HP-UX DMI 2.0 Developer's Guide

HP-UX/HP 9000 Computers

HP-UX Desktop Management Interface

Manufacturing Part Number: Not Assigned
August 2000

Legal Notices

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company 3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Use of this manual and flexible disc(s), compact disc(s), or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

Trademark Acknowledgments

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

MS-DOS and Microsoft are U.S. registered trademarks of Microsoft Corporation.

Windows NT® is a US registered trademark of MicroSoft Corporation.

PostScript is a trademark of Adobe Systems, Inc.

Intelis a registered trademark and Intel 80386 is a trademark of Intel Corporation.

Ethernet is a trademark of Xerox Corporation.

OSF/Motif is a trademark of the Open Software Foundation, Inc. in the U.S. and other countries. Certification for conformance with OSF/Motif user environment pending.

Contents

Contents

Contents

Contents

Contents

1 Introduction

This chapter defines some DMI-specific terms and provides an overview of the DMI system

Terminology

This section defines some of the DMI-specific terms used in this guide.

Component Interface API

DMI's term for the interface that manages communication between the service provider and manageable products allowing components to be seen and managed by DMI-enabled applications. (See also “Management Interface API” on page 14.)

Component

A component is a grouping of data that can describe any of the following:

- Any piece of hardware. Some examples are a CPU, a network interface card or a pointing device.
- Any software subsystem. Some examples are a file system, an operating system or print spooling.
- Any installed software application. Some examples are word processing programs, compilers or spread sheet applications.
- Any arbitrary grouping of data that a developer wants to retrieve or set through DMI.

Component Instrumentation (CI)

The component instrumentation is the software that provides the component provider functions. It gathers the dynamic data that makes up a component and provides it to the MIF database.

Direct Interface

A direct interface component instrumentation runs as a process independently of the DMI Service Provider. It contacts the DMI Service Provider via socket IPC to register its component ID and entry points, and to notify DMI that it will handle all subsequent requests for the specified information.

Desktop Management Interface (DMI)

The DMTF's standard for management of hardware and software components in a computer system. (The standard is available from the DMTF website: www.dmtf.org)

Distributed Management Task Force (DMTF)

An industry-wide consortium committed to making desktop computer systems easier to use, understand, configure and manage. The organization's website os www.dmtf.org.

Indication Server

See Management Application Provider API Client and Management Application Provider API Server.

Management Interface API

The DMI layer between management application and the Service Provider.

Management Interface Client

DMI's term for the client side of the Management Interface service which makes calls to the MI Server to manage communication between DMI-enabled applications and the DMI SP. It allows DMI-enabled applications to access, manage and control desktop systems, components and peripherals. The MI Client is part of the management application.

Management Interface Server

DMI's term for the server side of the Management Interface service that manages communication between DMI-enabled applications and the DMI SP. It allows DMI-enabled applications to access, manage, and control desktop systems, components and peripherals. The MI server is part of the DMI SP.

Management Interface Format (MIF)

DMTF's standard syntax for describing computer system components, groups and attributes.

MIF Database

The MIF database is a data store which the Service Provider maintains. It holds the following information:

- Names, descriptions, and ID numbers of installed components.
- Names, descriptions, classes and ID numbers of the groups within the installed components.
- Names, descriptions, types, values and ID numbers of the attributes within the groups within the installed components.
- Flags telling whether the attributes are stored in the MIF database (static data) or whether component instrumentation must be called to retrieve or set the attribute.
- Entry points into the component instrumentation that must be called to retrieve or set attributes (dynamic data).

MIF File

DMI component providers ship the MIF file with the component instrumentation to describe the instrumented component. A single MIF file describes exactly one component. The component can have any number of groups and groups can have any number of attributes. The Service Provider uses the MIF file to generate entries within the MIF Database that describe the component identified by the MIF file.

Overlay Component

Developers write an Overlay component as a shared library. The DMI SP loads the shared library to handle requests for information related to the component. As a shared library, the Overlay component is a part of the DMI Service Provider's process. DMI SP explicitly loads and unloads the shared library as needed for each data request.

Row

A row of data describes an instance of data in a group. Scalar group rows refer to all the attributes in the associated group. Table group rows refer to a particular set of all columns (Index attribute, Name attribute, Position attribute). For example, row 1 identifies the set of (1,"Frank","Electrical Engineer").

```
Index   Name      Position
=====   =====     ==============
  1     Frank     Electrical Engineer
  2     Howard    Electrical Engineer Intern
```

Scalar Group

This is a group that has one set of values for the described attributes. The lack of the MIF keyword key implies the group is scalar.

Service Provider (SP)

This is the HP-UX daemon that receives RPC calls from management applications and manages the MIF information found in its MIF database. This daemon also provides an interface to direct interface and overlay components by starting them and calling them when it receives attribute requests from management applications.

Table Group

This is a multidimensional group. It has more than one instance of the attributes described. This type of a group is implied when the MIF keyword key is used. This type of group is typically implemented as an array or other kind of multiple value structure (list, tree etc.). See Row, above.

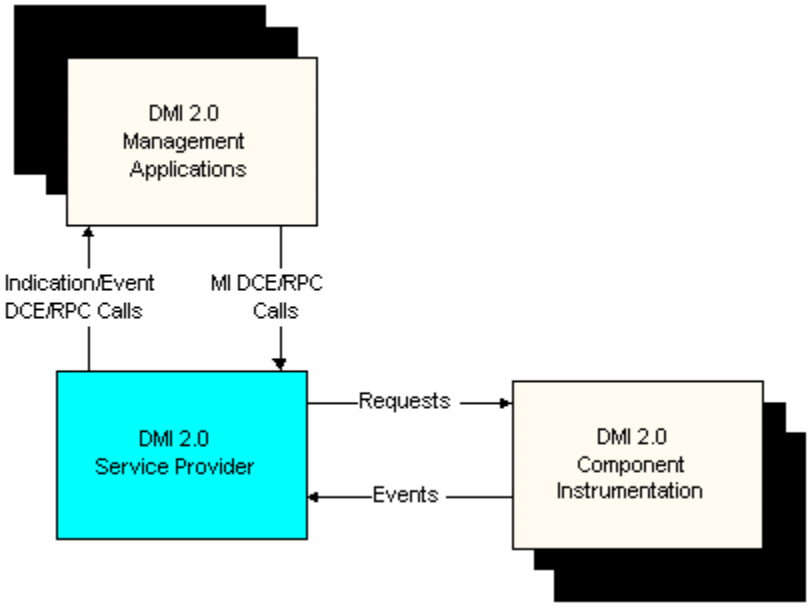
Overview of the Desktop Management Interface

The Desktop Management Interface, or DMI, is a layer of abstraction between management software, or management applications and a system's components that are able to be managed by DMI. Components are physical or logical entities on a system such as hardware, software or firmware.

Management application developers use a set of functions called the Management Interface, or MI to manage components. Likewise, component developers use a set of functions called the Component Interface, or CI to describe access to management information and enable a component to be managed.

Mediation between the Management Interface and the Component Interface is handled by an active, resident code set, or daemon, called the DMI Service Provider (SP). The DMI SP handles the run-time management which includes component installation, registration, request serialization and synchronization, and general flow control and housekeeping. HP implements the SP as the daemon process, dmisp. The SP is started at system boot.

Figure 1-1 Desktop Management Interface Block Diagram



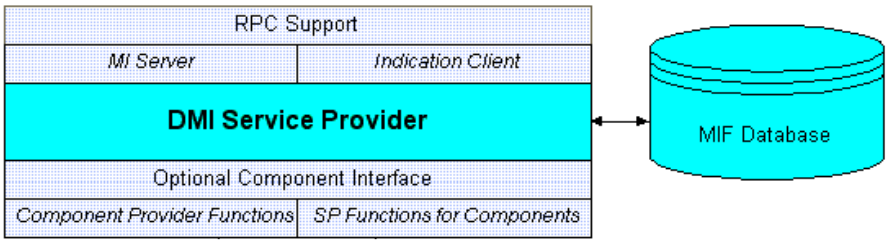
All managed components and the DMI Service Provider exist within a single system. The management applications may be command-line or graphical user interface programs, located on the local system or on remote management systems.

The relationships among management applications, the DMI SP and component instrumentation may be many-to-one. There may be many management applications issuing commands through a single DMI SP to manage many components.

Within DMI, information about all components is defined in a language called the Management Interface Format (MIF) and stored by the DMI SP in a MIF database. Each component has a unique MIF file which describes its manageable characteristics. Each component's MIF file is registered in a MIF database, and owned by the DMI SP. For more information on MIF files, see Writing Component MIF Files in the Component section of this Developer's Guide.

Within DMI, information about all components is defined in a language called the Management Interface Format (MIF) and stored by the DMI SP in a MIF database. Each component has a unique MIF file which describes its manageable characteristics. Each component's MIF file is registered in a MIF database, and owned by the DMI SP. For more information on MIF files, see Writing Component MIF Files in the Component section of this Developer's Guide.

Figure 1-2 Service Provider to MIF Relationship



Installing DMI

In order to use HP's Desktop Management Interface SDK, you must install the following three software packages:

1. swinstall the HP C/ANSI C Developer's Bundle for HP-UX 10.20 or the aC++ Compiler S800;
2. swinstall the DCE-BPRG fileset from the DCE Programming Environment and Libraries bundle;
3. Download the DMI SDK from the HP Software Depot website at http://www.software.hp.com.

When running the SDK, you must be root user to get or set the DMI client machine list in /var/dmi/dmiMachines.

2 SDK Tools

Three tools are provided with the DMI Software Developer's Kit. Each is described in one of the following sections:

The MIF Browser

The MIF browser is a graphical user interface designed for viewing the contents of the MIF database on a local or remote DMI-enabled system. It is invoked using the command:

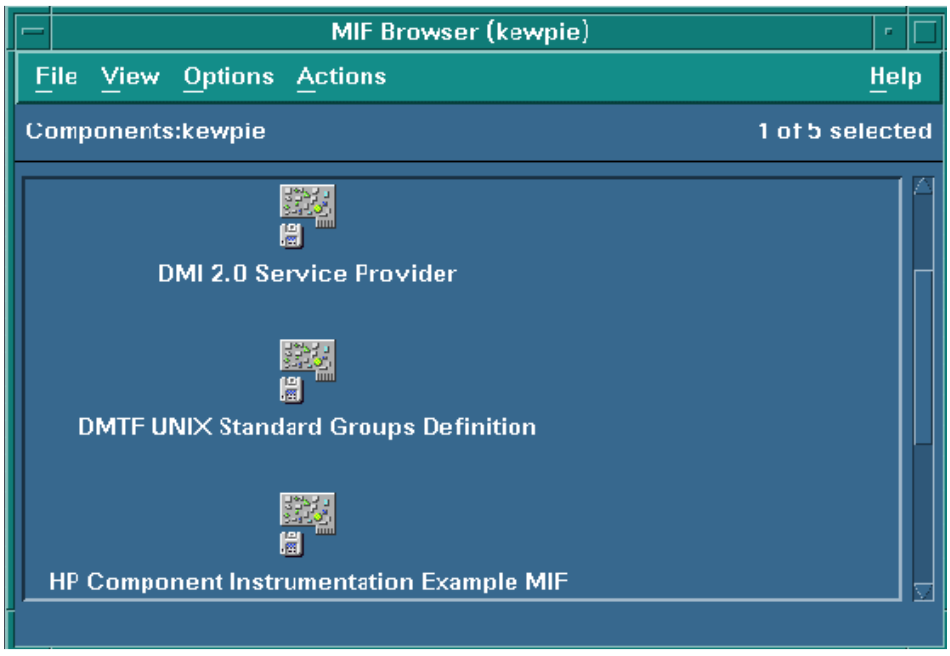
`/usr/dmi/bin/browser`

The browser initially displays a system view which lists the local system MIF database. You “Open” the view to remote systems via the **Actions** menu.

Double-clicking on a system icon causes the browser to list all the components on that machine. Similarly, you can open a component to list its groups and open a group to list its attributes and associated attribute values. You can not make changes to components from the MIF Browser.

The following illustration shows a component-level view of the system “kewpie.” This illustration shows three of five installed components. Note the scroll bar on the right side of the window.

Figure 2-1 Component-Level View of MIF Browser



QueryDB

QueryDB scans the MIF database belonging to the local machine and lists its components, groups, attributes, and attribute values through stdout.

It is invoked with the command:

/usr/dmi/bin/QueryDB -m *machine_name>*

where -m *machine_name* specifies a remote DMI-enabled system.

A partial listing looks similar to the following:

```
Dmi Specification Level:  DMI 2.0
Dmi Description:   HP-UX 10.x DMI 2.0 Service Provider,
Version 0.91 %P
Component id:   1
Component name:   DMI 2.0 Service Provider
            Group id:   1
            Group name:   ComponentID
            Class name:   DMTF|ComponentID|001
            description:   Defines attributes common to all
                               components.
                    Attribute id:1
                    Attribute name:Manufacturer
            description:   The organization that produced
                               this component.
                    Attribute storage:MIF_COMMON
                    Attribute access:MIF_READ_ONLY
                    Attribute type:MIF_DISPLAYSTRING
                    Attribute maxSize:64
                    Attribute id:2
                    Attribute name:Product
            description:   The name of this component or
                                 product.
                    Attribute storage:MIF_COMMON
                    Attribute access:MIF_READ_ONLY
     ...
     ...
     ...
```

DMI Log Viewer

The DMI Log Viewer is a graphical user interface that allows you to read and search specified DMI log files by message level, user and time range. The default log file is /var/dmi/log/dmisp_log. It is invoked with the command:

```
/usr/dmi/bin/dmilog_viewer
```

3 Component Interface Concepts

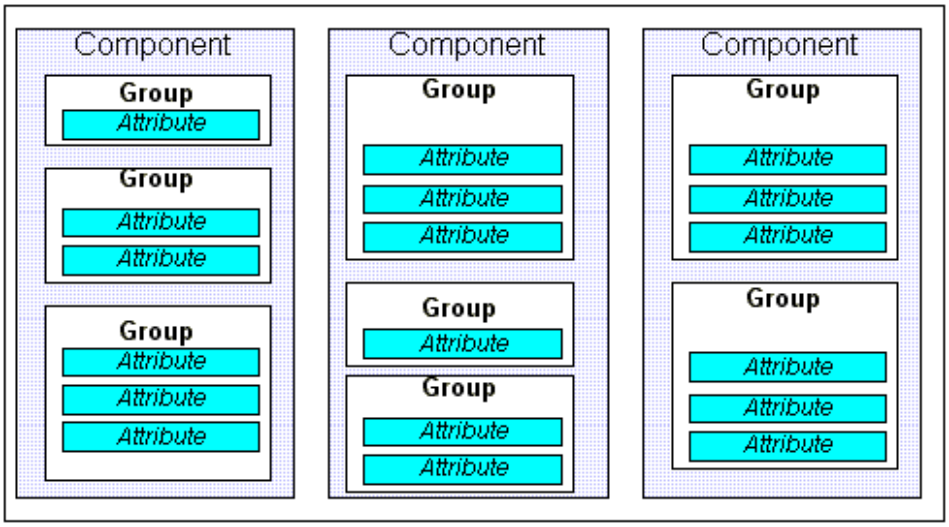
The Component Interface is an optional interface allowing managed components to connect directly to the DMI SP. It is a procedural interface that is completely synchronous with the SP acting as the broker to ensure that component code does not need to be re-entrant.

The following sections discuss the two types of data that describe components and the two types of components that developers may implement with HP-UX DMI 2.0. The last section discusses Component Interface security strategies.

Static vs. Dynamic Data

Components have one or more named attributes that collectively define the information available to a management application. Attributes are collected into named groups for ease of reference. So, within a system, there are many components, each with one or more groups. Each group has one or more attributes.

Figure 3-1 **System Components Showing Groups and Attributes**



The data stored within attribute values may be dynamic or static. If the data is unlikely to change, you should consider making the data static. If the data is likely to change often, you should make the data dynamic. The following paragraphs describe the differences between the two types of data and provide some examples.

Static data does not require component instrumentation to get and set its attribute values. Instead, the attribute values are stored in an associated MIF file and installed into the SP's MIF (information) database. The DMI SP gets and sets the data from the MIF.

The following example shows the attribute values of ComponentID: a group required by the MIF file. If a management application needs the attribute value for Manufacturer, the SP gets the attribute value directly from the MIF database and returns it to the management application:

```
Start Component
      Name = "DMTF Unix Standard Groups Definition"
      Description = "The DMTF system standard groups”
                    “definition for Unix vendors."
      Start Group
           Name = "ComponentID"
           Class = "DMTF|ComponentID|001"
```

```
         ID = 1
         Description = "System Memory Group"

        Start Attribute
            Name = "Manufacturer"
            ID = 1
             Description = "Manufacturer of this"
                         "component."
            Access = Read-Write
            Storage = Common
            Type = String(64)
            Value = "Hewlett-Packard, Co."
         End Attribute
// ===============================================
// ComponentID attributes 2-6 deleted for example clarity.
// ===============================================
      End Group
    End Component
```

Dynamic data requires component instrumentation intervention. The values do not reside in the MIF database. The attribute values in the MIF file are place holders for the real attribute values.

The following examples shows how the component instrumentation retrieves an attribute Names. The SP learns from the MIF database that the data is dynamic. It gets the data from the component instrumentation and passes it back to the management application.

```
Start Component
   .
   .
   .
   //================================================
   // ComponentID group deleted for example clarity.
   //================================================
        Start Group
            Name = "Names"
            ID   = 2
            Class = "DMTF|DevNames|2.0"
             Description = "DMTF Developers names.Direct"
                         "Interface Version"
            Key=1

            Start Attribute
                Name   = "Index"
                ID     = 1
                Description = "Index into names table."
                Access = READ-ONLY
                Type   = INTEGER
                Value  = 1
            End Attribute
              .
              .
              .
```

```
        End Group
    End Component
```

When you choose to use dynamic data, you must also decide what type of component instrumentation accesses the data. The HP-UX DMI 2.0 implementation supports two types of component instrumentation. They are Direct Interface (DI) and Overlay Interface. The following sections describe each type.

Direct Interface vs. Overlay Instrumentation

Both Direct Interface (DI) and Overlay Interface component instrumentation interact with the SP through the Component Interface to provide dynamic attribute data.

Direct-Interface Instrumentation

Direct Interface component instrumentation is implemented as a separate application loaded into memory independently of the SP. In HP-UX, this type of component instrumentation is an HP-UX daemon. DI is advantageous when system overhead to get or set attributes is high. However, with DI system resources remain in use.

HP recommends using DI component instrumentation only when the associated MIF file has large amounts of dynamic data or the system overhead for retrieving and setting the dynamic data is high.

Debugging Direct-Interface Instrumentation

The DMI SP has a configurable timeout for communication with a Direct CI. Adjust this value so that the `DMI SP` will not return the error code `DMIERR_SP_INACTIVE` to the Management Interface requesting information from the CI you are debugging. Modify the timeout environment variable, `DMISP_DITIMEOUT`, and restart the `DMI SP`:

```
Ksh $ export DMISP_DITIMEOUT=some_number
```

The maximum value is two hours and the default value is 100 seconds.

Be cautious of semaphores when debugging Direct Interface components. Terminate Component Instrumentation properly by calling `DmiUnregisterCi()` and exiting.

The current implementation of the Service Provider will not allow Direct Interface Component Instrumentation to remain running over a Service Provider restart. If the Service Provider goes down, the instrumentation will return from the `DmiMainLoop()` function.

Overlay Instrumentation

Overlay components are implemented as shared libraries that are identified by an associated MIF file and are loaded by the SP to get and

set dynamic data. The shared library remains loaded for a period of time following the attribute access.

The attributes that the SP gets and sets using the Overlay component instrumentation are also identified in the associated MIF file. The shared library must reside in /usr/dmi/overlay to run.

Overlay component instrumentation is advantageous when the overhead to get or set attribute values is low, and the frequency for getting and setting attributes is low. However, with Overlay instrumentation event-generating attributes can not be programmed.

Overlay component instrumentation is the preferred method for building component instrumentation. The conditions for which HP does not recommend using Overlay component instrumentation are when the MIF file defines event generating attributes and when the component defines large amounts of dynamic data.

Ill-behaved Overlay components can crash the SP if they receive segmentation violations or perform illegal operations.

The following example illustrates setting up a MIF Path statement and using the Path name as the value for each attribute that represents the dynamic data. For example:

```
Start Path
     Name = "Overlay Component Instrumentation"
     Value = *"Overlay Component Instrumentation"
     Unix = "/usr/dmi/bin/overlayMemoryExample"
End Path
```

For additional information on creating MIF files, see the following chapter, Writing Component MIF Files.

Group-Level Security

HP provides a group-level security mechanism for its implementation of DMI 2.0. In order to use this security feature, the management application and the component instrumentation developers must agree to not ship the security libraries separately. Furthermore, executables using this security feature statically link to the security libraries.

Intel recommends this type of security and it must be enforced by the component instrumentation developer.

The following sections describe how to implement group-level security. The description is derived from the paper DMI 2.0 Security Token Proposal written by Brodi Beartusk and John Keith of Intel Corporation.

Modifying Groups to Use Security Tokens

Component instrumentation supporting this group requires that management applications provide the Security Token attribute when making a call. This attribute is passed on as the keylist for the operation. Applications not in possession of the Security Token can not access any attributes in the group. Component instrumentation denies access to applications that request the first row of the group.

In order to secure a scalar group, you must first convert it to a tabular group by adding a Security Token attribute to the group definition, then specify this attribute as the key attribute for the group. The following example shows how to redefine the scalar System Memory Group as a tabular group with a Security Token.

Original Definition (scalar)

```
Start Group
      Name = "System Memory Group"
      Class = "HP|System Memory Group|001"
      ID = 2

      Start Attribute
            Name = "Total Physical Memory"
            ID = 1
            Type = int64
            Access = Read-Write
            Storage = Specific
            Value = 0
      End Attribute
End Group
```

Modified Definition (tabular)

```
Start Group
      Name = "System Memory Group"
      Class = "HP|System Memory Group|001"
      ID = 2
      Key = 2
      Start Attribute
            Name = "Total Physical Memory"
            ID = 1
            Storage = Specific
            Access = Read-Write
            Type = int64
            Value = 0
      End Attribute
      Start Attribute
            Name = "Security Token"
            ID = 2
            Type = "OctetString"
            Access = Read-Only
            Value = ""
      End Attribute
End Group
```

Securing an existing tabular group is much like securing a scalar group. First, add a new Security Token attribute to the group definition. Next, augment the existing keylist definition to include this new attribute. The Security Token then becomes one of the attributes comprising a unique row. This means that it is required for any get or set command on the table. Additionally, DmiAddRow() and DmiDeleteRow() calls require a valid keylist. Thus, they also require the Security Token.

Security Token Handling

Alternatively, HP provides a library with a security token-generating procedure and a security token validation procedure. Management applications can use these procedures to secure access to protected attribute values.

Additionally, HP provides a library to management application writers that wish to retrieve protected attributes from HP's implementation of the Unix Standards Group Definition (Unix MIF). This library contains the security token-generating procedure.

These two procedures are:

- `GenerateToken()`
- `ValidateToken()`

For detailed information, see the generateToken(3X) and validateToken(3X) manpages.

4 Writing Component MIF Files

Management Information Format (MIF) Files

Within DMI, information about all components is defined in a language called the Management Interface Format (MIF) and stored by the DMI SP in a MIF database. Each component has a unique MIF file which describes its manageable characteristics. Each MIF file describes one, and only one component. However, there may be multiple instances of the component on a single system.

When a management application or component instrumentation asks the SP to add a component, the SP compiles the associated MIF file containing schema description data, checks it for adherence to the DMI MIF format and installs it in the MIF database. The MIF database is then accessed when a management application asks the SP to list, get or set the attributes of installed components.

During initialization, if the SP detects that the MIF database is corrupt it removes the MIF database and creates a new MIF database containing the SP MIF and any MIF files that have been installed into the /usr/dmi/mif/C directory. If component instrumentation installs MIF files into the MIF database without copying the MIF file to the /usr/dmi/mif/C directory, that component instrumentation is responsible for reinstalling its MIF file into the MIF database after the SP detects a corrupt database.

The HP-UX DMI 2.0 Software Developer's Kit includes a Unix Standards Group Definition MIF and associated instrumentation (hpuxci), a software MIF and associated instrumentation (swci), the Service Provider MIF, as well as several example MIF files in the /usr/dmi/examples/ci directory.

Each MIF file contains one, and only one component definition. The component definition has the following syntax:

```
start component
          name = "component name"
          [description = " description string" ]
          [pragma = "pragma string" ]
          [component definition goes here]
end component
```

In addition, a MIF file contains some of the following definitions:

Table 4-1 Definitions Contained in a MIF File

Definition	Purpose
Path Definitions	Locate files used to manage the component. There may be one path definition for each callable function.
Enum Definitions	Allow strings to be associated with 32-bit integers. There may potentially be one enum definition for each enumeration list.
Group Definitions	Collection of one or more attributes, arranged into logical sets. There must be one group called ComponentID. There may be zero to many additional groups.
Pragma Statements	Provides additional information about the component, group or attribute.
Attribute Statements	Provides data about the data, and is contained within the group statement. Each group must have at least one attribute statement.

Every component MIF must contain a ComponentID group with ID 1. This group provides the base-level identification of the component and represents the minimum amount of information that a component developer should provide.

The ComponentID class string is “DMTF|ComponentID|001”. The six named attributes of the group are Manufacturer, Product, Version, Serial Number, Installation, and Verify. Refer to http://www.dmtf.org for conformance requirements.

Example MIF

The following is an example of a MIF file for an HP Overlay component.

```
Start Component
   Name  = "HP Overlay Component Example MIF"
   Description = "This is an example of an overlay component"
                      "instrumentation-type MIF."
   Start Path
      Name = "Overlay Component Instrumentation"
      Unix = "/usr/dmi/bin/OverlayMemoryExample"
   End Path

   Start Group
      Name = "ComponentID"
      Class = "DMTF|ComponentID|001"
      ID = 1
      Start Attribute
        Name = "Manufacturer"
        ID = 1
        Description = "Manufacturer of this component."
                  Access = Read-Write
                  Storage = Common
                  Type = String(64)
                  Value = "Hewlett-Packard Company"
      End Attribute
// ======================================================
// ComponentID attributes 2-6 deleted for example clarity
// ======================================================
   End Group
   Start Group
      Name = "System Memory Group"
      Class = "HP|SystemMemoryGroup|001"
      ID = 2
      Description = "System Memory Group"
      Start Attribute
        Name = "Total Available Memory"
        ID = 1
        Description = "The total available memory on"
                                 "this system."
        Type = int64
        Access = Read-Write
        Storage = Specific
        Value = *"Overlay Component Instrumentation"
      End Attribute
   End Group
End Component
```

For more information on writing MIFs, see the example MIF file `/var/dmi/mif/C/systemExample.mif`, or see the DMTF DMI Specification Version 2.0

5 Component Interface Data Structures

Within DMI data structures, all strings are stored in the form:

length data

where *length* is an unsigned 32-bit value giving the number of octets (8-bit untyped data) in the *data* part of the string. Note that the number of characters in the string depends on whether it is in ISO 8859-1 format (1 octet/character) or Unicode format (2 octets/character). For DMI 2.0, String *data* values must be NULL terminated in addition to the *length* specifier.

The data structures are described below:

DmiAccessData

This data structure contains group/attribute access ID for instrumentation registering the Direct Interface.

Table 5-1 DMI Access Data

Field Name	Description
groupId	Group that uses the Direct Interface. A value of zero indicates that all groups, except the ComponentID group, within this MIF use the Direct Interface, and the following AttributeId field is ignored.
attributeId	Attributes, within the group specified by GroupId, that use the Direct Interface. A value of zero indicates that all attributes within this group use the Direct Interface.

```
typedef struct DmiAccessData {
          DmiId_t     groupId;
          DmiId_t     attributeId;
    } DmiAccessData_t;
```

DmiAccessDataList

This data structure describes an array of DmiAccessData.

Table 5-2 DMIAccessDataList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiAccessDataList {
          DmiUnsigned_t         size;
          DmiAccessData_t*      list;
    } DmiAccessDataList_t;
```

DmiRegisterInfo

This data structure identifies entry points for registering CI Direct Interface code.

Table 5-3 DmiRegisterInfo

Field Name	Description
componentId	Identifier assigned by the service provider on component installation
ciGetAttribute	Address of the CiGetAttribute entry point
ciGetNextAttribute	Address of the CiGetNextAttribute entry point
ciReserveAttribute	Address of the CiReserveAttrribute entry point
ciReleaseAttribute	Address of the CiReleaseAttrribute entry point
ciSetAttribute	Address of the CiSetAttribute entry point
ciAddRow	Address of the CiAddRow entry point
ciDeleteRow	Address of the CiDeleteRow entry point
accessData	Array containing the groups and/or individual attributes that use the Direct Interface

```
typedef struct DmiRegisterInfo {
          DmiId_t                  componentId;
          CiGetFunc                ciGetAttribute;
          CiGetFunc                ciGetNextAttribute;
          CiResRelFunc             ciReserveAttribute;
          CiResRelFunc             ciReleaseAttribute;
          CiSetFunc                ciSetAttribute;
          CiAddDelFunc             ciAddRow;
          CiAddDelFunc             ciDeleteRow;
          DmiAccessDataList_t      *accessData;
   }DmiRegisterInfo_t;
```

6 Component Interface Functions

There are three classes of functions in the HP-UX DMI 2.0 Component Interface, they are as follows:

Service Provider Functions for Components	Implemented by the DMI SP and may be invoked by component developers. These include registration and indication origination functions.
Component Provider Functions	Implemented by component developers and may be invoked by the DMI SP.
Optional Component Interface Support Functions	Authorization techniques for the security provided by the DMI SP.

The following sections list and provide a brief description of each CI function call. For detailed information on a function, see the function man pages included with this SDK, or see the DMTF DMI Version 2.0 Specification.

Service Provider Functions for Components

Using the following Service Provider Functions for Components, component instrumentation code can register with the SP to override its current access mechanism for the registered attributes. Instead of manipulating the data in the MIF database or invoking programs, the SP calls the entry points provided in the registration call. Once the component unregisters, the SP returns to its normal method of processing requests for the data as defined in the SP's MIF. In this manner, component instrumentation temporarily interrupts normal processing to perform special functions.

DmiMainLoop()

Waits on the socket communication thread for component instrumentation or the Indication Server thread for management applications.

```
int DmiMainLoop()
```

DmiOriginateEvent()

Originates an event for filtering and delivery. This function performs any necessary indication filtering (or subsequent processing) prior to forwarding the event to the registered management applications.

```
DmiErrorStatus_t DMI_API DmiOriginateEvent(
   /* [in] */ DmiId_t                componentId,
   /* [in] */ DmiString_t     *      language,
   /* [in] */ DmiTimestamp_t  *      timestamp,
   /* [in] */ DmiRowData_t    *      rowData);
```

DmiRegisterCi()

Registers a callable interface for components that have resident instrumentation code and/or to get the version of the DMI SP. Components cannot register with regInfo->componentId=1. That ID is reserved for the Service Provider. The DMI SP sends the signal SIGUSR2 when it has come up after a planned restart. The CI must then use this call and re-register with the SP.

```
DmiErrorStatus_t DMI_API DmiRegisterCi(
     /* [in]  */   DmiRegisterInfo_t *    regInfo,
     /* [out] */   DmiHandle_t       *    handle,
     /* [out]  */   DmiString_t     **    dmiSpecLevel);
```

DmiUnregisterCi()

Instructs the SP to remove a Direct Interface table of registered interfaces.

```
DmiErrorStatus_t DMI_API DmiUnregisterCi(
     /* [in] */  DmiHandle_t           handle);
```

Component Provider Functions

These functions are the entry points into the component instrumentation that the SP calls to get or set attribute values, or to add or delete whole rows of information from the MIF database.

All of the Component Provider Functions for the Direct Interface Component Instrumentation example reside in the source file /usr/dmi/examples/ci/systemExample/ci_cpfc.c. Similarly, all of the Component Provider Functions for the Overlay Interface Component Instrumentation example reside in the source file /usr/dmi/examples/oi/names.c.

There are two types of functions for calling component instrumentation within the Component Provider Functions. The first type lends itself to getting and setting attribute values for single attributes in a group. These functions can be applied to either scalar or tabular data.

- "CiGetAttribute()" on page 44
- "CiGetNextAttribute()" on page 45
- "CiReleaseAttribute()" on page 45
- "CiReserveAttribute()" on page 45
- "CiSetAttribute()" on page 45
- "CiAddRow()" on page 46
- "CiDeleteRow()" on page 46

CiGetAttribute()

Corresponds directly to the Management Interface call DmiGetAttribute(). Gets the value of a single attribute within a group.

```
DmiErrorStatus_t DMI_API CiGetAttribute(
     /*  [in]  */  DmiId_t                componentId,
     /*  [in]  */  DmiId_t                    groupId,
     /*  [in]  */  DmiId_t                attributeId,
     /*  [in]  */  DmiString_t          *     language,
     /*  [in]  */  DmiAttributeValues_t  *      keyList,
     /*  [out] */  DmiAttributeData_t    **       data);
```

CiGetNextAttribute()

Gets the value of the attribute immediately following the currently referenced attribute.

```
DmiErrorStatus_t DMI_API CiGetNextAttribute(
     /* [in] */  DmiId_t                    componentId,
     /* [in] */  DmiId_t                        groupId,
     /* [in] */  DmiId_t                    attributeId,
     /* [in] */  DmiString_t            *      language,
     /* [in] */  DmiAttributeValues_t   *       keyList,
     /* [out] */  DmiAttributeData_t    **       data);
```

CiReleaseAttribute()

Releases the resources previously reserved with a call to CiReserveAttribute().

```
DmiErrorStatus_t DMI_API CiReleaseAttribute(
     /* [in] */  DmiId_t                    componentId,
     /* [in] */  DmiId_t                        groupId,
     /* [in] */  DmiId_t                    attributeId,
     /* [in] */  DmiAttributeValues_t *         keyList,
     /* [in] */  DmiAttributeData_t   *           data);
```

CiReserveAttribute()

Reserves the required resources to set the specified attribute.

```
DmiErrorStatus_t DMI_API CiReserveAttribute(
     /* [in] */  DmiId_t                   componentId
     /* [in] */  DmiId_t                       groupId,
     /* [in] */  DmiId_t                   attributeId,
     /* [in] */  DmiAttributeValues_t *        keyList,
     /* [in] */  DmiAttributeData_t    *          data);
```

CiSetAttribute()

Sets the specified attribute with the given value.

```
   DmiErrorStatus_t DMI_API CiSetAttribute
     /* [in] */  DmiId_t                    componentId,
     /* [in] */  DmiId_t                        groupId,
     /* [in] */  DmiId_t                    attributeId,
```

The second type of function is used to add or delete rows of information or all attribute values in a group. These functions can only manipulate tabular groups.

CiAddRow()

Adds a row of data to an existing table in the MIF database. Corresponds to the Management Interface DmiAddRow() call.

```
DmiErrorStatus_t DMI_API CiAddRow(
     /* [in] */  DmiRowData_t          *    rowData);
```

CiDeleteRow()

Deletes a row of data from an existing table in the MIF database. Corresponds to the Management Interface DmiDeleteRow() call.

```
DmiErrorStatus_t DMI_API CiDeleteRow(
     /* [in] */  DmiRowData_t          *     rowData);
```

Optional Component Interface Support Functions

validateToken()

Validates a security token that has been passed in by the management application to gain access to a protected attribute value.

```
DmiErrorStatus_t DMI_API validateToken(
     /* [in] */  DmiString_t  *        componentName,
     /* [in] */  DmiString_t  *        groupName,
     /* [in] */  DmiString_t  **       securityToken);
```

7 Management Interface Concepts

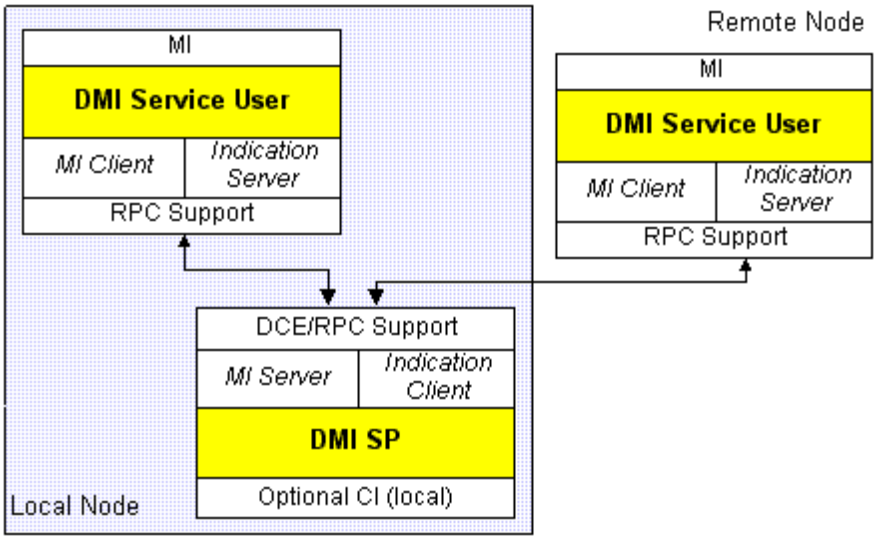
Management Interface is the DMI term for the procedural interface that manages communication between DMI-enabled applications and the Service Provider. This interface allows DMI-enabled applications to access, manage and control desktop systems. It is remotable based on remote procedure calls (RPC) and implemented using DCE. The Management Interface protects management application developers from needing to know specific information about the DMI Service Provider and Component Interface.

The following sections provide more detailed information about the MI's remotability and security mechanisms.

DCE/RPC Remotability

The DMI 2.0 procedural interfaces are designed to be remotely accessible through the use of RPC. HP-UX implemented these RPCs using the Distributed Computing Environment (DCE/RPC).

Figure 7-1 DCE/RPC Remote Procedural Interface



RPC is based on a client/server architecture. The client side includes a set of function calls which have interfaces with the same signatures as the function calls they represent on the server. The calls interact with the local RPC support to exchange the input parameters, the output parameters and the return codes with the remote procedure located on the server. A remote node acts as a client for procedural MI function calls and acts as a server when receiving indications. The managed node acts as a server for procedural MI function calls and acts as a client when delivering indications to a remote node.

MI Security

DMI does not provide primitives to own or lock resources over a sequence of commands. Multiple management applications may make simultaneous accesses to the interfaces. Grouping and scheduling of operations, other than the synchronization provided by the DMI SP, are the responsibility of the management application. Likewise, any desire for mutual exclusion to lock out certain accesses or to provide DMI database security in any form is the responsibility of the management application.

The HP-UX implementation of DMI 2.0 stresses the need to protect attribute values. The attribute names do not require protection as they do not provide useful information.

There are three levels of security for the HP-UX implementation of DMI 2.0. These are:

- Group-level security provided by the component developer.
- Machine list protection;
- User name check on HP-UX clients;

These levels are described in the following sections.

Group-Level Security

For detailed information on group-level security, see Group-Level Security in the Component Interface Concepts chapter.

Intel recommends this type of security and it must be enforced by the component instrumentation developer.

Access to protected attribute values requires a Security Token. Group level security requires that the Component Instrumentation developer and the management application developer agree on a Security Token. HP provides a library with a Security Token-generating procedure and a Security Token validation procedure.

Additionally, HP provides a library to management application writers that wish to retrieve protected attributes from HP's implementation of the Unix Standard Groups Definition. This library also has the Security Token generating procedure.

Machine List Protection

This type of security uses knowledge of the DCE/RPC binding to determine the machine making the DMI call. It then checks the machine name against an internal table of allowed machine names. If the machine name or IP address matches, access to attribute values is granted.

The system administrator grants access rights to machines by editing the file /var/dmi/dmiMachines. The format of this file is any valid hostname or complete internet address, one entry per line. The SP detects changes to this file when a client gets or sets an attribute value.

Any machine on the network may view the data description (attribute names) of the MIF database. The following calls are restricted by the machine list protection.

- DmiGetAttribute() / DmiSetAttribute()
- DmiGetMultiple() / DmiSetMultiple()
- DmiAddRow() / DmiDeleteRow()
- DmiAddGroup() / DmiDeleteGroup()
- DmiAddComponent() / DmiDeleteComponent()
- DmiAddLanguage() / DmiDeleteLanguage()

User Name Check on HP-UX Clients

This type of security verifies that a client making a DMI call is the root user. This keeps unknown users from accessing the MIF database. Only the following calls are restricted by the user name check.

- DmiGetAttribute() / DmiSetAttribute()
- DmiGetMultiple() / DmiSetMultiple()
- DmiAddRow() / DmiDeleteRow()
- DmiAddGroup() / DmiDeleteGroup()
- DmiAddComponent() / DmiDeleteComponent()
- DmiAddLanguage() / DmiDeleteLanguage()

8 Management Interface Data Structures

Within DMI data structures, all strings are stored in the form:

length data

where *length* is an unassigned 32-bit value giving the number of octets (8-bit untyped data) in the *data* part of the string. Note that the number of characters in the string depends on whether it is in ISO 8859-1 format (1 octet/character) or Unicode format (2 octets/character). For DMI 2.0, String *data* values must be NULL terminated in addition to the *length* specifier.

DMI Data Types

The DMI data types below adhere to the naming convention for DCE/RPC data types. DCE data types have the following size representations:

Table 8-1 DMI Data Types

IDL Datatypes	Size
char	8 bits
boolean	8 bits
long	32 bits
hyper	64 bits
unsigned long	32 bits
unsigned hyper	64 bits

```
typedef unsigned long           DmiCounter_t;
    typedef unsigned hyper          DmiCounter64_t;
    typedef unsigned long           DmiErrorStatus_t;
    typedef unsigned long           DmiGauge_t;
    typedef unsigned long           DmiHandle_t;
    typedef unsigned long           DmiId_t;
    typedef long                    DmiInteger_t;
    typedef hyper                   DmiInteger64_t;
    typedef unsigned long           DmiUnsigned_t;
    typedef boolean                 DmiBoolean_t;
```

DMI Enumerated Types

DmiAccessMode

This enumerated type defines the access modes for an attribute.

Table 8-2 DmiAccessMode

Field Name	Description
MIF_UNKNOWN_ACCESS	Unknown access mode.
MIF_READ_ONLY	Read access only.
MIF_READ_WRITE	Readable and writable.
MIF_WRITE_ONLY	Write access only.
MIF_UNSUPPORTED	Attribute is not supported.

```
typedef enum {
        MIF_UNKNOWN_ACCESS,
        MIF_READ_ONLY,
        MIF_READ_WRITE,
        MIF_WRITE_ONLY,
        MIF_UNSUPPORTED
    } DmiAccessMode_t;
```

DmiDataType

This enumerated type defines the data types referenced by `DmiDataUnion`.

Table 8-3 DmiDataType

Field Name	Description
MIF_DATATYPE_0	RESERVED
MIF_COUNTER	32-bit unsigned integer that never decreases.
MIF_COUNTER64	64-bit unsigned integer that never decreases.
MIF_GAUGE	32-bit unsigned integer that may increase or decrease.
MIF_DATATYPE_4	RESERVED
MIF_INTEGER	32-bit signed integer.
MIF_INTEGER64	64-bit signed integer.
MIF_OCTETSTRING	String of n octets, not necessarily displayable.
MIF_DISPLAYSTRING	Displayable string of n octets.
MIF_DATATYPE_9	RESERVED
MIF_DATATYPE_10	RESERVED
MIF_DATE	28-octet displayable string (yyyymmddhhmmss.uuuuuu+ooo)

```
typedef enum {
        MIF_DATATYPE_0,
        MIF_COUNTER,
        MIF_COUNTER64,
        MIF_GAUGE,
        MIF_DATATYPE_4,
        MIF_INTEGER,
        MIF_INTEGER64,
        MIF_OCTETSTRING,
        MIF_DISPLAYSTRING,
        MIF_DATATYPE_9,
        MIF_DATATYPE_10,
        MIF_DATE
    } DmiDataType_t;
```

DmiFileType

This data structure defines the DMI mapping file types.

Table 8-4 DmiFileType

Field Name	Description
DMI_FILETYPE_0	RESERVED
DMI_FILETYPE_1	RESERVED
DMI_MIF_FILE_NAME	File data is the name of the DMI MIF file data.
DMI_MIF_FILE_DATA	File data is the contents of the DMI MIF file data.
SNMP_MAPPING_FILE_NAME	File data is the name of an SNMP mapping file data.
SNMP_MAPPING_FILE_DATA	File data is the contents of an SNMP mapping file data.
DMI_GROUP_FILE_NAME	File data is the name of a DMI GROUP file data.
DMI_GROUP_FILE_DATA	File data is the contents of a DMI GROUP file data.
VENDOR_FORMAT_FILE_NAME	File data is the name of a Vendor-format data file data.
VENDOR_FORMAT_FILE_DATA	File data is the contents of a Vendor-format data file data.

```
typedef enum {
        DMI_FILETYPE_0,
        DMI_FILETYPE_1,
        DMI_MIF_FILE_NAME,
        DMI_MIF_FILE_DATA,
        SNMP_MAPPING_FILE_NAME,
        SNMP_MAPPING_FILE_DATA,
        DMI_GROUP_FILE_NAME,
        DMI_GROUP_FILE_DATA,
        VENDOR_FORMAT_FILE_NAME,
        VENDOR_FORMAT_FILE_DATA
    } DmiFileType_t;
```

DmiRequestMode

This data structure defines sequential access modes.

Table 8-5 DmiRequestMode

Field Name	Description
DMI_UNIQUE	Access the specified item or table row.
DMI_FIRST	Access the first item.
DMI_NEXT	Access the next item.

```
typedef enum {
        DMI_UNIQUE,
        DMI_FIRST,
        DMI_NEXT
    } DmiRequestMode_t;
```

DmiSetMode

This data structure describes set operations.

Table 8-6 DmiSetMode

Field Name	Description
DMI_RESERVE	Reserve resources for a set operation.
DMI_SET	Set data values.
DMI_RELEASE	Release previously reserved resources.

```
typedef enum {
        DMI_SET,
        DMI_RESERVE,
        DMI_RELEASE
    } DmiSetMode_t;
```

DmiStorageType

This data structure defines the storage type for an attribute.

Table 8-7 DmiStorageType

Field Name	Description
MIF_COMMON	Value is from a small set of possibilities.
MIF_SPECIFIC	Value is from a large set of possibilities.

```
typedef enum {
        MIF_COMMON,
        MIF_SPECIFIC
    } DmiStorageType_t;
```

DMI Data Structures

DmiAttributeData

This data structure describes an attribute ID, type, and value.

Figure 8-1 DMI Attribute ID, Type and value

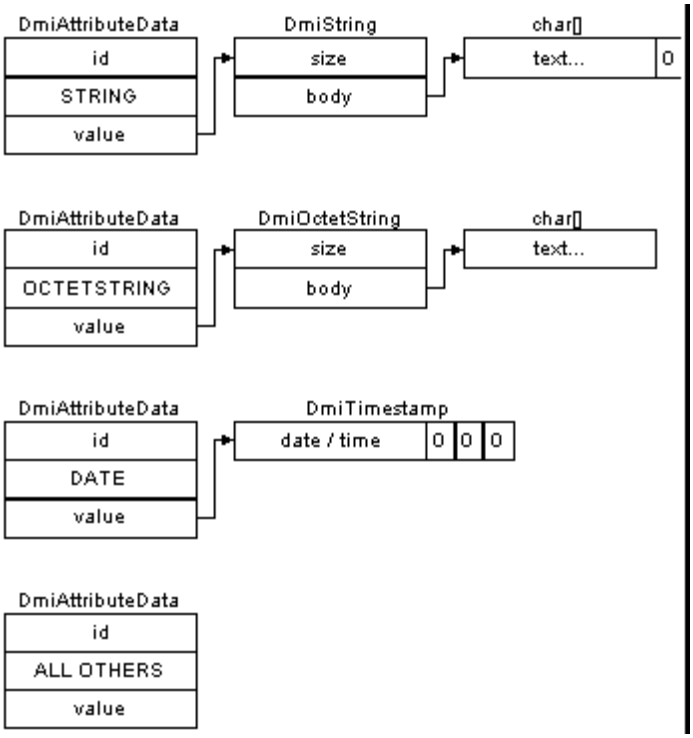


Table 8-8 DmiAttributeData

Field Name	Description
Id	Attribute ID
Data	Attribute type and value
Field Name	Description
Id	Attribute ID
Data	Attribute type and value

```
typedef struct DmiAttributeData {
          DmiId_t          id;
          DmiDataUnion_t   data;
    } DmiAttributeData_t;
```

DmiAttributeIds

This data structure describes a conformant array of DmiId_t.

Table 8-9 DmiAttributeIds

Field Name	Description
Size	Array elements
List	Array data

```
typedef struct DmiAttributeIds {
          DmiUnsigned_t   size;
          DmiId_t*        list;
    } DmiAttributeIds_t;
```

DmiAttributeInfo

This data structure holds information about an attribute.

Figure 8-2 Attribute Information Data Structure

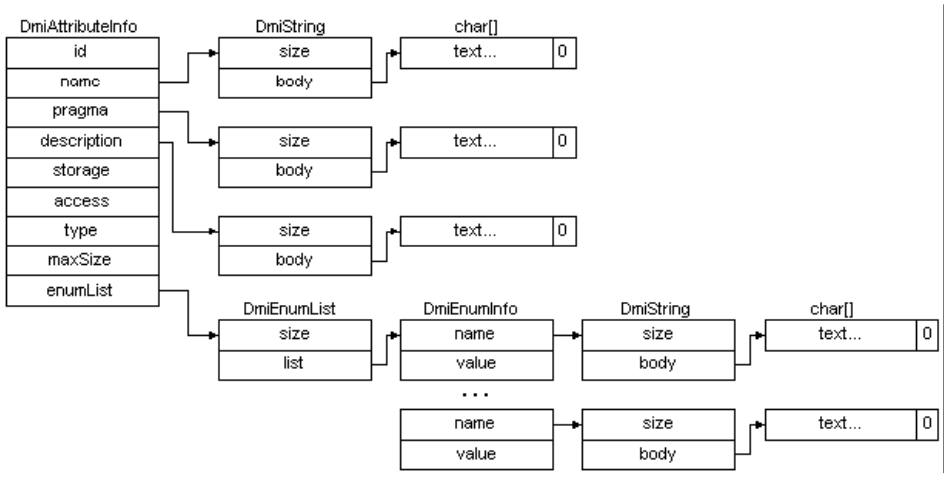


Table 8-10 DmiAttributeInfo

Field Name	Description
Id	Attribute ID
Name	Attribute name string
Pragma	Attribute pragma string [optional]
Description	Attribute description string [optional]
Storage	Common or specific storage
Access	read-only, read-write, etc.
Type	Counter, integer, etc.
MaxSize	Maximum length of the attribute
EnumList	EnumList for enumerated types [optional]

```
typedef struct DmiAttributeInfo {
          DmiId_t               id;
          DmiString_t*          name;
          DmiString_t*          pragma;
          DmiString_t*          description;
          DmiStorageType_t      storage;
          DmiAccessMode_t       access;
          DmiDataType_t         type;
          DmiUnsigned_t         maxSize;
          struct DmiEnumList*   enumList;
    } DmiAttributeInfo_t;
```

DmiAttributeList

This data structure describes a conformant array of DmiAttributeInfo_t.

Table 8-11 DmiAttributeList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiAttributeValues {
          DmiUnsigned_t         size;
          DmiAttributeData_t*   list;
    } DmiAttributeValues_t;
```

DmiAttributeValues

This data structure describes a conformant array of DmiAttributeData_t.

Table 8-12 DmiAttributeValues

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiAttributeValues {
          DmiUnsigned_t         size;
          DmiAttributeData_t*   list;
    } DmiAttributeValues_t;
```

DmiClassNameInfo

This data structure holds a group's ID and class string.

Table 8-13 DmiClassNameInfo

Field Name	Description
id	Group ID
className	Group class name string

```
typedef struct DmiClassNameInfo {
          DmiId_t        id;
          DmiString_t*   className;
    } DmiClassNameInfo_t;
```

DmiClassNameList

This data structure describes a conformant array of DmiClassNameInfo_t.

Table 8-14 DmiClassNameList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiClassNameList {
          DmiUnsigned_t         size;
          DmiClassNameInfo_t*   list;
    } DmiClassNameList_t;
```

DmiComponentInfo

This data structure holds information about a component.

Figure 8-3 Component Data Structure

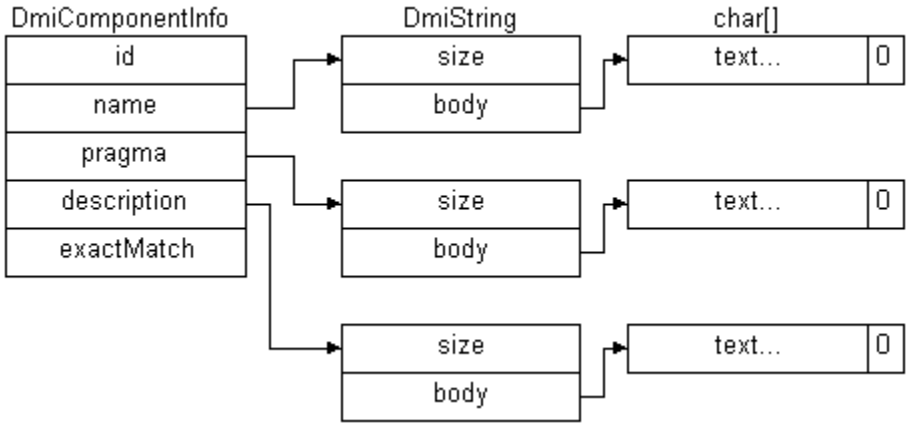


Table 8-15 DmiComponentInfo

Field Name	Description
id	Component ID
name	Component name string
pragma	Component pragma string [optional]
description	Component description string [optional]
exactMatch	TRUE = Exact match FALSE = Possible match
Field Name	Description
id	Component ID
name	Component name string
pragma	Component pragma string [optional]
description	Component description string [optional]
exactMatch	TRUE = Exact match FALSE = Possible match

```
typedef struct DmiComponentInfo {
          DmiId_t       id;
          DmiString_t*  name;
          DmiString_t*  pragma;
          DmiString_t*  description;
          DmiBoolean_t  exactMatch;
    } DmiComponentInfo_t;
```

DmiComponentList

This data structure describes a conformant array of DmiComponentInfo.

Table 8-16 DmiComponentList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiComponentList {
          DmiUnsigned_t       size;
          DmiComponentInfo_t* list;
    } DmiComponentList_t;
```

DmiDataUnion

This data structure is a discriminated union of DMI data types.

Table 8-17 DmiDataUnion

Field Name	Description
type	Discriminator for the union
value	Union of DMI attribute data types

```
typedef union
   switch (DmiDataType_t type) value {
   case MIF_COUNTER:       DmiCounter_t       counter;
   case MIF_COUNTER64:     DmiCounter64_t     counter64;
   case MIF_GAUGE:         DmiGauge_t         gauge;
   case MIF_INTEGER:       DmiInteger_t       integer;
   case MIF_INTEGER64:     DmiInteger64_t     integer64;
   case MIF_OCTETSTRING:   DmiOctetString_t*  octetstring;
   case MIF_DISPLAYSTRING: DmiString_t*       string;
   case MIF_DATE:          DmiTimestamp_t*    date;
} DmiDataUnion_t;
```

DmiEnumInfo

This data structure associates an integer value with descriptive text.

Table 8-18 DmiEnumInfo

Field Name	Description
name	Enumeration name
value	Enumeration value

```
typedef struct DmiEnumInfo {
          DmiString_t*  name;
          DmiInteger_t  value;
    } DmiEnumInfo_t;
```

DmiEnumList

This data structure describes a conformant array of DmiEnumInfo_t.

Table 8-19 DmiEnumList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiEnumList {
          DmiUnsigned_t   size;
          DmiEnumInfo_t*  list;
    } DmiEnumList_t;
```

DmiFileDataInfo

This data structure holds language file type and mapping data.

Table 8-20 DmiFileDataInfo

Field Name	Description
fileType	MIF file, SNMP mapping file, etc.
fileData	The file info (name or contents)

```
typedef struct DmiFileDataInfo {
          DmiFileType_t  fileType;
          DmiString_t*   fileData;
    } DmiFileDataInfo_t;
```

DmiFileDataList

This data structure describes a conformant array of DmiFileDataInfo_t.

Table 8-21 DmiFileDataList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiFileDataList {
          DmiUnsigned_t      size;
          DmiFileDataInfo_t* list;
    } DmiFileDataList_t;
```

DmiFileTypeList

This data structure describes a conformant array of DmiFileTypes. It is used by the DmiGetVersion function to return a list of file types supported by the DmiAddComponent, DmiAddLanguage, and DmiAddGroup functions.

Table 8-22 DmiFileTypeList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiFileTypeList {
          DmiUnsigned_t   size;
          DmiFileType_t*  list;
    } DmiFileTypeList_t;
```

DmiGroupInfo

This data structure holds information about a group.

Figure 8-4 Group Information Data Structure

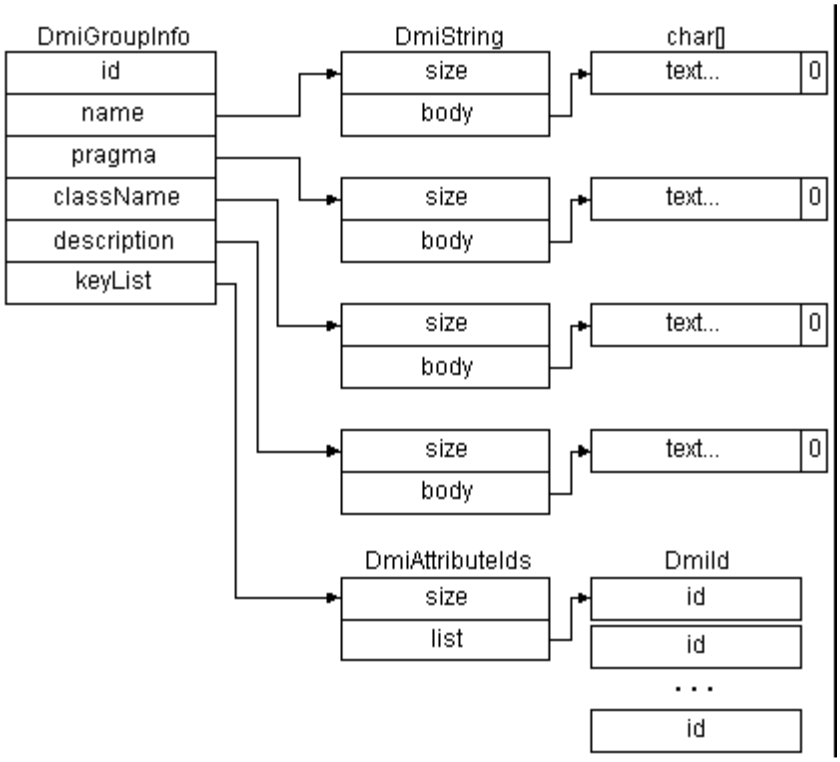


Table 8-23 DmiGroupInfo

Field Name	Description
id	Group ID
name	Group name string
pragma	Group pragma string [optional]
className	Group class name string
description	Group description string [optional]
keyList	Attribute Ids for table row keys

```
typedef struct DmiGroupInfo {          DmiId_t
id;
          DmiString_t*              name;
          DmiString_t*              pragma;
          DmiString_t*              className;
          DmiString_t*              description;
          struct DmiAttributesIds*  KeyList;
    } DmiGroupInfo_t;
```

DmiGroupList

This data structure describes a conformant array of DmiGroupInfo_t.

Table 8-24 DmiGroupList

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiGroupList {
          DmiUnsigned_t   size;
          DmiGroupInfo_t* list;
    } DmiGroupList_t;
```

DmiMultiRowData

This data structure describes a conformant array of DmiRowData_t.

Table 8-25 DmiMultiRowData

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiMultiRowData {
          DmiUnsigned_t   size;
          DmiRowData_t*   list;
    } DmiMultiRowData_t;
```

DmiMultiRowRequest

This data structure describes a conformant array of DmiRowRequest_t.

Table 8-26 DmiMultiRowRequest

Field Name	Description
size	Array elements
list	Array data

```
typedef struct DmiMultiRowRequest {
          DmiUnsigned_t    size;
          DmiRowRequest_t* list;
    } DmiMultiRowRequest_t;
```

DmiNodeAddress

This data structure describes addressing information for indication originators.

Table 8-27 DmiNodeAddress

Field Name	Description
address	Transport-dependent node address
rpc	Identifies the RPC (DCE, ONC, etc)
transport	Identifies the transport (TCP/IP, SPX, etc.)

```
typedef struct DmiNodeAddress {
          DmiString_t*  address;
          DmiString_t*  rpc;
          DmiString_t*  transport;
    } DmiNodeAddress_t;
```

DmiOctetString

This data structure defines the DMI octet string representation

Table 8-28 DmiOctetString

Field Name	Description
size	Number of octets in the string body
body	String contents

```
typedef struct DmiOctetString {
          DmiUnsigned_t  size;
          char*          body;
    } DmiOctetString_t;
```

DmiRowData

This data structure identifies {component, group, row, IDs} to set.

Figure 8-5 Row Data Structure

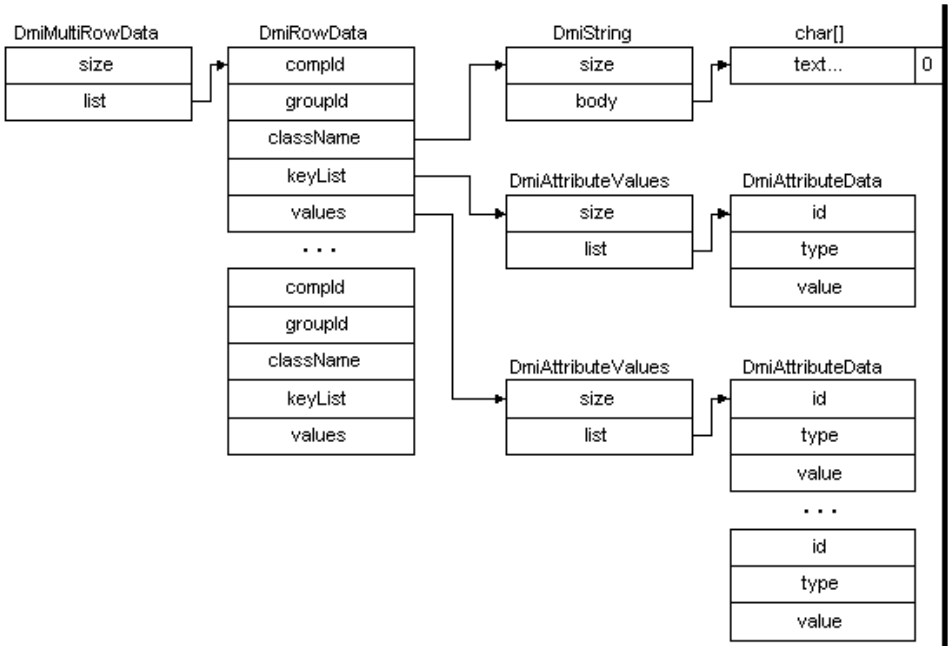


Table 8-29 DmiRowData

Field Name	Description
compId	Component ID
groupId	Group ID
className	Class name string for the group. Used for indications.
keyList	Array of values for key attributes
values	Array of values for data attributes

```
typedef struct DmiRowData {
          DmiId_t                    compId;
          DmiId_t                    groupId;
          DmiString_t*               className;
          struct DmiAttributeValues* keyList;
          struct DmiAttributeValues* values;
    } DmiRowData_t;
```

DmiRowRequest

This data structure identifies {component, group, row, IDs} to get.

Figure 8-6 Row Request Data Structure

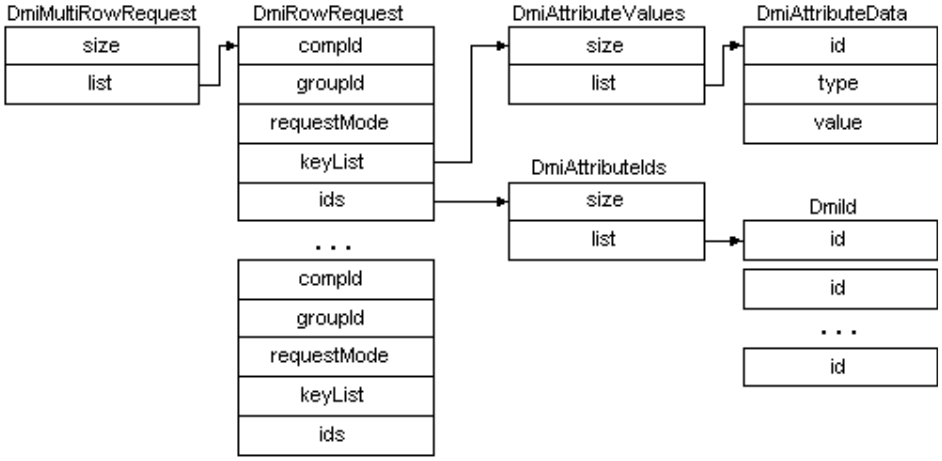


Table 8-30 DmiRowRequest

Field Name	Description
compId	Component ID
groupId	Group ID
requestMode	Get from specified row, first row, or next row
keyList	Array of values for key attributes
ids	Array of Ids for data attributes

```
typedef struct DmiRowRequest {
          DmiId_t                    compId;
          DmiId_t                    groupId;
          DmiRequestMode_t           requestMode;
          struct DmiAttributeValues* keyList;
          struct DmiAttributeIds*    ids;
    } DmiRowRequest_t;
```

DmiString

This data structure defines the DMI string representation. All DmiStrings must be null terminated.

Table 8-31 DmiString

Field Name	Description
size	Number of octets in the string body including the terminating null character (Note: null is 2 octets in Unicode)
body	String contents

```
typedef struct DmiString {
          DmiUnsigned_t  size;
          char*          body;
    } DmiString_t;
```

DmiStringList

This data structure describes a conformant array of DmiString_t*.

Table 8-32 DmiStringList

Field Name	Description
groupId	Group that uses the Direct Interface. A value of zero indicates that all groups, except the ComponentID group, within this MIF use the Direct Interface, and the following AttributeId field is ignored.
attributeId	Attributes, within the group specified by GroupId, that use the Direct Interface. A value of zero indicates that all attributes within this group use the Direct Interface.

```
typedef struct DmiStringList {
          DmiUnsigned_t  size;
          DmiString_t**  list;
    } DmiStringList_t;
```

DmiTimestamp

This data structure describes the time format used by DMI. The format of the time block is a 28-octet displayable string with ISO 8859-1 encoding. Each element is one or more printable characters.

For example, Wednesday May 25, 1994 at 1:30:15 PM EDT is represented as:

19940525133015.000000-300

A seconds value of 60 is used for leap seconds.

The offset from UTC (Coordinated Universal Time) is the number of minutes west (negative number) or east offset from UTC.

Values are zero-padded. If a value is not supplied for a field, each character in the field must be replaced with asterisk ('*') characters.

The DMI Server is not required to check the contents of this string for validity.

Table 8-33 DmiTimestamp

Field Name	Description
year	The year
month	The month ('1'..'12')
day	The day of the month ('1'..'31')
hour	The hour of the day ('0'..'23')
minutes	The minutes ('0'..'59')
seconds	The seconds ('0'..'60')
dot	A dot ('.'}
microSeconds	MicroSeconds ('0'..'999999')
plusOrMinus	'+' for east, or '-' west of UTC
utcOffset	Minutes ('0'..'720') from UTC
padding	Unused padding for 4-byte alignment

```
typedef struct DmiTimestamp {

          char year         [4];
          char month        [2];
          char day          [2];
          char hour         [2];
          char minutes      [2];
          char seconds      [2];
          char dot;
          char microSeconds [6];
          char plusOrMinus;
          char utcOffset    [3];
          char padding      [3];
   } DmiTimestamp_t;
```

9 Management Interface Functions

There are three classes of functions in HP's DMI 2.0 Management Interface. They are as follows:

Service Provider Functions for Management Applications	Implemented by the DMI SP and may be invoked by management application developers. These include initialization, listing, operation, and database administration functions.
Management Application Provider Functions	Implemented by the management application developers and may be invoked by the DMI SP.
Optional Management Interface Support Functions (Memory Handling, Security and Message Logging Functions)	HP's implementation of DMI 2.0 supports a set functions to ease memory handling and allocation, security and logging.

The following sections list and provide brief descriptions of each MI function call. For detailed information on a function, see the function man pages, or see the DMTF DMI Version 2.0 Specification.

Service Provider Functions for Management Applications

Management applications must register with the DMI Service Provider. The following functions provide this capability. These calls contain fields which rarely change between a manager and a client.

Initialization Functions

DmiGetConfig()

Retrieves the per-session string describing the current language in use.

```
DmiErrorStatus_t DMI_API DmiGetConfig (
    /* [in]  */  DmiHandle_t         handle,
    /* [out] */  DmiString_t**       language);
```

DmiGetVersion()

Retrieves information about the SP's specification level and the MIF file types supported.

```
DmiErrorStatus_t DMI_API DmiGetVersion(
   /* [in]  */ DmiHandle_t            handle,
   /* [out] */ DmiString_t**          dmiSpecLevel,
   /* [out] */ DmiString_t**          description,
   /* [out] */ DmiFileTypeList_t**    fileTypes);
```

DmiMainLoop()

Waits on the socket communication thread for component instrumentation or the Indication Server thread for management applications.

```
int DmiMainLoop()
```

DmiRegister()

Provides the management application or direct component instrumentation with a unique per-session handle. The SP uses this procedure to initialize its internal state for subsequent calls. Local use only.

```
DmiErrorStatus_t DMI_API DmiRegister(
   /* [out] */ DmiHandle_t*           handle);
```

DmiRemoteRegister()

Provides a unique per-session handle. The SP uses this procedure to initalize its internal state for subsequent calls to the remote node.

```
DmiErrorStatus_t DMI_API DmiRemoteRegister(
```

```
  /* [out] */ DmiHandle_t       *        handle,
  /* [in]  */ DmiNodeAddress_t  *        node);
```

DmiRemoteUnregister()

Unregisters the management application. The DMI SP uses this command to perform its end-of-session cleanup actions.

```
DmiErrorStatus_t DMI_API DmiRemoteUnregister(
  /* [in]  */ DmiHandle_t            handle);
```

DmiSetConfig()

Sets the per-session string describing the language required by the management application.

```
DmiErrorStatus_t DMI_API DmiSetConfig(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiString_t**          language);
```

DmiUnregister()

Unregisters the management application. The DMI SP uses this command to perform its end-of-session cleanup actions. Local use only.

```
DmiErrorStatus_t DMI_API DmiUnregister(
  /* [in]  */ DmiHandle_t            handle);
```

Listing Functions

The Listing Functions provide sequential or random access to component, group and attribute information.

DmiListAttributes()

Retrieves the properties for one or more attributes in a specified group.

```
DmiErrorStatus_t DMI_API DmiListAttributes(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiRequestMode_t      requestMode,
  /* [in]  */ DmiUnsigned_t         maxCount,
  /* [in]  */ DmiBoolean_t          getPragma,
  /* [in]  */ DmiBoolean_t          getDescription,
  /* [in]  */ DmiId_t               compId,
  /* [in]  */ DmiId_t               groupId,
  /* [in]  */ DmiId_t               attribId,
  /* [out] */ DmiAttributeList_t**  reply);
```

DmiListClassNames()

Retrieves the class name strings for all groups in a component.

```
DmiErrorStatus_t DMI_API DmiListClassNames(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiUnsigned_t         maxCount,
  /* [in]  */ DmiId_t               compId,
```

```
  /* [out] */ DmiClassNameList_t**  reply);
```

DmiListComponents()

Retrieves the name and (optionally) the description of components in a system.

```
DmiErrorStatus_t DMI_API DmiListComponents(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiRequestMode_t      requestMode,
  /* [in]  */ DmiUnsigned_t         maxCount,
  /* [in]  */ DmiBoolean_t          getPragma,
  /* [in]  */ DmiBoolean_t          getDescription,
  /* [in]  */ DmiId_t               compId,
  /* [out] */ DmiComponentList_t**  reply);
```

DmiListComponentsByClass()

Lists components which match specified criteria.

```
DmiErrorStatus_t DMI_API DmiListComponentsByClass(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiRequestMode_t      requestMode,
  /* [in]  */ DmiUnsigned_t         maxCount,
  /* [in]  */ DmiBoolean_t          getPragma,
  /* [in]  */ DmiBoolean_t          getDescription,
  /* [in]  */ DmiId_t               compId,
  /* [in]  */ DmiString_t*          className,
  /* [in]  */ DmiAttributeValues_t* keyList,
  /* [in]  */ DmiComponentList_t**  reply);
```

DmiListGroups()

Retrieves a list of groups within a component.

```
DmiErrorStatus_t DMI_API DmiListGroups(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiRequestMode_t      requestMode,
  /* [in]  */ DmiUnsigned_t         maxCount,
  /* [in]  */ DmiBoolean_t          getPragma,
  /* [in]  */ DmiBoolean_t          getDescription,
  /* [in]  */ DmiId_t               compId,
  /* [in]  */ DmiId_t               groupId,
  /* [out] */ DmiGroupList_t**      reply);
```

DmiListLanguages()

Retrieves the set of language mappings installed for the specified component.

```
DmiErrorStatus_t DMI_API DmiListLanguages(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiUnsigned_t          maxCount,
  /* [in]  */ DmiId_t                compId,
  /* [out] */ DmiStringList_t**      reply)
```

Operation Functions

This set of functions gets and sets the values of one or more attributes in the MIF database.

DmiAddRow()

Adds a row to an existing table.

```
DmiErrorStatus_t DMI_API DmiAddRow(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiRowData_t*         rowData)
```

DmiDeleteRow()

Removes a row from an existing table.

```
DmiErrorStatus_t DMI_API DmiDeleteRow(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiRowData_t*         rowData)
```

DmiGetAttribute()

Retrieves a single attribute value from the DMI SP.

```
DmiErrorStatus_t DMI_API DmiGetAttribute(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiId_t               compId,
  /* [in]  */ DmiId_t               groupId,
  /* [in]  */ DmiId_t               attribId,
  /* [in]  */ DmiAttributeValues_t* keyList
  /* [out] */ DmiDataUnion_t**      value);
```

DmiGetMultiple()

Retrieves value for a single attribute, or values for multiple attributes across groups, components or rows of a table from the DMI SP.

```
DmiErrorStatus_t DMI_API DmiGetMultiple(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiMultiRowRequest_t* request,
  /* [out] */ DmiMultiRowData_t**   rowData);
```

DmiSetAttribute()

Sets a single attribute value.

```
DmiErrorStatus_t DMI_API DmiSetAttribute(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiId_t               compId,
  /* [in]  */ DmiId_t               groupId,
  /* [in]  */ DmiId_t               attribId,
  /* [in]  */ DmiAttributeValues_t* keyList,
  /* [in]  */ DmiSetMode_t          setMode,
  /* [in]  */ DmiDataUnion_t*       value)
```

DmiSetMultiple()

Performs a set operation on an attribute or list of attributes.

```
DmiErrorStatus_t DMI_API DmiSetMultiple(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiSetMode_t           setMode,
  /* [in]  */ DmiMultiRowData_t*     rowData)
```

Database Administration Functions

The following functions modify the schema (data) in the SP's MIF database.

DmiAddComponent()

Adds a new component to the SP's MIF database.

```
DmiErrorStatus_t DMI_API DmiAddComponent(
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiFileDataList_t*    fileData,
  /* [out] */ DmiId_t*              compId,
  /* [out] */ DmiStringList_t**     errors);
```

DmiAddGroup()

Adds a new group to an existing component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiAddGroup("
  /* [in]  */ DmiHandle_t           handle,
  /* [in]  */ DmiFileDataList_t*    fileData,
  /* [in]  */ DmiId_t               compId,
  /* [out] */ DmiId_t               groupId,
  /* [out] */ DmiStringList_t**     errors);
```

DmiAddLanguage()

Adds a new language mapping for an existing component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiAddLanguage(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiFileDataList_t*     fileData,
  /* [in]  */ DmiId_t                compId,
  /* [out] */ DmiStringList_t**      errors);
```

DmiDeleteComponent()

Removes an existing component from the MIF database.

```
DmiErrorStatus_t DMI_API DmiDeleteComponent(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiId_t                compId)
```

DmiDeleteGroup()

Removes a group from a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiDeleteGroup(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiId_t                compId,
  /* [in]  */ DmiId_t                groupId)
```

DmiDeleteLanguage()

Removes a specific language mapping for a component from the MIF database.

```
DmiErrorStatus_t DMI_API DmiDeleteLanguage(
  /* [in]  */ DmiHandle_t            handle,
  /* [in]  */ DmiString_t*           language
  /* [in]  */ DmiId_t                compId)
```

Management Application Provider Functions

This section describes the Management Application Provider Functions that a client must provide to receive indications.

A management interface client receiving indications undergoes a role reversal where, in RPC terms, it becomes an indication delivery server. This DMI SP is a client of this interface.

Indication Functions

All indication functions have some information in common. All indication functions have an opaque handle returned to the application. This handle contains the SubscriberID attribute from the client's row in the SPIndicationSubscription table. Additionally, all indication functions have the sender's address which provides a mechanism for identifying the local or remote system that sent the indication.

DmiComponentAdded()

Notifies the subscribing node that a component has been added to the MIF database.

```
DmiErrorStatus_t DMI_API DmiComponentAdded(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiComponentInfo_t*    info);
```

DmiComponentDeleted()

Notifies the subscribing node that a component has been deleted from the MIF database.

```
DmiErrorStatus_t DMI_API DmiComponentDeleted(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiId_t                compId);
```

DmiDeliverEvent()

Delivers event data to a management application.

```
DmiErrorStatus_t DMI_API DmiDeliverEvent(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiString_t*           language,
  /* [in]  */ DmiId_t                compId
  /* [in]  */ DmiTimestamp_t*        timestamp,
  /* [in]  */ DmiMultiRowData_t*     rowData);
```

DmiGroupAdded()

Notifies the subscribing node that a group has been added to a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiGroupAdded(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiId_t                compId,
  /* [in]  */ DmiGroupInfo_t*        info);
```

DmiGroupDeleted()

Notifies the subscribing node that a group has been deleted from a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiGroupDeleted(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiId_t                compId,
  /* [in]  */ DmiId_t                groupId);
```

DmiLanguageAdded()

Notifies the subscribing node that a language mapping has been added to a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiLanguageAdded(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiId_t                compId,
  /* [in]  */ DmiString_t*           language);
```

DmiLanguageDeleted()

Notifies the subscribing node that a language mapping has been deleted from a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiLanguageDeleted(
  /* [in]  */ handle_t               bind_handle,
  /* [in]  */ DmiUnsigned_t          handle,
  /* [in]  */ DmiNodeAddress_t*      sender,
  /* [in]  */ DmiId_t                compId,
  /* [in]  */ DmiString_t*           language);
```

DmiSubscriptionNotice()

In order to receive indications, a managing node must have subscribed for indications with a managed node using the DmiAddRow() function. See the Events chapter for more information.

```
DmiErrorStatus_t DMI_API DmiSubscriptionNotice(
  /* [in]  */ handle_t               bind_handle,
```

```
/* [in]  */ DmiUnsigned_t          handle,
/* [in]  */ DmiBoolean_t           expired,
/* [in]  */ DmiRowData_t*          rowData);
```

Optional Management Interface Support Functions (Memory Handling, Security and Message Logging)

The following Support Functions provide the client writer with convenient memory allocation and security routines.

Memory Handling and Security Functions

DmiAlloc()

Allocates memory for use as input parameters to DMI function calls, or any other transient use.

```
void *DMI_API DmiAlloc (
  /* [in]  */ size_t               size,
  /* [in]  */ DmiMemDsc_t          memDsc,
  /* [out] */ DmiErrorStatus_t     *status);
```

DmiCopyAttributeData()

Copies the DmiAttributeData_t data structure.

```
DmiErrorStatus_t DmiCopyAttributeData(
   /* [in]  */ DmiAttributeData_t* dest,
   /* [in]  */ DmiAttributeData_t* src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiCopyAttributeIds()

Copies the DmiAttributeIds_t data structure.

```
DmiErrorStatus_t DmiCopyAttributeIds(
   /* [in]  */ DmiAttributeIds_t* dest,
   /* [in]  */ DmiAttributeIds_t* src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiCopyAttributeList()

Copies the DmiAttributeList_t data structure.

```
DmiErrorStatus_t DmiCopyAttributeList(
   /* [in]  */ DmiAttributeList_t* dest,
   /* [in]  */ DmiAttributeList_t* src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiCopyAttributeValues()

Copies the DmiAttributeValues_t data structure.

```
DmiErrorStatus_t DmiCopyAttributeValues(
   /* [in]  */ DmiAttributeValues_t* dest,
```

```
   /* [in]  */ DmiAttributeValues_t* src,
   /* [in]  */ DmiMemDsc_t           memDsc);
```

DmiCopyClassNameInfo()

Copies the DmiClassNameInfo_t data structure.

```
DmiErrorStatus_t DmiCopyClassNameInfo(
   /* [in]  */ DmiClassNameInfo_t* dest,
   /* [in]  */ DmiClassNameInfo_t* src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiCopyClassNameList()

Copies the DmiClassNameList_t data structure.

```
DmiErrorStatus_t DmiCopyClassNameList(
   /* [in]  */ DmiClassNameList_t* dest,
   /* [in]  */ DmiClassNameList_t* src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiCopyComponentInfo()

Copies the DmiComponentInfo_t data structure.

```
DmiErrorStatus_t DmiCopyComponentInfo(
   /* [in]  */ DmiComponentInfo_t* dest,
   /* [in]  */ DmiComponentInfo_t* src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiCopyComponentList()

Copies the DmiComponentList_t data structure.

```
DmiErrorStatus_t DmiCopyComponentList(
   /* [in]  */ DmiComponentList_t* dest,
   /* [in]  */ DmiComponentList_t* src,
   /* [in]  */ DmiMemDsc_t         memDsc)
```

DmiCopyDataUnion()

Copies the DmiCopyDataUnion_t data structure.

```
DmiErrorStatus_t DmiCopyDataUnion(
   /* [in]  */ DmiDataUnion_t* dest,
   /* [in]  */ DmiDataUnion_t* src,
   /* [in]  */ DmiMemDsc_t     memDsc);
```

DmiCopyEnumInfo()

Copies the DmiEnumInfo_t data structure.

```
DmiErrorStatus_t DmiCopyEnumInfo (
   /* [in]  */ DmiEnumInfo_t* dest,
   /* [in]  */ DmiEnumInfo_t* src,
   /* [in]  */ DmiMemDsc_t    memDsc);
```

DmiCopyEnumList()

Copies the DmiEnumList_t data structure.

```
DmiErrorStatus_t DmiCopyEnumList(
   /* [in]  */ DmiEnumList_t* dest,
   /* [in]  */ DmiEnumList_t* src,
   /* [in]  */ DmiMemDsc_t    memDsc);
```

DmiCopyFileDataInfo()

Copies the DmiFileDataInfo_t data structure.

```
DmiErrorStatus_t DmiCopyFileDataInfo(
   /* [in]  */ DmiFileDataInfo_t* dest,
   /* [in]  */ DmiFileDataInfo_t* src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiCopyFileDataList()

Copies the DmiFileDataList_t data structure.

```
DmiErrorStatus_t DmiCopyFileDataList(
   /* [in]  */ DmiFileDataList_t* dest,
   /* [in]  */ DmiFileDataList_t* src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiCopyFileTypeList()

Copies the DmiFileTypeList_t data structure.

```
DmiErrorStatus_t DmiCopyFileTypeList(
   /* [in]  */ DmiFileTypeList_t* dest,
   /* [in]  */ DmiFileTypeList_t* src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiCopyGroupInfo()

Copies the DmiGroupInfo_t data structure.

```
DmiErrorStatus_t DmiCopyGroupInfo(
   /* [in]  */ DmiGroupInfo_t* dest,
   /* [in]  */ DmiGroupInfo_t* src,
   /* [in]  */ DmiMemDsc_t     memDsc);
```

DmiCopyGroupList()

Copies the DmiGroupList_t data structure.

```
DmiErrorStatus_t DmiCopyGroupList(
   /* [in]  */ DmiGroupList_t* dest,
   /* [in]  */ DmiGroupList_t* src,
   /* [in]  */ DmiMemDsc_t     memDsc);
```

DmiCopyMultiRowData()

Copies the DmiMultiRowData_t data structure.

```
DmiErrorStatus_t DmiCopyMultiRowData(
   /* [in]  */ DmiMultiRowData_t* dest,
   /* [in]  */ DmiMultiRowData_t* src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiCopyMultiRowRequest()

Copies the DmiMultiRowRequest_t data structure.

```
DmiErrorStatus_t DmiCopyMultiRowRequest(
   /* [in]  */ DmiMultiRowRequest_t* dest,
   /* [in]  */ DmiMultiRowRequest_t* src,
   /* [in]  */ DmiMemDsc_t           memDsc);
```

DmiCopyNodeAddress()

Copies the DmiNodeAddress_t data structure.

```
DmiErrorStatus_t DmiCopyNodeAddress(
   /* [in]  */ DmiNodeAddress_t* dest,
   /* [in]  */ DmiNodeAddress_t* src,
   /* [in]  */ DmiMemDsc_t       memDsc);
```

DmiCopyOctetString()

Copies the DmiOctetString_t data structure.

```
DmiErrorStatus_t DmiCopyOctetString (
   /* [in]  */ DmiOctetString_t* dest,
   /* [in]  */ DmiOctetString_t* src,
   /* [in]  */ DmiMemDsc_t       memDsc);
```

DmiCopyRowData()

Copies the DmiRowData_t data structure.

```
DmiErrorStatus_t DmiCopyRowData(
   /* [in]  */ DmiRowData_t* dest,
   /* [in]  */ DmiRowData_t* src,
   /* [in]  */ DmiMemDsc_t   memDsc);
```

DmiCopyRowRequest()

Copies the DmiRowRequest_t data structure.

```
DmiErrorStatus_t DmiCopyRowRequest(
   /* [in]  */ DmiRowRequest_t* dest,
   /* [in]  */ DmiRowRequest_t* src,
   /* [in]  */ DmiMemDsc_t      memDsc)
```

DmiCopyString()

Copies the DmiString_t data structure.

```
DmiErrorStatus_t DmiCopyString (
   /* [in]  */ DmiString_t* dest,
   /* [in]  */ DmiString_t* src,
```

```
   /* [in]  */ DmiMemDsc_t  memDsc);
```

DmiCopyStringList()

Copies the DmiStringList_t data structure.

```
DmiErrorStatus_t DmiCopyStringList(
   /* [in]  */ DmiStringList_t* dest,
   /* [in]  */ DmiStringList_t* src,
   /* [in]  */ DmiMemDsc_t      memDsc);
```

DmiCopyTimestamp()

Copies the DmiTimestamp_t data structure.

```
DmiErrorStatus_t DmiCopyTimestamp (
   /* [in]  */ DmiTimestamp_t* dest,
   /* [in]  */ DmiTimestamp_t* src,
   /* [in]  */ DmiMemDsc_t     memDsc);
```

DmiCopyUnicodeString()

Copies the DmiUnicodeString_t data structure.

```
DmiErrorStatus_t DmiCopyUnicodeString (
   /* [in]  */ DmiString_t* dest,
   /* [in]  */ DmiString_t* src,
   /* [in]  */ DmiMemDsc_t  memDsc);
```

DmiDupAttributeData()

Duplicates the DmiAttributeData_t data structure.

```
DmiErrorStatus_t DmiDupAttributeData(
   /* [in]  */ DmiAttributeData_t** dest,
   /* [in]  */ DmiAttributeData_t*  src,
   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupAttributeIds()

Duplicates the DmiAttributeIDs_t data structure.

```
DmiErrorStatus_t DmiDupAttributeIds(
   /* [in]  */ DmiAttributeIds_t** dest,
   /* [in]  */ DmiAttributeIds_t*  src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiDupAttributeInfo()

Duplicates the DmiAttributeInfo_t data structure.

```
DmiErrorStatus_t DmiDupAttributeInfo(
   /* [in]  */ DmiAttributeInfo_t** dest,
   /* [in]  */ DmiAttributeInfo_t*  src,
   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupAttributeList()

Duplicates the DmiAttributeList_t data structure.

```
DmiErrorStatus_t DmiDupAttributeList(
   /* [in]  */ DmiAttributeList_t** dest,
   /* [in]  */ DmiAttributeList_t*  src,
   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupAttributeValues()

Duplicates the DmiAttributeValues_t data structure.

```
DmiErrorStatus_t DmiDupAttributeValues(
   /* [in]  */ DmiAttributeValues_t** dest,
   /* [in]  */ DmiAttributeValues_t*  src,
   /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiDupClassNameInfo()

Duplicates the DmiClassNameInfo_t data structure.

```
DmiErrorStatus_t DmiDupClassNameInfo(
   /* [in]  */ DmiClassNameInfo_t** dest,
   /* [in]  */ DmiClassNameInfo_t*  src,
   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupClassNameList()

Duplicates the DmiClassNameList_t data structure.

```
DmiErrorStatus_t DmiDupClassNameList(

   /* [in]  */ DmiClassNameList_t** dest,

   /* [in]  */ DmiClassNameList_t*  src,

   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupComponentInfo()

Duplicates the DmiComponentInfo_t data structure.

```
DmiErrorStatus_t DmiDupComponentInfo(
   /* [in]  */ DmiComponentInfo_t** dest,
   /* [in]  */ DmiComponentInfo_t*  src,
   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupComponentList()

Duplicates the DmiComponentList_t data structure.

```
DmiErrorStatus_t DmiDupComponentList(
   /* [in]  */ DmiComponentList_t** dest,
   /* [in]  */ DmiComponentList_t*  src,
   /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiDupDataUnion()

Duplicates the DmiDataUnion_t data structure.

```
DmiErrorStatus_t DmiDupDataUnion(
   /* [in]  */ DmiDataUnion_t** dest,
   /* [in]  */ DmiDataUnion_t*  src,
   /* [in]  */ DmiMemDsc_t      memDsc);
```

DmiDupEnumInfo()

Duplicates the DmiEnumInfo_t data structure.

```
DmiErrorStatus_t DmiDupEnumInfo (
   /* [in]  */ DmiEnumInfo_t** dest,
   /* [in]  */ DmiEnumInfo_t*  src,
   /* [in]  */ DmiMemDsc_t     memDsc);
```

DmiDupEnumList()

Duplicates the DmiEnumList_t data structure.

```
DmiErrorStatus_t DmiDupEnumList(
   /* [in]  */ DmiEnumList_t** dest,
   /* [in]  */ DmiEnumList_t*  src,
   /* [in]  */ DmiMemDsc_t     memDsc);
```

DmiDupFileDataInfo()

Duplicates the DmiFileDataInfo_t data structure.

```
DmiErrorStatus_t DmiDupFileDataInfo(
   /* [in]  */ DmiFileDataInfo_t** dest,
   /* [in]  */ DmiFileDataInfo_t*  src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiDupFileDataList()

Duplicates the DmiFileDataList_t data structure.

```
DmiErrorStatus_t DmiDupFileDataList(
   /* [in]  */ DmiFileDataList_t** dest,
   /* [in]  */ DmiFileDataList_t*  src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiDupFileTypeList()

Duplicates the DmiFileTypeList_t data structure.

```
DmiErrorStatus_t DmiDupFileTypeList(
   /* [in]  */ DmiFileTypeList_t** dest,
   /* [in]  */ DmiFileTypeList_t*  src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiDupGroupInfo()

Duplicates the DmiGroupInfo_t data structure.

```
DmiErrorStatus_t DmiDupGroupInfo(
   /* [in]  */ DmiGroupInfo_t** dest,
   /* [in]  */ DmiGroupInfo_t*  src,
   /* [in]  */ DmiMemDsc_t      memDsc);
```

DmiDupGroupList()

Duplicates the DmiGroupList_t data structure.

```
DmiErrorStatus_t DmiDupGroupList(
   /* [in]  */ DmiGroupList_t** dest,
   /* [in]  */ DmiGroupList_t*  src,
   /* [in]  */ DmiMemDsc_t      memDsc);
```

DmiDupMultiRowData()

Duplicates the DmiMultiRowData_t data structure.

```
DmiErrorStatus_t DmiDupMultiRowData(
   /* [in]  */ DmiMultiRowData_t** dest,
   /* [in]  */ DmiMultiRowData_t*  src,
   /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiDupMultiRowRequest()

Duplicates the DmiMultiRowRequest_t data structure.

```
DmiErrorStatus_t DmiDupMultiRowRequest(
   /* [in]  */ DmiMultiRowRequest_t** dest,
   /* [in]  */ DmiMultiRowRequest_t*  src,
   /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiDupNodeAddress()

Duplicates the DmiNodeAddress_t data structure.

```
DmiErrorStatus_t DmiDupNodeAddress(
   /* [in]  */ DmiNodeAddress_t** dest,
   /* [in]  */ DmiNodeAddress_t*  src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiDupOctetString()

Duplicates the DmiOctetString_t data structure.

```
DmiErrorStatus_t DmiDupOctetString (
   /* [in]  */ DmiOctetString_t** dest,
   /* [in]  */ DmiOctetString_t*  src,
   /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiDupRowData()

Duplicates the DmiRowData_t data structure.

```
DmiErrorStatus_t DmiDupRowData(
   /* [in]  */ DmiRowData_t** dest,
   /* [in]  */ DmiRowData_t*  src,
```

```
   /* [in]  */ DmiMemDsc_t    memDsc);
```

DmiDupRowRequest()

Duplicates the DmiRowRequest_t data structure.

```
DmiErrorStatus_t DmiDupRowRequest(
   /* [in]  */ DmiRowRequest_t** dest,
   /* [in]  */ DmiRowRequest_t*  src,
   /* [in]  */ DmiMemDsc_t       memDsc);
```

DmiDupString()

Duplicates the DmiString_t data structure.

```
DmiErrorStatus_t DmiDupString (
   /* [in]  */ DmiString_t** dest,
   /* [in]  */ DmiString_t*  src,
   /* [in]  */ DmiMemDsc_t   memDsc);
```

DmiDupStringList()

Duplicates the DmiStringList_t data structure.

```
DmiErrorStatus_t DmiDupStringList(
   /* [in]  */ DmiStringList_t** dest,
   /* [in]  */ DmiStringList_t*  src,
   /* [in]  */ DmiMemDsc_t       memDsc);
```

DmiDupTimestamp()

Duplicates the DmiTimestamp_t data structure.

```
DmiErrorStatus_t DmiDupTimestamp (
   /* [in]  */ DmiTimestamp_t** dest,
   /* [in]  */ DmiTimestamp_t*  src,
   /* [in]  */ DmiMemDsc_t      memDsc);
```

DmiDupUnicodeString()

Duplicates the DmiUnicodeString_t data structure.

```
DmiErrorStatus_t DmiDupUnicodeString (
   /* [in]  */ DmiString_t** dest,
   /* [in]  */ DmiString_t*  src,
   /* [in]  */ DmiMemDsc_t   memDsc);
```

DmiFree()

Frees previously allocated memory.

```
DmiErrorStatus_t DMI_API DmiFree (
  /* [in]  */ void          *ptr,
  /* [in]  */ DmiMemDsc_t   memDsc);
```

DmiFreeAttributeData()

Frees the DmiAttributeData_t data structure.

```
DmiErrorStatus_t DmiFreeAttributeData (
  /* [in]  */ DmiAttributeData_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeAttributeIds()

Frees the DmiAttributeIds_t data structure.

```
DmiErrorStatus_t DmiFreeAttributeIds (
  /* [in]  */ DmiAttributeIds_t     *ptr,
  /* [in]  */ DmiMemDsc_t           memDsc);
```

DMIFreeAttributeInfo()

Frees the DmiAttributeInfo_t data structure.

```
DmiErrorStatus_t DmiFreeAttributeInfo (
  /* [in]  */ DmiAttributeInfo_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeAttributeList()

Frees the DmiAttributeList_t data structure.

```
DmiErrorStatus_t DmiFreeAttributeList (
  /* [in]  */ DmiAttributeList_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeAttributeValues()

Frees the DmiAttributeValues_t data structure.

```
DmiErrorStatus_t DmiFreeAttributeValues (
  /* [in]  */ DmiAttributeValues_t     *ptr,
  /* [in]  */ DmiMemDsc_t              memDsc);
```

DmiFreeClassNameList()

Frees the DmiClassNameList_t data structure.

```
DmiErrorStatus_t DmiFreeClassNameList (
  /* [in]  */ DmiClassNameList_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeComponentList()

Frees the DmiComponentList_t data structure.

```
DmiErrorStatus_t DmiFreeComponentList (
  /* [in]  */ DmiComponentList_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeClassNameInfo()

Frees the DmiClassNameInfo_t data structure.

```
DmiErrorStatus_t DmiFreeClassNameInfo (
  /* [in]  */ DmiClassNameInfo_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeComponentInfo()

Frees the DmiComponentInfo_t data structure.

```
DmiErrorStatus_t DmiFreeComponentInfo (
  /* [in]  */ DmiComponentInfo_t     *ptr,
  /* [in]  */ DmiMemDsc_t            memDsc);
```

DmiFreeDataUnion()

Frees the DmiDataUnion_t data structure.

```
DmiErrorStatus_t DmiFreeDataUnion (
  /* [in]  */ DmiDataUnion_t     *ptr,
  /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiFreeEnumInfo()

Frees the DmiEnumInfo_t data structure.

```
DmiErrorStatus_t DmiFreeEnumInfo (
  /* [in]  */ DmiEnumInfo_t     *ptr,
  /* [in]  */ DmiMemDsc_t       memDsc)
```

DmiFreeEnumList()

Frees the DmiEnumList_t data structure.

```
DmiErrorStatus_t DmiFreeEnumList (
  /* [in]  */ DmiEnumList_t     *ptr,
  /* [in]  */ DmiMemDsc_t       memDsc);
```

DmiFreeFileDataInfo()

Frees the DmiFileDataInfo_t data structure.

```
DmiErrorStatus_t DmiFreeFileDataInfo (
  /* [in]  */ DmiFileDataInfo_t     *ptr,
  /* [in]  */ DmiMemDsc_t           memDsc);
```

DmiFreeFileDataList()

Frees the DmiFileDataList_t data structure.

```
DmiErrorStatus_t DmiFreeFileDataList (
  /* [in]  */ DmiFileDataList_t     *ptr,
  /* [in]  */ DmiMemDsc_t           memDsc);
```

DmiFreeFileTypeList()

Frees the DmiFileTypeList_t data structure.

```
DmiErrorStatus_t DmiFreeFileTypeList (
  /* [in]  */ DmiFileTypeList_t     *ptr,
  /* [in]  */ DmiMemDsc_t           memDsc);
```

DmiFreeGroupInfo()

Frees the DmiGroupInfo_t data structure.

```
DmiErrorStatus_t DmiFreeGroupInfo (
  /* [in]  */ DmiGroupInfo_t     *ptr,
  /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiFreeGroupList()

Frees the DmiGroupList_t data structure.

```
DmiErrorStatus_t DmiFreeGroupList (
  /* [in]  */ DmiGroupList_t     *ptr,
  /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiFreeMultiRowRequest()

Frees the DmiMultiRowRequest_t data structure.

```
DmiErrorStatus_t DmiFreeMultiRowRequest (
  /* [in]  */ DmiMultiRowRequest_t     *ptr,
  /* [in]  */ DmiMemDsc_t              memDsc);
```

DmiFreeMultiRowData()

Frees the DmiMultiRowData_t data structure.

```
DmiErrorStatus_t DmiFreeMultiRowData (
  /* [in]  */ DmiMultiRowData_t     *ptr,
  /* [in]  */ DmiMemDsc_t           memDsc);
```

DmiFreeNodeAddress()

Frees the DmiNodeAddress_t data structure.

```
DmiErrorStatus_t DmiFreeNodeAddress (
  /* [in]  */ DmiNodeAddress_t     *ptr,
  /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiFreeOctetString()

Frees the DmiOctetString_t data structure.

```
DmiErrorStatus_t DmiFreeOctetString (
  /* [in]  */ DmiOctetString_t     *ptr,
  /* [in]  */ DmiMemDsc_t          memDsc);
```

DmiFreeRowData()

Frees the DmiRowData_t data structure.

```
DmiErrorStatus_t DmiFreeRowData (
  /* [in]  */ DmiRowData_t     *rowData,
  /* [in]  */ DmiMemDsc_t      memDsc);
```

DmiFreeRowRequest()

Frees the DmiRowRequest_t data structure.

```
DmiErrorStatus_t DmiFreeRowRequest (
  /* [in]  */ DmiRowRequest_t     *ptr,
  /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiFreeString()

Frees the DmiString_t data structure.

```
DmiErrorStatus_t DmiFreeString (
  /* [in]  */ DmiString_t     * ptr,
  /* [in]  */ DmiMemDsc_t memDsc);
```

DmiFreeStringList()

Frees the DmiStringList_t data structure.

```
DmiErrorStatus_t DmiFreeStringList (
  /* [in]  */ DmiStringList_t     *ptr,
  /* [in]  */ DmiMemDsc_t         memDsc);
```

DmiFreeTimestamp()

Frees the DmiTimestamp_t data structure.

```
DmiErrorStatus_t DmiFreeTimestamp (
  /* [in]  */ DmiTimestamp_t     *ptr,
  /* [in]  */ DmiMemDsc_t        memDsc);
```

DmiFreeUnicodeString()

Frees the DmiString_t data structure.

```
DmiErrorStatus_t DmiFreeUnicodeString (
  /* [in]  */ DmiString_t     * ptr,
  /* [in]  */ DmiMemDsc_t memDsc);
```

DmiGetSubscriptionAddress()

Retrieves the address of the local network node.

```
DmiErrorStatus_t DMI_API DmiGetSubscriptionAddress(
  /* [in]  */ DmiString_t*  rpcType,
  /* [in]  */ DmiString_t*  transportType,
  /* [out] */ DmiString_t** address);
```

DmiIndicationListen()

Causes the local (management application) indication servers to start listening for indications.

```
DmiErrorStatus_t DMI_API DmiIndicationListen(
  /* [in]  */ void);
```

DmiListRpcTypes()

Lists all RPC types available through the client front end.

```
DmiErrorStatus_t DMI_API DmiListRpcTypes(
       /* [out]   */  DmiStringList_t ** rpcTypes)
```

DmiListTransportTypes()

Lists all available transports for the given RPC type.

```
DmiErrorStatus_t DMI_API DmiListTransportTypes(
  /* [in]  */ DmiString*        rpcTypes)
  /* [out] */ DmiStringList_t** transportTypes);
```

DmiNewAttributeData()

Allocates a new DmiAttributeData_t data structure.

```
DmiAttributeData_t     *DmiNewAttributeData (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t *status);
```

DmiNewAttributeIds()

Allocates a new DmiAttributeIds_t data structure.

```
DmiAttributeIds_t     *DmiNewAttributeIds (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewAttributeInfo()

Allocates a new DmiAttributeInfo_t data structure.

```
DmiAttributeInfo_t     *DmiNewAttributeInfo (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewAttributeList()

Allocates a new DmiAttributeList_t data structure.

```
DmiAttributeList_t     *DmiNewAttributeList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewAttributeValues()

Allocates a new DmiAttributeValues_t data structure.

```
DmiAttributeValues_t FAR *DmiNewAttributeValues (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewClassNameInfo()

Allocates a new DmiClassNameInfo_t data structure.

```
DmiClassNameInfo_t     *DmiNewClassNameInfo (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewClassNameList()

Allocates a new DmiClassNameList_t data structure.

```
DmiClassNameList_t     *DmiNewClassNameList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewComponentInfo()

Allocates a new DmiComponentInfo_t data structure.

```
DmiComponentInfo_t     *DmiNewComponentInfo (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewComponentList()

Allocates a new DmiComponentList_t data structure.

```
DmiComponentList_t     *DmiNewComponentList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewDataUnion()

Allocates a new DmiDataUnion_t data structure.

```
DmiDataUnion_t     *DmiNewDataUnion (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewEnumInfo()

Allocates a new DmiEnumInfo_t data structure.

```
DmiEnumInfo_t     *DmiNewEnumInfo (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewEnumList()

Allocates a new DmiEnumList_t data structure.

```
DmiEnumList_t     *DmiNewEnumList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewFileDataInfo()

Allocates a new DmiFileDataInfo_t data structure.

```
DmiFileDataInfo_t     *DmiNewFileDataInfo (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewFileDataList()

Allocates a new DmiFileDataList_t data structure.

```
DmiFileDataList_t     *DmiNewFileDataList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewFileTypeList

Allocates a new DmiFileTypeList_t data structure.

```
DmiFileTypeList_t     *DmiNewFileTypeList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewGroupInfo()

Allocates a new DmiGroupInfo_t data structure.

```
DmiGroupInfo_t     *DmiNewGroupInfo (
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewGroupList()

Allocates a new DmiGroupList_t data structure.

```
DmiGroupList_t     *DmiNewGroupList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewMultiRowData()

Allocates a new DmiMultiRowData_t data structure.

```
DmiMultiRowData_t     *DmiNewMultiRowData (
  /* [in]  */ size_t               size,
```

```
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewMultiRowRequest()

Allocates a new DmiMultiRowRequest_t data structure.

```
DmiMultiRowRequest_t     *DmiNewMultiRowRequest (
  /* [in]  */  size_t              size,
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewNodeAddress()

Allocates a new DmiNodeAddress_t data structure.

```
DmiNodeAddress_t     *DmiNewNodeAddress (
  /* [in]  */  const char     *     pAddress,
  /* [in]  */  const char     *     pRpc,
  /* [in]  */  const char     *     pTransport,
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewOctetString()

Allocates a new DmiOctetString_t data structure.

```
DmiOctetString_t     *DmiNewOctetString (
  /* [in]  */  size_t              size,
  /* [in]  */  const char          *src,
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewRowData()

Allocates a new DmiRowData_t data structure.

```
DmiRowData_t     *DmiNewRowData (
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewRowRequest()

Allocates a new DmiRowRequest_t data structure.

```
DmiRowRequest_t     *DmiNewRowRequest (
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewString()

Allocates a new DmiString_t data structure.

```
DmiString_t     *DmiNewString (
  /* [in]  */  const char          *src,
  /* [in]  */  DmiMemDsc_t         memDsc,
  /* [out] */  DmiErrorStatus_t    *status);
```

DmiNewStringList()

Allocates a new DmiStringList_t data structure.

```
DmiStringList_t     *DmiNewStringList (
  /* [in]  */  size_t               size,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewTimestamp()

Allocates a new DmiTimestamp_t data structure.

```
DmiTimestamp_t     *DmiNewTimestamp (
  /* [in]  */  DmiMemDsc_t memDsc,
  /* [out] */  DmiErrorStatus_t     *status);
```

DmiNewUnicodeString()

Allocates a new DmiUnicodeString_t data structure.

```
DmiString_t     *DmiNewUnicodeString (
  /* [in]  */  const char           *src,
  /* [in]  */  DmiMemDsc_t          memDsc,
  /* [out] */  DmiErrorstatus_t     *status);
```

DmiSetDefaultNode()

Allows management application to get and set the default SP node.

```
DmiErrorStatus_t DMI_API DmiSetDefaultNode(
  /* [in]  */ DmiNodeAddress_t *        node,
  /* [out] */ DmiNodeAddress_t **       oldNode);
```

DmiSetIndicationCallbacks()

Exchanges information on indication callback entry points between the application and client front-end.

```
DmiErrorStatus_t DMI_API DmiSetIndicationCallbacks(
  /* [in]  */ DmiIndicationFuncs_t* indicationFuncs,
  /* [out] */ DmiIndicationFuncs_t* oldIndicationFuncs);
```

DmiStopIndicationListening()

Causes local (management application) indication servers in all available RPCs to stop listening for indications.

```
DmiErrorStatus_t DMI_API DmiStopIndicationListening( void )
```

generateToken()

Generates a security token that will be used by the component instrumentation to validate access to the protected attribute.

```
DmiErrorStatus_t DMI_API generateToken(
  /* [in]  */ DmiString_t  *        componentName,
  /* [in]  */ DmiString_t  *        groupName,
  /* [out] */ DmiString_t  **       securityToken);
```

dmiLogMessage()

Writes a message to the specified logfile. The message's parameters can be passed after `msgNumber`.

```
DmiLogMessage(
    /* [in] */ char *    msgLogName,
    /* [in] */ char *    msgCatalogName,
    /* [in] */ char *    msgText,
    /* [in] */ char *    msgLevel,
    /* [in] */ char *    msgCallerTag,
    /* [in] */ int       msgSetNumber,
    /* [in] */ int       msgNumber,
                                ...);
```

10 Events and Indications

About DMI 2.0 Events and Indications

DMI generates events and indications which can be used by DMI management application developers to monitor computer systems. The event indications make up the management application provider API. These events are listed below.

Event Indications

DmiComponentAdded()

Generated by SP when a new component is added to the MIF database.

```
DmiErrorStatus_t DMI_API DmiComponentAdded(
  /* [in] */  handle_t              bind_handle,
  /* [in] */  DmiUnsigned_t         handle,
  /* [in] */  DmiNodeAddress_t*     sender,
  /* [in] */  DmiComponentInfo_t*   info);
```

DmiComponentDeleted()

Generated by SP when a component is deleted from the MIF database.

```
DmiErrorStatus_t DMI_API DmiComponentDeleted(
  /* [in] */  handle_t              bind_handle,
  /* [in] */  DmiUnsigned_t         handle,
  /* [in] */  DmiNodeAddress_t*     sender,
  /* [in] */  DmiId_t               compId);
```

DmiDeliverEvent()

Initiated by instrumentation when a previously agreed upon condition is detected within the monitored component. This event is passed from the instrumentation to SP. SP then generates this event.

```
DmiErrorStatus_t DMI_API DmiDeliverEvent(
  /* [in] */  handle_t              bind_handle,
  /* [in] */  DmiUnsigned_t         handle,
  /* [in] */  DmiNodeAddress_t*     sender,
  /* [in] */  DmiString_t*          language,
  /* [in] */  DmiId_t               compId
  /* [in] */  DmiTimestamp_t*       timestamp,
  /* [in] */  DmiMultiRowData_t*    rowData);
```

DmiGroupAdded()

Generated by SP when a group is added to a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiGroupAdded(
  /* [in] */  handle_t               bind_handle,
  /* [in] */  DmiUnsigned_t          handle,
  /* [in] */  DmiNodeAddress_t*      sender,
  /* [in] */  DmiId_t                compId,
  /* [in] */  DmiGroupInfo_t*        info);
```

DmiGroupDeleted()

Generated by SP when a group is deleted from a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiGroupDeleted(
  /* [in] */  handle_t               bind_handle,
  /* [in] */  DmiUnsigned_t          handle,
  /* [in] */  DmiNodeAddress_t*      sender,
  /* [in] */  DmiId_t                compId,
  /* [in] */  DmiId_t                groupId);
```

DmiLanguageAdded()

Generated by SP when a new language is added to component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiLanguageAdded(
  /* [in] */  handle_t               bind_handle,
  /* [in] */  DmiUnsigned_t          handle,
  /* [in] */  DmiNodeAddress_t*      sender,
  /* [in] */  DmiId_t                compId,
  /* [in] */  DmiString_t*           language);
```

DmiLanguageDeleted()

Generated by SP when a language is deleted from a component in the MIF database.

```
DmiErrorStatus_t DMI_API DmiLanguageDeleted(
  /* [in] */  handle_t               bind_handle,
  /* [in] */  DmiUnsigned_t          handle,
  /* [in] */  DmiNodeAddress_t*      sender,
  /* [in] */  DmiId_t                compId,
  /* [in] */  DmiString_t*           language);
```

DmiSubscriptionNotice()

Generated by SP when the dates for warning and/or expiring the management application's entries in the SP indication subscription and SP filter information tables expire.

```
DmiErrorStatus_t DMI_API DmiSubscriptionNotice(
  /* [in] */  handle_t              bind_handle,
  /* [in] */  DmiUnsigned_t         handle,
  /* [in] */  DmiBoolean_t          expired,
  /* [in] */  DmiRowData_t*         rowData);
```

Monitoring Events

In order for a management application to monitor these events it must:

1. Register with the SP of the host it will monitor.
2. Create a DCE/RPC end point for these indication RPCs.
3. Add a row to SP's indication subscription table giving SP the address of the created DCE/RPC end point, and add a row to SP's filter information table indicating which component ID and which class name it wishes to monitor (wild cards can be used to monitor all components and classes).
4. Register procedural entry points for the monitored indications.

Registering With a Host's SP

The management application registers with a host's SP by using the DmiRemoteRegister() call as described in the Management Interface section.

Creating a DCE/RPC Endpoint

The DCE/RPC end point is created so that SP can issue a DCE/RPC call to the management application. The end point is set up by calling DmiIndicationListen(). This procedure has no parameters.

Adding Rows to the SP Tables

When SP generates indications, it needs the following information:

The addressing information of management applications that are registered to receive indications.

The components a registered management application is monitoring.

The group classes a registered management application is monitoring.

There are a certain number of addressing entries in the SP Indication Subscription table. There are a certain number of component and group entries in the SP Filter Information table. There must be one entry in the SP Indication Subscription table for each management application that is monitoring events. There must be at least one corresponding SP Filter Information table entry and there can be more than one filter. To determine which subscription entries belong to which filter entries, you look at the first four attributes. You add entries into these tables using DmiAddRow().

Wild cards can be used in SP Filter Information table entries to describe components and group classes being monitored.

For example:

- A component ID of -1 matches all components in the MIF Database.
- A group class of "||" matches all group class names in the MIF database.
- A group class of "HP||" matches all HP defined group class names.
- A group class of "|Host Disk Storage Table|" matches all Host Disk Storage Table group classes.

Matching the first four attributes in both the SP Indication Subscription table and the SP Filter Information table define corresponding entries.

In order to clarify what values would be entered in these tables, the attributes are presented below with potential valid entries for the HP-UX implementation of DMI 2.0.

The SP Indication Subscription table holds the following information:

Table 10-1 SP Indication Subscription Table Content

Subscriber RPC Type	"dce" or "local"
Subscriber Transport Type	"ncacn_ip_tcp" or "ncalrpc"
Subscriber Addressing	Call DmiGetSubscriptionAddress() to get a valid subscriber address
Subscriber ID	List the current SP Indication Table entries and use an integer that is not currently being used
Subscription Expiration Warning Date Stamp	Date & Time when the application wants to be warned of pending SP Indication Table entry expiration
Subscription Expiration Date Stamp	Date & Time when the SP Indication Subscription entry expires
Indication Failure Threshold	Number of indication transmission failures to tolerate before this entry and corresponding entries from the SP Filter Information table are deleted

The SP Filter Information table holds the following information:

Table 10-2 SP Filter Information Table Content

Subscriber RPC Type	"dce" or "local"
Subscriber Transport Type	"ncacn_ip_tcp" or "ncalrpc"
Subscriber Addressing	Call DmiGetSubscriptionAddress() to get a valid subscriber address
Subscriber ID	List the current SP Indication Table entries and use an integer that is not currently being used
Component ID	The component ID number of the monitored component (-1 for all, another valid component ID integer for specific monitored components)

Table 10-2 SP Filter Information Table Content

Group Class String	The group class string of the monitored group ("||" for all, "DMTF||" for all DMTF defined groups etc...)
Event Severity	The severity level being monitored

Procedural Entry Points

A management application passes the following data structure:

```
typedef struct DmiIndicationFuncs {
   DmiDeliverEventFunc          DeliverEventFunc;
   DmiComponentAddedFunc        componentAddedFunc;
   DmiComponentDeletedFunc      componentDeletedFunc;
   DmiLanguageAddedFunc         languageAddedFunc;
   DmiLanguageDeletedFunc       languageDeletedFunc;
   DmiGroupAddedFunc            groupAddedFunc;
   DmiGroupDeletedFunc          groupDeletedFunc;
   DmiSubscriptionNoticeFunc    subscriptionNoticeFunc;
} DmiIndicationFuncs_t;
```

Each function pointer represents a callback to handle indications for the management application. To receive the event, the management application assigns the corresponding pointer to a valid procedure. If the management application does not want the event, it assigns the corresponding pointer to NULL. Once the structure is filled out as desired, call DmiSetIndicationCallbacks() with the new DmiIndicationFuncs_t structure and with a pointer to a structure that will receive the old DmiIndicationFuncs_t data structure, or NULL if the old callbacks are not needed.

11 Error Handling and Messaging

The following table contains DMI non-error messages.

Table 11-1 DMI Non-Error Messages

SYMBOL	VALUE	DESCRIPTION
DMI_NO_ERROR	0x00000	Success
DMI_NO_ERROR_MORE_ DATA	0x00001	More data is available
DMI_DEFAULT_LANGUAGE-RETURNED	0x00002	The item requested did not have a language mapping installed that matched the one requested. The value was returned using the default language

The following table contains DMI error messages.

Table 11-2 DMI Error Messages

SYMBOL	VALUE	DESCRIPTION
DMIERR_OVERLAY_NOT_ FOUND	0x00001	Unable to retrieve the overlay path name or statement from the database
DMIERR_ATTRIBUTE_NOT_ FOUND	0x00100	Attribute not found
DMIERR_VALUE_EXCEEDS_ MAXSIZE	0x00101	Value exceeds maximum size
DMIERR_COMPONENT_NOT_ FOUND	0x00102	Component ID is not found
DMIERR_ENUM_ERROR	0x00103	Enumeration error
DMIERR_GROUP_NOT_ FOUND	0x00104	Group not found
DMIERR_ILLEGAL_KEYS	0x00105	Illegal keys
DMIERR_ILLEGAL_TO_SET	0x00106	Illegal to set
DMIERR_OVERLAY_NAME_ NOT_FOUND	0x00107	Component instrumentation not found
DMIERR_ILLEGAL_TO_GET	0x00108	Illegal to get
DMIERR_ROW_NOT_FOUND	0x0010a	Row not found
DMIERR_DIRECT_INTERFACE_ NOT_REGISTERED	0x0010b	Direct interface not registered
DMIERR_DATABASE_ CORRUPT	0x0010c	MIF database is corrupt

Table 11-2 DMI Error Messages

SYMBOL	VALUE	DESCRIPTION
DMIERR_ATTRIBUTE_NOT_SUPPORTED	0x0010d	Attribute is not supported
DMIERR_VALUE_UNKNOWN	0x0010f	Value for this attribute is not known
DMIERR_BUFFER_FULL	0x00200	Buffer full
DMIERR_ILL_FORMED_COMMAND	0x00201	Ill-formed command (NULL parameter(s))
DMIERR_ILLEGAL_COMMAND	0x00202	Illegal command
DMIERR_ILLEGAL_HANDLE	0x00203	Illegal handle
DMIERR_OUT_OF_MEMORY	0x00204	Out of memory
DMIERR_NULL_COMPLETION_FUNCTION	0x00205	No confirm function
DMIERR_NULL_RESPONSE_BUFFER	0x00206	No response buffer
DMIERR_CMD_HANDLE_IN_USE	0x00207	Command handle is already in use
DMIERR_ILLEGAL_DMI_LEVEL	0x00208	DMI version mismatch
DMIERR_UNKNOWN_CI_REGISTRY	0x00209	Unknown Ci registry
DMIERR_COMMAND_CANCELLED	0x0020a	Command has been cancelled
DMIERR_INSUFFICIENT_PRIVILEGES	0x0020b	Insufficient privileges
DMIERR_NULL_ACCESS_FUNCTION	0x0020c	No access function provided
DMIERR_FILE_ERROR	0x0020d	OS file I/O error
DMIERR_EXEC_FAILURE	0x0020e	Could not spawn a new task
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE	0x0020f	Ill-formed SCHEMA
DMIERR_INVALID_FILE_TYPE	0x00210	Invalid file type
DMIERR_SP_INACTIVE	0x00211	Service provider is inactive
DMIERR_CANT_UNINSTALL_SP_COMPONENT	0x00213	Unable to remove the service provider component
DMIERR_NULL_CANCEL_FUNCTION	0x00214	No cancel function provided
DMIERR_INVALID_POOL	0x00215	Memory pool handle is invalid
DMIERR_INVALID_PTR	0x00216	Memory Ptr passes are invalid
DMIERR_NO_POOL	0x00217	Memory pool is required for use with this function
DMIERR_FILE_TYPE_NOT_SUPPORTED	0x00218	The passed file type is legal, but not supported by this implementation

Table 11-2 DMI Error Messages

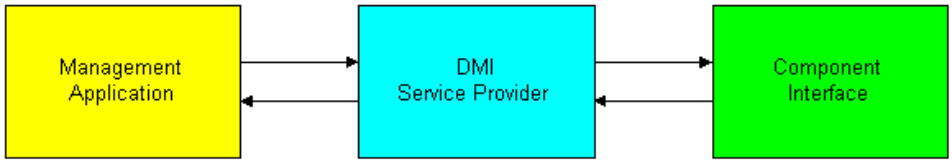
SYMBOL	VALUE	DESCRIPTION
DMIERR_CANT_UNINSTALL_ COMPONENT_LANGUAGE	0x00219	Unable to install a component's language mapping
DMIERR_CANT_UNINSTALL_ GROUP	0x0021a	Unable to install the group
DMIERR_UNABLE_TO_ADD_ ROW	0x0021b	The add row failed due to either a database problem or a component limitation
DMIERR_UNABLE_TO_DELETE_ ROW	0x0021c	The delete row failed to to either a database problem or a component limitation
DMIERR_RPC_COMM_FAILURE	0x00500	Failure to start or join a thread or bind to a server
DMIERR_RPC_TRANSPORT_ UNAVAILABLE	0x00502	Requested transport type is unavailable
DMIERR_RPC_SERVER_ UNAVAILABLE	0x00503	Server required to perform the requested operation is unavailable or not reponding
DMIERR_RPC_GENERAL_ FAILURE	0x00506	General RPC failure
DMIERR_CFE_RPC_NOT_ SUPPORTED	0x00803	Requested RPC type os not supported by the DCE runtime library

A An HP-UX DMI 2.0 Example Code

Hewlett-Packard has provided a set of Component and management application example files which developers can compile and run with the HP-UX DMI 2.0 SDK.

The example includes a management application which subscribes with the DMI SP to receive indications when new components are added. The management application registers with the DMI SP, sets up an indication server and adds entries to the required groups. The example component installs itself into the SP's MIF database. The management application then receives notification of the component's ID, system memory size, processors, and how long each processor is in sleep, run, zombie, or idle state.

Figure A-1 High Level Block Diagram

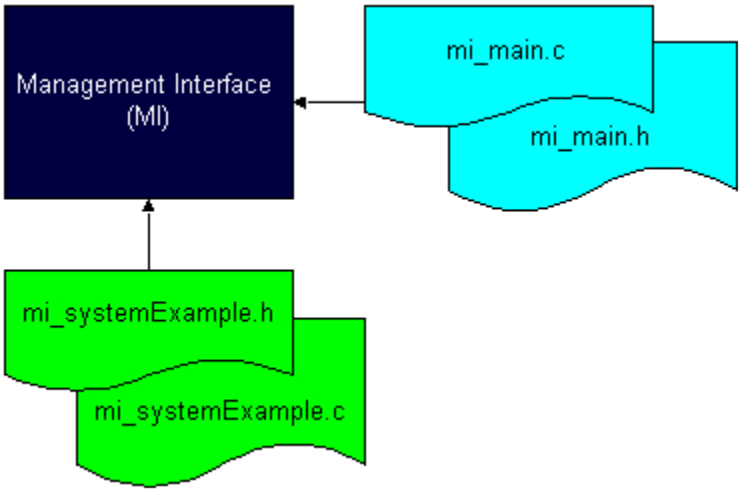


The Example Management Application

DMI-enabled management applications are different from components in that they are each unique to the purpose they serve. There is no recipe or formula for building a DMI-enabled management application. The management application used in the System Example, however, at least illustrates how various calls and indication subscriptions can be used.

The example management application includes a set of four files. The mi_main.c file remote registers the management application with the DMI SP. It then makes a call to set up an indication server from mi_systemExample.c file. Finally, it makes calls to list installed components and loop, again from mi_systemExample.c. These two files each have an associated header file; mi_main.h and mi_systemExample.h respectively.

Figure A-2 Example Management Block Diagram



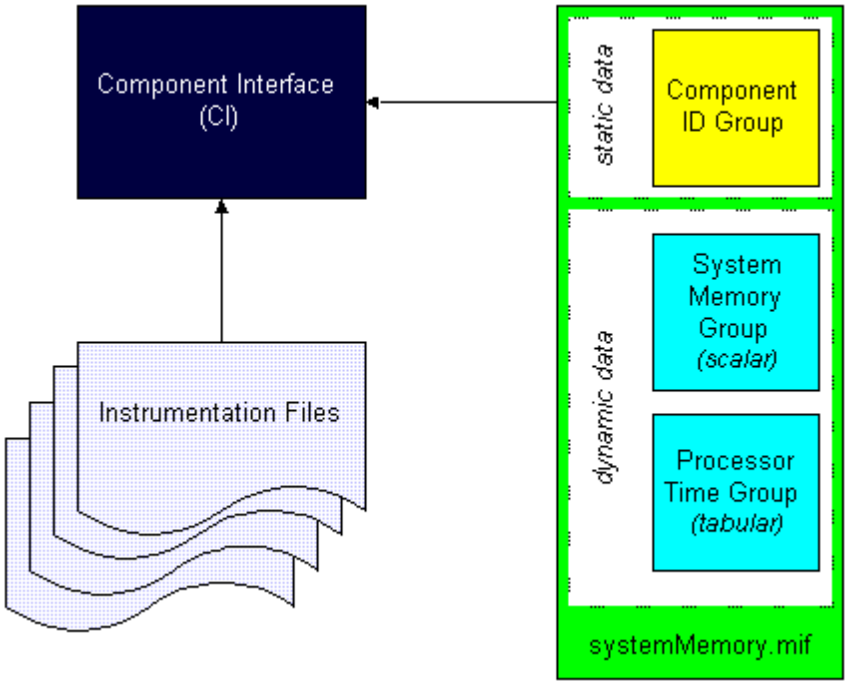
The Example Component

The component used in HP's System Example, like all components, is created from two types of elements in the Component Interface: the component's MIF file and the various instrumentation files.

The systemExample.mif, like all components, contains the Component ID Group. This static group contains the six attributes required to identify a component. They are Manufacturer, Product, Version, Serial Number, Installation, and Verify.

The example component MIF also contains two dynamic data groups: the scalar System Group and the tabular Processor Time Group. The System Group contains the attributes Total Physical Memory and Total Processors. The Processor Time Group contains the attributes Processor ID, Sleep Time, Run Time, Zombie Time, and Idle Time. The attributes within these two groups are accessed dynamically and require component instrumentation.

Figure A-3 Example Component



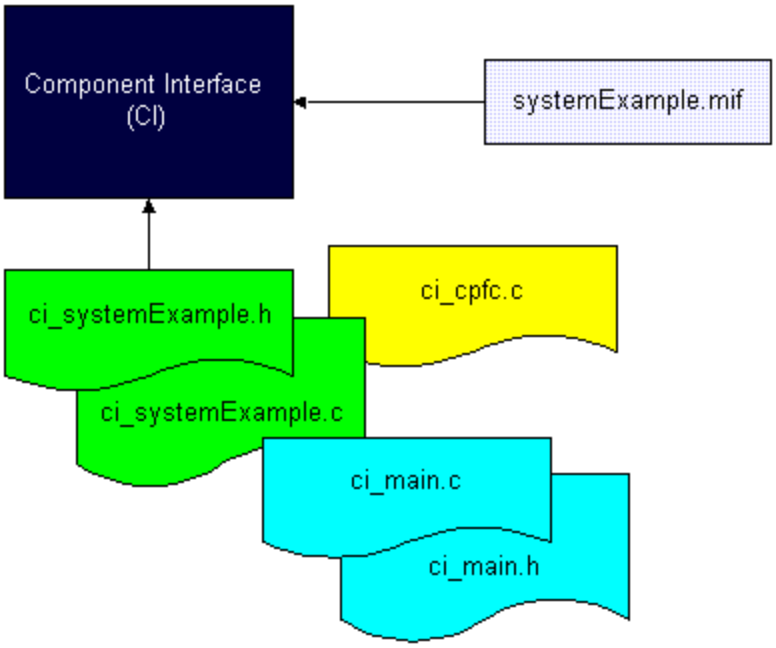
Component instrumentation for the dynamic data in this example is handled in the ci_systemExample.h file. Instrumentation provided by the developer is identified as following between headers and footers like those illustrated below. The corresponding ci_systemExample.c file does not need to be modified.

```
/*================================================
  START OF DEVELOPER DEFINED CODE
  ================================================ */


 /* ================================================
  END OF DEVELOPER DEFINED CODE.
  ================================================ */
```

Component functions are defined in the ci_cpfc.c file. For our System Example, we have also defined a set of helper functions in a file called ci_systemExample.c. This example makes calls to the ci_systemExample.c file from the ci_cpfc.c file. Again, these developer provided changes are identified between the “developer defined” headers and footers.

Figure A-4 Component Structure



The ci_main.c and ci_main.h files are used to install the component into the SP's MIF database and do not need to be modified by the developer.

Compiling the Example

To compile the example, you must have the following software installed on your system:

HP C/ANSI C Developer's Bundle for HP-UX 10.20 or the aC++ Compiler S800;

DCE-BPRG fileset from the DCE Programming Environment and Libraries bundle.

To compile the management application example, execute the make in the examples/mi directory of the SDK. To compile the component example, execute the make in the examples/ci/systemExample directory of the SDK.

Running the Example

To run the example, do the following:

1. Invoke the management application to begin monitoring and receiving indications. In the examples/mi directory, execute:

   mi_memoryExample

2. Invoke the Component Interface to add a new component. In the examples/ci/systemExample directory execute:

   ci_memoryExample /var/dmi/mif/C

The management application should display information similar to the following via std out:

```
Monitoring incoming DmiComponentAdded()events
      Component id: 3
      Processor: 1 SLEEP: 0  RUN: 0  ZOMBIE: 0  IDLE: 0
      Processor: 1 SLEEP: 0  RUN: 0  ZOMBIE: 0  IDLE: 0
      Processor: 1 SLEEP: 0  RUN: 0  ZOMBIE: 0  IDLE: 0
      DmiComponentAdded Event:
      Component id: 3
      Name: HP Component Instrumentation Example MIF
```

You can verify the component by viewing its groups, attributes and attribute values in the MIF browser or by executing QueryDB.

B Component Interface Skeleton

The HP-UX DMI 2.0 SDK includes a set of skeleton files that can be used to develop DMI Components. Those files and their related directories are as follows:

Directory

/usr/dmi/examples/ci/skeleton

This directory contains the source, headers and Makefile.

Files

/var/dmi/mif/C/ci_skeleton.mif

This is a skeleton MIF file. It has the ComponentID group and some boilerplate information for groups and attributes.

/usr/dmi/examples/ci/skeletonCode/Makefile

This Makefile builds the source files into an executable named ci_YourCi. It has no function because it is only a skeleton. The executable links against the correct libraries and includes the correct header files. You may want to rename the files for your component implementation.

/usr/dmi/examples/ci/skeletonCode/ci_skeletonMain.h

This is the header for the main routine file.

/usr/dmi/examples/ci/skeletonCode/ci_skeletonCallbacks.h

This header contains prototypes for the developer-defined procedures.

/usr/dmi/examples/ci/skeletonCode/ci_skeletonCallbacks.c

This file contains the procedures that are called by the code in ci_skeletonSpecific.c.

/usr/dmi/examples/ci/skeletonCode/ci_skeletonSpecific.c

This file contains the procedures which are called by the API library on behalf of the Service Provider in response to a management application's requests. The functions in this file are described in Section 8 of the DMTF Desktop Management Interface Specification Version 2.0.

These functions then call whatever developer-defined functions are required to perform the request from the Service Provider.

/usr/dmi/examples/ci/skeletonCode/ci_skeletonMain.c

This is the main routine file. It sets up the signal handler, starts a session with the SP, loads the MIF and gets a componentID for the MIF, ends its session, and registers the CI.

C HP-UX Systems MIF Groups Quick Reference

This appendix provides a graphical view of the groups in the HP-UX Software MIF. It is a quick reference view of the attributes in each group and their data types. For easy reference, HP-UX System MIF groups are presented in four divisions:

Component Information Groups

Component ID
Manufacturer String (64)
Product String (64)
Version String (64)
Serial Number String (64)
Installation Date
Verify Enum (Verify_Type)

General Information
System Name String (64)
System Location String (64)
System Primary User Name String 64)
System Primary User Phone String (64)
System Boot Up Time (Date)
System Date Time (Date)
System Secondary User Name String(64)
System Secondary User Phone String(64)
System Primary Pager String(64)
System Secondary Pager String(64)
Security Token OctetString(64) **[Key]**
System Model String(64)
System Serial Number String(64)
System Software Identifier String(64)

System Information Groups

Operating System
Operating System Index Integer
Operating System Name String (64)
Operating System Version String (64)
Primary Operating System (BOOL)
Operating System Description String (128)
Operating System Capability String (64)

Host Physical Memory
Physical Main Memory Size Integer

Host Logical Memory
Logical Memory Index Integer **[Key]**
Swap Space Name DisplayString (256)
Swap Type Enum
Swap Priority Integer
Swap Space Size Int(64)
Swap Space Minimum Size Int(64)
Swap Space Maximum Size Int(64)
Swap Space Reserved Size Int(64)
File System Index Integer
Storage Index Integer
Logical Index Integer

Event Generation
Event Type Enum
Event Severity Enum
Is Event State-Based? BOOL
Event State [Key] Integer
Associated Group String (64) **[Key]**
Event System String (64) *
Event Subsystem String (64) *
Event Solution Enum
Instance Data Present BOOL

Host Processor
Host Processor Index Integer **[Key]**
Processor Firmware ID String (64)
Processor Load Integer
Processor Allocated BOOL

Host System
System Uptime Int(64)
Initial Load Device Integer
Initial Load Parameters String (128)
Number of Users (Gauge)
Number of Processes (Gauge)
Max Processes Integer
Mounted File Systems Gauge
Security Token OctetString (64) **[Key]**
Architecture String(64)
Language String(64)
Timezone String(64)
Hardware Capability String(64)
[Key]board String(64)
Graphics Resolution String(64)
Graphics Planes Integer
CPU Speed Integer

Host Storage
Host Storage Index Integer **[Key]**
Storage Type Enum
Description DisplayString (256)
Allocation Units Size Int (64)
Total Allocation Units Int (64)
Allocation Units Used Int (64)
Storage Allocation Failures Counter

Host Device
Host Device Index Integer **[Key]**
Device Type Enum
Device Description String (64)
Device ID String (64)
Device Status Enum
Device Errors Counter
Hardware Path String(64)
Hardware Type String(64)
Device Class String(64)
Associated Driver String(64)

Host Disk Storage
Host Disk Index Integer **[Key]**
Disk Storage Access Enum
Disk Storage Media Enum
Disk Storage Removable Integer
Disk Storage Capacity Integer

* Type determined by instrumentation

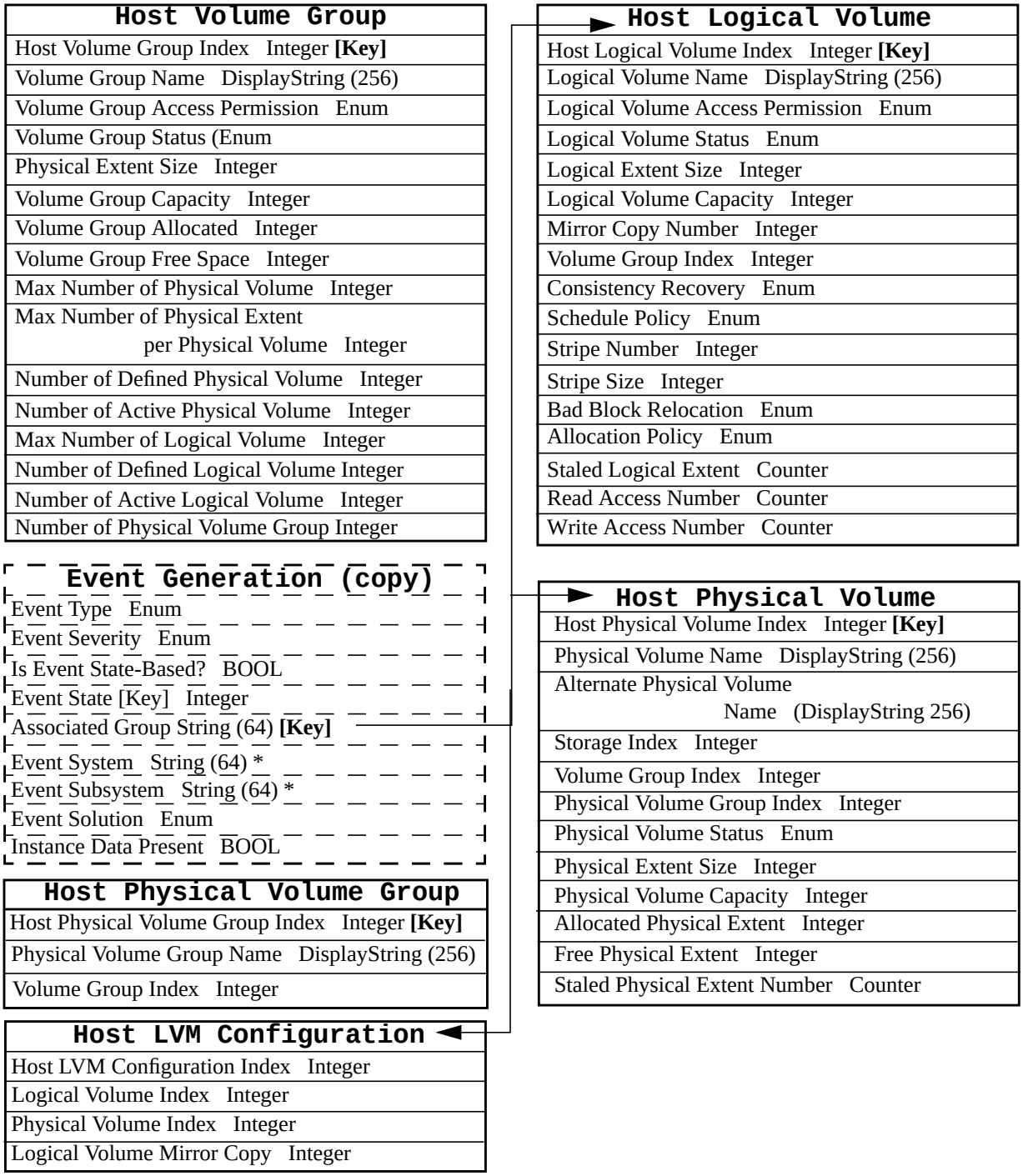
Host File System
Host File System Index Integer **[Key]**
Mount Point Name DisplayString (256)
Mounted Special Device Name DisplayString (256)
Remote Mount Point Name DisplayString (256)
File System Type String (128)
File System Access Enum
File System Bootable Enum
Storage Index Integer
Last Full Backup Date
Last Partial Backup Date
Logical Index Integer
Total INodes Int(64)
Free INodes Int (64)
Data Capacity Int(64)
Free Capacity Int(64)
Reserved Data Capacity Int(64)

Host Disk Storage (Copy)
Host Disk Index Integer **[Key]**
Disk Storage Access Enum
Disk Storage Media Enum
Disk Storage Removable Integer
Disk Storage Capacity Integer

Host HFS Tuning Parameters
Host HFS Tuning Parameters Index Integer **[Key]**
Host File System Index Integer
Long File Name Flag Enum
Large File Feature Enum
Minifree Integer
Block Size Integer
Fragment Size Integer
Bytes per Inode Integer
Sectors per Track Integer
Tracks per Cylinder Integer
Disk Cylinders per Cylinder Group Integer
Disk Revolutions per Second Integer
Rotational Delay Integer

Process Information
Process ID String (6) **[Key]**
Parent Process ID String (6)
Process Group ID String (6)
Real User ID Int (64)
Process TTY DisplayString (32)
Process Name DisplayString (256)
Module Path DisplayString (2048)
Parameters DisplayString (1024)
Process State Enum
Process Priority Int (64)
Process Nice Value Integer
Process CPU Time Integer
Process System Time Int (64)
Process User Time Int (64)
Process Real Total Int (64)
Process Real Text Int (64)
Process Real Data Int (64)
Process Real Stack Int (64)
Process Virtual Text Int (64)
Process Virtual Data Int (64)
Process Virtual Stack Int (64)
Process Virtual Shared Memory Int (64)
Process Virtual Memory Mapped File Size Int (64)

Logical Volume Manager Groups



Network Configuration Groups

Network Interface
Network Interface Index Integer **[Key]**
Interface Name DisplayString (8)
IP Address DisplayString (64)
Subnet Mask DisplayString (16)
Broadcast Address DisplayString (16)
Interface State Enum
DHCP Enabled Enum
Station Address DisplayString (16)
Interface Card Hardware Path DisplayString (256)
Hostname DisplayString (256)

NIS Configuration
Domain Name DisplayString (256)
Master Server Enum
Slave Server Enum
Server Wait Flag Enum

System Contact Information
Contact Index Integer **[Key]**
Contact Name String (64)
Contact Type Enum
Contact Information String (512)
Security Token OctetString (64) **[Key]**

Kernel Configure
Kernel Configure Index Integer **[Key]**
Parameter Name DisplayString(256)
Parameter Value DisplayString(256)

Host Device
Host Device File Index Integer **[Key]**
Device File Type Enum
Device File Name DisplayString(1024)
Host Device Group Index Integer

Static Routing Definition
Static Routing Definition Index Integer **[Key]**
Route Destination DisplayString (64)
Route Mask DisplayString (64)
Route Gateway DisplayString (16)
Route Count Enum
Route Argument DisplayString (1024)

DNS Configuration
Domain Name DisplayString (256)
Search DisplayString (256)
Server IP Address(es) DisplayString (256)

NTP Configuration
Server Address DisplayString (256)

D HP-UX Software MIF Groups Quick Reference

This appendix provides a graphical view of the groups in the HP-UX Software MIF. It is a quick reference view of the attributes in each group and their data types. For easy reference, HP-UX Software MIF groups are presented in five divisions:

General Groups

ComponentID
Manufacturer String (64)
Product String (64)
Version String (64)
Serial Number String (64)
Installation Date
Verify Enum

Software Location
Path String (256)
Catalog String (256)
Dfiles String (64)
Layout Version String (64)
Pfiles String (64)

Vendors
Tag String (64) **[Key]**
Index Integer **[Key]**
Title String (256)
Description String (8096)

Bundle Groups

Bundle Contents
Bundle Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Content String (1024)

Bundles
Bundle Software Specification String (1024) **[Key]**
Tag String (64)
Architecture String (64)
Location String (1024)
Qualifier String (64)
Revision String (64)
Vendor Tag String (64)
Create Time String (16)
Description String (8096)
Modification Time String (16)
Size String (32)
Title String (256)
Copyright String (8096)
Directory String (256)
Instance Identifier String (16)
Is Locatable String (8)
Layout Version String (64)
Machine Type String (64)
Number String(64)
Operating System Name(64)
Operating System Release String (64)
Operating System Version String (64)
Is Patch String(8)
Install Source String(128)
Data Model Revision String(8)
Install Date String(16)

Product and Subproduct Groups

Products
Product Software Specification String (1024) **[Key]**
Tag String (64)
Architecture String (64)
Location String (1024)
Qualifier String (64)
Revision String (64)
Vendor Tag String (64)
Create Time String (16)
Description String (8096)
Modification Time String (16)
Size String (32)
Title String (256)
All Filesets String (8096)
Control Directory String (256)
Copyright String (8096)
Directory String (256)
Instance Identifier String (16)
Is Locatable String (8)
Post Kernel Path String (256)
Layout Version String (64)
Machine Type String (64)
Number String (64)
Operating System Name String (64)
Operating System Release String(64)
Operating System Version String (64)
Is Patch String(8)
Install Source String(128)
Data Model Revision String(8)
Install Date String(16)

Product Contents
Product Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Content String (1024)
Content Type Enum

Product Control Files
Product Software Specification String (1024) **[Key]**
Tag String (64) **[Key]**
Cksum String (16)
Compressed Cksum String (16)
Compressed Size String (16)
Compression State String (16)
Compression Type String (64)
Revision String (64)
Size String (16)
Source String (256)
Interpreter String (256)
Path String (256)
Result String (16))

Subproducts
Subproduct Software Specification String (1024) **[Key]**
Tag String (64)
Create Time String (16)
Description String (8096)
Modification Time String (16)
Size String (32)
Title String (256)
Contents String (8096)
Install Source String(128)
Data Model Revision String(8)
Install Date String(16)
Is Patch String(8)

Fileset Groups

Filesets
Fileset Software Specification String (1024) **[Key]**
Tag String (64)
Create Time String (16)
Description String (8096)
Modification Time String (16)
Size String (32)
Title String (256)
Control Directory String (256)
Is Kernel String (8)
Is Locatable String (8)
Is Reboot String (8)
Location String (1024)
Media Sequence List String (1024)
Revision String (64)
State String (16)
Data Model Revision String(8)
Instance Identifier String(16)
Install Date String(16)
Architecture String(64)
Machine Type String(64)
Operating System Name String(64)
Operating System Release String(64)
Operating System Version String(64)
Install Source String(128)
Is Patch String(8)
Is Sparse String(8)
Patch State STring(16)
Applied To String(1024)
Superseded By String(1024)
Saved Files Directory(256)

Fileset Ancestors
Fileset Software Specification String (1024)
Index Integer
Ancestor Software Specification String (1024)

Fileset Contents
Fileset Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Content String (1024)
Content Type Enum

Fileset Dependencies
Fileset Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Dependency String (1024)
Dependency Type Enum

Fileset Control Files
Fileset Software Specification String (1024) **[Key]**
Tag String (64) **[Key]**
Cksum String (16)
Compressed Cksum String (16)
Compressed Size String (16)
Compression State String (16)
Compression Type String (64)
Revision String (64)
Size String (16)
Source String (256)
Interpreter String (256)
Path String (256)
Result String (16))

Fileset Files
Fileset Software Specification String (1024) **[Key]**
Path String (256) **[Key]**
Cksum String (16)
Compressed Cksum String (16)
Compressed Size String (16)
Compression State String (16)
Compression Type String (64)
Revision String (64)
Size String (16)
Source String (256)
Gid String (16)
Group String (256)
Is Volatile String (8)
Link Source String (256)
Major String (16)
Minor String (16)
Mode String (16)
Mtime String (16)
Owner String (256)
File Type String (8)
Uid String (16)
Archive Path String(16)

Fileset Supersedes
Fileset Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Supersedes Software Specification String (1024)

Fileset Applied Patches
Fileset Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Applied Patch Software Specification String (1024)

Category Groups

Categories
Tag String (64) **[Key]**
Index Integer **[Key]**
Title String (256)
Description String (8096)
Revision String (64)

Product Category Tags
Product Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Category Tag String (1024)

Bundle Category Tags
Bundle Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Category Tag String (1024)

Subproduct Category Tags
Subproduct Software Specification String(1024) **[Key]**
Index Integer **[Key]**
Category Tag String (1024)

Fileset Category Tags
Fileset Software Specification String (1024) **[Key]**
Index Integer **[Key]**
Category Tag String (1024)