The CIM Schema

The WBEM DMI Provider uses the following schema to represent the MIF component group attribute information:

DmiNode

Static class defined in MOF. Used to establish the network address of the managed node.

DmiNodeData

Dynamic singleton class with a dynamic instance. Contains **DmiGetConfig** data. Has methods that allow you to add and delete components, and set the default language.

DmiComponent

A dynamic class with dynamic instances. Contains component and component ID group data. Has methods that allow you to add a group, add and delete a language, and extract the component ID group attribute enumeration.

DmiLanguage

A dymanic class with dynamic instances. Contains language data.

DmiGroupRoot

This is an abstract class that does not have any instances. All dynamic group classes are derived from this class.

Dynamic DmiGroup Classes

A dynamic class is created for each group on the managed node. Each dynamic group class is given a name in the form:

```
Component<Component Id>__Group<Group Id>__<Class String>.
```

A dynamic instance is created for each row of a tabular group. For a scalar group, only one instance is created.

DmiBindingRoot

This is an abstract class from which all bindings in the namespace are derived. The classes derived from DmiBindingRoot are used to bind a DmiComponent class to a DmiNode class; a DmiGroup class to a DmiComponent class, etc.

DmiLanguageBinding

Instances of this dynamic class bind **DmiLanguage** instances to instances of **DmiComponent**.

Dynamic DmiGroupBinding Classes

A dynamic group binding class is created for each group on the managed node. Each dynamic group binding class is given a name in the form:

```
Component<Component Id>__Group<Group Id>__<Class String>__Binding.
```

Dynamic instances of these classes are created for each row of a tabular group. For a scalar group, only one instance is created. An instance of a dynamic group binding class binds an instance of a dynamic group class to an instance of a DmiComponent.

DmiAddMethodParams

Instances of this dynamic class contain the parameters required by the add language, add group, and delete language methods.

DmiGetEnumParams

Instances of this dynamic class contain the parameters required by the get attribute enumeration method.

DmiEnum class

A dynamic class created for attribute enumeration. Instances of this class contain the value string pair of a DMI attribute's enumeration.

DmiLanguageMethodParams

Instances of this dynamic class are used in setting the default language.

DmiEvent

Instances of this dynamic class are generated when a DMI event is generated.

DMI Operations using Wbemdmip.dll

This section briefly describes the WBEM methods used for common DMI management tasks. See the DMITEST sample application included in this package for an example of how to use the DMI access methods.

DmiGetVersion

Use **IWbemServices::CreateInstanceEnum** with the class set to the singleton instance of **DmiNodeData**.

DmiGetConfig

Use **IWbemServices::GetObject** with the path for the singleton instance of **DmiNodeData**.

DmiSetConfig

Use the **IWbemServices::ExecMethod** with the path for the singleton instance of **DmiNodeData**, the method **SetDefaultLanguage**, and the instance of **DmiLanguageMethodParams** with the **Language** property set to the desired language string.

DmiListComponents

Use the **IWbemServices::CreateInstanceEnum** method with the class set to **DmiComponent**.

DmiListLanguages

Use the **IWbemServices::CreateInstanceEnum** method with the class set to **DmiLanguage**.

DmiListGroups

Use the **IWbemServices::CreateClassEnum** method with the superclass set to **DmiGroupRoot**.

DmiListAttributes

Use the **IWbemServices::GetObject** method with the path set to the desired instance of the desired dynamic group class.

DmiGetAttribute

Use the **IWbemServices::GetObject** method with the path set to the desired instance of the desired dynamic group class.

Note:

This gives you all the attributes in the group. There is no way to access just one attribute. Once you get the object that contains all the attributes, you can perform the Get operations on the object to get the class property representing a given DMI attribute.

DmiSetAttribute

Use the **IWbemServices::GetObject** method with the path set to the desired instance of the desired dynamic group class. Use the WBEM's **PUT** method on the object to modify the instance returned, then use **IWbemServices::PutInstance**.

DmiGetMultiple

Use the **IWbemServices::GetObject** method with the path set to the desired instance of the desired dynamic group class.

Note:

This gives you all the attributes in the group.

DmiSetMultiple

Use the **IWbemServices::GetObject** method with the path set to the desired instance of the desired dynamic group class. Modify the instance returned. Then use **IWbemServices::PutInstance**.

DmiAddRow

Use the **IWbemServices::GetObject** method with the path set to the desired instance of the desired dynamic group class. Modify the instance returned then use **IWbemServices::PutInstance**.

Note:

The **RowId** property must be changed to a nonexistent **RowId**.

DmiDeleteRow

Use the **IWbemServices::DeleteInstance** with the path set to the desired dynamic instance of the dynamic class.

DmiAddComponent

Use the **IWbemServices::ExecMethod** with the path for the singleton instance of **DmiNodeData**, the method of **AddComponent**, and the **MifFile** property of the instance of the **DmiAddMethodParams** set to the desired MIF file path.

DmiAddLanguage

Use the **IWbemServices::ExecMethod** with the path to the instance of the component to which the language is to be added. Set the method to **AddLanguage**, and the **MifFile** property of the instance of the **DmiAddMethodParams** set to the desired MIF file path.

DmiAddGroup

Use the **IWbemServices::ExecMethod** with the path to the instance of the component to which the language is to be added. Set the method to **AddGroup**, and the **MifFile** property of the instance of the **DmiAddMethodParams** set to the desired MIF file path.

DmiDeleteComponent

Use **IWbemServices::DeleteInstance** with the path set to the desired dynamic instance of the **DmiComponent** class.

DmiDeleteLanguage

Use the **IWbemServices::ExecMethod** with the path to the instance of component to which the language is going to be deleted. Set the method to **DeleteLanguage**, and the **Language** property of the instance of the **DmiLanguageMethodParams** set to the language string to be deleted.

DmiDeleteGroup

Use **IWbemServices::DeleteClass** with the path set to the desired dynamic group class.

DMITEST Code Sample

This section describes the dmitest code sample (DMITEST.EXE), a Win32 console application that uses the WBEM API to access and manipulate a DMI 2.0 enabled Win32 platform. The sample includes examples of DMI 2.0 MIF files that will be used to install components in the DMI database.
The same code reads input data from a script file, dmitest.scr, processes the command script, and makes appropriate WBEM API calls to access the DMI database. The application then will store the output data into a file called dmitest.out.

The dmitest application relies on the user to ensure that the DMI 2.0 Service Provider is already running on the system and that the WBEMDMIP.MOF file has already been compiled and installed into the CIMOM's repository to enable DMI operations.

The dmitest code sample was developed using Microsoft Visual C++ version 5.0.

The DMITEST' s MIF data

The application uses two MIF files dbtypes1.mif and tempcom1.mif to install three components in the DMI database and access them through the DMI provider. The dbtypes1.mif is installed twice. The DMITEST.SCR script uses hard coded path of C:\TESTMIFS for the location of the above two MIF files. To access the MIF files from a different directory, change the DMITEST.SCR file to look for the correct path.

Location of the Dmitest Code sample

The source files and executables for the Dmitest Code sample are in the subdirectory

\WBEM\SDK\COM\DMITEST

The files included in this sample

- **DMITEST.CPP,** the main source file for the DMITEST application responsible for reading and processing command scripts from the DMITEST.SCR input file and submitting DMI requests to WBEM.
- **DMITEST.H,** the header file for DMITEST.CPP module.
- **DMITEST.SCR,** the file containing the DMI command scripts processed by the DMITEST application.
- **DMITEST.OUT,** the DMITEST application generated output file containing the results from executing the DMI service requests.
- **DATATYPES.CPP,** a module that contains code for manipulating the OLE data types used by the DMITEST.CPP module.
- **DATATYPES.H,** the header file for DATATYPES.CPP module.
- **DEFINES.H,** a header file containing definitions used to turn off some sections of the WINDOWS.H header file that are not used by the application.
- **WBEMSVC.H,** the header file containing the CIMOM interface definitions.
- **DBTYPES1.MIF,** the MIF file used to install a DMI component called "First Database Types MIF"
- **TEMPCOM1.MIF,** a MIF file used to install a DMI component called "Test System Temporary Component"
- **DMITEST.DSP, DMITEST.DSW,** the Visual C++ 5.0 project file used to build the DMITEST.EXE executable.

Building and using the DMITEST Code sample with Visual C++ * v5.0 IDE

To build the Dmitest module, perform these steps:

1. Run the Microsoft Visual C++ 5.0 Development Studio program.
2. Load the DMITEST.DSP project file.
3. From the BUILD menu, select SETTINGS. Choose the Preprocessor option from the C/C++ Tabbed dialog box. Set the Additional Include Directory entry to the INCLUDE directory of the WBEM SDK installed on your system.
4. Replace the WBEMUUID.LIB module in the project files list with the one in the LIB directory of the WBEM SDK installed on your system. To build this library, run the "nmake" program from the \wbem\include subdirectory.
5. From the BUILD menu, select UPDATE ALL DEPENDENCIES.
6. Rebuild the project to generate the DMITEST.EXE module.
7. Verify that DMITEST.SCR file is in the same directory as the DMITEST.EXE module.
8. Run the DMITEST application and view DMITEST.OUT file for the results after the program terminates.
9. Open the DMITEST.CPP module and set breakpoints on the *ProcessScriptFile()* function, run the DMITEST application again, and observe how the application accesses DMI information through WBEM.

Using The WBEMTEST* Application To View DMI Data

To view the DMI data on the local or remote node, connect to the machines residing in the DMINODES namespace. For example, to view the DMI data on the local machine, use the following connect string:

Root\dminodes\local

After you establish connection, you can query and view the DMI classes specified earlier in this document. For example, to view a list of the DMI components installed in the DMI database, click on the "EnumInstances" button, and enter "dmicomponent" without quotes for the class name. To view a list of the DMI groups, click "EnumClasses" button, and enter "dmigrouproot" without quotes for the class name.

Subscribing For DMI Events and Notifications

You can use the WBEMTEST application to subscribe for DMI events and notifications. To subscribe for events, enable the "Execute asynchronously" check box, click on the "Notification Query" button and enter the appropriate event subscription queries. The following example shows how to subscribe for a "DmiComponent Added" notification:

*Select * from __InstanceCreationEvent Where TargetInstance isa "DmiComponent"*

The following example shows how to subscribe for a "DmiComponent Added" notification:

*Select * from __InstanceDeletionEvent Where TargetInstance isa "DmiComponent"*

The “DmiGroup Added/Deleted” notifications map to ClassCreation and ClassDeletion events in CIMOM. Since CIMOM does not support the ClassCreation and ClassDeletion events, the DMI provider does not support the above DMI group notifications.

The following example shows how to subscribe for all kinds of DMI extrinsic events including the extrinsic DMI events:

*Select * from DmiEvent*

Intel Corporation, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124-6497. Intel Corporation assumes no responsibility for errors or omissions in this document; nor does Intel make any commitment to update the information contained herein.