

CIM INSTRUMENTATION FOR LINUX : FUNCTIONAL SPECIFICATION

PROVIDER SPECIFICATION AND IMPLEMENTATION DETAILS

Document Version : 1.0
Date Last Changed : November 18, 2003

Document Owner : Heidi Neumann
Department : LTC Systems Management
e-mail : heidineu@de.ibm.com
Web Page : oss.software.ibm.com/developerworks/projects/sblim

Master Copy is located on PC of document owner

Table of Contents

TABLE OF CONTENTS	2
TABLE OF FIGURES	5
NOTION DEFINITION	6
1 DOCUMENT CONTROL	7
1.1 CHANGE HISTORY	7
1.2 REVIEWERS LIST	7
1.3 CONTRIBUTING AUTHORS.....	7
2 INTRODUCTION	8
2.1 COMMON INFORMATION MODEL (CIM).....	8
2.2 CIM INFRASTRUCTURE	8
2.3 MANAGEMENT INSTRUMENTATION.....	9
2.3.1 <i>Architecture</i>	9
2.3.2 <i>Interfaces</i>	10
2.3.2.1 Instance Provider.....	10
2.3.2.2 Association Provider	11
2.3.2.3 Method Provider	11
2.3.2.4 Indication Provider	11
3 CMPI-BASE	13
3.1 PACKAGE CHARACTERISTICS.....	13
3.1.1 <i>Supported Platforms</i>	13
3.1.2 <i>Dependencies</i>	13
3.2 CIM SCHEMA.....	13
3.2.1 <i>DMTF CIM Schema Dependency</i>	13
3.2.2 <i>Linux Schema Definition</i>	13
3.2.2.1 <i>Linux_ComputerSystem</i>	14
3.2.2.2 <i>Linux_OperatingSystem</i>	15
3.2.2.3 <i>Linux_UnixProcess</i>	16
3.2.2.4 <i>Linux_Processor</i>	17
3.2.2.5 <i>Linux_RunningOS</i>	18
3.2.2.6 <i>Linux_OSProcess</i>	18
3.2.2.7 <i>Linux_CSProcessor</i>	19
3.3 MANAGEMENT INSTRUMENTATION.....	19
3.3.1 <i>Linux_ComputerSystem</i>	19
3.3.1.1 Instance Interface	19
3.3.1.2 Method Support.....	20
3.3.1.3 Class Properties.....	20
3.3.2 <i>Linux_OperatingSystem</i>	21
3.3.2.1 Instance Interface	21
3.3.2.2 Method Support.....	22
3.3.2.3 Class Properties.....	22
3.3.3 <i>Linux_UnixProcess</i>	23
3.3.3.1 Instance Interface	23
3.3.3.2 Method Support.....	24
3.3.3.3 Class Properties.....	24
3.3.4 <i>Linux_Processor</i>	25
3.3.4.1 Instance Interface	25
3.3.4.2 Method Support.....	26
3.3.4.3 Class Properties.....	27

3.3.5 <i>Linux_RunningOS</i>	28
3.3.5.1 Instance Interface	28
3.3.5.2 Association Interface.....	29
3.3.6 <i>Linux_OSProcess</i>	30
3.3.6.1 Instance Interface	30
3.3.6.2 Association Interface.....	31
3.3.7 <i>Linux_CSProcessor</i>	31
3.3.7.1 Instance Interface	31
3.3.7.2 Association Interface.....	32
3.4 TOOL LIBRARY.....	33
3.4.1 <i>CIM dependent Layer</i>	33
3.4.1.1 Static Variables	33
3.4.1.2 Tool Functions	33
3.4.1.3 Generic 1-to-N Association Provider Functions.....	34
3.4.2 <i>Resource Access Layer</i>	36
4 CMPI-FSVOL.....	38
4.1 PACKAGE CHARACTERISTICS.....	38
4.1.1 <i>Supported Platforms</i>	38
4.1.2 <i>Dependencies</i>	38
4.2 CIM SCHEMA.....	38
4.2.1 <i>DMTF CIM Schema Dependency</i>	38
4.2.2 <i>Linux Schema Definition</i>	38
4.2.2.1 <i>Linux_Ext2FileSystem, Linux_Ext3FileSystem, Linux_ReiserFileSy stem</i>	39
4.2.2.2 <i>Linux_NFS</i>	40
4.2.2.3 <i>Linux_HostedFileSystem</i>	42
4.2.2.4 <i>Linux_BootOSFromFS</i>	42
4.3 MANAGEMENT INSTRUMENTATION.....	42
4.3.1 <i>Linux_Ext2FileSystem, Linux_Ext3FileSystem, Linux_ReiserFileSystem</i>	42
4.3.1.1 Instance Interface	42
4.3.1.2 Class Properties.....	43
4.3.2 <i>Linux_NFS</i>	44
4.3.2.1 Instance Interface	44
4.3.2.2 Class Properties.....	45
4.3.3 <i>Linux_HostedFileSystem</i>	46
4.3.3.1 Instance Interface	46
4.3.3.2 Association Interface.....	47
4.3.4 <i>Linux_BootOSFromFS</i>	47
4.3.4.1 Instance Interface	47
4.3.4.2 Association Interface.....	48
4.4 TOOL LIBRARY.....	49
4.4.1 <i>CIM dependent Layer</i>	49
4.4.2 <i>Resource Access Layer</i>	50
5 CMPI-NETWORK.....	51
5.1 PACKAGE CHARACTERISTICS.....	51
5.1.1 <i>Supported Platforms</i>	51
5.1.2 <i>Dependencies</i>	51
5.2 CIM SCHEMA.....	51
5.2.1 <i>DMTF CIM Schema Dependency</i>	51
5.2.2 <i>Linux Schema Definition</i>	51
5.2.2.1 <i>Linux_LocalLoopbackPort</i>	53
5.2.2.2 <i>Linux_EthernetPort</i>	54
5.2.2.3 <i>Linux_TokenRingPort</i>	55
5.2.2.4 <i>Linux_IPProtocolEndpoint</i>	56
5.2.2.5 <i>Linux_CSNetworkPort</i>	57
5.2.2.6 <i>Linux_NetworkPortImplementsIPEndpoint</i>	57
5.3 MANAGEMENT INSTRUMENTATION.....	57
5.3.1 <i>Linux_LocalLoopbackPort</i>	57

5.3.1.1 Instance Interface	57
5.3.1.2 Method Support.....	58
5.3.1.3 Class Properties.....	59
5.3.2 <i>Linux_EthernetPort</i>	60
5.3.2.1 Instance Interface	60
5.3.2.2 Method Support.....	61
5.3.2.3 Class Properties.....	61
5.3.3 <i>Linux_TokenRingPort</i>	62
5.3.3.1 Instance Interface	62
5.3.3.2 Method Support.....	63
5.3.3.3 Class Properties.....	64
5.3.4 <i>Linux_IPProtocolEndpoint</i>	64
5.3.4.1 Instance Interface	64
5.3.4.2 Class Properties.....	65
5.3.5 <i>Linux_CSNetworkPort</i>	66
5.3.5.1 Instance Interface	66
5.3.5.2 Association Interface.....	67
5.3.6 <i>Linux_NetworkPortImplementsIPEndpoint</i>	68
5.3.6.1 Instance Interface	68
5.3.6.2 Association Interface.....	69
5.4 TOOL LIBRARY.....	70
5.4.1 <i>CIM dependent Layer</i>	70
5.4.2 <i>Resource Access Layer</i>	70

Table of Figures

FIGURE 2.1 : CIM INFRASTRUCTURE	9
FIGURE 3.1 : BASE MODEL	14
FIGURE 4.1 : FILE SYSTEM MODEL	39
FIGURE 5.1 : LINUX NETWORK MODEL.....	52

Notion Definition

DMTF	Distributed Management Task Force at www.dmtf.org
CIM	Common Information Model (see chapter 2.1 Common Information Model (CIM))
Class	CIM classes describe computer system entities or certain aspects of them.
Property	Properties describe the attributes of managed entities and contain the values.
Association	Associations describe the relations of CIM classes to each other.
Method	A class can define extrinsic methods. A method can act class or instance specific.
Indication	Is the capability of managed entities to create indications if a certain event occurred.
CMPI	Common Manageability Programming Interface(see
SBLIM	Standards Based Linux Instrumentation for Manageability at oss.software.ibm.com/developerworks/projects/sblim
Object Path	Is an implementation construct and defines the path to the system (hostname and CIM namespace) and the key properties. An object path can have two states : empty key values or filled key values. The first is used to access class and namespace information. The second identifies a specific instance of the class.
Instance	An instance represents an existing entity of the class.
Object	The object is type less and used for both, an object path or an instance.

1 Document Control

1.1 Change History

Revision	Date	Description
1	Nov 2003	Initial Version

1.2 Reviewers List

Name	Role & Responsibility

1.3 Contributing Authors

Name	Date	e-mail address	Description

2 Introduction

2.1 Common Information Model (CIM)

The Common Information Model (CIM) was developed and standardized by the industry organization Distributed Management Task Force, Inc. (DMTF). More information about the DMTF can be found under www.dmtf.org.

CIM is an object-oriented model to describe overall management entities of enterprise environments and enables the platform-independent and technology-neutral exchange of management information. CIM consists of a specification and a schema. The specification describes integration aspects. The schema consists of an extensive set of modeled enterprise entities. More information about CIM can be found under <http://www.dmtf.org/standards/cim>.

2.2 CIM Infrastructure

A major advantage of a CIM enabled enterprise environment is the capability of one CIM Client Application to remotely manage all different kinds of CIM enabled platforms.

Figure 2.1 shows the different parts of a CIM infrastructure. A managed system is CIM enabled, if a CIM Object Manager (CIMOM) and the necessary Management Instrumentation (Provider) are installed and running. CIMOM and CIM Client “talk” CIM over HTTP. This protocol is a DMTF standard. The communication between CIMOM and CIM Client takes place as XML stream.

The CIMOM is responsible for the communication to the CIM Client, manages the schema and routes requests down to the provider, who is responsible for the requested function. At the moment, there are four different Open Source CIMOMs available.

- | | | |
|----------------|------|--|
| • openCimom | Java | www.opengroup.org/snia-cimom |
| • Pegasus | C++ | www.openpegasus.org |
| • OpenWBEM | C++ | openwbem.sourceforge.net |
| • WbemServices | Java | wbemservices.sourceforge.net |

The Common Manageability Programming Interface (CMPPI) is a standardized C provider interface. It was defined to write CIMOM independent Management Instrumentation. Each of the four Open Source CIMOMs defines its own provider API. To avoid duplicating work by porting the same provider functionality to each of the CIMOM specific provider API, a common C provider interface was defined and standardized by the WBEMsource initiative (www.wbemsource.org).

A provider (management instrumentation) is a plugin to the CIMOM and implements the platform specific resource access.

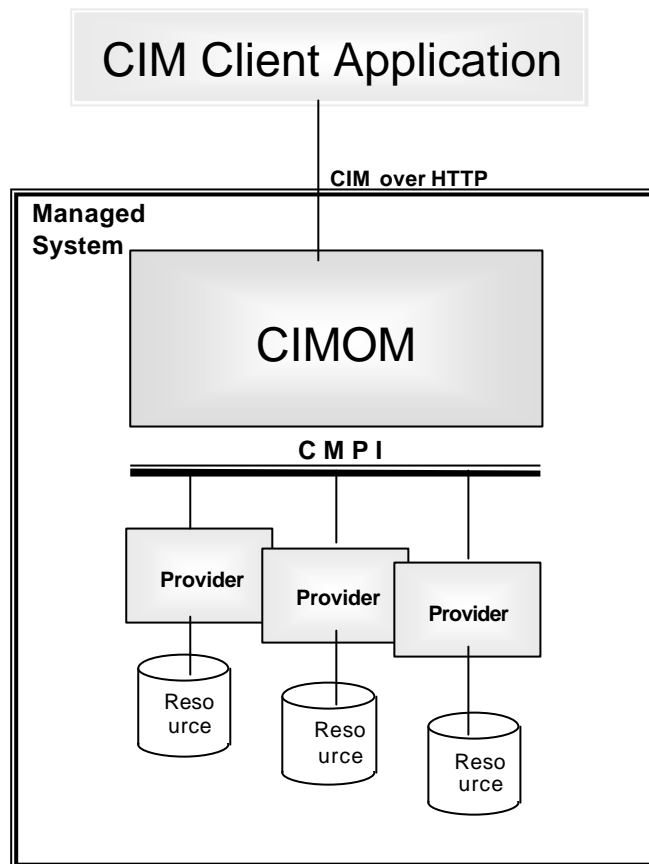


Figure 2.1 : CIM Infrastructure

2.3 Management Instrumentation

In a CIM enabled environment, the model and the upper parts of the CIM infrastructure can be as platform-independent as possible. The CIM Management Instrumentation (Provider) is the glue between the model and the managed system and maps the platform independent model to the platform specific resources.

2.3.1 Architecture

As different environments may require different CIMOMs, it is recommended to write CMPI management instrumentation (provider) to become independent of the underlying CIMOM technology. The provider described in this paper, implement the CMPI.

The architecture document describing the provider implementation concept can be found on the SBLIM web page under <http://oss.software.ibm.com/developerworks/oss/sblim/doc/ProviderArchitecture.pdf>.

2.3.2 Interfaces

It exist four different types of provider interfaces. Each one represents a certain capability of a class / association.

2.3.2.1 Instance Provider

The instance interface is responsible for enumerating the available resource entities and performing actions on these instances. The following interfaces must be implemented by an instance provider :

- **Cleanup()**

Deinitialization routine.

- **EnumInstanceNames()**

The provider is responsible for reporting all instances of the class that are available at this point in time. It may happen, that the provider reports different instances at different points in time. It is not guaranteed, that an older instance still exists at a later point in time. The returned objects are object paths.

- **EnumInstances()**

The provider is responsible for reporting all instances of the class that are available at this point in time. It may happen, that the provider reports different instances at different points in time. It is not guaranteed, that an older instance still exists at a later point in time. The returned objects are instances.

- **GetInstance()**

The provider is responsible for returning the requested instance, which is specified by the input object path.

- **SetInstance()**

The provider is responsible for setting new property values. The behavior depends on the semantic of the implemented class.

- **CreateInstance()**

The provider is responsible for creating the new instance. The behavior depends on the semantic of the implemented class.

- **DeleteInstance()**

The provider is responsible for deleting the specified instance. The behavior depends on the semantic of the implemented class.

- **ExecQuery()**

The provider is responsible for parsing the input query string (filter) and reports only the objects, which achieve the filter. The returned objects are instances.

2.3.2.2 Association Provider

The association interface represents the knowledge about the relations certain resource entities have to each other. The following interfaces must be implemented by an association provider :

- **AssociationCleanup()**

Deinitialization routine.

- **Associators()**

The provider gets a source object path and is responsible for reporting all target objects, staying in the relation to the source object. The relation is defined by the association class. The returned objects are instances of the target class.

- **AssociatorNames()**

The provider gets a source object path and is responsible for reporting all target objects, staying in the relation to the source object. The relation is defined by the association class. The returned objects are object paths of the target class.

- **References()**

The provider gets a source object path and is responsible for reporting all association objects, where the specified source object stay in relation to target object(s). The relation is defined by the association class. The returned objects are instances of the association class.

- **ReferenceNames()**

The provider gets a source object path and is responsible for reporting all association objects, where the specified source object stay in relation to target object(s). The relation is defined by the association class. The returned objects are object paths of the association class.

2.3.2.3 Method Provider

The method interface is responsible for executing extrinsic methods on a certain class or instance, for example restarting the operating system. These interfaces must be implemented by a method provider :

- **MethodCleanup()**

Deinitialization routine.

- **InvokeMethod()**

The provider is responsible for executing the requested extrinsic method either on the specified object or the class.

2.3.2.4 Indication Provider

The indication interface allows the provider to monitor certain events and create indications. For example, a client subscribes to the event of file system usage higher than 90 % and will be informed once this happens.

- **IndicationCleanup()**
Deinitialization routine.
- **ActivateFilter()**
TBD
- **AuthorizeFilter()**
TBD
- **DeActivateFilter()**
TBD
- **MustPoll()**
TBD

3 CMPI-BASE

3.1 Package Characteristics

3.1.1 Supported Platforms

- xSeries
- pSeries
- zSeries

3.1.2 Dependencies

The providers depend on a CIMOM, supporting the Common Manageability Programming Interface (CMPI).

3.2 CIM Schema

3.2.1 DMTF CIM Schema Dependency

The Linux schema requires the CIM Schema version 2.7 or higher. The necessary sub schema parts are :

- Core
- System
- Device

3.2.2 Linux Schema Definition

The cmpi-base package implements the very base classes, needed to describe a system. This includes the computer system container itself, the operating system, the currently executed processes and the processor.

The following UML diagram shows the Linux Base Model and the CIM Inheritance tree for the classes. The inheritance tree of the associations is not shown. The red marked properties are already deprecated and will be removed from the CIM model at a later point in time.

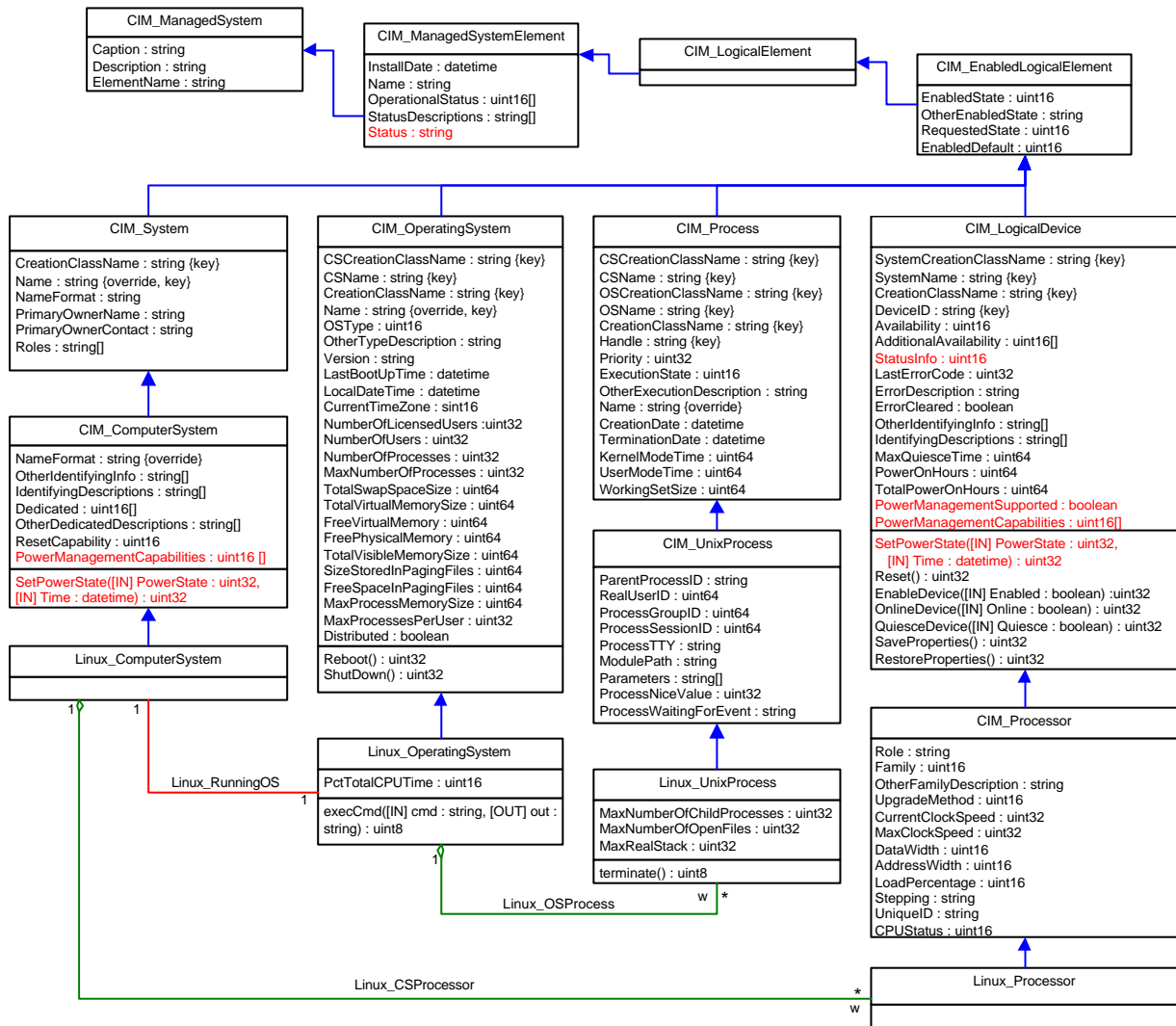


Figure 3.1 : Base Model

3.2.2.1 Linux_ComputerSystem

The class Linux_ComputerSystem represents the computer system container and acts as an aggregation point to associate one or more of the following elements : file systems, operating system, processors and memory (volatile and / or non volatile).

The CIM inheritance tree can be found in figure 3.1 Base Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the computer system.	Y
Description	string	Textual description of the computer system.	Y
ElementName	string	Contains a user-friendly name of the computer system.	Y
InstallDate	datetime	Indicates the install date of the computer system.	N
Name	string key	Serves as unique key in and enterprise environment and therefore contains the full qualified and unique hostname of the computer system : <host.domain>	Y

OperationalStatus	uint16[]	Indicates the current status of the computer system (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the computersystem	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the computer system. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the computer system.	Y
EnabledDefault	uint16	The administrator's default / startup state of the computer system	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
NameFormat	string	Identifies the heuristic used to get / create the unique system name .	Y
PrimaryOwnerName	string	Name of the primary system owner.	Y
PrimaryOwnerContact	string	Contact to the primary system owner (e.g email address ...).	Y
Roles	string[]	Specifies the administrator defined roles, the system plays in a managed environment.	N
OtherIdentifyingInfo	string[]	Additional data to identify the computer system.	N
IdentifyingDescriptions	string[]	Textual descriptions to the OtherIdentifyingInfo value(s).	N
Dedicated	uint16[]	Indicates if the computer system is a special purpose system (value map).	Y
OtherDedicatedDescriptions	string[]	Textual descriptions to the Dedicated value(s).	N
ResetCapability	uint16	If enabled (4) the computer system can be reset via hardware (value map).	N
PowerManagementCapabilities	uint16[]	Indicates power management capabilities of the computer system (value map).	N

3.2.2.2 Linux_OperatingSystem

The class Linux_OperatingSystem represents the installed and current running Linux operating system, independent of the distribution (SuSE, Red Hat, ...).

The CIM inheritance tree can be found in figure 3.1 Base Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the operating system.	Y
Description	string	Textual description of the operating system.	Y
ElementName	string	Contains a user-friendly name of the operating system.	Y
InstallDate	datetime	Indicates the install date of the operating system.	Y
Name	string key	Contains the full qualified and unique hostname of the operating system : <host.domain>	Y
OperationalStatus	uint16[]	Indicates the current status of the operating system (value map).	Y
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the operating system.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the operating system. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the operating system.	Y
EnabledDefault	uint16	The administrator's default / startup state of the operating system.	Y
CSCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
CSName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
OSType	uint16	Indicates the type of the operating system (value map).	Y
OtherTypeDescription	string	Textual description, if OSType is set to 1 ('Other') or 59 ('Dedicated').	Y
Version	string	Contains the version number of the operating system.	Y

LastBootUpTime	datetime	Time when the operating system was last booted.	Y
LocalDateTime	datetime	The operating system's notion of the local date and time of day.	Y
CurrentTimeZone	uint16	Indicates the number of minutes, the operating system is offset from Greenwich Mean Time.	Y
NumberOfLicensedUsers	uint32	Number of user licenses for this operating system instance.	N
NumberOfUsers	uint32	Contains the current number of user sessions.	Y
NumberOfProcesses	uint32	Number of processes currently executing in the operating system's scope.	Y
MaxNumberOfProcesses	uint32	Max number of process contexts the operating system can support.	Y
TotalSwapSpaceSize	uint64	Total swap space in Kbytes.	Y
TotalVirtualMemorySize	uint64	Total virtual memory in Kbytes of.	Y
FreeVirtualMemory	uint64	Number of Kbytes of virtual memory currently unused and available.	Y
FreePhysicalMemory	uint64	Number of Kbytes of physical memory currently unused and available.	Y
TotalVisibleMemorySize	uint64	Total physical memory in Kbytes available to the operating system.	Y
SizeStoredInPagingFiles	uint64	Total number of Kbytes that can be stored in paging files.	Y
FreeSpaceInPagingFiles	uint64	Number of Kbytes of page space currently free.	Y
MaxProcessMemorySize	uint64	Max number of Kbytes of memory that can be allocated to a process.	Y
MaxProcessesPerUser	uint32	Indicates the max number of processes a user can have associated with it.	N
Distributed	boolean	Indicates if the operating system is distributed across several computer systems.	Y
PctTotalCPUTime	uint16	Total CPU time in percentage.	Y

3.2.2.3 Linux_UnixProcess

The class Linux_UnixProcess represents the currently running processes of the operating system. The user will typically see a process as an application or task. Within an operating system, a process is defined by a workspace of memory resources and environment settings allocated to it. On a multitasking system, the workspace prevents intrusion of resources by other processes.

The CIM inheritance tree can be found in figure 3.1 Base Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the process.	Y
InstallDate	datetime	Indicates the install date of the process.	N
Name	string	Contains a user-friendly name of the process.	Y
OperationalStatus	uint16[]	Indicates the current status of the process (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the process.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the process. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the process.	Y
EnabledDefault	uint16	The administrator's default / startup state of the process.	Y
CSCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
CSName	string key	Contains the Name of the scoping computer system.	Y
OSCreationClassName	string key	Contains the CreationClassName of the scoping operating system.	Y
OSName	string key	Contains the Name of the scoping operating system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y

Handle	string Key	Contains the unique Process ID.	Y
Priority	uint32	Indicates the importance of the process. The range is from -20 (high) to 20 (low).	Y
ExecutionState	uint16	Indicates the current state of the process (value map).	Y
OtherExecutionDescription	string	Indicates the state, when ExecutionState is set to 1 ('Other').	N
CreationDate	datetime	Time the process was created.	Y
TerminationDate	datetime	Time the process was stopped or terminated.	N
KernelModeTime	uint64	Milliseconds the process executed in kernel mode.	Y
UserModeTime	uint64	Milliseconds the process executed in user mode.	Y
WorkingSetSize	uint64	Amount of bytes the process needs for an efficient execution.	N
ParentProcessID	string	Parent's process ID.	Y
RealUserID	uint64	Real user ID of this process.	Y
ProcessGroupID	uint64	Group ID of this process.	Y
ProcessSessionID	uint64	Session ID of this process.	Y
ProcessTTY	string	TTY currently associated with this process.	Y
ModulePath	string	The command path of this process.	Y
Parameters	string[]	Parameters given to the process (argv[] values).	Y
ProcessNiceValue	uint32	Nice value of this process. Used to compute the priority.	Y
ProcessWaitingForEvent	string	Description of the event, the process is currently sleeping for.	N
MaxNumberOfChildProcesses	uint32	Max number of child processes, this process can have.	(Y)
MaxNumberOfOpenFiles	uint32	Max number of files, this process can open during the execution.	(Y)
MaxRealStack	uint32	Max number of Kbytes of memory that can be allocated to a process.	(Y)

3.2.2.4 Linux_Processor

The class Linux_Processor represents the CPUs available (visible) to the operating system.

The CIM inheritance tree can be found in figure 3.1 Base Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the processor.	Y
InstallDate	datetime	Indicates the install date of the processor.	N
Name	string	Contains a user-friendly name of the processor.	Y
OperationalStatus	uint16[]	Indicates the current status of the processor (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the processor.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the processor. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the processor.	Y
EnabledDefault	uint16	The administrator's default / startup state of the processor.	Y
SystemCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
SystemName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
DeviceID	string Key	Contains the unique Processor ID.	Y
Availability	uint16	Primary availability and status of the processor (value map).	N

AdditionalAvailability	uint16[]	Additional information about the availability of the processor.	N
StatusInfo	uint16	Indicates the enabled / disabled state of the processor.	N
LastErrorCode	uint32	Last error code reported by the processor.	N
ErrorDescription	string	Textual description of the error code.	N
ErrorCleared	boolean	Last error is cleared now.	N
OtherIdentifyingInfo	string[]	Additional information to identify the processor.	N
IdentifyingDescriptions	string[]	Textual description of the OtherIdentifyingInfo value(s).	N
MaxQuiesceTime	uint64	Max time in milliseconds the processor can run in 'Quiesced' state.	N
PowerOnHours	uint64	Number of hours the processor has been powered, since last power cycle.	N
TotalPowerOnHours	uint64	Total number of hours the processor has been powered.	N
PowerManagementSupported	boolean	Processor can be power managed (value map).	N
PowerManagementCapabilities	uint16	Indicates the power management capabilities of the processor.	N
Role	string	Indicates the role of the processor.	Y
Family	uint16	Indicates the family of the processor (value map).	Y
OtherFamilyDescription	string	Indicates the family, if Family is set to 1 ('Other').	Y
UpgradeMethod	uint16	CPU socket information with data how the processor can be upgraded.	N
CurrentClockSpeed	uint32	Current speed in MHz of this processor.	Y
MaxClockSpeed	uint32	Max speed in MHz of this processor.	Y
DataWidth	uint16	Processor data width in bits.	N
AddressWidth	uint16	Processor address width in bits.	N
LoadPercentage	uint16	Load on this processor, averaged over the last minute.	Y
Stepping	string	Indicating the revision level of the processor within the family.	Y
UniqueID	string	A globally unique identifier for the processor.	Y
CPUSStatus	uint16	Indicates the current status of the processor (value map).	Y

3.2.2.5 Linux_RunningOS

The association Linux_RunningOS indicates the currently executing operating system on the computer system. Within the scope of the Linux base model, the one instance of this association always exists.

The CIM inheritance tree is Linux_RunningOS => CIM_RunningOS => CIM_Dependency.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
Antecedent	REF key	Linux_OperatingSystem	The operating system currently running on the computer system.
Dependent	REF key	Linux_ComputerSystem	The computer system container.

3.2.2.6 Linux_OSProcess

The aggregation Linux_OSProcess is a link between the operating system and the process(es) running in the context of this operating system.

The CIM inheritance tree is Linux_OSProcess => CIM_OSProcess => CIM_Component.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
GroupComponent	REF key	Linux_OperatingSystem	The currently running operating system.
PartComponent	REF key	Linux_UnixProcess	The process(es) running in the context of the operating system.

3.2.2.7 Linux_CSProcessor

The aggregation Linux_CSProcessor indicates the processor(s) of the computer system container.

The CIM inheritance tree is Linux_CSProcessor => CIM_SystemDevice => CIM_SystemComponent => CIM_Component.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
GroupComponent	REF key	Linux_ComputerSystem	The computer system.
PartComponent	REF key	Linux_Processor	The processor(s) of the computer system.

3.3 Management Instrumentation

3.3.1 Linux_ComputerSystem

The class Linux_ComputerSystem is implemented by the provider `cmpiOSBase_ComputerSystemProvider`. The provider supports the instance and method interfaces. It consists of three modules (`cmpiOSBase_ComputerSystemProvider`, `cmpiOSBase_ComputerSystem`, `OSBase_ComputerSystem`) and depends on the tool library `cmpiOSBase_Commom` (see chapter 3.4 Tool Library).

3.3.1.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the one object path / instance of Linux_ComputerSystem.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of Linux_ComputerSystem.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **CreateInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **DeleteInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **ExecQuery()**
CMPI_RC_ERR_NOT_SUPPORTED

3.3.1.2 Method Support

The method interface is supported and part of the provider `cmptOSBase_ComputerSystemProvider`.

- **MethodCleanup()**
No functionality.
- **Invoke Method()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- o `uint32 SetPowerState([IN] uint32 PowerState, [IN] datetime Time)`
`CMPI_RC_ERR_NOT_SUPPORTED`

3.3.1.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s `SetInstance()` function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	DESCRIPTION
Caption	string	no	Computer System
Description	string	no	A class derived from <code>ComputerSystem</code> that represents the single node container of the Linux OS.
ElementName	string	no	<i>same value as key property “Name”</i>
Name	string key	no	<i>calls <code>get_system_name()</code> of tool library <code>cmptOSBase_Common</code>; this function returns the value <host.domain></i>
Status	string	no	NULL
EnabledState	uint16	no	2 (“Enabled”)
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as “EnabledState”</i>
EnabledDefault	uint16	no	<i>same value as “EnabledState”</i>
CreationClassName	string key	no	Linux_ComputerSystem

NameFormat	string	no	IP
PrimaryOwnerName	string	no	root
PrimaryOwnerContact	string	no	root@<host.domain>
Dedicated	uint16[]	no	0

3.3.2 Linux_OperatingSystem

The class `Linux_OperatingSystem` is implemented by the provider `cmpiOSBase_OperatingSystemProvider`. The provider supports the instance and method interfaces. It consists of three modules (`cmpiOSBase_OperatingSystemProvider`, `cmpiOSBase_OperatingSystem`, `OSBase_OperatingSystem`) and depends on the tool library `cmpiOSBase_Commom` (see chapter 3.4 Tool Library).

3.3.2.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the one object path / instance of `Linux_OperatingSystem`.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_OperatingSystem`.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

3.3.2.2 Method Support

The method interface is supported and part of the provider `cmptOSBase_OperatingSystemProvider`.

- **MethodCleanup()**

No functionality.

- **InvokeMethod()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- o `uint32 Reboot()`

`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 ShutDown()`

`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint8 execCmd([IN] string cmd, [OUT] string out)`

Execute the command `cmd` in a bash shell.

The method distinguishes the following success cases. It returns 0 and the `stdout` result of the command execution in `out`. It returns 1 and the `stderr` result of the command execution in `out`. If the execution failed it returns 2 and a human readable error message.

3.3.2.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s `SetInstance()` function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Operating System
Description	string	no	A class derived from <code>OperatingSystem</code> to represents the running Linux OS.
ElementName	string	no	<i>same value as “Version”</i>
InstallDate	datetime	no	<i>Red Hat : the install date of the release (rpm -qi redhat-release grep Install)</i>
Name	string key	no	<i>calls <code>get_os_name()</code> of tool library <code>cmptOSBase_Common</code>; this function returns the value <host.domain></i>
OperationalStatus	uint16[]	no	<i>if <code>PctTotalCPUtime > 90</code> then value = 4 (“Stressed”) else value = 2 (“OK”)</i>
Status	string	no	NULL
EnabledState	uint16	no	2 (“Enabled”)
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as “EnabledState”</i>
EnabledDefault	uint16	no	<i>same value as “EnabledState”</i>
CSCreationClassName	string key	no	CreationClassName of <code>Linux_ComputerSystem</code>
CSName	string	no	Name of <code>Linux_ComputerSystem</code>

	key		
CreationClassName	string key	no	Linux_OperatingSystem
OSType	uint16	no	36 ("Linux")
OtherTypeDescription	string	no	NULL
Version	string	no	content of the release file in the /etc directory (cat /etc/ls /etc/ grep release')
LastBootUpTime	datetime	no	implemented by _get_os_boottime() of the tool library cmpiOSBase_Common.so; this function gets the btime value out of the /proc/stat file
LocalDateTime	datetime	no	calls gettimeofday() function; first parameter contains the current date and time
CurrentTimeZone	uint16	no	calls gettimeofday() function; second parameter contains the timezone
NumberOfUsers	uint32	no	who -u wc -l
NumberOfProcesses	uint32	no	/proc/loadavg contains "1min 5min 10min runningProcess/NumberOfProcesses ..."
MaxNumberOfProcesses	uint32	no	content of the /proc/sys/fs/file-max file
TotalSwapSpaceSize	uint64	no	0
TotalVirtualMemorySize	uint64	no	TotalVisibleMemorySize + SizeStoredInPagingFiles
FreeVirtualMemory	uint64	no	FreePhysicalMemory + FreeSpaceInPagingFiles
FreePhysicalMemory	uint64	no	entry number 10 in /proc/meminfo
TotalVisibleMemorySize	uint64	no	entry number 8 in /proc/meminfo
SizeStoredInPagingFiles	uint64	no	entry number 15 in /proc/meminfo
FreeSpaceInPagingFiles	uint64	no	entry number 17 in /proc/meminfo
MaxProcessMemorySize	uint64	no	calls getrlimit() function; this returns an rlimit structure; rlimit.rlim_max contains the value
Distributed	boolean	no	false
PctTotalCPUTime	uint16	no	/proc/stat contains "cpu user system nice idle ..."; (100 * user+system+nice) / (user+system+nice+idle)

3.3.3 Linux_UnixProcess

The class Linux_UnixProcess is implemented by the provider cmpiOSBase_UnixProcessProvider. The provider supports the instance and method interfaces. It consists of three modules (cmpiOSBase_UnixProcessProvider, cmpiOSBase_UnixProcess, OSBase_UnixProcess) and depends on the tool library cmpiOSBase_Common (see chapter 3.4 Tool Library).

3.3.3.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of Linux_UnixProcess as processes are currently running in the operating system. The listing is based on the ps command.

In the case of failure, the function returns with CMPI_RC_ERR_FAILED and a human readable error message.

The EnumInstances() input parameter properties is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_UnixProcess` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

3.3.3.2 Method Support

The method interface is supported and part of the provider `cmplOSBase_UnixProcessProvider`.

- **MethodCleanup()**

No functionality.

- **InvokeMethod()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- o `uint8 terminate()`

Kills the process indicated by the object path.

In the case of success, the function returns 0. Otherwise it returns 1.

3.3.3.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s `SetInstance()` function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Linux (Unix) Process
Description	string	no	This class represents instances of currently running programmes.
ElementName	string	no	same value as "Name"
Name	string	no	<i>ps -p<PID> -o comm</i> delivers the command executed by this PID
Status	string	no	NULL
EnabledState	uint16	no	2 ("Enabled")
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	same value as "EnabledState"
EnabledDefault	uint16	no	same value as "EnabledState"
CSCreationClassName	string key	no	CreationClassName of Linux_ComputerSystem
CSName	string key	no	Name of Linux_ComputerSystem
OSCreationClassName	string key	no	CreationClassName of Linux_OperatingSystem
OSName	string key	no	Name of Linux_OperatingSystem
CreationClassName	string key	no	Linux_OperatingSystem
Handle	string key	no	contains the unique Process ID (PID)
Priority	uint32	no	<i>ps -p<PID> -o pri</i> delivers the priority of this PID
ExecutionState	uint16	no	<i>ps -p<PID> -o state</i> delivers the state of this PID; supported types : 3 (Running) = R, 4 (Blocked) = D, 6 (Suspended Ready) = S, 7 (Terminated) = Z, 8 (Stopped) = T
CreationDate	datetime	no	(entry number 22 in /proc/<PID>/stat in Jiffies (/100)) + (system boot time)
KernelModeTime	uint64	no	entry number 16 in /proc/<PID>/stat
UserModeTime	uint64	no	entry number 15 in /proc/<PID>/stat
ParentProcessID	string	no	<i>ps -p<PID> -o ppid</i> delivers the parent PID of this PID
RealUserID	uint64	no	<i>ps -p<PID> -o uid</i> delivers the user ID of this PID
ProcessGroupID	uint64	no	<i>ps -p<PID> -o gid</i> delivers the process group ID of this PID
ProcessSessionID	uint64	no	<i>ps -p<PID> -o session</i> delivers the session ID of this PID
ProcessTTY	string	no	<i>ps -p<PID> -o tty</i> delivers the TTY of this PID
ModulePath	string	no	content of the /proc/<PID>/exe file
Parameters	string[]	no	<i>ps -p<PID> -o args</i> delivers the parameters given to the command of this PID
ProcessNiceValue	uint32	no	<i>ps -p<PID> -o nice</i> delivers the nice value of this PID
MaxNumberOfChildProcesses	uint32	no	entry number 1 in /proc/sysman/pid_rlimit (optional)
MaxNumberOfOpenFiles	uint32	no	entry number 2 in /proc/sysman/pid_rlimit (optional)
MaxRealStack	uint32	no	entry number 3 in /proc/sysman/pid_rlimit (optional)

3.3.4 Linux_Processor

The class `Linux_Processor` is implemented by the provider `cmpioSBase_ProcessorProvider`. The provider supports the instance and method interfaces. It consists of three modules (`cmpioSBase_ProcessorProvider`, `cmpioSBase_Processor`, `OSBase_Processor`) and depends on the tool library `cmpioSBase_Commom` (see chapter 3.4 Tool Library).

3.3.4.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of Linux_Processor as processors are available to the operating system. The listing is based on the entries in the `/proc/cpuinfo` file.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_Processor` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

3.3.4.2 Method Support

The method interface is supported and part of the provider `cmpiOSBase_ProcessorProvider`.

- **MethodCleanup()**

No functionality.

- **InvokeMethod()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- `uint32 SetPowerState([IN] uint16 PowerState, [IN] datetime Time)`

`CMPI_RC_ERR_NOT_SUPPORTED`

- o uint32 Reset()
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 EnableDevice([IN] boolean Enabled)
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 OnlineDevice([IN] boolean Online)
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 QuiesceDevice([IN] boolean Quiesce)
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 SaveProperties()
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 RestoreProperties()
CMPI_RC_ERR_NOT_SUPPORTED

3.3.4.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s SetInstance() function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Linux Processor
Description	string	no	This class represents instances of available processors.
ElementName	string	no	<i>Intel : corresponding entry 'model name' in /proc/cpuinfo zSeries : corresponding entry 'vendor_id' in /proc/cpuinfo PowerPC : corresponding entry 'cpu' in /proc/cpuinfo</i>
Name	string	no	<i>same value as "DeviceID"</i>
Status	string	no	NULL
EnabledState	uint16	no	2 ("Enabled")
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as "EnabledState"</i>
EnabledDefault	uint16	no	<i>same value as "EnabledState"</i>
SystemCreationClassName	string key	no	CreationClassName of Linux_ComputerSystem
System Name	string key	no	Name of Linux_ComputerSystem
CreationClassName	string key	no	Linux_OperatingSystem
Device	string key	no	contains the unique Processor ID
Role	string	no	Central Processor
Family *	uint16	no	<i>Intel, zSeries: corresponding entry 'vendor_id' in /proc/cpuinfo PowerPC : corresponding entry 'cpu' in /proc/cpuinfo</i>
OtherFamilyDescription	string	no	NULL

CurrentClockSpeed	uint32	no	Intel : corresponding entry 'cpu MHz' in /proc/cpuinfo zSeries : corresponding entry 'bogomips per cpu' in /proc/cpuinfo PowerPC : corresponding entry 'clock' in /proc/cpuinfo
MaxClockSpeed	uint32	no	same value as "CurrentClockSpeed"
LoadPercentage	uint16	no	corresponding entry in /proc/stat; "cpu<ID> user system nice idle ..."; (100 * user+system+nice) / (user+system+nice+idle)
Stepping	string	no	Intel : corresponding entry 'stepping' in /proc/cpuinfo zSeries, PowerPC : return "no stepping"
UniqueID	string	no	NULL
CPUSStatus	uint16	no	1 ("CPU Enabled")

* Supported Family types and their names :

"Other"	1
"Unknown"	2
"80486"	6
"Pentium(R) Brand"	11
"Pentium(R) Pro"	12
"Pentium(R) II"	13
"Pentium(R) Processor with MMX(TM) Technology"	14
"Celeron(TM)"	15
"Pentium(R) II Xeon(TM)"	16
"Pentium(R) III"	17
"Pentium(R) III Xeon(TM)"	176
"Pentium(R) III Processor with Intel(R) SpeedStep(TM) Technology"	177
"K5 Family"	24
"K6 Family"	25
"K6-2"	26
"K6-3"	27
"AMD Athlon(TM) Processor Family"	28
"AMD Duron(TM) Processor Family"	29
"POWER"	32
"S390"	200

3.3.5 Linux_RunningOS

The association Linux_RunningOS is implemented by the provider `cmpiOSBase_RunningOSProvider`. The provider supports the instance and association interfaces. It consists of one module (`cmpiOSBase_RunningOSProvider`) and depends on the tool library `cmpiOSBase_Common` (see chapter 3.4 Tool Library).

3.3.5.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the one object path / instance of Linux_RunningOS.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of Linux_RunningOS.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, `GetInstance()` returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

3.3.5.2 Association Interface

- **AssociationCleanup()**

No functionality.

- **AssociatorNames(), Associators()**

In the case of success, it returns the object path / instance of the target class. If the object path of the `Linux_ComputerSystem` instance (source) was given to the provider, the object path / instance of `Linux_OperatingSystem` (target) is returned. If the object path of the `Linux_OperatingSystem` instance (source) was given to the provider, the object path / instance of `Linux_ComputerSystem` (target) is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path / instance of `Linux_RunningOS`. This instance is only returned if the source object path is either the `Linux_ComputerSystem` or the `Linux_OperatingSystem` instance.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

3.3.6 Linux_OSProcess

The association `Linux_OSProcess` is implemented by the provider `cmpiOSBase_OSProcessProvider`. The provider supports the instance and association interfaces. It consists of one module (`cmpiOSBase_OSProcessProvider`) and depends on the tool library `cmpiOSBase_Common` (see chapter 3.4 Tool Library).

3.3.6.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of `Linux_OSProcess` as processes, represented by `Linux_UnixProcess`, are running on the operating system.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_OSProcess` defined by the input object path.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, `GetInstance()` returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

3.3.6.2 Association Interface

- **AssociationCleanup()**

No functionality.

- **AssociatorNames(), Associators()**

In the case of success, it returns the object path(s) / instance(s) of the target class. Their number depends on the source object. If the object path of the Linux_OperatingSystem instance (source) was given to the provider, all object paths / instances of Linux_UnixProcess (target) are returned. If the object path of a Linux_UnixProcess instance (source) was given to the provider, the object path / instance of Linux_OperatingSystem (target) is returned.

If the referenced instance (source) do not exist the function returns with CMPI_RC_ERR_NOT_FOUND. In the case of failure, the function returns with CMPI_RC_ERR_FAILED and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path(s) / instance(s) of Linux_OSProcess. Their number depends on the source object. If the object path of the Linux_OperatingSystem instance (source) was given to the provider, as many object paths / instances of Linux_OSProcess are returned, as Linux_UnixProcess (target) instances exist. If the object path of a Linux_UnixProcess instance (source) was given to the provider, one object path / instance of Linux_OSProcess is returned.

If the referenced instance (source) do not exist the function returns with CMPI_RC_ERR_NOT_FOUND. In the case of failure, the function returns with CMPI_RC_ERR_FAILED and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

3.3.7 Linux_CSProcessor

The association `Linux_CSProcessor` is implemented by the provider `cmpiOSBase_CSProcessorProvider`. The provider supports the instance and association interfaces. It consists of one module (`cmpiOSBase_CSProcessorProvider`) and depends on the tool library `cmpiOSBase_Commom` (see chapter 3.4 Tool Library).

3.3.7.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of Linux_CSProcessor as processors, represented by Linux_Processor, are available to the operating system.

In the case of failure, the function returns with CMPI_RC_ERR_FAILED and a human readable error message.

The EnumInstances() input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of Linux_CSProcessor defined by the input object path.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with CMPI_RC_ERR_NOT_FOUND. In the case of any other failure, GetInstance() returns with CMPI_RC_ERR_FAILED and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **CreateInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **DeleteInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **ExecQuery()**

CMPI_RC_ERR_NOT_SUPPORTED

3.3.7.2 Association Interface

- **AssociationCleanup()**

No functionality.

- **AssociatorNames(), Associators()**

In the case of success, it returns the object path(s) / instance(s) of the target class. Their number depends on the source object. If the object path of the Linux_ComputerSystem instance (source) was given to the provider, all object paths / instances of Linux_Processor (target) are returned. If the object path of a Linux_Processor instance (source) was given to the provider, the object path / instance of Linux_ComputerSystem (target) is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path(s) / instance(s) of `Linux_CSProcessor`. Their number depends on the source object. If the object path of the `Linux_ComputerSystem` instance (source) was given to the provider, as many object paths / instances of `Linux_CSProcessor` are returned, as `Linux_Processor` (target) instances exist. If the object path of a `Linux_Processor` instance (source) was given to the provider, one object path / instance of `Linux_CSProcessor` is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

3.4 Tool Library

The tool library is called `cmpiOSBase_Common`. It consists of two modules. The one containing all CIM dependent functions is `cmpiOSBase_Common`. The other one containing the resource access dependent functions is `OSBase_Common`.

3.4.1 CIM dependent Layer

The CIM dependent layer externalizes the following static variables and functions.

3.4.1.1 Static Variables

```
char * CSCreationClassName = "Linux_ComputerSystem";
char * OSCreationClassName = "Linux_OperatingSystem";
```

```
unsigned char CMPI_true = 1;
unsigned char CMPI_false = 0;
```

3.4.1.2 Tool Functions

```
void _check_system_key_value_pairs(
    CMPIBroker * _broker, CMPIObjectPath * ref, char * creationClassName,
    char * className, CMPIStatus * rc )
```

This function is useful for providers implementing classes that have a weak dependency to ComputerSystem. It checks the correctness of the key-value-pairs in `ref` for the weak properties `CSCreationClassName` and `CSName`. It depends on the subclass, how the weak properties are named. Therefore the property names are submitted via `creationClassName` and `className`.

In addition the function is prepared to test the OperatingSystem's key-value-pairs of `CreationClassName` and `Name`. The reason is both classes call the same function to get the `Name` value.

The result is reported through the `rc` parameter. If successful, it returns `CMPI_RC_OK`. If not, it returns `CMPI_RC_ERR_FAILED` or `CMPI_RC_ERR_NOT_FOUND` and a human readable error message.

3.4.1.3 Generic 1-to-N Association Provider Functions

The instance and association interface of association classes, where the referenced classes stay in a 1-to-n relation to each other, are implemented in a generic manner. The major advantage of this approach is the high degree of reuse and the portability. The utility functions are platform independent and dynamic configurable.

The input parameters in the utility functions have the following meaning :

- `_ClassName` : class name of the association
- `_RefLeftClass` : class name of the left reference
- `_RefRightClass` : class name of the right reference
- `_RefLeft` : property name of the left reference
- `_RefRight` : property name of the right reference

```
int _assoc_create_inst_1toN(
    CMPIBroker * _broker, CMPIContext * ctx, CMPIResult * rslt,
    CMPIObjectPath * cop, char * _ClassName,
    char * _RefLeftClass, char * _RefRightClass,
    char * _RefLeft, char * _RefRight,
    int left, int inst,
    CMPIStatus * rc)
```

This function is the generic implementation of `EnumInstances()` and `EnumInstanceNames()` for 1-to-N association classes. It delivers all possible object path(s) / instance(s) of the requested association.

The function is configured by the constellation of the `left` and `inst` parameters to each other. The `left` parameter identifies the class reference representing the single instance end. The values and their effect are :

- (`left = 0`) -> right class is single instance end
- (`left = 1`) -> left class is single instance end
- (`inst = 0`) -> `EnumInstanceNames()`
- (`inst = 1`) -> `EnumInstances()`

If successful, the function returns with 0. In the case of any error, it returns with -1. The error is reported through the `rc` parameter. It contains the error code and a human readable error message.

```
CMPIInstance * _assoc_get_inst(
    CMPIBroker * _broker, CMPIContext * ctx, CMPIObjectPath * cop,
    char * _ClassName, char * _RefLeft, char * _RefRight,
    CMPIStatus * rc )
```

This function is the generic implementation of the `GetInstance()` call for (only) association classes.

If successful, the function returns the requested instance. In the case of any error, it returns with `NULL`. The error is reported through the `rc` parameter. It contains the error code and a human readable error message.

```
int _assoc_create_refs_1toN(
    CMPIBroker * _broker, CMPIContext * ctx, CMPIResult * rslt,
    CMPIObjectPath * ref, char * _ClassName,
    char * _RefLeftClass, char * _RefRightClass,
    char * _RefLeft, char * _RefRight,
    int inst, int associators,
    CMPIStatus * rc)
```

This function is the generic implementation of the association interface (`Associator()`, `AssociatorNames()`, `References()`, `ReferenceNames()`) for 1-toN association classes. It delivers the object path(s) / instance(s) of the target class or the association itself. For the behavior of the association interface calls see chapter 2.3.2.2 Association Provider.

The function is configured by the constellation of the `inst` and `associators` parameters to each other. The values and their effect are :

- (`inst = 0`) && (`associators = 0`) -> `ReferenceNames()`
- (`inst = 1`) && (`associators = 0`) -> `References()`
- (`inst = 0`) && (`associators = 1`) -> `AssociatorNames()`
- (`inst = 1`) && (`associators = 1`) -> `Associators()`

If successful, the function returns with 0. In the case of any error, it returns with -1. The error is reported through the `rc` parameter. It contains the error code and a human readable error message.

```
char * _assoc_targetClass_Name(
    CMPIBroker * _broker, CMPIObjectPath * ref, char * _RefLeftClass,
    char * _RefRightClass, CMPIStatus * rc)
```

This function receives the source object path `ref` and returns the name of the target class. The target class can either be the left or the right class reference. If `ref` is not an instance of one of the two classes or a subclass of one of them the function returns `NULL`.

```
CMPIObjectPath * _assoc_targetClass_OP(
    CMPIBroker * _broker, CMPIObjectPath * ref, char * _RefLeftClass,
    char * _RefRightClass, CMPIStatus * rc )
```

This function is based on `_assoc_targetClass_Name()`. But it returns the object path of the target class.

```
int _assoc_check_parameter_const(
    CMPIBroker * _broker, CMPIObjectPath * cop,
    char * _RefLeft, char * _RefRight,
    char * _RefLeftClass, char * _RefRightClass,
    char * resultClass, char * role, char * resultRole,
    CMPIStatus * rc )
```

A client has the possibility to reduce the returned amount of objects by an intelligent input parameter configuration. This function implements the algorithm to check the different parameter constellations and indicates if the provider is responsible for the request.

If the association provider is responsible for the incoming call, the function returns 1. If not responsible it returns with 0.

3.4.2 Resource Access Layer

The resource access layer externalizes the following functions.

```
#define _OSBASE_TRACE( int level, char * output )
```

This macro defines the tracing facility. The level is set externally in the runtime environment.

```
char * get_system_name()
```

This function returns the full qualified and unique name of the system. The syntax is <hostname>.<domain>. The function calls `gethostname()` and analyzes the returned string for completeness. If the string does not have the domain defined at this time `dnsdomainname()` is called to get the domain.

```
char * get_os_name()
```

This function returns the full qualified and unique name of the operating system. Because the operating system name and release are not unique in an enterprise environment, the syntax is <hostname>.<domain>.

```
signed short get_os_timezone()
```

This function calls the `gettimeofday()` function. Its return value offers the current time and the time zone. The time zone is returned as the number of minutes west of Greenwich Mean Time. This value is multiplied by -1 to get the CIM conform value. The function returns the minutes away from Greenwich Mean Time. Eastern are positive and west of it are negative values.

```
unsigned long _get_os_boottime()
```

This function parses the `/proc/stat` file for the `btime` value, which contains the point in time, the system was booted last.

In the case of success, the function returns the time of the last start up in seconds passed since 00:00:00 01/01/1970. Else it returns 0.

```
void _cat_timezone( char * str, signed short zone )
```

This function adds the time zone `zone` to the end of the string `str`.

```
int runcommand( const char * cmd, char ** in, char *** out, char *** err )
```

This function executes shell commands. The commands can be submitted by the string `cmd` or the input file description `in`. The result of the command is returned via the `out` and `err` parameter.

In the case of success, the function returns 0. In the case of any error, it returns the error code of the executed command.

```
void freeresultbuf( char ** buf )
```

Deallocates the memory of the buffer `buf`. The buffer is organized as a array of `char *` to string values.

```
char ** line_to_array( char * line, int c )
```

Converts the line `line` to an array of `char *`, where each `char *` points to a string. The separator to identify a new string is identified by `c`.

If successful, the function returns the pointer to the array. If failed, it returns NULL.

```
int get_system_parameter( char * path, char * filename,  
                          char * buf, int bufsize )
```

This function reads the content of the file `path/filename` and stores it in the buffer `buf`. The maximum number of bytes that can be read into the buffer is submitted by `bufsize`.

In the case of success, the function returns the number of bytes read. In the case of an error, it returns -1.

```
int set_system_parameter( char * path, char * filename, char * buf )
```

This function writes the content of buffer `buf` to the file `path/filename`.

In the case of success, the function returns the number of bytes written. In the case of any error, it returns -1.

4 CMPI-FSVOL

4.1 Package Characteristics

4.1.1 Supported Platforms

- xSeries
- pSeries
- zSeries

4.1.2 Dependencies

The providers depend on a CIMOM, supporting the Common Manageability Programming Interface (CMPI).

The providers require the cmpi-base package installed on the system.

4.2 CIM Schema

4.2.1 DMTF CIM Schema Dependency

The Linux schema requires the CIM Schema version 2.7 or higher. The necessary sub schema parts are :

- Core
- System

4.2.2 Linux Schema Definition

The cmpi-fsvol package implements the classes, needed for file system management. This includes support for different types of file systems.

The following UML diagram shows the Linux File System Model and the CIM Inheritance tree for the classes. The inheritance tree of the associations is not shown. The red marked properties are already deprecated and will be removed from the CIM model at a later point in time.

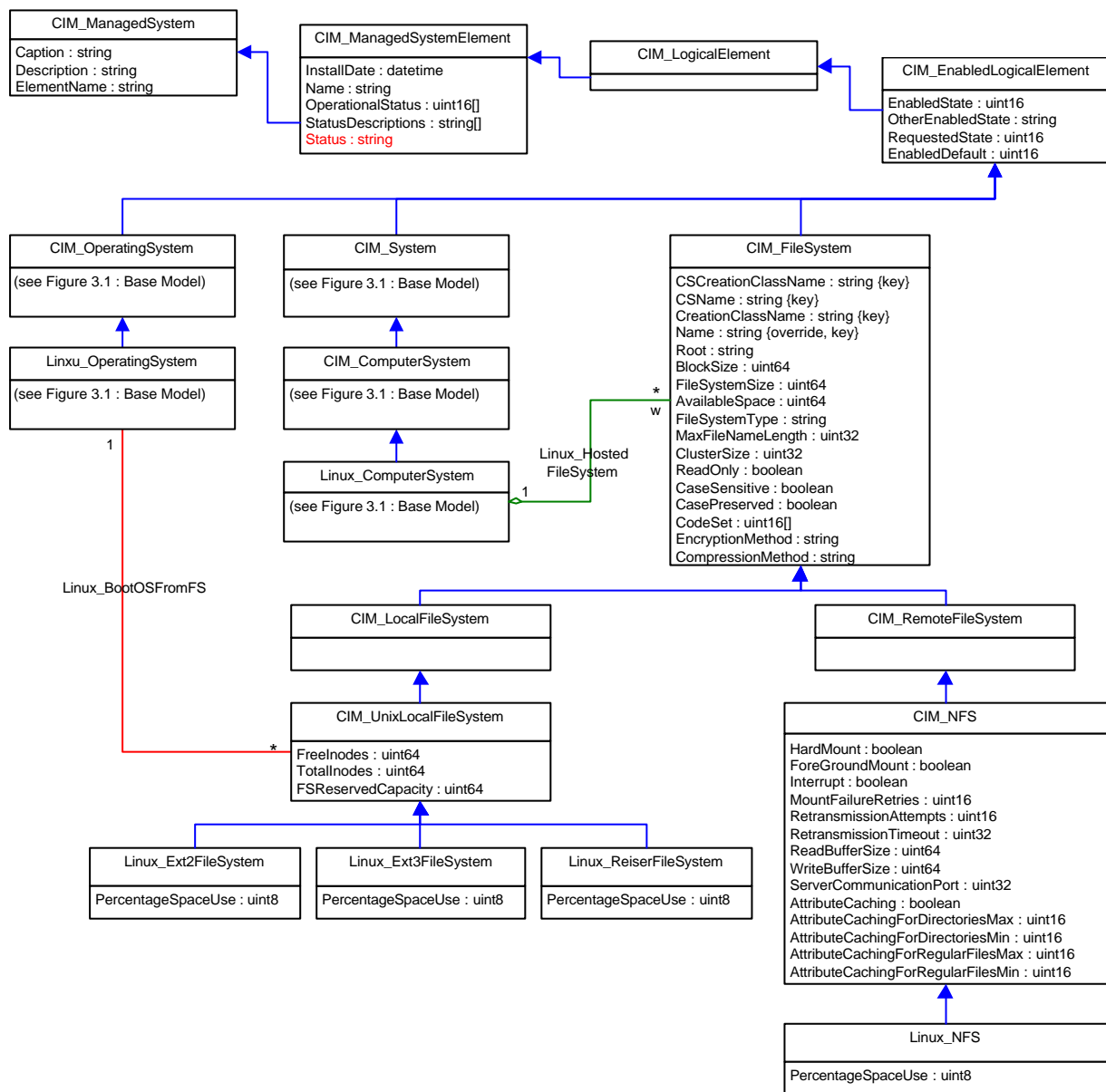


Figure 4.1 : File System Model

4.2.2.1 Linux_Ext2FileSystem, Linux_Ext3FileSystem, Linux_ReiserFileSystem

The class `Linux_Ext2FileSystem` represents the local file systems of type `ext2`. The class `Linux_Ext3FileSystem` represents the local file systems of type `ext3`. The class `Linux_ReiserFileSystem` represents the local file systems of type `reiserfs`.

The CIM inheritance tree can be found in figure 4.1 File System Model. The table lists all inherited and class specific properties. The three classes are implemented by one provider (see chapter 4.3.1 `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem`, `Linux_ReiserFileSystem`) and do not define different properties in the file system type specific subclasses. Therefore the following table is shown only once for all classes. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the file system.	Y
InstallDate	datetime	Indicates the install date of the file system.	N
Name	string key	Contains the full qualified partition path as unique identifier of the file system.	Y
OperationalStatus	uint16[]	Indicates the current status of the file system (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the file system	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the file system. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the file system	Y
EnabledDefault	uint16	The administrator's default / startup state of the file system	Y
CSCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
CSName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
Root	string	Contains the full qualified path of the mount point.	Y
BlockSize	uint64	Contains the file system's block size to store and retrieve data.	Y
FileSystemSize	uint64	Contains the total size in bytes.	Y
AvailableSpace	uint64	Contains the free space in bytes.	Y
FileSystemType	string	Contains the type of the file system (supported are ext2, ext3 and reiserfs).	Y
MaxFileNameLength	uint32	Contains the max length of a file name within the file system.	Y
ClusterSize	uint32	The min file allocation size.	N
ReadOnly	boolean	Indicates if the file system is mounted read only.	Y
CaseSensitive	boolean	Indicates that case sensitive file names are supported.	N
CasePreserved	boolean	Indicates that the case of file names are preserved.	N
CodeSet	uint16[]	Defines the supported character sets or encoding.	N
EncryptionMethod	string	A free form string indicating the algorithm or tool used to encrypt the file system.	N
CompressionMethod	string	A free form string indicating the algorithm or tool used to compress the file system.	N
FreeInodes	uint64	Number of free inodes.	N
TotalInodes	uint64	Total number of inodes.	N
FSReservedCapacity	uint64	The reserve data capacity of the file system in bytes.	N
PercentageSpaceUse	uint8	Space usage of file system in percent.	Y

4.2.2.2 Linux_NFS

The class Linux_NFS represents the mounted NFS file systems.

The CIM inheritance tree can be found in figure 4.1 File System Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the NFS file system.	Y
Description	string	Textual description of the NFS file system.	Y
ElementName	string	Contains a user-friendly name of the NFS file system.	Y
InstallDate	datetime	Indicates the install date of the NFS file system.	N

Name	string key	Contains the full qualified and unique server path of the nfs file system (server:"mount point on server")	Y
OperationalStatus	uint16[]	Indicates the current status of the NFS file system (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the NFS file system.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the NFS file system. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the NFS file system.	Y
EnabledDefault	uint16	The administrator's default / startup state of the NFS file system.	Y
CSCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
CSName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
Root	string	Contains the full qualified path of the local mount point.	Y
BlockSize	uint64	Contains the file system's block size to store and retrieve data.	Y
FileSystemSize	uint64	Contains the total size in bytes.	Y
AvailableSpace	uint64	Contains the free space in bytes.	Y
FileSystemType	string	Contains the type of the file system (nfs).	Y
MaxFileNameLength	uint32	Contains the max length of a file name within the file system.	Y
ClusterSize	uint32	The min file allocation size.	N
ReadOnly	boolean	Indicates if the file system is mounted read only.	Y
CaseSensitive	boolean	Indicates that case sensitive file names are supported.	N
CasePreserved	boolean	Indicates that the case of file names are preserved.	N
CodeSet	uint16[]	Defines the supported character sets or encoding.	N
EncryptionMethod	string	A free form string indicating the algorithm or tool used to encrypt the file system.	N
CompressionMethod	string	A free form string indicating the algorithm or tool used to compress the file system.	N
HardMount	boolean	TRUE : once the file system is mounted, NFS requests are retried until the hosting system responds. FALSE : once the file system is mounted, an error is returned if the hosting system does not respond.	N
ForeGroundMount	boolean	TRUE : retries are performed in the foreground. FALSE : if the first mount attempt fails, retries are performed in the background.	N
Interrupt	boolean	TRUE : interrupts are permitted for hard mounts. FALSE : interrupts are ignored for hard mounts.	N
MountFailureRetries	uint16	Max number of mount failure retries allowed.	N
RetransmissionAttempts	uint16	Max number of NFS retransmissions allowed.	N
RetransmissionTimeout	uint32	NFS timeout in tenths of a second.	N
ReadBufferSize	uint32	Read buffer size in bytes.	N
WriteBufferSize	uint64	Write buffer size in bytes.	N
ServerCommunicationPort	uint64	The remote computer system's (NFS File Server's) UDP port number.	N
AttributeCaching	boolean	Control attribute caching is enabled / disabled.	N
AttributeCachingForDirectoriesMax	uint16	Max number of seconds that cached attributes are held after directory update.	N
AttributeCachingForDirectoriesMin	uint16	Minimum number of seconds that cached attributes are held after directory update.	N
AttributeCachingForRegularFilesMax	uint16	Max number of seconds that cached attributes are held after file modification.	N
AttributeCachingForRegularFilesMin	uint16	Min number of seconds that cached attributes are held after file modification.	N

4.2.2.3 Linux_HostedFileSystem

The aggregation Linux_HostedFileSystem indicates the file system(s) that is (are) part of the computer system container.

The CIM inheritance tree is Linux_HostedFileSystem => CIM_HostedFileSystem => CIM_SystemComponent => CIM_Component.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
GroupComponent	REF key	Linux_ComputerSystem	The computer system.
PartComponent	REF key	CIM_FileSystem	The file system(s) that is (are) part of the system and hosted on it.

4.2.2.4 Linux_BootOSFromFS

The association Linux_BootOSFromFS is a link between the operating system and the file system(s) from which this operating system is loaded.

The CIM inheritance tree is Linux_BootOSFromFS => CIM_BootOSFromFS => CIM_Dependency.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
Antecedent	REF key	CIM_UnixLocalFileSystem	The file system(s) from which the operating system is loaded.
Dependent	REF key	Linux_OperatingSystem	The operating system.

4.3 Management Instrumentation

4.3.1 Linux_Ext2FileSystem, Linux_Ext3FileSystem, Linux_ReiserFileSystem

The classes Linux_Ext2FileSystem, Linux_Ext3FileSystem and Linux_ReiserFileSystem are implemented by the provider `cmpiOSBase_LocalFileSystemProvider`. The provider supports the instance interface. It consists of three modules (`cmpiOSBase_LocalFileSystemProvider`, `cmpiOSBase_LocalFileSystem`, `OSBase_LocalFileSystem`) and depends on the tool libraries `cmpiOSBase_Common` (see chapter 3.4 Tool Library) and `cmpiOSBase_CommonFsvol` (see chapter 4.4 Tool Library).

4.3.1.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object path(s) / instance(s) of Linux_Ext2FileSystem, Linux_Ext3FileSystem and / or Linux_ReiserFileSystem as entries have

been found in the `/etc/fstab` and `/etc/mtab` files. The algorithm is able to avoid duplicates. If only one of the supported classes is requested, the provider returns only object path(s) / instance(s) of the specified class.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem` or `Linux_ReiserFileSystem` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

4.3.1.2 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s `SetInstance()` function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Ext2 / Ext3 / Reiser local file system
Description	string	no	Represents the file store controlled by a computer system through local means.
ElementName	string	no	same value as “Root”
Name	string key	no	unique file system name, for example /dev/hda1; readsentry in /etc/fstab or /etc/mtab by getmntent(); entry mnt_fsname of returned struct mntent
Status	string	no	NULL
EnabledState	uint16	no	if the file system is currently mounted : 2 (“Enabled”); else 3 (“Disabled”)
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	same value as property EnabledState

EnabledDefault	uint16	no	<i>if the file system is listed in /etc/fstab and option set to automatic mount at start up : 2 ("Enabled"); else 3 ("Disabled")</i>
CSCreationClassName	string key	no	CreationClassName of Linux_ComputerSystem
CSName	string key	no	Name of Linux_ComputerSystem
CreationClassName	string key	no	Linux_Ext2FileSystem or Linux_Ext3FileSystem or Linux_ReiserFileSystem
Root	string	no	<i>read the entry in /etc/fstab or /etc/mtab by getmntent(); entry mnt_dir of returned struct mntent.</i>
BlockSize	uint64	no	<i>call statfs() with mnt_dir; entry f_bsize of returned struct statfs</i>
FileSystemSize	uint64	no	<i>call statfs() with mnt_dir; (f_blocks * f_bsize) of returned struct statfs</i>
AvailableSpace	uint64	no	<i>call statfs() with mnt_dir; (f_bavail * f_bsize) of returned struct statfs</i>
FileSystemType	string	no	<i>read the entry in /etc/fstab or /etc/mtab by getmntent(); entry mnt_type of returned struct mntent.</i>
MaxFileNameLength	uint32	no	<i>call statfs() with mnt_dir; entry f_namelen of returned struct statfs</i>
ReadOnly	boolean	no	<i>call hasmntopt() with mntent; scans entry mnt_opts of struct mntent for "ro" (read only) flag</i>
PercentageSpaceUse	uint8	no	<i>call statfs() with mnt_dir; ((f_bfree*100)/f_blocks) of returned struct statfs</i>

4.3.2 Linux_NFS

The classes Linux_NFS is implemented by the provider `cmptiOSBase_NFSProvider`. The provider supports the instance interface. It consists of three modules (`cmptiOSBase_NFSProvider`, `cmptiOSBase_NFS`, `OSBase_NFS`) and depends on the tool libraries `cmptiOSBase_Common` (see chapter 3.4 Tool Library) and `cmptiOSBase_CommonFsvol` (see chapter 4.4 Tool Library).

4.3.2.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object path(s) / instance(s) of Linux_NFS as entries have been found in the `/etc/fstab` and `/etc/mtab` files. The algorithm is able to avoid duplicates.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of Linux_NFS defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **CreateInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **DeleteInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **ExecQuery()**

CMPI_RC_ERR_NOT_SUPPORTED

4.3.2.2 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s SetInstance() function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	NFS File System
Description	string	no	The class represents the nfs mounts of the system.
ElementName	string	no	<i>same value as “Root”</i>
Name	string key	no	<i>unique file system name, for example nfsserver:/home/pub; reads entry in /etc/fstab or /etc/mtab by getmntent(); entry mnt_fsname of returned struct mntent</i>
Status	string	no	NULL
EnabledState	uint16	no	<i>if the file system is currently mounted : 2 (“Enabled”); else 3 (“Disabled”)</i>
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as property EnabledState</i>
EnabledDefault	uint16	no	<i>if the file system is listed in /etc/fstab and option set to automatic mount at start up : 2 (“Enabled”); else 3 (“Disabled”)</i>
CSCreationClassName	string key	no	<i>CreationClassName of Linux_ComputerSystem</i>
CSName	string key	no	<i>Name of Linux_ComputerSystem</i>
CreationClassName	string key	no	<i>Linux_NFS</i>
Root	string	no	<i>read the entry in /etc/fstab or /etc/mtab by getmntent(); entry mnt_dir of returned struct mntent.</i>
BlockSize	uint64	no	<i>call statfs() with mnt_dir; entry f_bsize of returned struct statfs</i>
FileSystemSize	uint64	no	<i>call statfs() with mnt_dir; (f_blocks * f_bsize) of returned struct statfs</i>
AvailableSpace	uint64	no	<i>call statfs() with mnt_dir; (f_bavail * f_bsize) of returned struct statfs</i>
FileSystemType	string	no	<i>read the entry in /etc/fstab or /etc/mtab by getmntent(); entry mnt_type of returned struct mntent.</i>
MaxFileNameLength	uint32	no	<i>call statfs() with mnt_dir; entry f_namelen of returned struct statfs</i>
ReadOnly	boolean	no	<i>call hasmntopt() with mntent; scans entry mnt_opts of struct mntent for “ro” (read only) flag</i>
PercentageSpaceUse	uint8	no	<i>call statfs() with mnt_dir; ((f_bfree*100) / f_blocks) of returned struct statfs</i>

4.3.3 Linux_HostedFileSystem

The aggregation `Linux_HostedFileSystem` is implemented by the provider `cmpiOSBase_HostedFileSystemProvider`. The provider supports the instance and association interfaces. It consists of one module (`cmpiOSBase_HostedFileSystemProvider`) and depends on the tool library `cmpiOSBase_Common` (see chapter 3.4 Tool Library).

4.3.3.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of `Linux_HostedFileSystem`, as file systems (`Linux_Ext2FileSystem`, `Linux_Ext3FileSystem`, `Linux_ReiserFileSystem`, `Linux_NFS`) are hosted by the computer system.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_HostedFileSystem` defined by the input object path.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, `GetInstance()` returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

4.3.3.2 Association Interface

- **AssociationCleanup()**

No functionality.

- **AssociatorNames(), Associators()**

In the case of success, it returns the object path(s) / instance(s) of the target class. Their number depends on the source object. If the object path of the `Linux_ComputerSystem` instance (source) was given to the provider, all object paths / instances of `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem`, `Linux_ReiserFileSystem` and `Linux_NFS` (target) are returned. If the object path of a `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem`, `Linux_ReiserFileSystem` or `Linux_NFS` instance (source) was given to the provider, the object path / instance of `Linux_ComputerSystem` (target) is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path(s) / instance(s) of `Linux_HostedFileSystem`. Their number depends on the source object. If the object path of the `Linux_ComputerSystem` instance (source) was given to the provider, as many object paths / instances of `Linux_HostedFileSystem` are returned, as `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem`, `Linux_ReiserFileSystem` and `Linux_NFS` (target) instances exist. If the object path of a `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem`, `Linux_ReiserFileSystem` or `Linux_NFS` instance (source) was given to the provider, one object path / instance of `Linux_HostedFileSystem` is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

4.3.4 Linux_BootOSFromFS

The association `Linux_BootOSFromFS` is implemented by the provider `cmpioSBase_BootOSFromFSProvider`. The provider supports the instance and association interfaces. It consists of two modules (`cmpioSBase_BootOSFromFSProvider`, `cmpioSBase_BootOSFromFS`) and depends on the tool libraries `cmpioSBase_Common` (see chapter 3.4 Tool Library) and `cmpioSBase_CommonFsvol` (see chapter 4.4 Tool Library).

4.3.4.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the object path(s) / instance(s) of Linux_BootOSFromFS, as local file systems (Linux_Ext2FileSystem, Linux_Ext3FileSystem or Linux_ReiserFileSystem) are responsible to load the operating system at start up.

In the case of failure, the function returns with CMPI_RC_ERR_FAILED and a human readable error message.

The EnumInstances() input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of Linux_BootOSFromFS defined by the input object path.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with CMPI_RC_ERR_NOT_FOUND. In the case of any other failure, GetInstance() returns with CMPI_RC_ERR_FAILED and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **CreateInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **DeleteInstance()**

CMPI_RC_ERR_NOT_SUPPORTED

- **ExecQuery()**

CMPI_RC_ERR_NOT_SUPPORTED

4.3.4.2 Association Interface

- **AssociationCleanup()**

No functionality.

- **AssociatorNames(), Associators()**

In the case of success, it returns the object path(s) / instance(s) of the target class. Their number depends on the source object. If the object path of the Linux_OperatingSystem instance (source) was given to the provider, the object paths / instances of Linux_Ext2FileSystem, Linux_Ext3FileSystem and / or Linux_ReiserFileSystem (target) are returned, that are responsible for the correct loading of the operating system at start up. If the object path of a Linux_Ext2FileSystem, Linux_Ext3FileSystem or Linux_ReiserFileSystem instance (source) was given to the provider, the object path / instance of Linux_OperatingSystem (target) is only then

returned, if the given source object is a local file system, that is responsible for loading the operating system.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path(s) / instance(s) of `Linux_BootOSFromFS`. Their number depends on the source object. If the object path of the `Linux_OperatingSystem` instance (source) was given to the provider, as many object paths / instances of `Linux_BootOSFromFS` are returned, as `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem` and / or `Linux_ReiserFileSystem` (target) instances exist and are responsible for the correct loading of the operating system. If the object path of a `Linux_Ext2FileSystem`, `Linux_Ext3FileSystem` or `Linux_ReiserFileSystem` instance (source) was given to the provider, the object path / instance of `Linux_HostedFileSystem` is only returned, if the given source object is a local file system, that is responsible for loading the operating system.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

4.4 Tool Library

The tool library is called `cmpiOSBase_CommonFsvol`. It consists of two modules. The one, containing all CIM dependent functions is `cmpiOSBase_CommonFsvol`. The other one, containing the resource access dependent functions is `OSBase_CommonFsvol`.

4.4.1 CIM dependent Layer

The CIM dependent layer externalizes the following function.

```
int get_fs_of_dir( char * dir, char ** fscsname, char ** fsname)
```

This function indicates the file system where the directory `dir` resides on and returns the key-value-pairs of the file system specific properties `CreationClassName` and `Name`. This utility function is useful for providers, implementing classes with a weak dependency to file systems, for example `CIM_LogicalFile`.

If successful, `fscsname` and `fsname` point to the values for `CreationClassName` and `Name` of the corresponding file system instance and the function returns with 0. In the case of any error, `fscsname` and `fsname` are NULL pointer and it returns with -1.

4.4.2 Resource Access Layer

The resource access layer externalizes the following definitions and functions.

```
#define PROC_MOUNTS      "/proc/ mounts"
#define ETC_MTAB         "/etc/ mtab"
#define ETC_FSTAB       "/etc/ fstab"
#define PROC_DEVICES    "/proc/ devices"
#define PROC_PARTITIONS "/proc/ partitions"
```

```
struct mntlist {
    struct mntent * me;
    struct mntlist * next;
}
```

```
int enum_all_fs ( struct mntlist ** lptr )
```

This function enumerates all file systems known to the system, independent of their file system type. The enumeration is based on the `ETC_FSTAB` and `ETC_MTAB` files. If `ETC_MTAB` is not available the `PROC_MOUNTS` file is taken instead. Labels are replaced by the native device path. Duplicates are filtered and thereby avoided.

If successful the function returns 0 and `lptr` points to the starting point of the chained list of file system entries. In the case of an error, it returns with -1 and `lptr` is a NULL pointer.

```
int get_fs_data( struct mntent ** sptr, char * name )
```

This function collects the file system data of the entry requested by `name`.

If successful the function returns 0 and `sptr` points to the data. In the case of an error, it returns with -1 and `sptr` is a NULL pointer.

```
void free_mntlist( struct mntlist * lptr )
```

Deallocates the storage of the chained list `lptr` of file system entries.

```
void free_mntent( struct mntent * sptr )
```

Deallocates the storage of a file system entry.

```
unsigned char fs_mount_status( char * fsname )
```

This function indicates the current mount status of the file system `fsname` (e.g. `/dev/hda1`). If the file system is currently mounted, the function returns 1. If currently not mounted, it returns 0.

```
unsigned char fs_default_mount_status( char * fsname )
```

This function indicates the default option for the file system `fsname` at start up. If the file system is mounted automatically at start up, the function returns 1. If not mounted at start up, it returns 0.

5 CMPI-NETWORK

5.1 Package Characteristics

5.1.1 Supported Platforms

- xSeries
- pSeries
- zSeries

5.1.2 Dependencies

The providers depend on a CIMOM, supporting the Common Manageability Programming Interface (CMPI).

The providers require the cmpi-base package installed on the system.

5.2 CIM Schema

5.2.1 DMTF CIM Schema Dependency

The Linux schema requires the CIM Schema version 2.7 or higher. The necessary sub schema parts are :

- Core
- System
- Device
- Network

5.2.2 Linux Schema Definition

The cmpi-network package implements the classes, needed for network management. This includes support for different types of network ports and IP protocol endpoints.

The following UML diagram shows the Linux Network Model and the CIM Inheritance tree for the classes. The inheritance tree of the associations is not shown. The red marked properties are already deprecated and will be removed from the CIM model at a later point in time.

5.2.2.1 Linux_LocalLoopbackPort

The class Linux_LocalLoopbackPort is the logical representation of a Linux local loopback device.

The CIM inheritance tree can be found in figure 5.1 Network Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the port.	Y
InstallDate	datetime	Indicates the install date of the port.	N
Name	string	Contains a user-friendly name of the port.	Y
OperationalStatus	uint16[]	Indicates the current status of the port (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the port.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the port. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the port.	Y
EnabledDefault	uint16	The administrator's default / startup state of the port.	Y
SystemCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
SystemName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
DeviceID	string Key	Contains the uniqueport name.	Y
Availability	uint16	Primary availability and status of the port (value map).	N
AdditionalAvailability	uint16[]	Additional information about the availability of the port.	N
StatusInfo	uint16	Indicates the enabled / disabled state of the port.	N
LastErrorCode	uint32	Last error code reported by the port.	N
ErrorDescription	string	Textual description of the errorcode.	N
ErrorCleared	boolean	Last error is cleared now.	N
OtherIdentifyingInfo	string[]	Additional information to identify the port.	N
IdentifyingDescriptions	string[]	Textual description of the OtherIdentifyingInfo value(s).	N
MaxQuiesceTime	uint64	Max time in milliseconds the port can run in 'Quiesced' state.	N
PowerOnHours	uint64	Number of hours the port has been powered, since last power cycle.	N
TotalPowerOnHours	uint64	Total number of hours the port has been powered.	N
PowerManagementSupported	boolean	Port can be power managed (value map).	N
PowerManagementCapabilities	uint16	Indicates the power management capabilities of the port.	N
Speed	uint64	The current bandwidth of the port in Bits per Second.	Y
MaxSpeed	uint64	The maximum bandwidth of the port in Bits per Second.	Y
PortType	uint16	Contains the type of the port (value map).	N
OtherNetworkPortType	string	Indicates the port type, if PortType is set to 1 ('Other').	N
PortNumber	uint16	Contains the number, if port is numbered relative to the network element.	N
LinkTechnology	uint16	Contains the type of the underlying link technology (value map).	Y
OtherLinkTechnology	string	Contains the type description, if LinkTechnology is set to 1 ('Other').	Y
PermanentAddress	string	Contains the hard coded port address.	N
NetworkAddresses	string[]	Indicates the network addresses for this port.	N
FullDuplex	boolean	Port can operate in full duplex mode.	N
AutoSense	boolean	Port can automatically determine characteristics of the network.	N
SupportedMaximumTransmissionUnit	uint64	Contains the supported maximum transmission unit (MTU).	N
ActiveMaximumTransmissionUnit	uint64	Contains the current supported maximum transmission unit (MTU).	N

5.2.2.2 Linux_EthernetPort

The class Linux_EthernetPort is the logical representation of an ethernet network device.

The CIM inheritance tree can be found in figure 5.1 Network Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the port.	Y
InstallDate	datetime	Indicates the install date of the port.	N
Name	string	Contains a user-friendly name of the port.	Y
OperationalStatus	uint16[]	Indicates the current status of the port (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the port.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the port. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the port.	Y
EnabledDefault	uint16	The administrator's default / startup state of the port.	Y
SystemCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
SystemName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
DeviceID	string Key	Contains the unique port name.	Y
Availability	uint16	Primary availability and status of the port (value map).	N
AdditionalAvailability	uint16[]	Additional information about the availability of the port.	N
StatusInfo	uint16	Indicates the enabled / disabled state of the port.	N
LastErrorCode	uint32	Last error code reported by the port.	N
ErrorDescription	string	Textual description of the error code.	N
ErrorCleared	boolean	Last error is cleared now.	N
OtherIdentifyingInfo	string[]	Additional information to identify the port.	N
IdentifyingDescriptions	string[]	Textual description of the OtherIdentifyingInfo value(s).	N
MaxQuiesceTime	uint64	Max time in milliseconds the port can run in 'Quiesced' state.	N
PowerOnHours	uint64	Number of hours the port has been powered, since last power cycle.	N
TotalPowerOnHours	uint64	Total number of hours the port has been powered.	N
PowerManagementSupported	boolean	Port can be power managed (value map).	N
PowerManagementCapabilities	uint16	Indicates the power management capabilities of the port.	N
Speed	uint64	The current bandwidth of the port in Bits per Second.	Y
MaxSpeed	uint64	The maximum bandwidth of the port in Bits per Second.	Y
PortType	uint16	Contains the specific mode enabled for the port (value map).	N
OtherNetworkPortType	string	Indicates the port type, if PortType is set to 1 ('Other').	N
PortNumber	uint16	Contains the number, if port is numbered relative to the network element.	N
LinkTechnology	uint16	Contains the type of the underlying link technology (value map).	Y
OtherLinkTechnology	string	Contains the type description, if LinkTechnology is set to 1 ('Other').	Y
PermanentAddress	string	Contains the hard coded port address.	N
NetworkAddresses	string[]	Ethernet/802.3 MAC addresses formatted as twelve hexadecimal digits.	N
FullDuplex	boolean	Port can operate in full duplex mode.	N
AutoSense	boolean	Port can automatically determine characteristics of the network.	N
SupportedMaximumTransmissionUnit	uint64	Contains the supported maximum transmission unit (MTU).	N
ActiveMaximumTransmissionUnit	uint64	Contains the current supported maximum transmission unit (MTU).	N
MaxDataSize	uint32	Contains the max size of the transmitted / received INFO field.	N

Capabilities	uint16[]	Contains the port capabilities.	N
CapabilityDescriptions	string[]	Descriptions of the item(s) in the property Capabilities.	N
EnabledCapabilities	uint16[]	Specifies the enabled capabilities .	N
OtherEnabledCapabilities	string[]	Descriptions of the item(s) in the property EnabledCapabilities.	N

5.2.2.3 Linux_TokenRingPort

The class Linux_TokenRingPort is the logical representation of a token ring network device.

The CIM inheritance tree can be found in figure 5.1 Network Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the port.	Y
InstallDate	datetime	Indicates the install date of the port.	N
Name	string	Contains a user-friendly name of the port.	Y
OperationalStatus	uint16[]	Indicates the current status of the port (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the port.	Y
EnabledState	uint16	Indicate the current enabled / disabled state of the port. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the port.	Y
EnabledDefault	uint16	The administrator's default / startup state of the port.	Y
SystemCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
SystemName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
DeviceID	string Key	Contains the unique port name.	Y
Availability	uint16	Primary availability and status of the port (value map).	N
AdditionalAvailability	uint16[]	Additional information about the availability of the port.	N
StatusInfo	uint16	Indicates the enabled / disabled state of the port.	N
LastErrorCode	uint32	Last error code reported by the port.	N
ErrorDescription	string	Textual description of the error code.	N
ErrorCleared	boolean	Last error is cleared now.	N
OtherIdentifyingInfo	string[]	Additional information to identify the port.	N
IdentifyingDescriptions	string[]	Textual description of the OtherIdentifyingInfo value(s).	N
MaxQuiesceTime	uint64	Max time in milliseconds the port can run in 'Quiesced' state.	N
PowerOnHours	uint64	Number of hours the port has been powered, since last power cycle.	N
TotalPowerOnHours	uint64	Total number of hours the port has been powered.	N
PowerManagementSupported	boolean	Port can be power managed (value map).	N
PowerManagementCapabilities	uint16	Indicates the power management capabilities of the port.	N
Speed	uint64	The current bandwidth of the port in Bits per Second.	Y
MaxSpeed	uint64	The maximum bandwidth of the port in Bits per Second.	Y
PortType	uint16	Contains the specific mode enabled for the port (value map).	N
OtherNetworkPortType	string	Indicates the port type, if PortType is set to 1 ('Other').	N
PortNumber	uint16	Contains the number, if port is numbered relative to the network element.	N
LinkTechnology	uint16	Contains the type of the underlying link technology (value map).	Y
OtherLinkTechnology	string	Contains the type description, if LinkTechnology is set to 1 ('Other').	Y

PermanentAddress	string	Contains the hard coded port address.	N
NetworkAddresses	string[]	Token Ring/802.5 MAC addresses formatted as twelve hexadecimal digits.	N
FullDuplex	boolean	Port can operate in full duplex mode.	N
AutoSense	boolean	Port can automatically determine characteristics of the network.	N
SupportedMaximumTransmissionUnit	uint64	Contains the supported maximum transmission unit (MTU).	N
ActiveMaximumTransmissionUnit	uint64	Contains the current supported maximum transmission unit (MTU).	N
MaxDataSize	uint32	Contains the max size of the transmitted / received INFO field.	N
Capabilities	uint16[]	Contains the port capabilities.	N
CapabilityDescriptions	string[]	Descriptions of the item(s) in the property Capabilities.	N
EnabledCapabilities	uint16[]	Specifies the enabled capabilities.	N
OtherEnabledCapabilities	string[]	Descriptions of the item(s) in the property EnabledCapabilities.	N
RingStatus	uint32	Contains the current status (value map).	N
RingState	uint16	Contains the current entering / leaving state (value map).	N
RingOpenStatus	uint16	Indicates the success or failure reason for the most resent attempt to enter the ring (value map).	N
RingSpeed	uint16	Contains the ring's bandwidth in Bits per Second.	N

5.2.2.4 Linux_IPProtocolEndpoint

The class Linux_IPProtocolEndpoint represents IP communication points to send and receive data.

The CIM inheritance tree can be found in figure 5.1 Network Model. The table lists all inherited and class specific properties. The last column indicates if the property is implemented (Y) or not (N) by the provider.

NAME	TYPE	DESCRIPTION	I
Caption	string	Short description of the object.	Y
Description	string	Textual description of the object.	Y
ElementName	string	Contains a user-friendly name of the protocol endpoint.	Y
InstallDate	datetime	Indicates the install date of the protocol endpoint.	N
Name	string	Contains the unique port name with the prefix IPv4_.	Y
OperationalStatus	uint16[]	Indicates the current status of the protocol endpoint (value map).	N
StatusDescription	string[]	Textual descriptions to the OperationalStatus value(s).	N
Status	string	Describes the current status of the protocol endpoint.	N
EnabledState	uint16	Indicate the current enabled / disabled state of the protocol endpoint. The underlying value map is base for RequestedState and EnabledDefault too.	Y
OtherEnabledState	string	Indicates the enabled / disabled state, if EnabledState is set to 1 ('Other').	Y
RequestedState	uint16	Indicates the last requested / desired state of the protocol endpoint.	Y
EnabledDefault	uint16	The administrator's default / startup state of the protocol endpoint.	Y
SystemCreationClassName	string key	Contains the CreationClassName of the scoping computer system.	Y
SystemName	string key	Contains the Name of the scoping computer system.	Y
CreationClassName	string key	Indicates the name of the class itself.	Y
NameFormat	string	Contains the naming heuristic to make the property Name unique.	Y
ProtocolType	string	Defines the type of endpoint (value map).	Y
OtherTypeDescription	string	Textual description of the type, if ProtocolType is set to 1 ('Other').	Y
IPv4Address	string	Contains the IPv4 address represented by this protocol endpoint.	Y
IPv6Address	string	Contains the IPv6 address represented by this protocol endpoint.	Y
Address	string	Contains the IP address represented by this protocol endpoint.	N
SubnetMask	string	Contains the IPv4 mask.	Y
PrefixLength	uint8	Contains the IPv6 prefix length.	N
AddressType	uint16	Contains the format of the property Address (value map).	N
IPVersionSupport	uint16	Identifies the supported IP version (value map).	Y

5.2.2.5 Linux_CSNetworkPort

The aggregation Linux_CSNetworkPort indicates the network port(s) of the computer system container.

The CIM inheritance tree is Linux_CSNetworkPort => CIM_SystemDevice => CIM_SystemComponent => CIM_Component.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
GroupComponent	REF key	Linux_ComputerSystem	The computer system.
PartComponent	REF key	CIM_NetworkPort	The network port(s) that is (are) part of the system and hosted on it.

5.2.2.6 Linux_NetworkPortImplementsIPEndpoint

The association Linux_NetworkPortImplementsIPEndpoint makes the relation between one or more IP protocol endpoints and the network port, that implements the protocol endpoint, explicit.

The CIM inheritance tree is Linux_NetworkPortImplementsIPEndpoint => CIM_PortImplementsEndpoint => CIM_DeviceSAPImplementation => CIM_Dependency.

NAME	TYPE	REFERENCED CLASS	DESCRIPTION
Antecedent	REF key	CIM_NetworkPort	The logical network port represents the device behind the IP protocol endpoint.
Dependent	REF key	Linux_IPProtocolEndpoint	The IP protocol endpoint implemented on the logical network port.

5.3 Management Instrumentation

5.3.1 Linux_LocalLoopbackPort

The class Linux_LocalLoopbackPort is implemented by the provider cmpiOSBase_LocalLoopbackPortProvider. The provider supports the instance and method interface. It consists of two modules (cmpiOSBase_LocalLoopbackPortProvider, cmpiOSBase_LocalLoopbackPort) and depends on the tool libraries cmpiOSBase_Common (see chapter 3.4 Tool Library) and OSBase_CommonNetwork (see chapter 5.4 Tool Library).

5.3.1.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object path(s) / instance(s) of Linux_LocalLoopbackPort as entries with the prefix `lo`, indicating the type "Local Loopback", have been found in the `/proc/net/dev` file.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_LocalLoopbackPort` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

5.3.1.2 Method Support

The `method` interface is supported and part of the provider `cmplOSBase_LocalLoopbackPortProvider`.

- **MethodCleanup()**

No functionality.

- **InvokeMethod()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- o `uint32 SetPowerState([IN] uint16 PowerState, [IN] datetime Time)`

`CMPI_RC_ERR_NOT_SUPPORTED`

- o `uint32 Reset()`

`CMPI_RC_ERR_NOT_SUPPORTED`

- o uint32 EnableDevice([IN] boolean Enabled)
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 OnlineDevice([IN] boolean Online)
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 QuiesceDevice([IN] boolean Quiesce)
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 SaveProperties()
CMPI_RC_ERR_NOT_SUPPORTED
- o uint32 RestoreProperties()
CMPI_RC_ERR_NOT_SUPPORTED

5.3.1.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s SetInstance() function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Linux LocalLoopbackPort
Description	string	no	This class represents instances of available LocalLoopback Ports.
ElementName	string	no	<i>same value as key property “DeviceID”</i>
Name	string	no	<i>same value as key property “DeviceID”</i>
Status	string	no	OK
EnabledState	uint16	no	<i>if currently started : 2 (“Enabled”); if not started 3 (“Disabled”); else 0 (“Unknown”)</i>
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as property EnabledState</i>
EnabledDefault	uint16	no	2 (“Enabled”)
SystemCreationClassName	string key	no	<i>CreationClassName of Linux_ComputerSystem</i>
System Name	string key	no	<i>Name of Linux_ComputerSystem</i>
CreationClassName	string key	no	Linux_LocalLoopbackPort
DeviceID	string Key	no	<i>Contains the unique port name of the syntax : lo<ID>.</i>
Speed	uint64	no	0 <i>(it is not possible to find out the bandwidth of the local loopback)</i>
MaxSpeed	uint64	no	<i>same value as property Speed</i>
LinkTechnology	uint16	no	1
OtherLinkTechnology	string	no	Local Loopback

5.3.2 Linux_EthernetPort

The class `Linux_EthernetPort` is implemented by the provider `cmpiOSBase_EthernetPortProvider`. The provider supports the instance and method interface. It consists of two modules (`cmpiOSBase_EthernetPortProvider`, `cmpiOSBase_EthernetPort`) and depends on the tool libraries `cmpiOSBase_Common` (see chapter 3.4 Tool Library) and `OSBase_CommonNetwork` (see chapter 5.4 Tool Library).

5.3.2.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object path(s) / instance(s) of `Linux_EthernetPort` as entries with the prefix `eth`, indicating the type "Ethernet", have been found in the `/proc/net/dev` file.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_EthernetPort` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

5.3.2.2 Method Support

The method interface is supported and part of the provider `cmptOSBase_EthernetPortProvider`.

- **MethodCleanup()**

No functionality.

- **InvokeMethod()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- o `uint32 SetPowerState([IN] uint16 PowerState, [IN] datetime Time)`
`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 Reset()`
`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 EnableDevice([IN] boolean Enabled)`
`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 OnlineDevice([IN] boolean Online)`
`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 QuiesceDevice([IN] boolean Quiesce)`
`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 SaveProperties()`
`CMPI_RC_ERR_NOT_SUPPORTED`
- o `uint32 RestoreProperties()`
`CMPI_RC_ERR_NOT_SUPPORTED`

5.3.2.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s `SetInstance()` function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Linux EthernetPort
Description	string	no	This class represents instances of available Ethernet Ports.
ElementName	string	no	<i>same value as key property “DeviceID”</i>
Name	string	no	<i>same value as key property “DeviceID”</i>
Status	string	no	OK

EnabledState	uint16	no	<i>if currently started : 2 ("Enabled"); if not started 3 ("Disabled"); else 0 ("Unknown")</i>
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as property EnabledState</i>
EnabledDefault	uint16	no	2 ("Enabled")
SystemCreationClassName	string key	no	CreationClassName of Linux_ComputerSystem
System Name	string key	no	Name of Linux_ComputerSystem
CreationClassName	string key	no	Linux_EthernetPort
DeviceID	string Key	no	Contains the unique port name of the syntax : eth <ID>.
Speed	uint64	no	<i>calls /sbin/mii-tool -v <DeviceID>; result is parsed : 100base => 100*1024*1024, 10base => 10*1024*1024, else 0</i>
MaxSpeed	uint64	no	<i>same value as property Speed</i>
LinkTechnology	uint16	no	2 ('Ethernet')
OtherLinkTechnology	string	no	NULL

5.3.3 Linux_TokenRingPort

The class `Linux_TokenRingPort` is implemented by the provider `cmpiOSBase_TokenRingPortProvider`. The provider supports the instance and method interface. It consists of two modules (`cmpiOSBase_TokenRingPortProvider`, `cmpiOSBase_TokenRingPort`) and depends on the tool libraries `cmpiOSBase_Common` (see chapter 3.4 Tool Library) and `OSBase_CommonNetwork` (see chapter 5.4 Tool Library).

5.3.3.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object path(s) / instance(s) of `Linux_TokenRingPort` as entries with the prefix `tr`, indicating the type "Token Ring", have been found in the `/proc/net/dev` file.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_TokenRingPort` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **CreateInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **DeleteInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **ExecQuery()**
CMPI_RC_ERR_NOT_SUPPORTED

5.3.3.2 Method Support

The method interface is supported and part of the provider `cmpiOSBase-TokenRingPortProvider`.

- **MethodCleanup()**
No functionality.
- **InvokeMethod()**

If a wrong method name was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. The provider knows the following method names :

- o `uint32 SetPowerState([IN] uint16 PowerState, [IN] datetime Time)`
CMPI_RC_ERR_NOT_SUPPORTED
- o `uint32 Reset()`
CMPI_RC_ERR_NOT_SUPPORTED
- o `uint32 EnableDevice([IN] boolean Enabled)`
CMPI_RC_ERR_NOT_SUPPORTED
- o `uint32 OnlineDevice([IN] boolean Online)`
CMPI_RC_ERR_NOT_SUPPORTED
- o `uint32 QuiesceDevice([IN] boolean Quiesce)`
CMPI_RC_ERR_NOT_SUPPORTED
- o `uint32 SaveProperties()`
CMPI_RC_ERR_NOT_SUPPORTED

- o uint32 RestoreProperties()
- CMPI_RC_ERR_NOT_SUPPORTED

5.3.3.3 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s SetInstance() function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Linux TokenRingPort
Description	string	no	This class represents instances of available TokenRing Ports.
ElementName	string	no	<i>same value as key property “DeviceID”</i>
Name	string	no	<i>same value as key property “DeviceID”</i>
Status	string	no	OK
EnabledState	uint16	no	<i>if currently started : 2 (“Enabled”); if not started 3 (“Disabled”); else 0 (“Unknown”)</i>
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as property EnabledState</i>
EnabledDefault	uint16	no	2 (“Enabled”)
SystemCreationClassName	string key	no	CreationClassName of Linux_ComputerSystem
System Name	string key	no	Name of Linux_ComputerSystem
CreationClassName	string key	no	Linux_EthernetPort
DeviceID	string Key	no	Contains the unique port name of the syntax : eth <ID>.
Speed	uint64	no	16*1024*1024
MaxSpeed	uint64	no	<i>same value as property Speed</i>
LinkTechnology	uint16	no	7 (‘Token Ring’)
OtherLinkTechnology	string	no	NULL

5.3.4 Linux_IPProtocolEndpoint

The class Linux_IPProtocolEndpoint is implemented by the provider cmpiOSBase_IPProtocolEndpointProvider. The provider supports the instance interface. It consists of three modules (cmpiOSBase_IPProtocolEndpointProvider, cmpiOSBase_IPProtocolEndpoint, OSBase_IPProtocolEndpoint) and depends on the tool libraries cmpiOSBase_Common (see chapter 3.4 Tool Library) and OSBase_CommonNetwork (see chapter 5.4 Tool Library).

5.3.4.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object path(s) / instance(s) of Linux_IPProtocolEndpoint as network port entries have been found in the `/proc/net/dev` file.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_IPProtocolEndpoint` defined by the input object path.

If a wrong object path was given to the provider, it returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

`CMPI_RC_ERR_NOT_SUPPORTED`

5.3.4.2 Class Properties

The table shows only the properties implemented by the provider. The “writable” column indicates, if the property can be changed by using the provider’s `SetInstance()` function. The “System Resource / Value” column contains either a static value or a *description how to access the value*.

NAME	TYPE	WRITABLE	SYSTEM RESOURCE / VALUE
Caption	string	no	Protocol Endpoint for IP (Internet Protocol)
Description	string	no	A communication point to send and receive data. This class is dedicated to link IP interfaces to Logical Networks.
ElementName	string	no	<i>same value as key property “Name”</i>
Name	string	no	<i>Contains the unique name of the syntax : IPv4_<portname>.</i>
EnabledState	uint16	no	<i>if currently started : 2 (“Enabled”); if not started 3 (“Disabled”); else 0 (“Unknown”)</i>
OtherEnabledState	string	no	NULL
RequestedState	uint16	no	<i>same value as property EnabledState</i>
EnabledDefault	uint16	no	2 (“Enabled”)

SystemCreationClassName	string key	no	CreationClassName of Linux_ComputerSystem
System Name	string key	no	Name of Linux_ComputerSystem
CreationClassName	string key	no	Linux_EthernetPort
NameFormat	string	no	IP
ProtocolType	string	no	2 ('IPv4')
OtherTypeDescription	string	no	NULL
IPv4Address	string	no	<i>calls /sbin/ifconfig <portname>; result is parsed for 'inet addr:.'; if not found NULL</i>
IPv6Address	string	no	NULL
SubnetMask	string	no	<i>calls /sbin/ifconfig <portname>; result is parsed for 'Mask:.'; if not found NULL</i>
IPVersionSupport	uint16	no	1 ('IPv4 only')

5.3.5 Linux_CSNetworkPort

The aggregation `Linux_CSNetworkPort` is implemented by the provider `cmpiOSBase_CSNetworkPortProvider`. The provider supports the instance and association interfaces. It consists of one module (`cmpiOSBase_CSNetworkPortProvider`) and depends on the tool library `cmpiOSBase_Common` (see chapter 3.4 Tool Library).

5.3.5.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of `Linux_CSNetworkPort`, as network ports (`Linux_LocalLoopbackPort`, `Linux_EthernetPort`, `Linux_TokenRingPort`) are hosted by the computer system.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_CSNetworkPort` defined by the input object path.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, `GetInstance()` returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **DeleteInstance()**
CMPI_RC_ERR_NOT_SUPPORTED
- **ExecQuery()**
CMPI_RC_ERR_NOT_SUPPORTED

5.3.5.2 Association Interface

- **AssociationCleanup()**
No functionality.
- **AssociatorNames(), Associators()**

In the case of success, it returns the object path(s) / instance(s) of the target class. Their number depends on the source object. If the object path of the `Linux_ComputerSystem` instance (source) was given to the provider, all object paths / instances of `Linux_LocalLoopbackPort`, `Linux_EthernetPort` and `Linux_TokenRingPort` (target) are returned. If the object path of a `Linux_LocalLoopbackPort`, `Linux_EthernetPort` or `Linux_TokenRingPort` instance (source) was given to the provider, the object path / instance of `Linux_ComputerSystem` (target) is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path(s) / instance(s) of `Linux_CSNetworkPort`. Their number depends on the source object. If the object path of the `Linux_ComputerSystem` instance (source) was given to the provider, as many object paths / instances of `Linux_CSNetworkPort` are returned, as `Linux_LocalLoopbackPort`, `Linux_EthernetPort` and `Linux_TokenRingPort` (target) instances exist. If the object path of a `Linux_LocalLoopbackPort`, `Linux_EthernetPort` or `Linux_TokenRingPort` (source) was given to the provider, one object path / instance of `Linux_CSNetworkPort` is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

5.3.6 Linux_NetworkPortImplementsIPEndpoint

The association `Linux_NetworkPortImplementsIPEndpoint` is implemented by the provider `cmpiOSBase_NetworkPortImplementsIPEndpointProvider`. The provider supports the instance and association interfaces. It consists of two modules (`cmpiOSBase_NetworkPortImplementsIPEndpointProvider`, `cmpiOSBase_NetworkPortImplementsIPEndpoint`) and depends on the tool libraries `cmpiOSBase_Common` (see chapter 3.4 Tool Library) and `OSBase_CommonNetwork` (see chapter 5.4 Tool Library).

5.3.6.1 Instance Interface

- **Cleanup()**

No functionality.

- **EnumInstanceNames(), EnumInstances()**

In the case of success, it returns the total amount of object paths / instances of `Linux_NetworkPortImplementsIPEndpoint`, as network ports (`Linux_LocalLoopbackPort`, `Linux_EthernetPort`, `Linux_TokenRingPort`) implement instance(s) of `Linux_IPProtocolEndpoint`. In most cases this is a 1-to-1 relation, one network port implements one IP protocol endpoint. But it can happen that a certain IP protocol endpoint has no relation to a network port, because the network port is not one of the supported types.

In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The `EnumInstances()` input parameter `properties` is not supported.

- **GetInstance()**

In the case of success, it returns the instance of `Linux_NetworkPortImplementsIPEndpoint` defined by the input object path.

If the referenced instances do not exist or a wrong object path was given to the provider, the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of any other failure, `GetInstance()` returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `properties` is not supported.

- **SetInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **CreateInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **DeleteInstance()**

`CMPI_RC_ERR_NOT_SUPPORTED`

- **ExecQuery()**

CMPI_RC_ERR_NOT_SUPPORTED

5.3.6.2 Association Interface

- **AssociationCleanup()**

No functionality.

- **AssociatorNames(), Associators()**

In the case of success, it returns the object path(s) / instance(s) of the target class. Their number depends on the source object. If an object path of a `Linux_LocalLoopbackPort`, `Linux_EthernetPort` or `Linux_TokenRingPort` instance (source) was given to the provider, the object path / instance of `Linux_IPProtocolEndpoint` (target) is returned, that is implemented by the specific network port. If the object path of a `Linux_IPProtocolEndpoint` instance (source) was given to the provider, the corresponding object path / instance of `Linux_LocalLoopbackPort`, `Linux_EthernetPort` or `Linux_TokenRingPort` (target) is returned, that implements the specified IP protocol endpoint. If a relation between the specified source instance and the possible target instances does not exist, nothing is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass`, `resultClass`, `role` and `resultRole` are supported. The `Associators()` input parameter `propertyList` is not supported.

- **ReferenceNames(), References()**

In the case of success, it returns the object path(s) / instance(s) of `Linux_NetworkPortImplementsIPEndpoint`. Their number depends on the source object. If the object path of a `Linux_LocalLoopbackPort`, `Linux_EthernetPort` or `Linux_TokenRingPort` instance (source) was given to the provider, the object path / instance of `Linux_NetworkPortImplementsIPEndpoint` is returned, that shows the relation between the implemented IP protocol endpoint and this specific network port. If the object path of a `Linux_IPProtocolEndpoint` instance (source) was given to the provider, the object path / instance of `Linux_NetworkPortImplementsIPEndpoint` is returned, that shows the relation to the corresponding `Linux_LocalLoopbackPort`, `Linux_EthernetPort` or `Linux_TokenRingPort` (target) instances implementing this IP protocol endpoint. If a relation between the specified source instance and the possible target instances does not exist, nothing is returned.

If the referenced instance (source) do not exist the function returns with `CMPI_RC_ERR_NOT_FOUND`. In the case of failure, the function returns with `CMPI_RC_ERR_FAILED` and a human readable error message.

The input parameter `assocClass` and `role` are supported. The `References()` input parameter `propertyList` is not supported.

5.4 Tool Library

The tool library is called `OSBase_CommonNetwork`. It consists of one module, that contains the resource access dependent functions.

5.4.1 CIM dependent Layer

NOT AVAILABLE

5.4.2 Resource Access Layer

The resource access layer externalizes the following function.

```
struct cim_netPort {
    char          * name;
    unsigned short linkTec;
    unsigned short type;
    unsigned short enabled;
    unsigned long long speed;
    unsigned long long maxSpeed;
}

struct netPortList {
    struct cim_netPort * sptr ;
    struct netPortList * next ;
}
```

unsigned short _get_port_status(char * id)

This function indicates the current status of the network port `id` (e.g. `eth0`). If the network port is currently enabled, the function returns 2. If currently disabled, it returns 3 and if the status is unknown it returns 0.

char * _get_ipProtocolEndpoint_name(char * id)

This function adds the prefix "IPv4_" to `id` and returns the pointer to the new string.

int enum_all_netPorts(struct netPortList ** lptr)

This function enumerates all network ports of type local loopback, ethernet and tokenring known to the system. The enumeration is based on the `/proc/net/dev` file.

If successful the function returns 0 and `lptr` points to the starting point of the chained list of network port entries. In the case of an error, it returns with -1 and `lptr` is a NULL pointer.

int get_netPort_data(char * id, unsigned short type, struct cim_netPort ** sptr)

This function collects the network port data requested by the name `id` and the type `type`.

If successful the function returns 0 and `sptr` points to the data. In the case of an error, it returns with -1 and `sptr` is a NULL pointer.

void free_netPortList(struct netPortList * lptr)

Deallocates the storage of the chained list `lptr` of network port entries.

void free_netPort(struct cim_netPort * sptr)

Deallocates the storage of a network port entry.