HP Client Management Interface Technical White Paper

Introduction

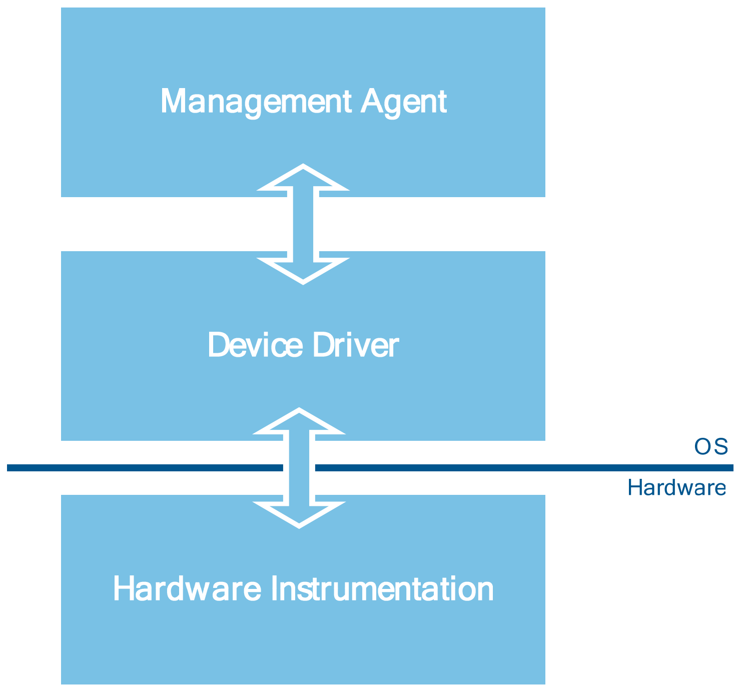
This white paper provides technical information on the HP Client Management Interface (HP CMI). HP CMI is an open architecture for gathering client computer inventory, monitoring health events, and managing BIOS configuration settings on HP business class client computers. This interface is included standard on select new models beginning with the HP Compaq dc7600 series and dx7200 series business desktops, and the HP xw4300 workstation. An HP CMI Software Provider SoftPaq is also available for legacy models and may be downloaded from HP.com.

This document describes the business need which drove development of HP CMI, benefits of the interface, architectural details, examples of how HP CMI can be used to carry out various client management tasks, and the security model.

Background

Historically, it has been a challenge for customers to easily integrate HP client computers with systems management tools and applications they are using. The typical management software model relies on a "software management agent" installed on the client computer. This software agent exposes management instrumentation through a proprietary driver and hardware interface and communicates with the systems management tool console. Often, the software agent must be updated and redeployed as new manageability features and new computer models are introduced.

Traditional Management Software Model



This traditional management software model contains three tightly integrated components: the computer hardware to be instrumented, an operating system specific driver to surface the instrumentation, and a software agent to expose and communicate the instrumented data with the management software console. In most cases these software agents are further specialized by the manner they surface the instrumented data to applications. This approach has made integration of advanced management features into commercial management software slow, and development of feature-rich custom-developed management applications difficult to accomplish.

Recognizing the need for a better solution, HP has developed the HP Client Management Interface. HP CMI provides a zero-footprint, programmatic interface built on industry standards that systems management tools and custom management applications can access to gather inventory information, heath alerts, and manage BIOS configuration.

Benefits

HP business-class client computers equipped with HP Client Management Interface technology provide an unprecedented level of out-of-the-box management capability. HP CMI provides the following benefits:

Flexible and open

- Built on industry standards for gathering inventory and health status information.
- Simple and scriptable instrumentation allows IT professionals to easily integrate with existing management tools or develop custom management applications.
- Client computer instrumentation can be made available to a central management console application and/or locally at the client computer.
- Computer health events are sent in real-time — no waiting for the management agent to poll for client status.

Consistent

- Provides a common interface to management information across HP business-class client computers equipped with HP CMI.
- Provides a stable foundation to future hardware management features.
- Interface behavior is consistent between 32-bit and 64-bit versions of Windows, and the next version of the Windows operating system.

Easy to manage

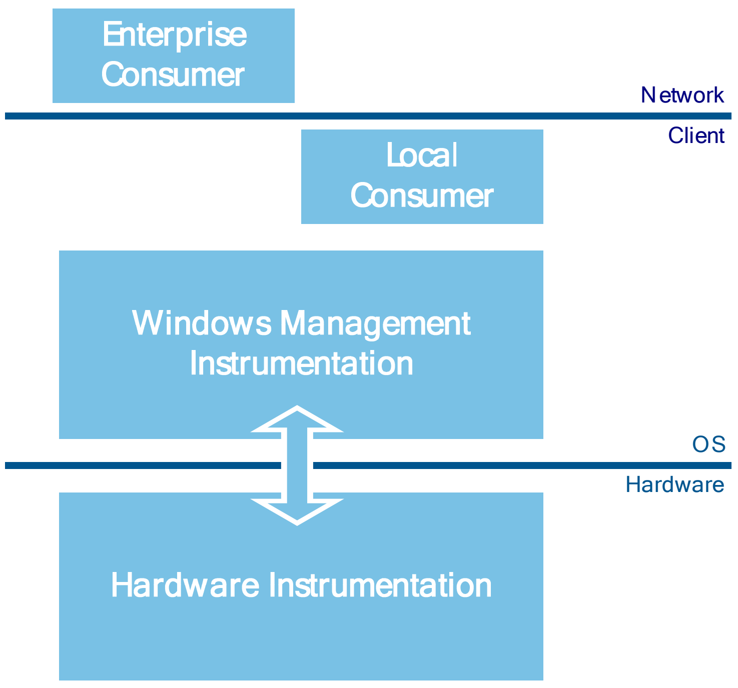
- No software agent is required to access client computer inventory information, health status and manage BIOS configuration.
- New client computers seamlessly integrate into the managed environment without re-tooling management software.
- Leverages operating system policies for configuration and security.

Architecture

Systems management technology has matured in recent years with the widespread adoption of the Common Information Model (CIM) and Web-Based Enterprise Management (WBEM) as a vendor-neutral method for describing the myriad of management elements available across the enterprise from client systems to storage area networks. This trend has made management of enterprise resources easier, and systems management applications more powerful in their ability to interpret heterogeneous management information.

Windows Management Instrumentation is Microsoft's implementation of the WBEM initiative, and is available as a component of the operating system. WMI uses the CIM standard to represent systems, applications, networks, devices, and other managed components. WMI can be used to automate administrative tasks in an enterprise environment. WMI features query-based information retrieval, relationship and data modeling, event subscription services, and access from any programming language capable of supporting Component Object Model (COM)[1], such as C++, Visual Basic, or scripting languages under Windows Scripting Host.

HP Client Management Interface Model



HP Client Management Interface leverages WMI to surface management information directly from the hardware and system BIOS, and in doing so gains all the benefits associated with the WMI interface to management information.

[1] Component Object Model is a specification developed by Microsoft. It provides the framework for technologies such as ActiveX.

HP CMI exposes three classifications of management information about the underlying hardware platform:

- Hardware sensor information

  This includes information about physical sensors within the client computer. The interface supports a wide variety of sensor types, including both numerical sensors such as fan speed (rpm) and sensors based on a physical state, such as the state of a case lock (open, closed). Sensor data is surfaced as an enumeration in WMI, which provides flexibility in the number and types of sensors reported from platform to platform.

- Hardware configuration options

  Instrumentation information related to configuring hardware options includes a multitude of features. These features are exposed in both a general and specialized manner to systems management applications. As with sensor data, hardware configuration options are surfaced as an enumeration in WMI, which provides flexibility in the number and types of sensors reported from platform to platform.

- System health events

  System health events are dynamically surfaced based on triggered hardware events. WMI provides a convenient, low-bandwidth mechanism for management information consumers to subscribe to these hardware events and be notified in real-time. These events can be monitored at the local client computer or by a remote console.

Hardware Sensor Information

HP CMI defines a base model for surfacing hardware sensor data to management applications. This model supports extension in two ways:

1. Enumeration of sensors physically attached to a given platform can be generalized by querying against the base class. This allows management tools to automatically detect available sensor devices without recoding.
2. Definition of new sensor types will extend the general definitions provided by the interface. This approach guarantees that properties known and understood today will continue to possess the same characteristics and behaviors as new features are introduced.

The following table defines the basic set of services provided by HP CMI to support the surfacing of hardware sensor data to management applications.

Management Class	**Description**
HP_BIOSSensor	Defines the basic set of properties common to all types of hardware sensors. All hardware sensors available on a particular platform can be surfaced by enumerating this class.
HP_BIOSStateSensor	This class provides access to a category of sensors that our monitored by state changes only. State changes are defined by an array of possible values within each sensor definition. Examples of state-based sensors would include POST warnings, physical switches, or solenoids.
HP_BIOSNumericSensor	Defines the category of sensors that return numerical measurements.

Hardware Configuration Options

One of the most compelling features of HP CMI is the power to manipulate and change hardware configuration options in an open and adaptable manner. WMI provides a foundation for scriptable administration of operating system options that is well proven in the enterprise management community. HP CMI leverages that foundation to provide the IT administrator with an unprecedented degree of control in managing configurations across the enterprise. As with sensor information, the mechanisms provided for collecting and manipulating hardware configuration options are designed with forward compatibility and future capabilities in mind.

The following table illustrates the inherent capability and flexibility of HP CMI in dealing with various types of hardware configuration options.

Management Class	**Description**
HP_BIOSSetting	Defines the basic set of properties common to all forms of BIOS settings. All hardware configuration options supported by the platform can be surfaced by enumerating this class.
HP_BIOSString	Extension of HP_BIOSSetting to support string-based hardware configuration options. This would include such capabilities as: ownership tag, asset tracking number, and UUID.
HP_BIOSInteger	Extension of HP_BIOSSetting to support numeric hardware configuration options.
HP_BIOSEnumeration	Most hardware configuration options fall into an enumeration category. Enumerations are collections of possible values for a setting (usually expressed as human-readable text). Example enumerations would be: “On, Off” or “Enable, Disable”.
HP_BIOSOrderedList	This class extends the HP_BIOSSetting to support such hardware features as boot order.
HP_BIOSPassword	While password values are never exposed through the HP CMI model, this class exists to help determine the existence of various password options on the client platform. This class would be queried to determine if a setup password was currently set on the platform, for example.
HP_BIOSSettingInterface	This class provides access to the WMI methods exposed by HP CMI. This includes methods to set individual settings and reset all settings to a default state.

System Health Events

Traditionally, system health monitoring has required a management agent to poll hardware features at some deterministic frequency in order to discover possible warning and error conditions within the system. This polling model of health measurement requires system resources and, potentially, network resources to monitor and maintain. In addition, the longer the polling interval, the longer it takes to potentially discover the triggering event. HP CMI breaks this mold of system health notification by surfacing events directly from the hardware when they are discovered. Management applications designed to consume WMI events can be configured to subscribe to the events generated by HP CMI without impacting system performance or network bandwidth.

The following table outlines some of the event monitoring capabilities provided by HP CMI. Like all HP CMI features, new events can be introduced with new hardware platforms or through system firmware updates without impacting the design of the interface or management tools designed to consume the events.

Event Type	Description
Sensor related	Events associated with health-monitoring sensors on the hardware platform. Sensor related events include: over-temperature, fan stalls, and chassis intrusion detection.
Device failures	Device failures refers to events for devices that are not monitored through hard-wired sensors, such as an ECC memory failure.
Configuration changes	When hardware configuration options are modified, or an attempt is made to modify these options without proper authority, HP CMI will generate related events to provide a mechanism to monitor and audit the interface from within the enterprise.

Available Software Products

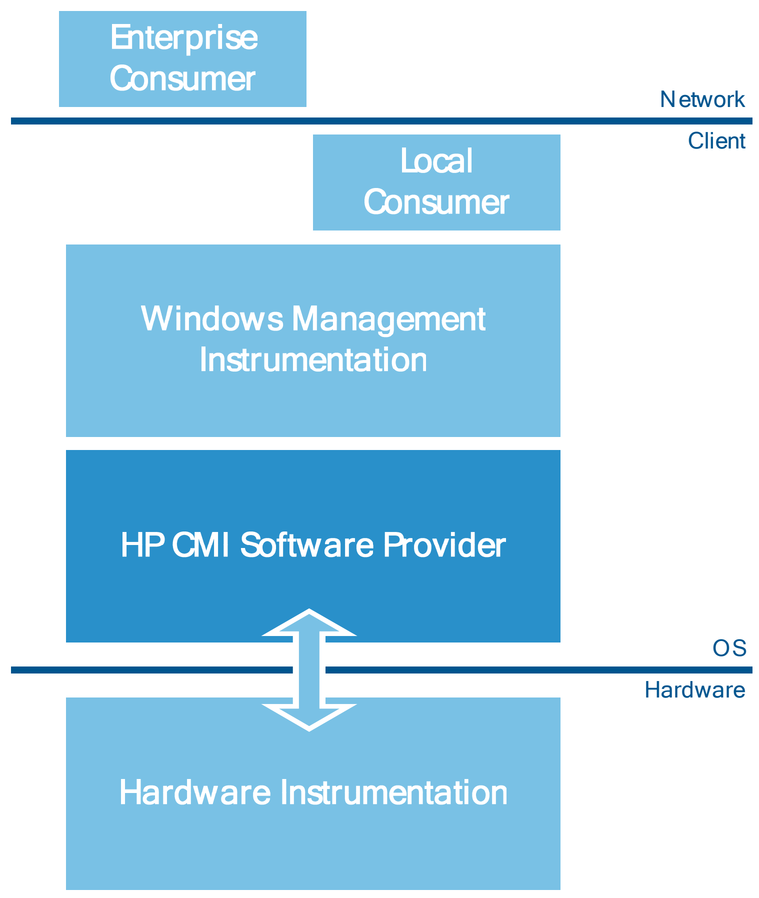
HP Client Management Interface is included standard on select new HP business client computer models. An HP CMI Software Provider is also available for legacy models. In addition, several HP Client Management Solutions leverage the features of HP CMI.

HP Client Management Interface Software Provider

The HP CMI Software Provider, available as a SoftPaq downloadable from HP.com, extends many of the capabilities of the HP Client Management Interface for legacy HP business computers. While the HP CMI Software Provider does not contain all the features and flexibility of native HP CMI support, the Software Provider allows IT managers to reap many of the same benefits exposed by HP CMI on their existing hardware infrastructure.

The following diagram shows how HP CMI Software Provider is a hybrid of the traditional management agent framework and the WMI instrumentation management model exposed by HP CMI.

HP Client Management Interface Model



HP CMI Software Provider creates the necessary definitions to support the hardware configuration option classes defined by the native HP CMI architecture. This allows for hardware configuration scripts and management application to interact with instrumented configuration data without the necessity of understanding if the underlying implementation is hardware or software based.

HP BIOS Configuration for ProtectTools

HP BIOS Configuration for ProtectTools version 2.0 provides the capability to access and modify BIOS configuration details from within the HP ProtectTools interface. This application utilizes the data-driven model used within HP CMI. Because HP CMI is designed to expose information in a consistent manner regardless of varying platform feature sets, BIOS Configuration is capable of supporting a wide range of features and platforms with minimal, if any, need to upgrade the application.

HP Client Manager

HP Client Manager provides centralized hardware management of HP business PCs, notebooks and workstations. Features include the ability to get in-depth hardware inventory information, monitor system health status, run diagnostic tests, remotely install drivers and manage BIOS settings updates without visiting each client computer. Beginning with version 6.1, HP Client Manager takes advantage of HP CMI exposed features and capabilities.

HP System Software Manager

HP System Software Manager (SSM) is a valuable tool in the custom IT solution arsenal for managing HP client computers. HP SSM supports automation of software and BIOS updates, and an ability to report and modify BIOS settings through a text-based file format. Available later this year, SSM 2.0 will leverage HP CMI to provide BIOS configuration support to 32-bit and 64-bit versions of Windows in a manner familiar to users of the utility.

Developing Custom Solutions

The following sections describe the HP CMI architecture at a detailed level for application developers and IT professionals familiar with WMI and CIM concepts. There are several reference links at the end of this paper to learn more information on CIM and the capabilities of WMI-based management solutions.

The management classes are described that are surfaced through HP CMI in managed object format (MOF) syntax. HP CMI relies on the object inheritance capability of CIM to create a flexible and extensible interface to hardware instrumentation details.

Following the explanation of the properties and methods provided via HP CMI, several examples are provided to demonstrate the capability of HP CMI in handling client management tasks.

The examples presented herein are based on Windows Scripting Host technology. However, any development environment capable of connecting to WMI could be used instead. Microsoft Visual Basic Scripting Edition is used to simplify the example scenarios.

Hardware Sensor Information

MOF Definition

```
#pragma namespace("\\\\.\\root\\HP\\InstrumentedBIOS");

[abstract]
class HP_BIOSSensor
{
  [read] string Name;
  [read] string Description;
  [read, ValueMap {"0","1","2","3","4","5","6","7","8","9",
   "10","11","12"}, Values {"Unknown","Other","Temperature",
   "Voltage","Current","Tachometer","Counter","Switch","Lock",
   "Humidity","Smoke Detection","Presence","Air Flow"}]
  uint32 SensorType;
  [read] string OtherSensorType;
  [read, ValueMap {"0","1","2","3","4","5","6","7","8","9",
   "10","11","12","13","14","15","16","17","18","..",
   "0x8000.."}, Values {"Unknown","Other","OK","Degraded",
   "Stressed","Predictive Failure","Error",
   "Non-Recoverable Error","Starting","Stopping","Stopped",
   "In Service","No Contact","Lost Communication","Aborted",
   "Dormant","Supporting Entity in Error","Completed",
   "Power Mode","DMTF Reserved","Vendor Reserved"}]
  uint32 OperationalStatus;
  [read] string CurrentState;
  [read] string PossibleStates[];
};

class HP_BIOSStateSensor : HP_BIOSSensor
```

```
{
};

class HP_BIOSNumericSensor : HP_BIOSSensor
{
  [read, ValueMap {"0","1","2","3","4","5","6","7","8","9",
   "10","11","12","13","14","15","16","17","18","19","20",
   "21","22","23","24","25","26","27","28","29","30","31",
   "32","33","34","35","36","37","38","39","40","41","42",
   "43","44","45","46","47","48","49","50","51","52","53",
   "54","55","56","57","58","59","60","61","62","63","64",
   "65"}, Values {"Unknown","Other","Degrees C","Degrees F",
   "Degrees K","Volts","Amps","Watts","Joules","Coulombs",
   "VA","Nits","Lumens","Lux","Candelas","kPa","PSI",
   "Newtons","CFM","RPM","Hertz","Seconds","Minutes",
   "Hours","Days","Weeks","Mils","Inches","Feet",
   "Cubic Inches","Cubic Feet","Meters","Cubic Centimeters",
   "Cubic Meters","Liters","Fluid Ounces","Radians",
   "Steradians","Revolutions","Cycles","Gravities","Ounces",
   "Pounds","Foot-Pounds","Ounce-Inches","Gauss","Gilberts",
   "Henries","Farads","Ohms","Siemens","Moles","Becquerels",
   "PPM (parts/million)","Decibels","DbA","DbC","Grays",
   "Sieverts","Color Temperature Degrees K","Bits","Bytes",
   "Words (data)","DoubleWords","QuadWords","Percentage"}]
  uint32 BaseUnits;
  [read] sint32 UnitModifier;
  [read] uint32 CurrentReading;
};
```

Property Details

Class Property	Description
Name	Name identifying the sensor being reported. Typically this will follow the format <device>_<device number> (ex: Fan_01).
Description	A textual description of the sensor. This may indicate which entity this sensor monitors in a form-factor that could have multiple sensors of the same type.
SesnorType	The type of the Sensor, e.g. Voltage or Temperature Sensor. If the type is set to "Other", then the OtherSensorType can be used to further identify the type, or if the Sensor has numeric readings, then the type of the Sensor can be implicitly determined by the Units. A description of the different Sensor types is as follows: A Temperature Sensor measures the environmental temperature. Voltage and Current Sensors measure electrical voltage and current readings. A Tachometer measures speed/revolutions of a device. For example, a Fan Device can have an associated Tachometer which measures its speed. A Counter is a general purpose Sensor that measures some numerical property of a Device. A Counter value can be cleared, but it never decreases. A Switch Sensor has states like Open/Close, On/Off, or Up/Down. A Lock has states of Locked/Unlocked. Humidity, Smoke Detection and Air Flow Sensors measure the equivalent environmental characteristics. A Presence Sensor detects the presence of a physical element.
OtherSensorType	A string describing the Sensor type. Used when the SensorType property is set to "Other".
OperationalStatus	Indicates the current status(es) of the element. Various operational

	statuses are defined. Many of the enumeration's values are self-explanatory. However, a few are not and are described in more detail. "Stressed" indicates that the element is functioning, but needs attention. Examples of "Stressed" states are overload, overheated, etc. "Predictive Failure" indicates that an element is functioning nominally but predicting a failure in the near future. "In Service" describes an element being configured, maintained, cleaned, or otherwise administered. "No Contact" indicates that the monitoring system has knowledge of this element, but has never been able to establish communications with it. "Lost Communication" indicates that the ManagedSystemElement is known to exist and has been contacted successfully in the past, but is currently unreachable. "Stopped" and "Aborted" are similar, although the former implies a clean and orderly stop, while the latter implies an abrupt stop where the element's state and configuration may need to be updated. "Dormant" indicates that the element is inactive or quiesced. "Supporting Entity in Error" describes that this element may be "OK" but that another element, on which it is dependent, is in error. An example is a network service or endpoint that cannot function due to lower layer networking problems. "Completed" indicates the element has completed its operation. This value should be combined with either OK, Error, or Degraded so that a client can till if the complete operation passed (Completed with OK), and failure (Completed with Error). Completed with Degraded would imply the operation finished, but did not complete OK or report an error. "Power Mode" indicates the element has additional power model information contained in the Associated PowerManagementService association.
PossibleStates	PossibleStates enumerates the string outputs of the Sensor. For example, a "Switch" Sensor may output the states "On", or "Off". Another implementation of the Switch may output the states "Open", and "Close". Another example is a NumericSensor supporting thresholds. This Sensor can report the states like "Normal", "Upper Fatal", "Lower Non-Critical", etc. A NumericSensor that does not publish readings and thresholds, but stores this data internally, can still report its states.
CurrentState	The current state indicated by the Sensor. This is always one of the "PossibleStates”.
BaseUnits	The base unit of the values returned by this Sensor. All the values returned by this Sensor are represented in the units obtained by (BaseUnits * 10 raised to the power of the UnitModifier). For example, if BaseUnits is Volts and the UnitModifier is -6, then the units of the values returned are MicroVolts.
UnitModifier	The unit multiplier for the values returned by this Sensor. All the values returned by this Sensor are represented in the units obtained by (BaseUnits * 10 raised to the power of the UnitModifier). For example, if BaseUnits is Volts and the Unit Modifier is -6, then the units of the values returned are MicroVolts.
CurrentReading	The current value indicated by the sensor.

MOF Definition

```
#pragma namespace("\\\\.\\root\\HP\\InstrumentedBIOS");

[abstract]
class HP_BIOSSetting
{
  [read] string Name;
  [read] string Value;
  [read] string Path;
  [read] uint32 IsReadOnly;
  [read] uint32 DisplayInUI;
  [read] uint32 RequiresPhysicalPresence;
  [read] uint32 Sequence;
  [read] string Prerequisites[];
};

class HP_BIOSString : HP_BIOSSetting
{
  [read] uint32 MinLength;
  [read] uint32 MaxLength;
};

class HP_BIOSInteger : HP_BIOSSetting
{
  [read] uint32 LowerBound;
  [read] uint32 UpperBound;
  [read] uint32 IntValue;
};

class HP_BIOSEnumeration : HP_BIOSSetting
{
  [read] string CurrentValue;
  [read] uint32 Size;
  [read] string PossibleValues[];
};

class HP_BIOSOrderedList : HP_BIOSSetting
{
  [read] uint32 Size;
  [read, ArrayType("orderlist")] string Elements[];
};

class HP_BIOSPassword : HP_BIOSSetting
{
  [read] uint32 MinLength;
  [read] uint32 MaxLength;
  [read] string SupportedEncoding[];
  [read] uint32 IsSet;
};

[abstract, singleton]
class HP_BIOSSettingInterface
{
  [implemented] void SetBIOSSetting(
    [out, ValueMap {"0","1","2","3","4","5","6"},
     Values {"Success","Not Supported","Unspecified Error",
     "Timeout","Failed","Invalid Parameter","Access Denied"}]
    uint32 Return,
    [in] string Name,
    [in] string Value,
    [in, optional] string Password);

  [implemented] void SetSystemDefaults(
    [out: ToSubclass ToInstance,
    [out, ValueMap {"0","1","2","3","4","5","6"},
     Values {"Success","Not Supported","Unspecified Error",
     "Timeout","Failed","Invalid Parameter","Access Denied"}]
    uint32 Return,
    [in, optional] string Password);
};
```

In the MOF definition provided, notice that all of the class properties are read only. These classes do not support update dynamic instance updates via the WMI _Put method. To change any of the instances requires using the methods surfaced from the HP_BIOSSetingInterface class. Example scripts provided later in this paper will demonstrate how to use this class appropriately.

Property Details

Class Property	**Description**
Name	This property contains the human readable name for the BIOS setting. This text should be similar to what is exposed through the F10 Computer Setup application. Setting names are unique in nature, as this value is used to identify the entity to change or update in calls through calls to the SetBIOSSetting() method.
Value	This property contains a string representation of the intended BIOS setting. List entities are separated by commas. Enumeration selections are designated by the presence of an asterisk character (ex: "*Enable, Disable" denotes a setting is enabled in an enumeration setting.
Path	This property provides a string representation of the setting hierarchy that encapsulates this instance data. Each level of the hierarchy is separated by a backslash. This hierarchy will usually follow the appearance and grouping of items within F10 Computer Setup.
IsReadOnly	Value indicating if this setting is supported by the interface method HP_BIOSSettingInterface.SetBIOSSetting(). A value of 1 indicates that this particular setting instance cannot be changed, otherwise the property is 0.
DisplayInUI	Flag indicating this component should be visible within a BIOS configuration user interface application. This property field is used by utilities such as HP BIOS Configuration for ProtectTools to filter elements that are not applicable to a given platform.
RequiresPhysicalPresence	A value of 1 indicates that attempts to modify this setting will require interactive acknowledgement during the next system startup. Otherwise the property is 0. This property is provided for future compatibility.
Sequence	This property provides an ordering sequence for the instances being enumerated through WMI. It is used in conjunction with the "Path" property to help generate UI representations of the BIOS setting data. The values are for all instances are arranged in ascending order and

	gaps in the sequence are acceptable. In the event that multiple setting instances share the same Sequence value, or the value is NULL, the Path and Name information is used to determine order.
Prerequisites	This property array allows the system BIOS to define prerequisite conditions that affect the use of the current instance. This property is provided for future compatibility.
MinLength	This property identifies the minimum string length allowed when modifying this BIOS setting. Otherwise the value is zero.
MaxLength	This property defines the maximum string length in characters.
LowerBound	This property defines the lower limit when modifying this setting.
UpperBound	This property defines the upper limit when modifying this setting.
IntValue	This property contains an integer representation of the string stored in the Value base class property.
CurrentValue	This property contains the string representation of the current active state for this BIOS setting.
PossibleValues	This property contains a string array representing the possible setting states.
Elements	This property contains a string array representing the ordered list of elements. The first entry (Element[0]) represents the first item in the ordered list.
Size	The value contained in this property denotes the number of elements contained within a corresponding array property. This field is used in conjunction with either the Elements or PossibleValues array properties.
SupportedEncoding	This property contains an array of strings representing the encoding tags the BIOS supports for denoting a password paramter string. Encoding tags are used to denote the format of a password string that is being passed into the BIOS and follow the syntax <tag/>, where tag is defined by the array element entries. "kbd" denotes a string in hexadecimal format containing keyboard scan code input. This feature should be supported by all BIOS implementations. An example of a password structured in this format would be "<kbd/>321539191E1F1F11181320", which is "my password" in US keyboard scan codes. This field provides the ability to surface new BIOS capabilities in defining password argument syntax without changing the interface design. As new elements are added to this array, new encoding features such as parameter encryption will be introduced.
IsSet	This property indicates whether a particular password setting instance is curently set (1) or blank (0). Use this property to determine that state of a password setting, since the "Value" property for a password instance will always be blank.

System Health Events

MOF Definition

```
#pragma namespace("\\\\.\\root\\WMI ");

class HPBIOS_BIOSEvent : HP_BIOSEvent
{
};

class HPBIOS_BIOSEvent : HP_BIOSEvent
{
  [read] string Name;
  [read] string Description;
  [read ValueMap {"0","1","2","3","4"}, Values {"Unknown",
   "Configuration Change","Button Pressed","Sensor",
   "BIOS Settings"}]
  uint32 Category;
  [read, ValueMap {"0","5","10","15","20","25","30"},
   Values {"Unknown","OK","Degraded/Warning",
   "Minor Failure","Major Failure","Critical Failure",
   "Non-recoverable Error"}]
  uint32 Severity;
  [read, ValueMap {"0","1","2","3","4","5","6","7","8",
   "9","10","11","12","13","14","15","16","17","18","..",
   "0x8000.."}, Values {"Unknown","Other","OK","Degraded",
   "Stressed","Predictive Failure","Error",
   "Non-Recoverable Error","Starting","Stopping","Stopped",
   "In Service","No Contact","Lost Communication","Aborted",
   "Dormant","Supporting Entity in Error","Completed",
   "Power Mode","DMTF Reserved","Vendor Reserved"}]
  uint32 Status;
};
```

Property Details

Class Property	**Description**
Name	Descriptive tag identifying the class of event.
Description	Descriptive text associated with the event, such as an error message or the physical location of the entity being evented.
Category	Provides a mechanism for ffiltering events for subscription purposes. One consumer may only be interested in button notifications, while another may be interested in BIOS setting notificatiosn. While providing additional flexibility, it is still possible to subscribe to all events.
Severity	Indicates the current health of the element. This attribute expresses the health of this element but not necessarily that of its subcomponents. The possible values are 0 to 30, where 5 means the element is entirely healthy and 30 means the element is completely non-functional. The following continuum is defined: "Non-recoverable Error" (30) - The element has completed failed and recovery is not possible. All functionality provided by this element has been lost.

	"Critical Failure" (25) - The element is non-functional and recovery MAY NOT be possible. "Major Failure" (20) - The element is failing. It is possible the some or all of the functionality of this component is degraded or not working. "Minor Failure" (15) - All functionality is available but some MAY be degraded. "Degraded/Warning" (10) - The element is in working order and all functionality is provided. However, the element is not working to the best of its abilities. For example, the element may not be operating at optimal performance or it may be reporting recoverable errors. "OK" (5) - The element is fully functional and is operating within normal operational parameters and without error. "Unknown" (0) - The implementation can not report on Severity at this time.
Status	Indicates the current status(es) of the element. Various operational statuses are defined. Many of the enumeration's values are self-explanatory. However, a few are not and are described in more detail. "Stressed" indicates that the element is functioning, but needs attention. Examples of "Stressed" states are overload, overheated, etc. "Predictive Failure" indicates that an element is functioning nominally but predicting a failure in the near future. "In Service" describes an element being configured, maintained, cleaned, or otherwise administered. "No Contact" indicates that the monitoring system has knowledge of this element, but has never been able to establish communications with it. "Lost Communication" indicates that the ManagedSystemElement is known to exist and has been contacted successfully in the past, but is currently unreachable. "Stopped" and "Aborted" are similar, although the former implies a clean and orderly stop, while the latter implies an abrupt stop where the element's state and configuration may need to be updated. "Dormant" indicates that the element is inactive or quiesced. "Supporting Entity in Error" describes that this element may be "OK" but that another element, on which it is dependent, is in error. An example is a network service or endpoint that cannot function due to lower layer networking problems. "Completed" indicates the element has completed its operation. This value should be combined with either OK, Error, or Degraded so that a client can till if the complete operation passed (Completed with OK), and failure (Completed with Error). Completed with Degraded would imply the operation finished, but did not complete OK or report an error. "Power Mode" indicates the element has additional power model information contained in the Associated PowerManagementService association.

Example applications

Retrieving BIOS Settings

The following script will enumerate all the available settings within a computer. This example uses semi-synchronous access for the purpose of simplifying the example. However, the interface supports either semisynchronous or asynchronous access.

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

strService = "winmgmts:{impersonationlevel=impersonate}//"
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSetting"

Set objWMIService = GetObject(strService & strComputer & _
    strNamespace)

Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

Counter = 1
For Each objItem In colItems
    WScript.Echo Counter & vbTab & objItem.Name & _
        " = " & objItem.Value
    Counter = Counter + 1
Next
```

Changing the Ownership Tag

Here is a sample script to change the ownership tag setting. The value field may need some modification. Note that “E302E020304” is the keyboard scan code for the keys “abc123”.

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

strService = "winmgmts:{impersonationlevel=impersonate}//"
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSettingInterface"

Set objWMIService = GetObject(strService & _
                      strComputer & strNamespace)
Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

‘ "Enter Ownership Tag" is the name of the BIOS setting
‘ instance object that we want to update.  The correct
‘ names of available settings are found by enumerating
‘ all instances of HP_BIOSSetting.
For each objItem in colItems
    objItem.SetBiosSetting oReturn, _
      "Enter Ownership Tag", _
      "Some environment-specific inventory code", _
      "<kbd/>1E302E020304"
Next

Dim strReturn
Select Case oReturn
  Case 0 strReturn = "Success"
  Case 1 strReturn = "Not Supported"
  Case 2 strReturn = "Unspecified Error"
  Case 3 strReturn = "Timeout"
  Case 4 strReturn = "Failed"
  Case 5 strReturn = "Invalid Parameter"
  Case 6 strReturn = "Access Denied"
  Case Else strReturn = "..."
End Select
WScript.Echo "SetBiosSetting() returned: (" & oReturn _
      & ") " & strReturn
```

Changing the Boot Order

Here is a sample script to change the boot order. The value field may need some modification. Note that "1E302E020304" is the keyboard scan code for the keys "abc123".

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

strService = "winmgmts:{impersonationlevel=impersonate}//"
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSettingInterface"

Set objWMIService = GetObject(strService & _
                      strComputer & strNamespace)
Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

For each objItem in colItems
    objItem.SetBiosSetting oReturn, _
      "Boot Order", _
      "Diskette,Hard Drive,Network Controller,Multibay", _
      "<kbd/>1E302E020304"
Next

Dim strReturn
Select Case oReturn
  Case 0 strReturn = "Success"
  Case 1 strReturn = "Not Supported"
  Case 2 strReturn = "Unspecified Error"
  Case 3 strReturn = "Timeout"
  Case 4 strReturn = "Failed"
  Case 5 strReturn = "Invalid Parameter"
  Case 6 strReturn = "Access Denied"
  Case Else strReturn = "..."
End Select
WScript.Echo "SetBiosSetting() returned: (" & oReturn _
      & ") " & strReturn
```

Enabling Hyper-Threading

Here is a sample script to change the hyper-threading setting. The value field may need some modification. Note that "E302E020304" is the keyboard scan code for the keys "abc123".

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

strService = "winmgmts:{impersonationlevel=impersonate}//"
```

```
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSettingInterface"

Set objWMIService = GetObject(strService & _
                      strComputer & strNamespace)
Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

For each objItem in colItems
    objItem.SetBiosSetting oReturn, _
      "Hyper-Threading", _
      "Enable", _
      "<kbd/>1E302E020304"
Next

Dim strReturn
Select Case oReturn
  Case 0 strReturn = "Success"
  Case 1 strReturn = "Not Supported"
  Case 2 strReturn = "Unspecified Error"
  Case 3 strReturn = "Timeout"
  Case 4 strReturn = "Failed"
  Case 5 strReturn = "Invalid Parameter"
  Case 6 strReturn = "Access Denied"
  Case Else strReturn = "..."
End Select
WScript.Echo "SetBiosSetting() returned: (" & oReturn _
      & ") " & strReturn
```

Setting BIOS Defaults

Here is a sample script that will reset the BIOS settings to factory defaults (or the last saved default configuration).

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

strService = "winmgmts:{impersonationlevel=impersonate}//"
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSettingInterface"

Set objWMIService = GetObject(strService & _
    strComputer & strNamespace)
Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

For each objItem in colItems
    objItem.SetSystemDefaults oReturn, "<kbd/>1E302E020304"
Next

Dim strReturn
Select Case oReturn
  Case 0 strReturn = "Success"
  Case 1 strReturn = "Not Supported"
  Case 2 strReturn = "Unspecified Error"
  Case 3 strReturn = "Timeout"
  Case 4 strReturn = "Failed"
  Case 5 strReturn = "Invalid Parameter"
  Case 6 strReturn = "Access Denied"
  Case Else strReturn = "..."
End Select
WScript.Echo "SetSystemDefaults() returned: (" & oReturn _
    & ") " & strReturn
```

Monitoring Events

Monitoring system health events is one of the more advanced aspects of the HP Client Management Interface. The approach to monitoring event presented here is designed to illustrate the capabilities of the interface, however, in an enterprise environment a more robust event consumer model would be recommended to monitor events without impacting system resources. HP CMI supports semi-synchronous and asynchronous event notifications. For more information on WMI event consumers, consult the Microsoft WMI SDK.

```
on error resume next

strService = "winmgmts:\\"
strComputer = "."
strNamespace = "\root\WMI"
strQuery = "select * from HPBIOS_BIOSEvent"

set objWMIService = GetObject( strService & strComputer _
                      & strNamespace )

set events = objWMIService.ExecNotificationQuery( strQuery )

if err <> 0 then
    WScript.Echo Err.Description, Err.Number, Err.Source
end if

WScript.Echo "Waiting for CMI Events..."
WScript.Echo "Press Ctrl-C to exit."
WScript.Echo ""

Dim strCategory
Dim strSeverity
Dim strStatus

Counter = 1

do
    ' Note this next call will wait indefinitely.
    set CMIEvent = events.nextevent

    if err <> 0 then
        WScript.Echo Err.Number, Err.Description, Err.Source
        Exit Do
    else
        Select Case CMIEvent.category
          Case 0 strCategory = "Unknown"
          Case 1 strCategory = "Configuration Change"
          Case 2 strCategory = "Button Pressed"
          Case 3 strCategory = "Sensor"
          Case 4 strCategory = "BIOS Settings"
          Case Else strCategory = "..."
        End Select
        Select Case CMIEvent.severity
          Case 0 strSeverity = "Unknown"
          Case 5 strSeverity = "OK"
          Case 10 strSeverity = "Degraded/Warning"
          Case 15 strSeverity = "Minor Failure"
          Case 20 strSeverity = "Major Failure"
          Case 25 strSeverity = "Critical Failure"
          Case 30 strSeverity = "Non-recoverable Error"
          Case Else strSeverity = "..."
        End Select
        Select Case CMIEvent.status
          Case 0 strStatus = "Unknown"
          Case 1 strStatus = "Other"
          Case 2 strStatus = "OK"
          Case 3 strStatus = "Degraded"
          Case 4 strStatus = "Stressed"
          Case 5 strStatus = "Predictive Failure"
          Case 6 strStatus = "Error"
          Case 7 strStatus = "Non-Recoverable Error"
          Case 8 strStatus = "Starting"
          Case 9 strStatus = "Stopping"
          Case 10 strStatus = "Stopped"
          Case 11 strStatus = "In Service"
          Case 12 strStatus = "No Contact"
          Case 13 strStatus = "Lost Communication"
          Case 14 strStatus = "Aborted"
          Case 15 strStatus = "Dormant"
          Case 16 strStatus = "Supporting Entity in Error"
          Case 17 strStatus = "Completed"
          Case 18 strStatus = "Power Mode"
          Case Else strStatus = "..."
        End Select
        Wscript.Echo "Event received... Count: " & Counter
        Wscript.Echo vbTab & "Name:" & vbTab & vbTab _
          & CMIEvent.name
        Wscript.Echo vbTab & "Description:" & vbTab _
          & CMIEvent.description
        Wscript.Echo vbTab & "Category:" & vbTab & strCategory
        Wscript.Echo vbTab & "Severity:" & vbTab & strSeverity
        Wscript.Echo vbTab & "Status:" & vbTab & vbTab _
          & strStatus
        Counter = Counter + 1
    end if
loop
```

Security

While the HP Client Management Interface provides a high level of control over client management instrumentation, that power must be guarded to prevent malicious, unauthorized usage.

HP CMI relies on two forms of authorization: OS level security and the BIOS administrative (F10 Setup) password assigned to each client system. Either of these security measures can be used alone, or combined to create an additional level of protection over the interface.

Preserving Password Integrity

Many of the example applications of HP CMI presented in the previous section contained the encoded password “E302E020304” that corresponds to the keyboard scan codes for the keys “abc123”. Notice that this is a form of encoding, not encryption. These examples were presented in this manner to convey the simplicity in developing custom solutions based on HP CMI. However, in an enterprise environment you probably do not want to leave traces of the Setup Password credential scattered throughout your script files.

To help preserve the integrity of the Setup Password credential, HP recommends using one of the following strategies.

Remote Execution

Executing scripts and applications from a central location such as an administrative console is more practical and secure than distributing sample scripts to individual clients through software deployment mechanisms and executing them locally. WMI supports remote invocation from any Windows system and follows the same domain and local system security policies. The calling interface is secured with stream based encryption. And by default remote method execution is prevent from WMI for all but domain administrator accounts.

Use Dynamic Arguments

Another method to preserve password integrity is to avoid carrying extra copies of the Setup Password in code, regardless of whether that code is script-based or complied. The following code fragment demonstrates using command-line arguments to modify BIOS settings.

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

Dim oArguments, strSetting, strValue, strPassword
set oArguments = WScript.Arguments

strSetting = oArguments(0)
strValue = oArguments(1)
strPassword = oArguments(2)

strService = "winmgmts:{impersonationlevel=impersonate}//"
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSettingInterface"

Set objWMIService = GetObject(strService & _
                      strComputer & strNamespace)
```

```
Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

For each objItem in colItems
    objItem.SetBiosSetting oReturn, _
      strSetting, strValue, strPassword
Next

Dim strResult
Select Case oReturn
  Case 0 strReturn = "Success"
  Case 1 strReturn = "Not Supported"
  Case 2 strReturn = "Unspecified Error"
  Case 3 strReturn = "Timeout"
  Case 4 strReturn = "Failed"
  Case 5 strReturn = "Invalid Parameter"
  Case 6 strReturn = "Access Denied"
  Case Else strReturn = "..."
End Select
WScript.Echo "SetBiosSetting() returned: (" & oReturn _
      & ") " & strReturn
```

Note that in the above example arguments are separated by spaces. To overcome this issue enclose individual arguments within quotation (") marks. An example command line based on the above script would look like:

```
C:\>cscript example.vbs "Enter Ownership Tag" "Test"
"<kbd/>1E302E020304"
```

HP Client Management Password Control

HP Client Management Interface Password Control provides two modes of operation. Stand-alone, the component can be used to convert keyboard scan codes into password text strings through a dedicated UI and cut-and-paste into the tool or code being used to modify BIOS settings. The control also provides an interactive mode of operation, in which the control can be invoked through an automation interface within a calling script or management application. The following code fragment demonstrates invoking the control from within a management script.

```
Const wbemFlagReturnImmediately = 16
Const wbemFlagForwardOnly = 32
lFlags = wbemFlagReturnImmediately + wbemFlagForwardOnly

strService = "winmgmts:{impersonationlevel=impersonate}//"
strComputer = "."
strNamespace = "/root/HP/InstrumentedBIOS"
strQuery = "select * from HP_BIOSSettingInterface"

Dim oPwdCtl, strPassword
Set oPwdCtl = CreateObject("hpPwdCtl.PasswordEdit")
oPwdCtl.GetPassword "Enter the Computer Setup Password:", _
      strPassword

Set objWMIService = GetObject(strService & _
                      strComputer & strNamespace)
Set colItems = objWMIService.ExecQuery(strQuery,,lFlags)

For each objItem in colItems
    objItem.SetBiosSetting oReturn, _
      "Hyper-Threading", _
      "Enable", _
      strPassword
Next

Dim strResult
Select Case oReturn
  Case 0 strReturn = "Success"
  Case 1 strReturn = "Not Supported"
  Case 2 strReturn = "Unspecified Error"
  Case 3 strReturn = "Timeout"
  Case 4 strReturn = "Failed"
  Case 5 strReturn = "Invalid Parameter"
  Case 6 strReturn = "Access Denied"
  Case Else strReturn = "..."
End Select
WScript.Echo "SetBiosSetting() returned: (" & oReturn _
      & ") " & strReturn
```

Note that in order to use the HP Client Management Interface Password Control as an automation component, it must be registered as an ActiveX automation component. To do this, just execute "hppwdctl.exe /install".

Configuring WMI Security

Windows Management Instrumentation (WMI) security is based on namespaces. The WMI schema is logically partitioned into namespaces for organizational and security purposes. This partitioning allows for varying security configurations to be applied to each namespace within the schema, or common security configurations to be inherited between namespaces within the schema. The WMIMGMT.MMC Microsoft Management Console (MMC) snap-in allows system administrators to modify the security attributes on WMI namespaces. In this tool, you can set security that is based off of the root or select individual namespaces. You can also use inheritance that is based on namespace hierarchy.

Use the following steps to modify WMI namespace security:

1. Click **Start**, click **Run**, type **wmimgmt.msc**, and then click **Enter**.
2. Right-click **WMI Control**, and then click on **Properties** from the context menu.
3. Click the **Security** tab to see the namespace navigation pane.
4. Highlight a namespace and click the **Security** button to see the allowable permissions.
5. Set the inheritance on the namespace.

   Enable: To grant read access to objects within the namespace.
   Execute Methods: Allows object methods exported from the CIM Object Manager to be run.
   Full Control: To grant full read/write/delete access to all CIM objects, classes, and instances.
   Partial Write: To grant write access to static objects in the repository.
   Provider Write: To grant write access to objects that are provided by the provider.
   Read Security: To grant read-only access to WMI security information.
   Edit Security: To grant read/write access to WMI security information.
   Remote Access: To grant a remote computer the same rights that are allowed when connecting from a local computer.

6. Click **Advanced**, click the specified user for whom you wish to edit the access control list, and then click **Edit**.
7. Choose the permissions that you want to grant or deny, and then under **Apply Onto**, you see the following options:

   This namespace only
   This namespace and subnamespaces
   Subnamespaces only

For more information

www.hp.com/go/easydeploy

HP Client Management Solutions

www.hp.com/go/clientmanager

HP Client Manager

www.hp.com/go/ssm

HP System Software Manager

www.hp.com/products/security

HP Business PC Security Solutions
HP ProtectTools

msdn.microsoft.com/library/en-us/dnanchor/html/anch_wmi.asp

Microsoft WMI SDK documentation

msdn.microsoft.com/downloads/list/webdev.asp

Microsoft Windows Scripting Host

www.microsoft.com/technet/scriptcenter

Microsoft Script Center

www.dmtf.org/standards/cim

Common Information Model Specification

www.dmtf.org/standards/wbem

Web-Based Enterprise Management

Call to action

HP invites you to create solutions that extend your client management capabilities with HP Client Management Interface and our additional management solutions designed to leverage and extend HP CMI within your environment.

At HP we value your ideas and suggestions on how to improve the business of enterprise client management. You are welcome to submit comments and questions related to HP CMI to cmi@hp.com.

399115 002, 07/2005