C++ Provider OpenPegasus

Roger Kumpf
Hewlett-Packard

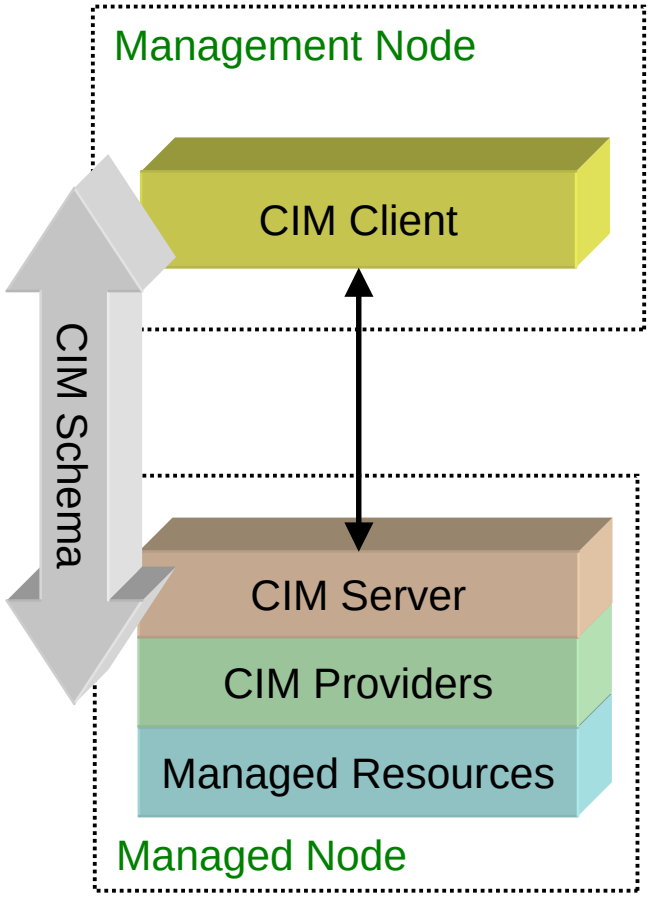
THE Open GROUP

Module Content

C++ Provider Overview

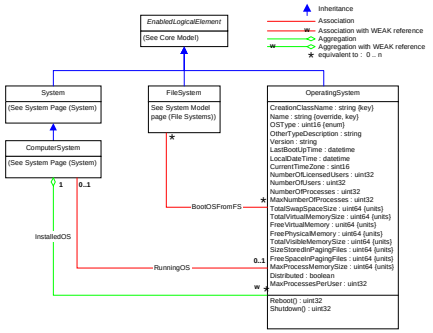
- **Concept Overview**
- Provider Example
- Instance Provider API
- Method Provider API

CIM Operations



A **CIM Client** issues CIM Operation requests and receives and processes CIM Operation responses.

What is the value of CIM_OperatingSystem.NumberOfProcesses?



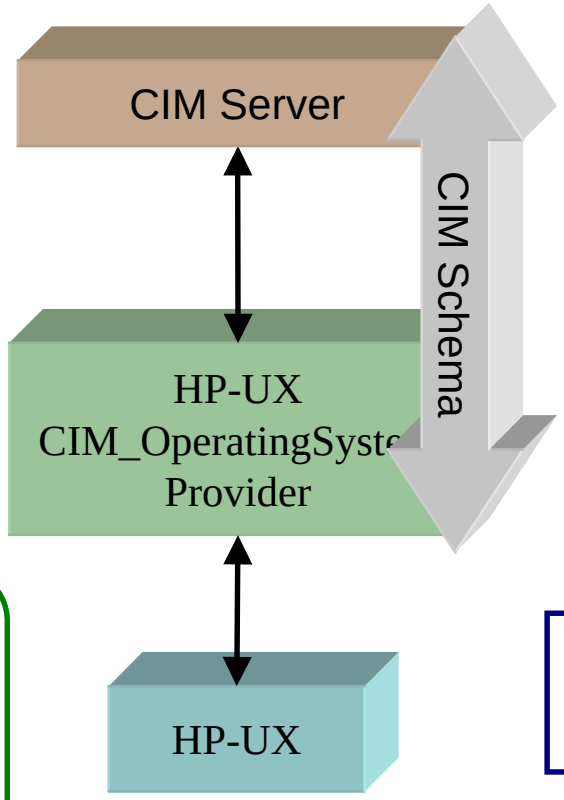
A **CIM Server** receives and processes CIM Operation requests and issues CIM Operation responses.

CIM Operations

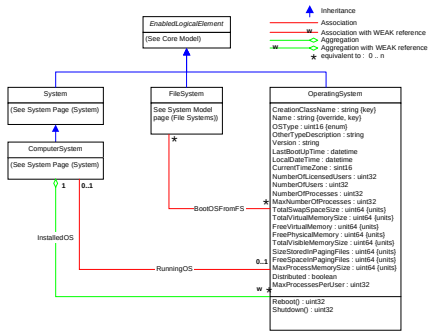
A **CIM Server** receives CIM Operation requests, coordinates the processing of requests and responses among the Providers and sends CIM Operation responses back to the CIM Client.

A **CIM Provider** translates CIM formatted requests into resource specific operations and translates resource-specific responses into CIM formatted responses.

A **Managed Resource** is a manageable entity (e.g., memory, process, system, application, network) plus the resource-specific instrumentation capable of monitoring and controlling the resource.

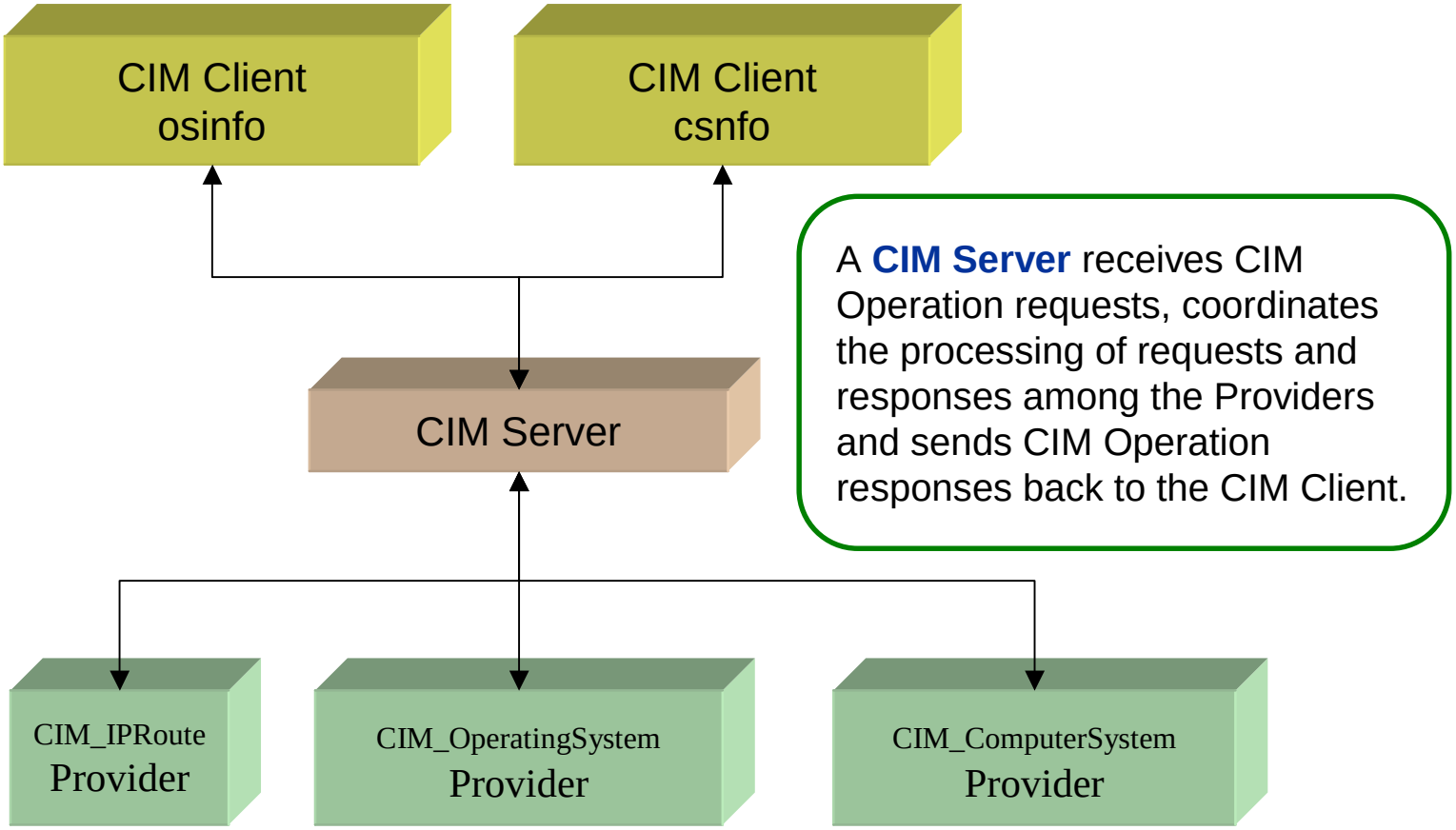


What is the value of CIM_OperatingSystem. NumberOfProcesses?



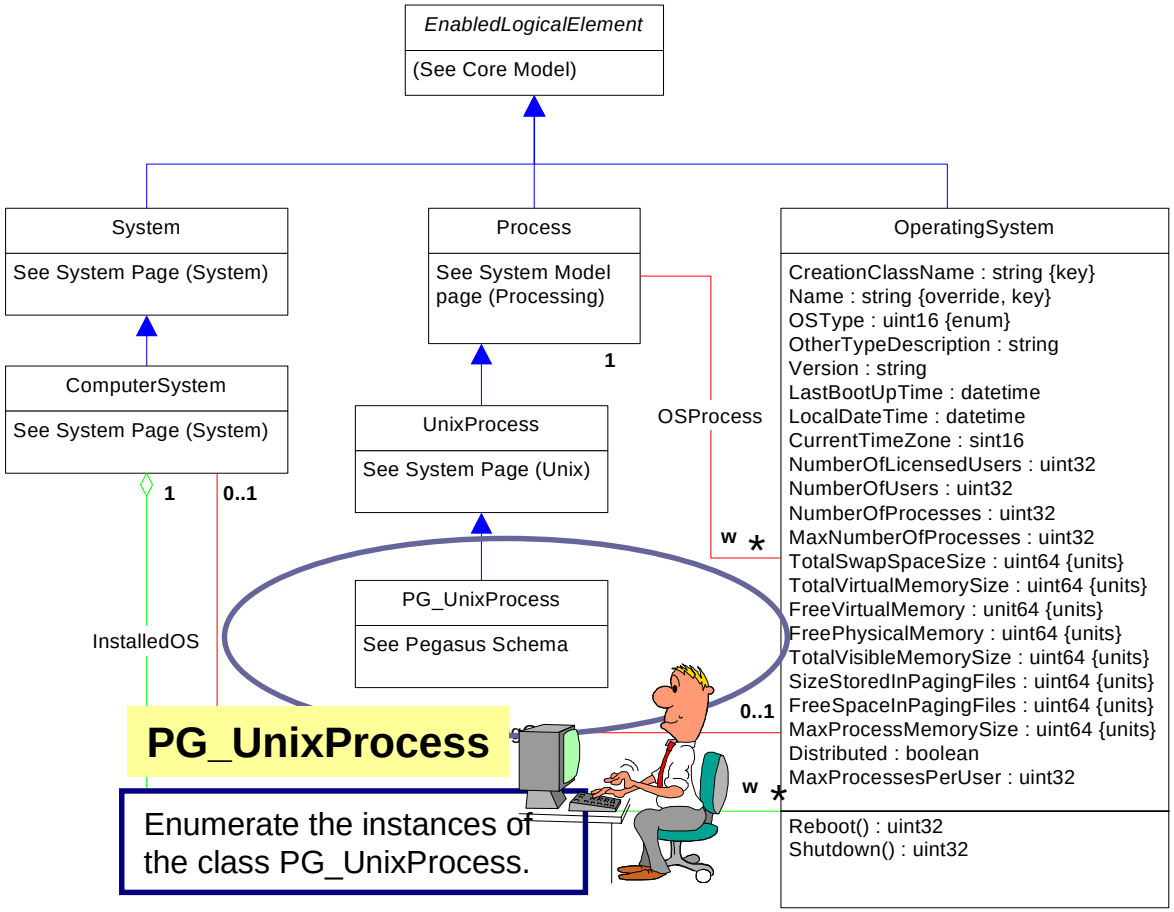
Get the value of psd_activeprocs using pstat_getdynamic.

CIM Server Role

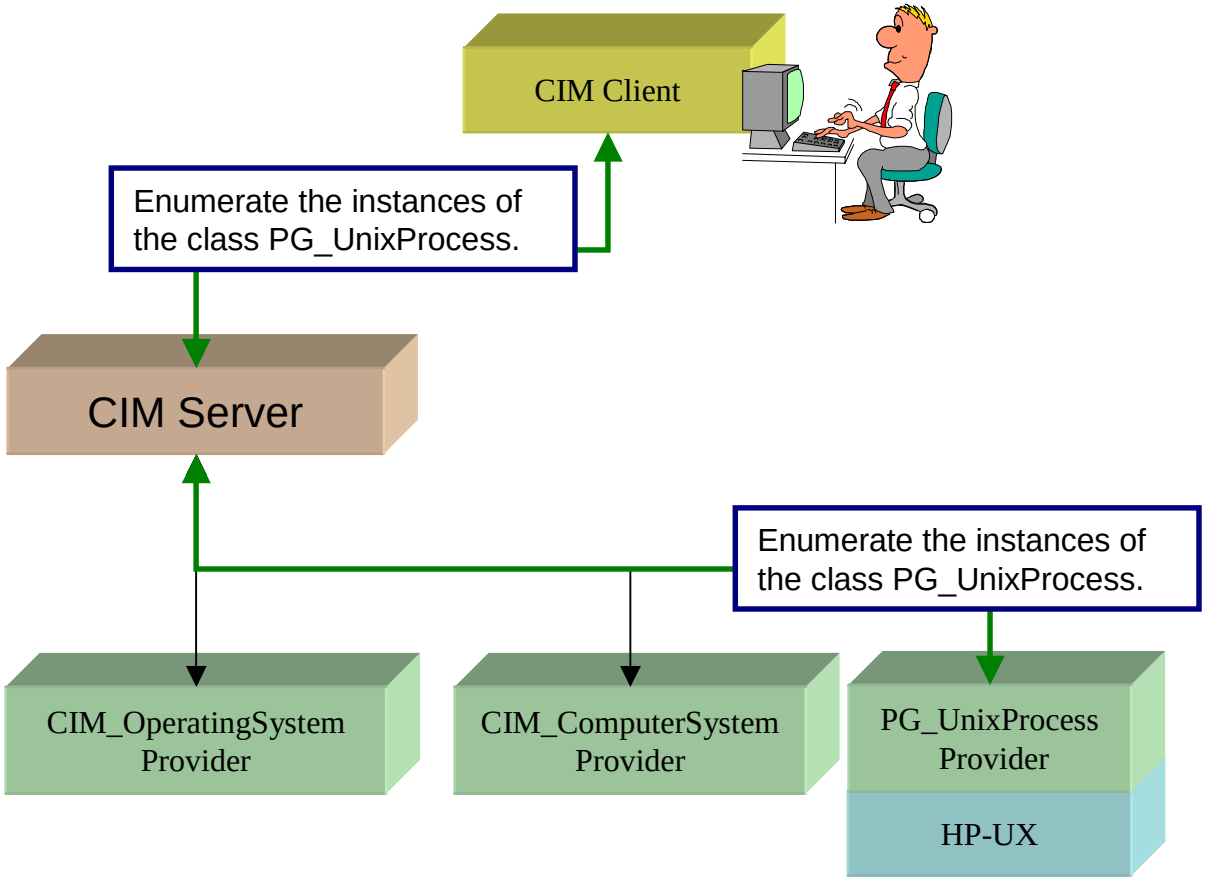


A **CIM Server** receives CIM Operation requests, coordinates the processing of requests and responses among the Providers and sends CIM Operation responses back to the CIM Client.

CIM Server Role



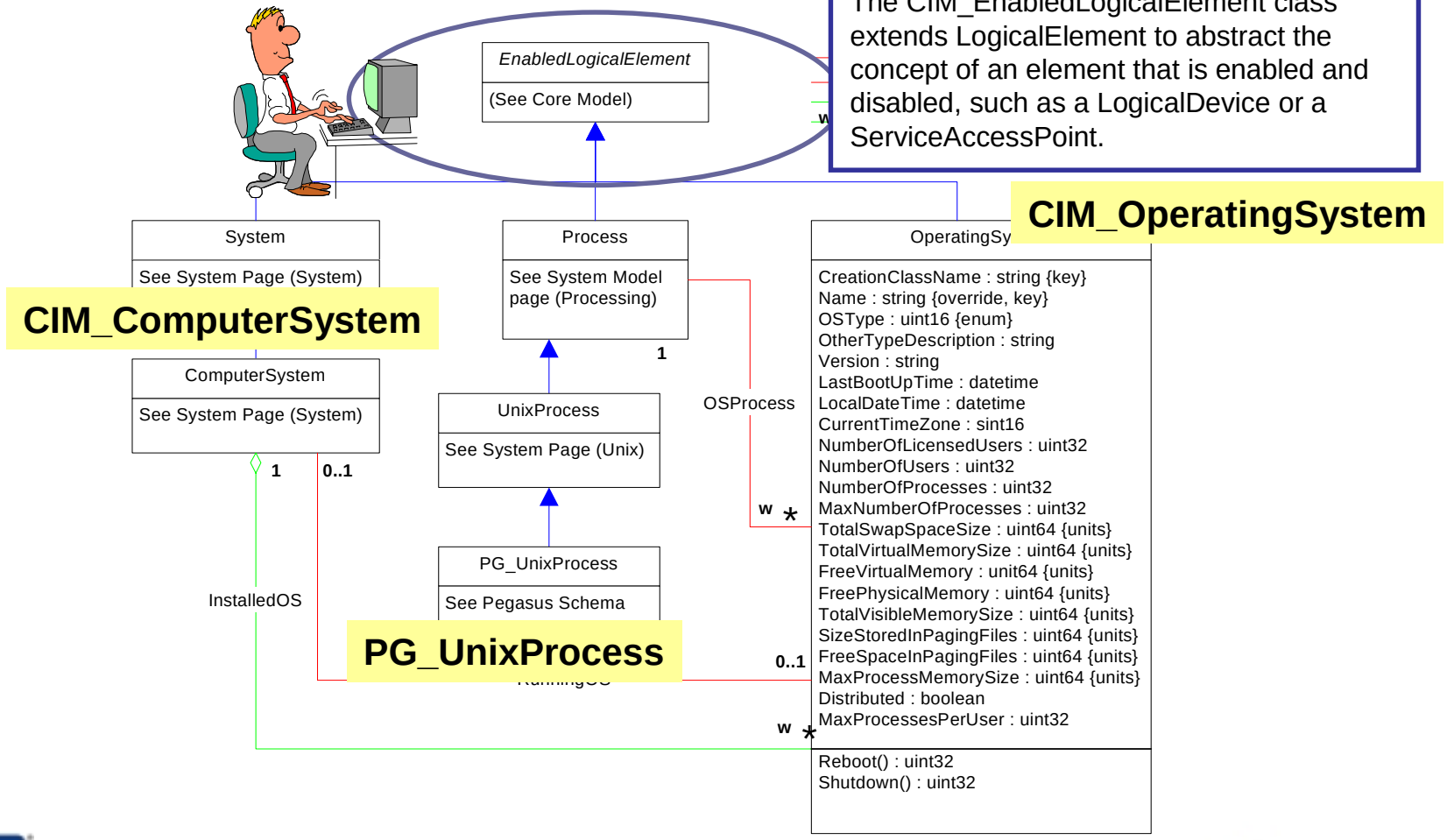
CIM Server Role



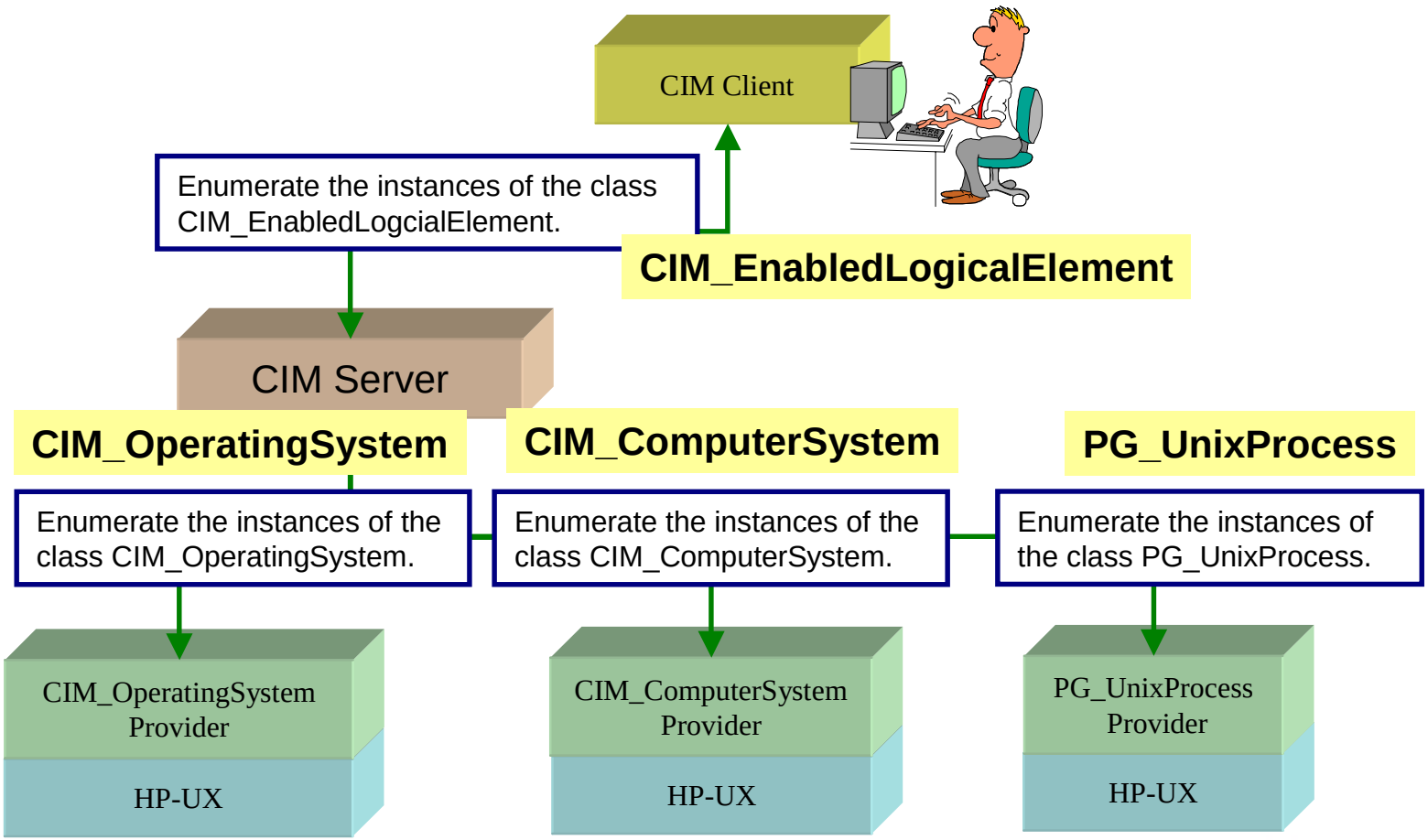
CIM Server Role

CIM_EnabledLogicalElement

The CIM_EnabledLogicalElement class extends LogicalElement to abstract the concept of an element that is enabled and disabled, such as a LogicalDevice or a ServiceAccessPoint.

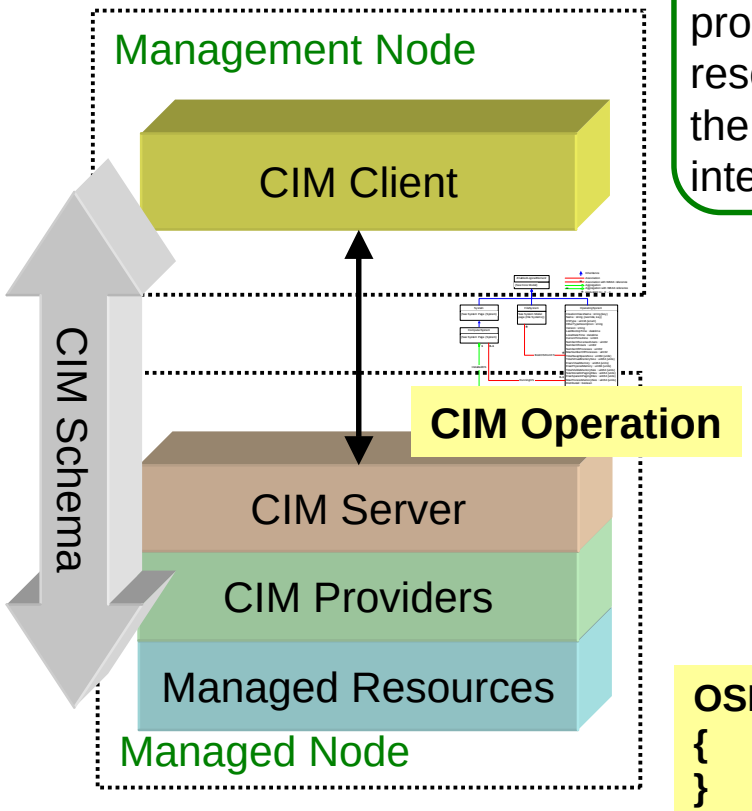


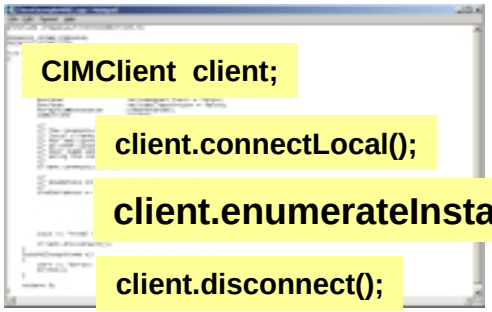
CIM Server Role

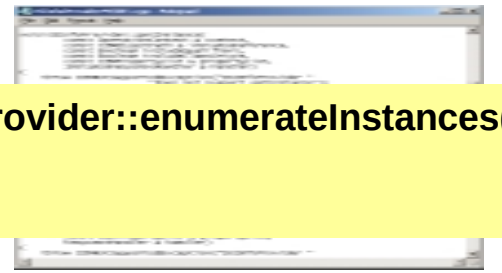


CIM Provider Role

A **CIM Provider** is responsible for the actual processing of CIM Operations on managed resources. It provides the mapping between the CIM interface and the resource-specific interface.







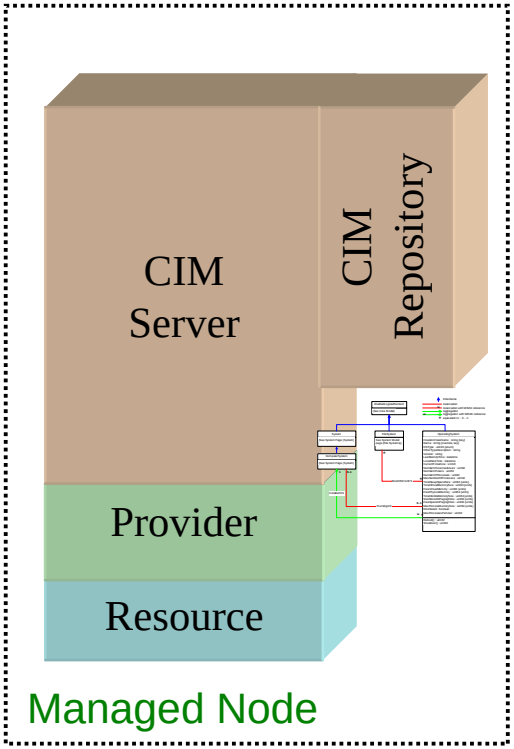
```
OSInfoProvider::enumerateInstances()
{
}
```

CIM Provider Types

CIM Operation	Implementation Owner
GetClass, CreateClass, ModifyClass, DeleteClass, GetQualifier, SetQualifier, DeleteQualifier, EnumerateQualifier, EnumerateClasses, EnumerateClassNames,	**CIMOM**
GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, ModifyInstance, DeleteInstance	**Instance Provider**
InvokeMethod	**Method Provider**
References, ReferenceNames, Associators, AssociatorNames	**Association Providers**

A CIM Provider contains the **implementation** for a set of CIM Operations for a defined set of managed resources.

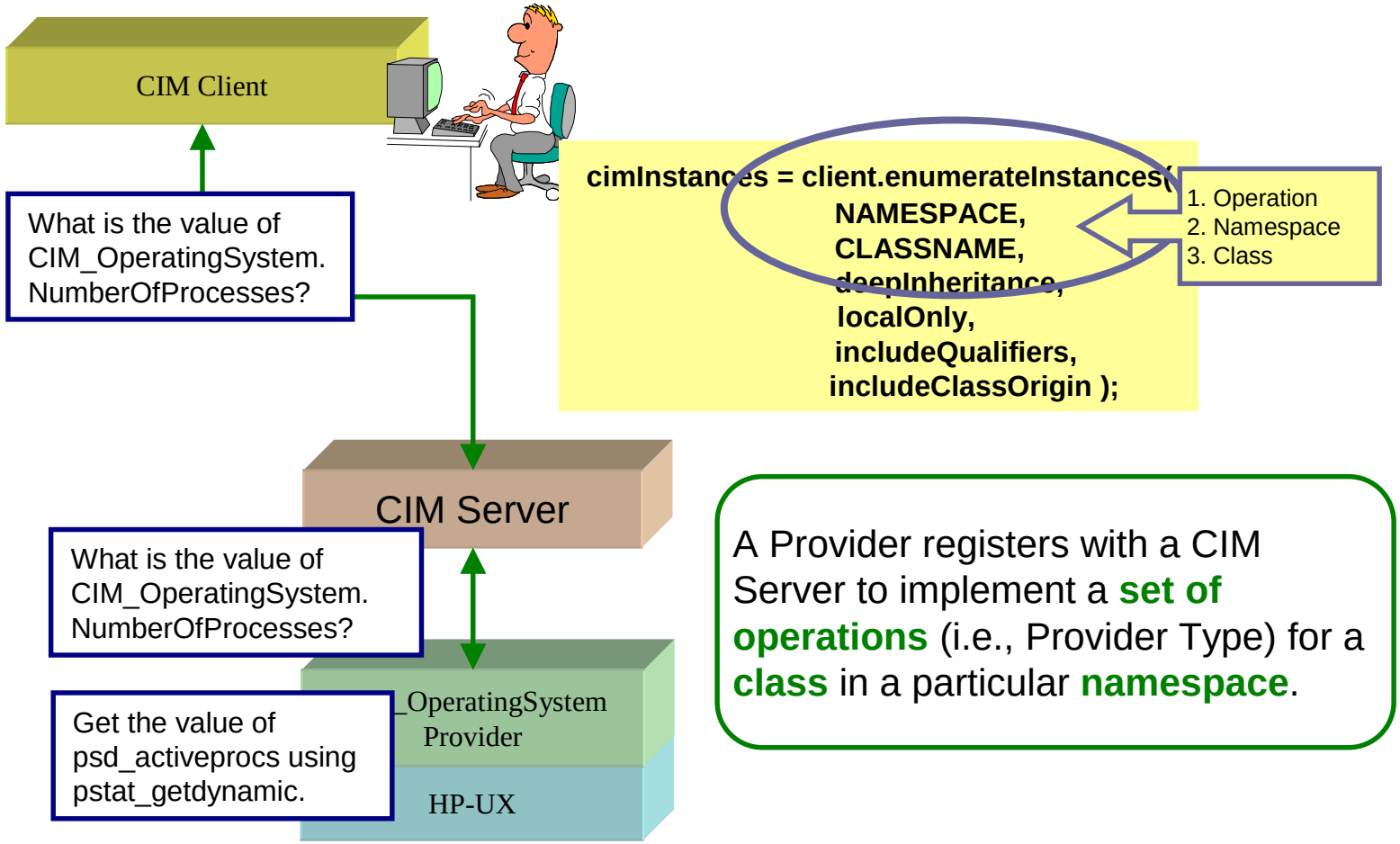
Provider Registration



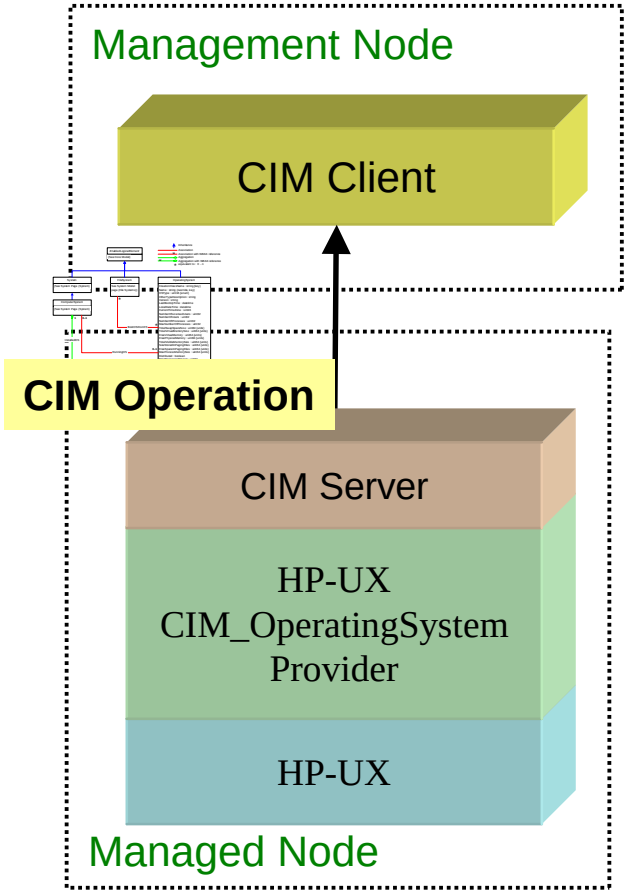
A Provider **registers** with a CIM Server to implement a set of operations (i.e., Provider Type) for a class in a particular namespace.

HP-UX
CIM_OperatingSystem
Provider

Provider Registration



Provider Registration



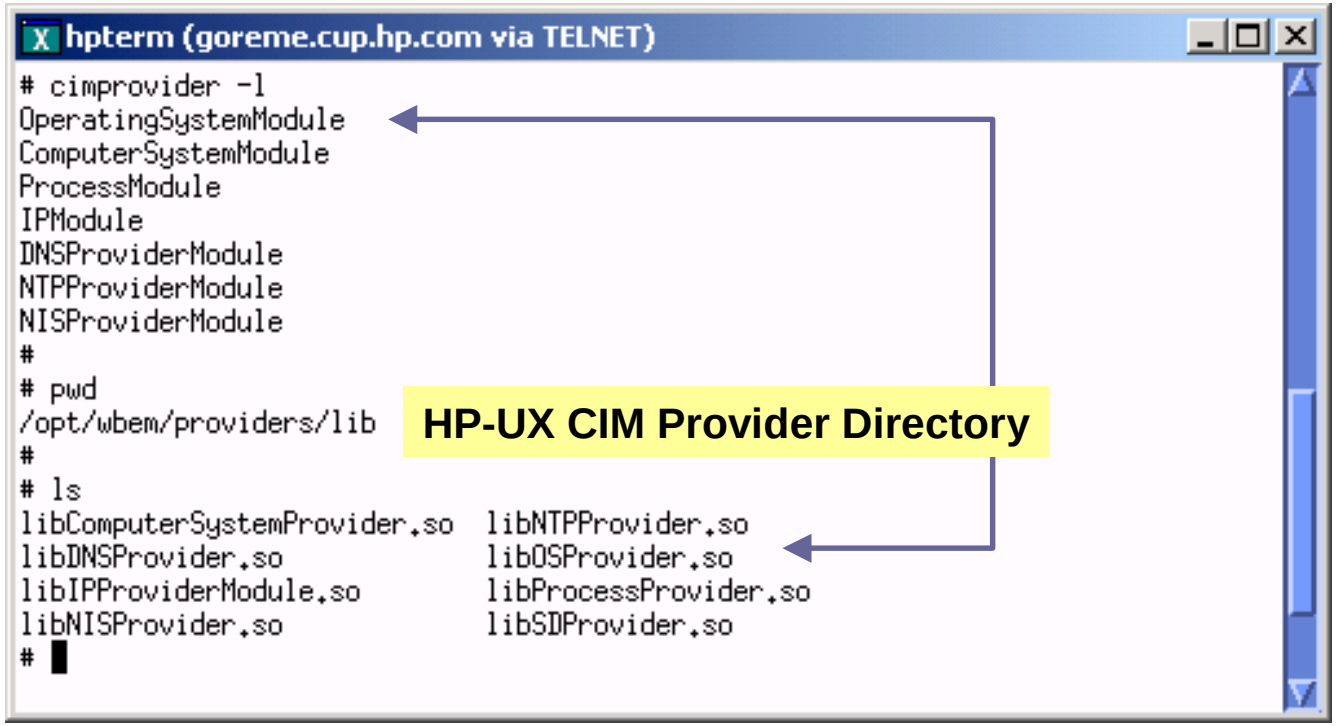
Description of Provider Capabilities

A Provider registers with a CIM Server to implement a **set of operations** (i.e., Provider Type) for a **class** in a particular **namespace**.

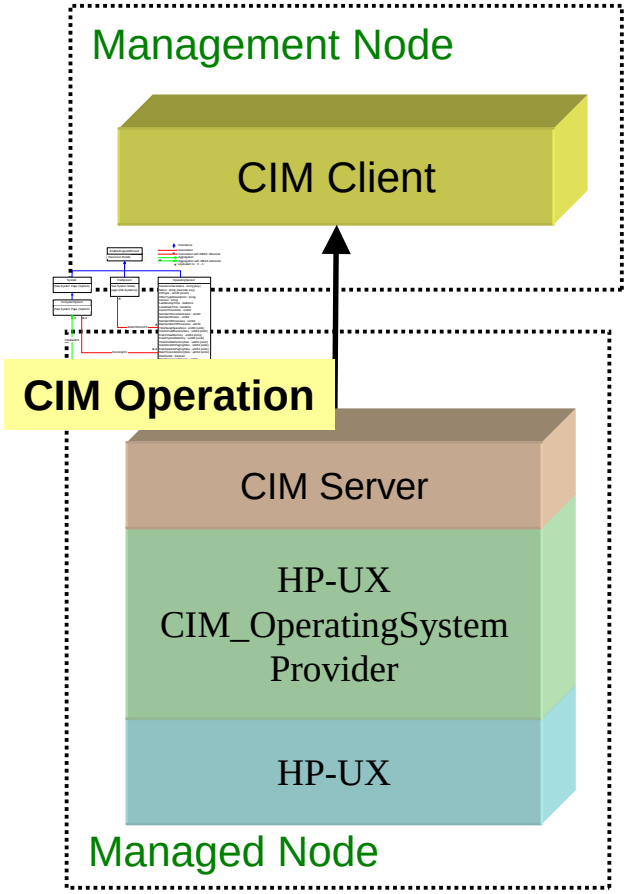
CIM_OperatingSystem Provider Registration	
Set of Operations	Instance Provider Operations
Class	CIM_OperatingSystem
Namespace	root/cimv2

CIM Providers

HP WBEM Services for HP-UX Fact: CIM Providers are implemented as shared libraries.



Provider Registration

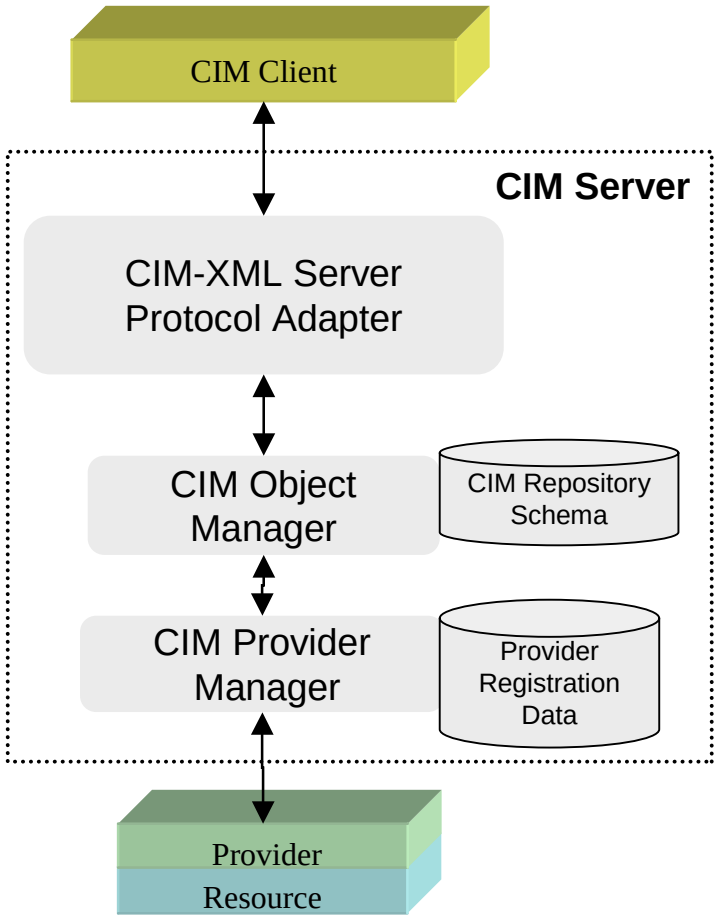


Description of Physical Package (e.g., Shared Library)

Provider register includes "implementation" information (e.g., location of the shared library).

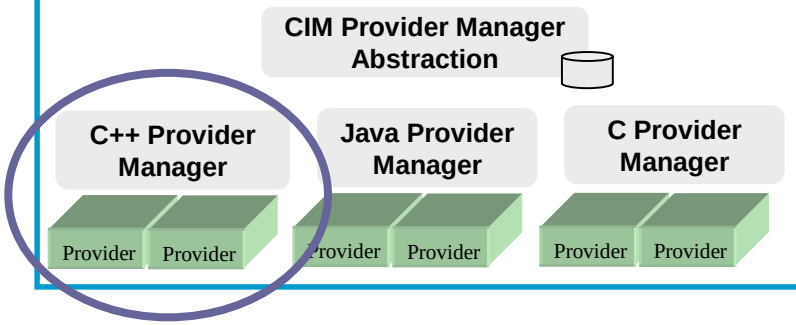
Operating System Provider	
Interface Type	
Interface Version	
Location	libOSProvider

CIM Provider Manager

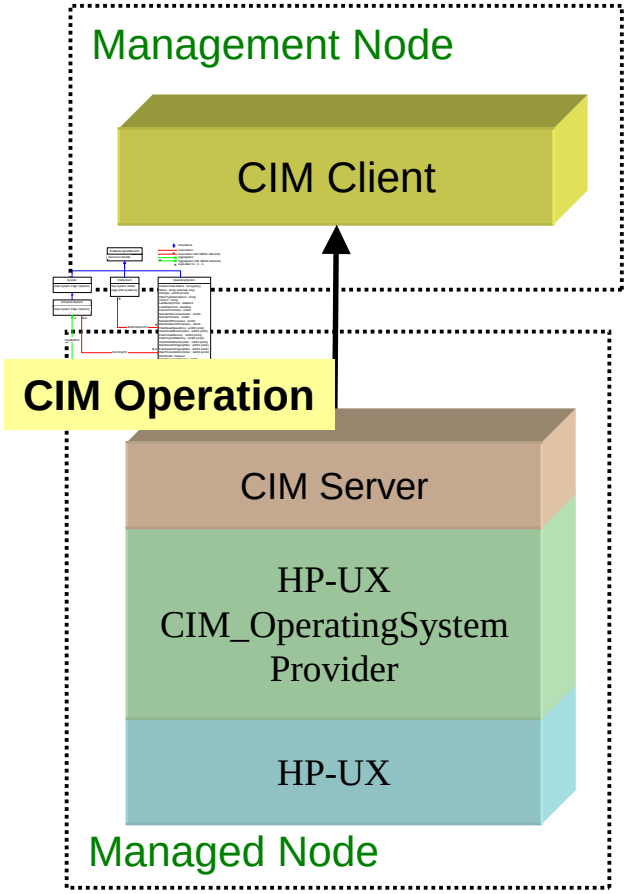


The **CIM Provider Manager** is responsible for Provider Registration, Provider Loading and Unloading, and the interface between the CIMOM and CIM Providers.

Note: A CIM Server may support multiple **provider interfaces**.



Provider Registration

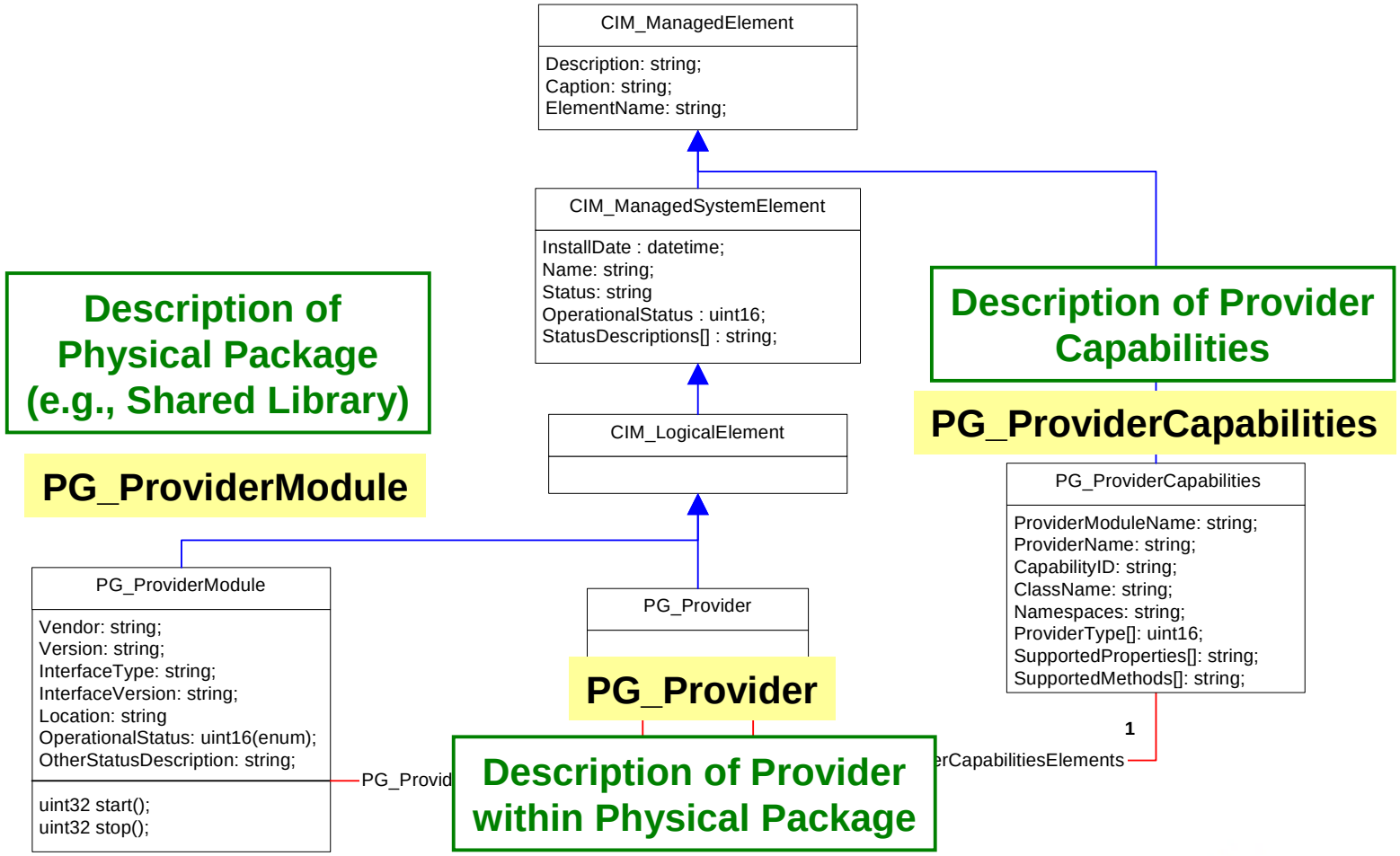


Description of Physical Package (e.g., Shared Library)

Provider register includes "implementation" information (e.g., interface type and version).

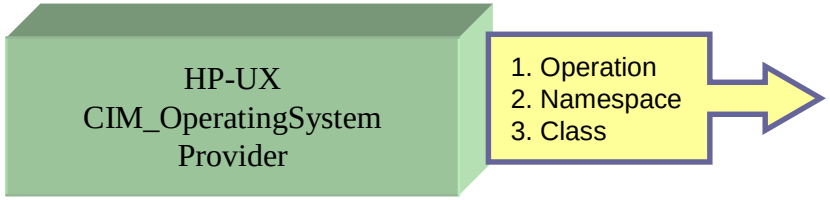
Operating System Provider	
Interface Type	C++Default
Interface Version	2.1.0
Location	OSProvider

Provider Registration Schema



Provider Capabilities

Operating System Provider	
Set of Operations	Instance Provider Operations
Class	CIM_OperatingSystem
Namespace	root/cimv2



PG_ProviderCapabilities

PG_ProviderCapabilities
ProviderModuleName: string; ProviderName: string; CapabilityID: string; ClassName: string; Namespaces: string; ProviderType[]: uint16; SupportedProperties[]: string; SupportedMethods[]: string;

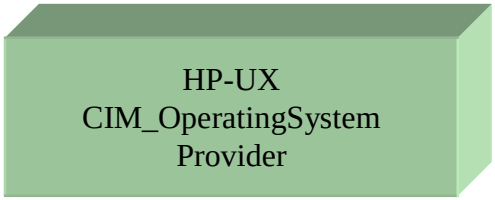
```
instance of PG_ProviderCapabilities
{
   ProviderModuleName = "OperatingSystemModule";
   ProviderName = "PG_OperatingSystemProvider";
   CapabilityID = "1";
   ClassName = "CIM_OperatingSystem";
   Namespaces = {"root/cimv2"};
   ProviderType = { 2, 5 }; // Instance and Method
   SupportedProperties = NULL; // All properties
   SupportedMethods = NULL; // All methods
};
```

Description of Provider Capabilities

Provider Module

Operating System Provider	
Interface Type	C++Default
Interface Version	2.1.0
Location	OSProvider

Description of Physical Package (e.g., Shared Library)



1. Interface Type
2. Interface Version
3. Location

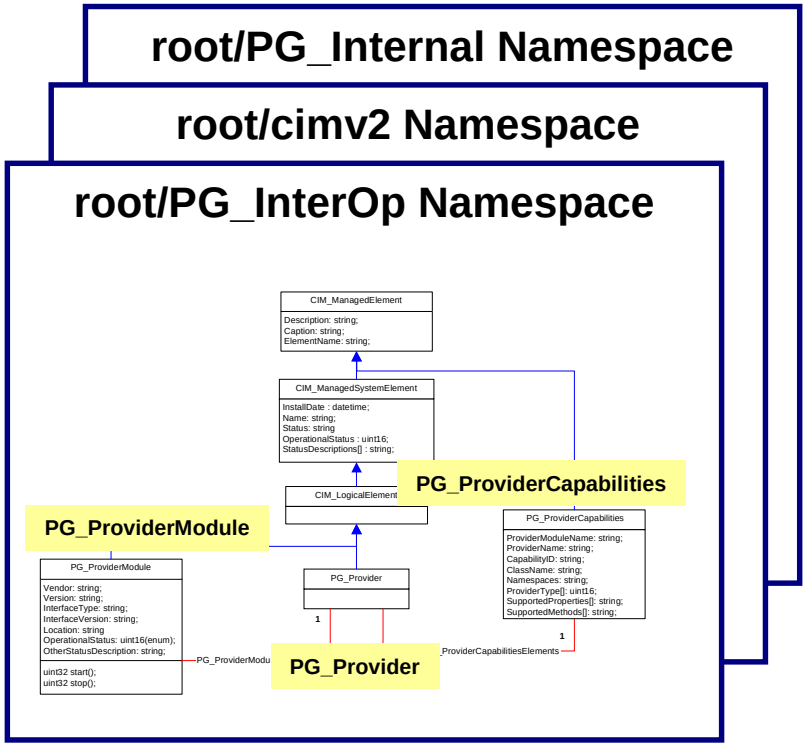
PG_ProviderModule

PG_ProviderModule
Vendor: string; Version: string; InterfaceType: string; InterfaceVersion: string; Location: string OperationalStatus: uint16(enum); OtherStatusDescription: string;
uint32 start(); uint32 stop();

```
instance of PG_ProviderModule
{
   Name = "OperatingSystemModule";
   Vendor = "Pegasus Community";
   Version = "2.0.0";
   InterfaceType = "C++Default";
   InterfaceVersion = "2.1.0";
   Location = "OSProvider";
};
```

Namespaces

The Provider Registration Schema is located in the **root/PG_InterOp** Namespace.



OSProvider Registration

cimmof –nroot/PG_InterOp PG_OperatingSystem20R.mof

PG_OperatingSystem20R.mof

PG_ProviderModule

```
instance of PG_ProviderModule
{
   Name = "OperatingSystemModule";
   Vendor = "Pegasus Community";
   Version = "2.0.0";
   InterfaceType = "C++Default";
   InterfaceVersion = "2.1.0";
   Location = "OSProvider";
};

instance of PG_Provider
{
   ProviderModuleName = "OperatingSystemModule";
   Name = "PG_OperatingSystemProvider";
};
```

PG_Provider

PG_ProviderCapabilities

```
instance of PG_ProviderCapabilities
{
   ProviderModuleName = "OperatingSystemModule";
   ProviderName = "PG_OperatingSystemProvider";
   CapabilityID = "1";
   ClassName = "CIM_OperatingSystem";
   Namespaces = {"root/cimv2"};
   ProviderType = { 2, 5 }; // Instance and Method
   SupportedProperties = NULL; // All properties
   SupportedMethods = NULL; // All methods
};

instance of PG_ProviderCapabilities
{
   ProviderModuleName = "OperatingSystemModule";
   ProviderName = "PG_OperatingSystemProvider";
   CapabilityID = "2";
   ClassName = "PG_OperatingSystem";
   Namespaces = {"root/cimv2"};
   ProviderType = { 2, 5 }; // Instance and Method
   SupportedProperties = NULL; // All properties
   SupportedMethods = NULL; // All methods
};
```

cimprovider Utility

cimprovider – disable, enable, remove or list registered CIM Providers or CIM Provider modules.

biscayne

```
cimprovider(1)

NAME
     cimprovider - disable, enable, remove or list registered CIM providers
     or CIM provider modules and module status

SYNOPSIS
     cimprovider -d -m module

     cimprovider -e -m module

     cimprovider -r -m module [-p provider]

     cimprovider -l [-s | -m module]

   Remarks
     The -l option for this command can be executed by any user(s). All
     other options require superuser permissions.

     This command disables, enables, or removes one CIM provider module or
     CIM provider at a time.
```

cimprovider Utility

hpterm (goreme.cup.hp.com via TELNET)

```
# cimprovider -l
OperatingSystemModule
ComputerSystemModule
ProcessModule
IPModule
DNSProviderModule
NTPProviderModule
NISProviderModule
#
# cimprovider -l -m ComputerSystemModule
PG_ComputerSystemProvider
#
# cimprovider -l -m ProcessModule
ProcessProvider
ProcessStatProvider
#
# cimprovider -l -m OperatingSystemModule
PG_OperatingSystemProvider
#
# cimprovider -l -m IPModule
IPPEpProvider
BIPTLEpProvider
IPRouteProvider
#
```

List Provider Modules

List Providers

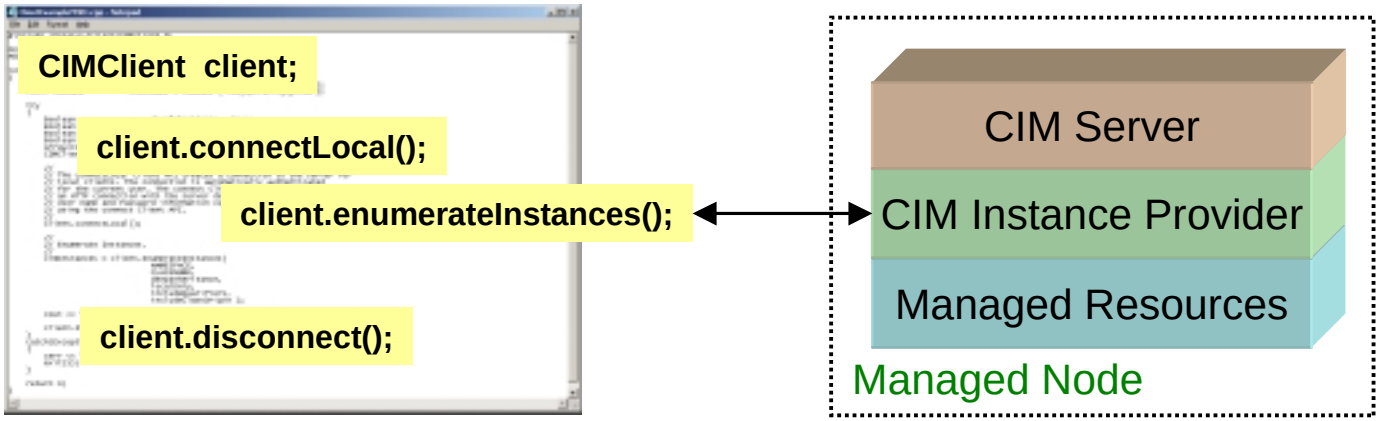
Module Content

C++ Provider Overview

- Concept Overview
- **Provider Example**
- Instance Provider API
- Method Provider API

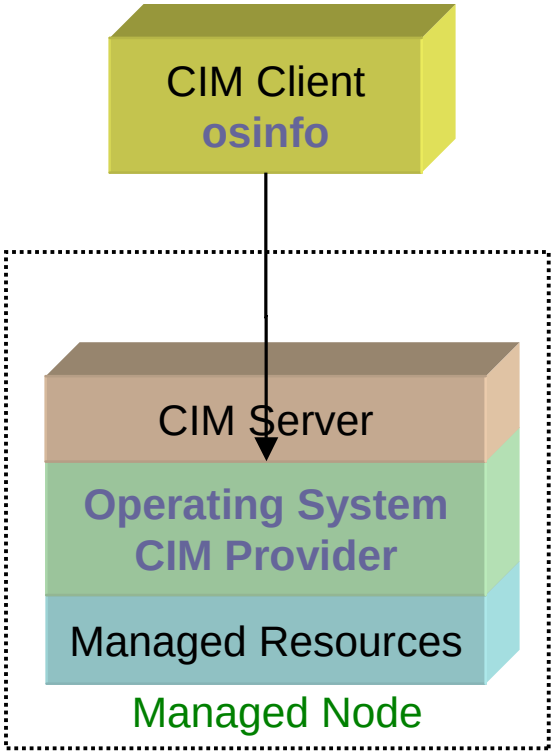
Instance Provider

CIM Operation	Implementation Owner
GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, ModifyInstance, DeleteInstance	**Instance Provider**



Instance Provider Example

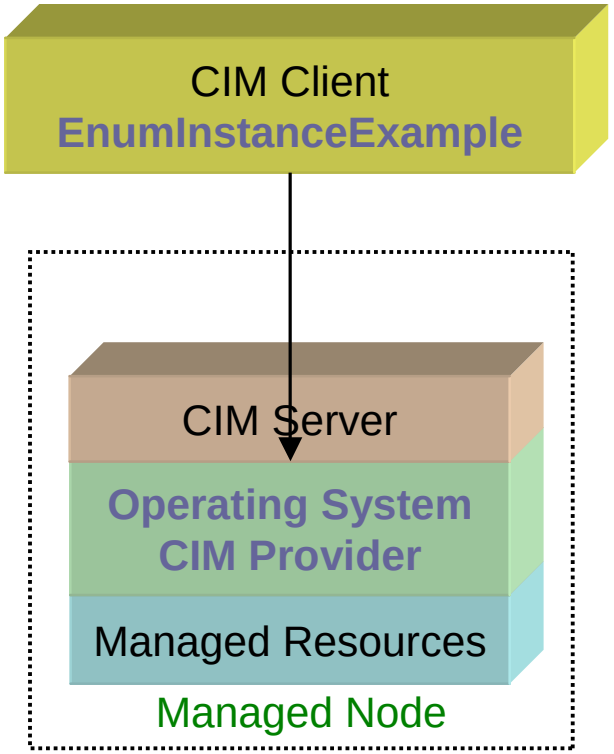
```
bodie
# osinfo
OperatingSystem Information
  Host: bodie.cup.hp.com
  Name: HP-UX
  Version: B.11.00
  UserLicense: Unlimited user license
  OSCapability: 32 bit
  LastBootTime: May 12, 2003  8:54:59 (-0700)
  LocalDateTime: Jun 14, 2003  10:46:5 (-0700)
  SystemUpTime: 2857866 seconds = 33 days, 1 hr, 51 mins, 6 secs
#
```

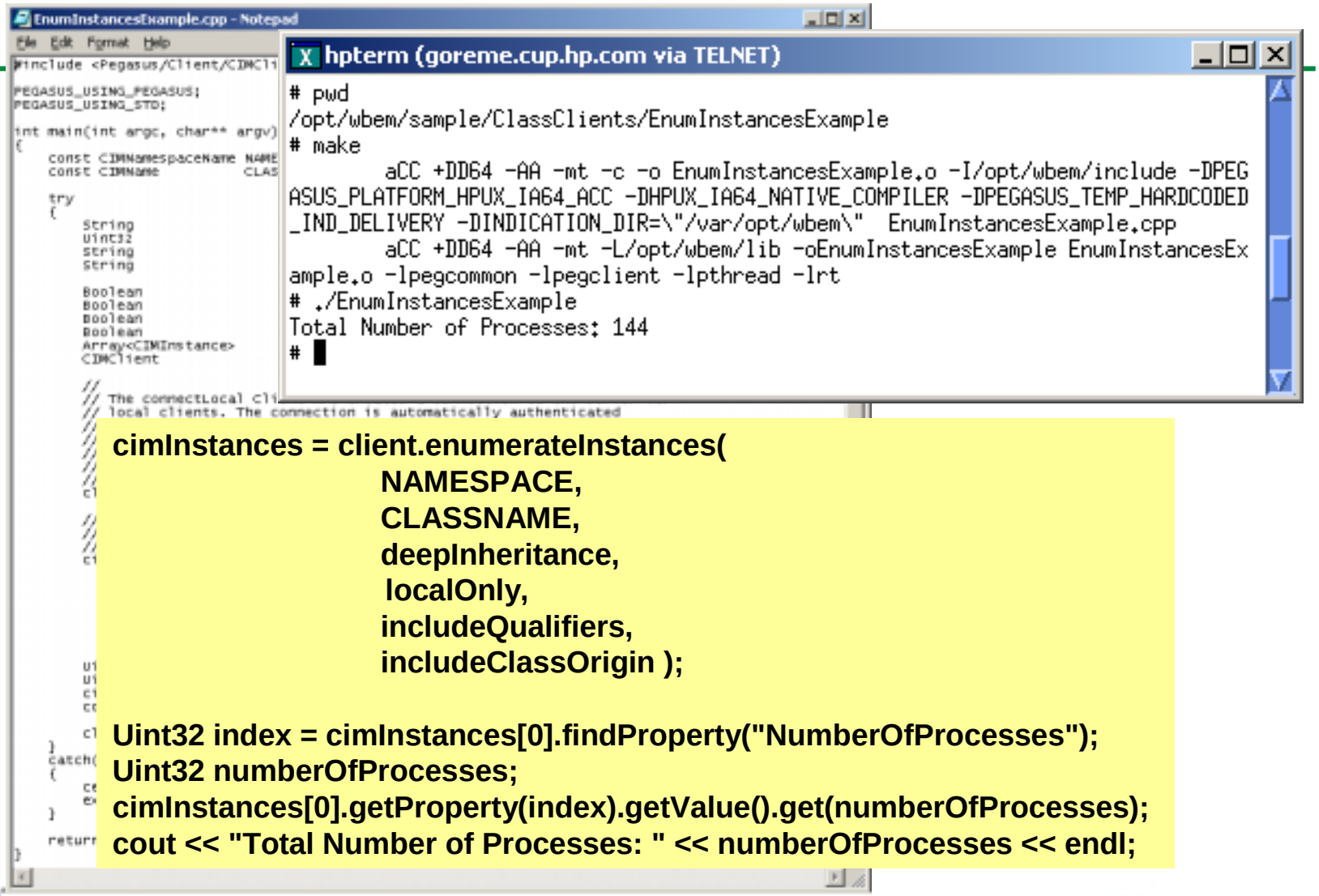


Packaged with CIM Server is an **Operating System Provider** that serves as an **Instance Provider** for the **CIM_OperatingSystem** and **PG_OperatingSystem** classes in the **root/cimv2** namespace.

Instance Provider Example

In the C++ Client Overview Module, we developed a client, **EnumInstanceExample**, that accessed the OS Provider packaged with HP WBEM Services.





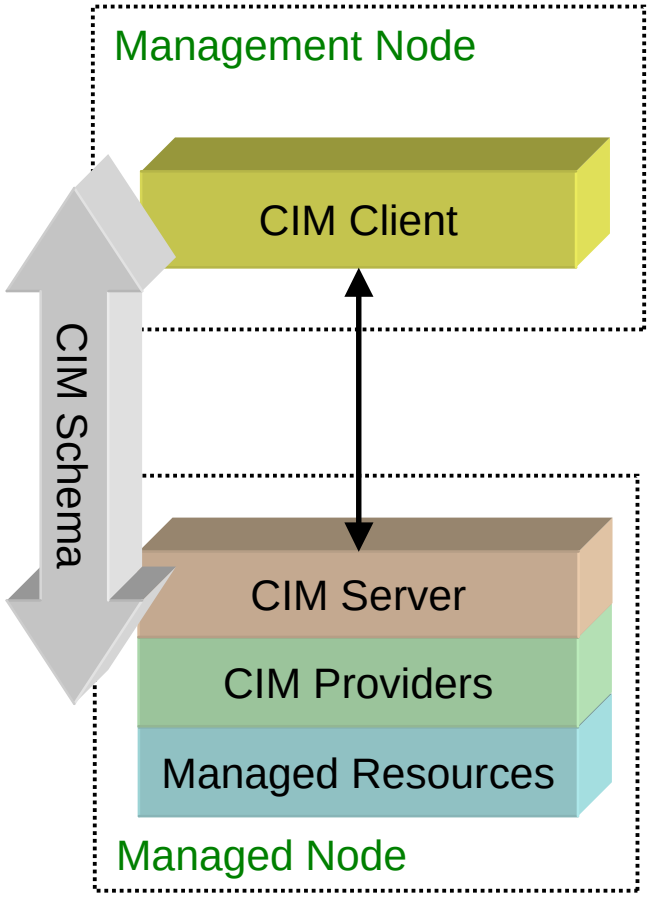

```
cimInstances = client.enumerateInstances(
                        NAMESPACE,
                        CLASSNAME,
                        deepInheritance,
                        localOnly,
                        includeQualifiers,
                        includeClassOrigin );

Uint32 index = cimInstances[0].findProperty("NumberOfProcesses");
Uint32 numberOfProcesses;
cimInstances[0].getProperty(index).getValue().get(numberOfProcesses);
cout << "Total Number of Processes: " << numberOfProcesses << endl;
```

Module Example

In this module , we'll replace the packaged Operating System Provider with our own implementation.

We'll use **EnumInstanceExample** to test our new Provider.

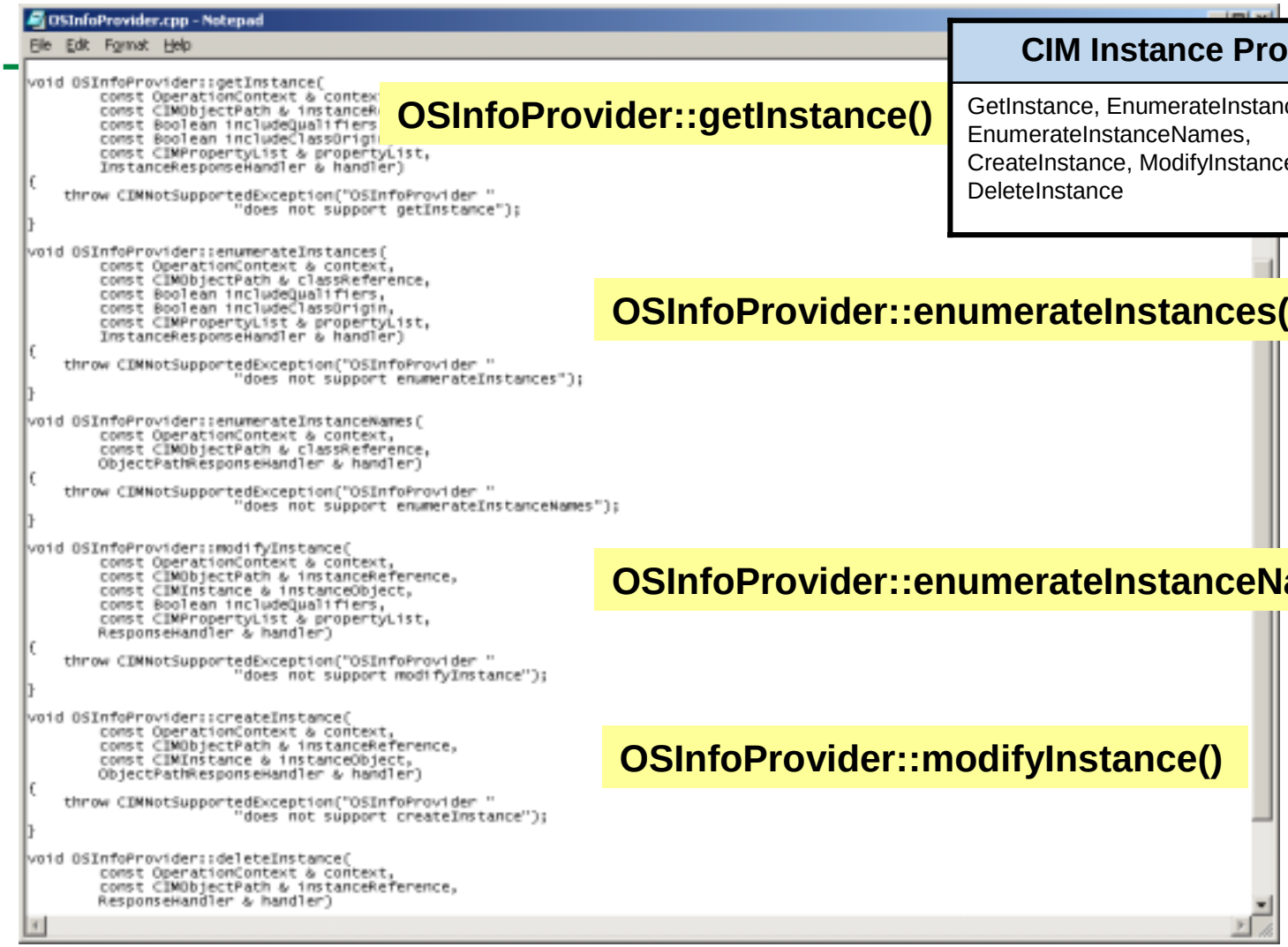
OS Provider Implementation



Let's start with a simple, but rather uninteresting OS Provider implementation.

What is the value of CIM_OperatingSystem.NumberOfProcesses?

CIM_ERR_NOT_SUPPORTED



CIM Instance Provider

GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, ModifyInstance, DeleteInstance

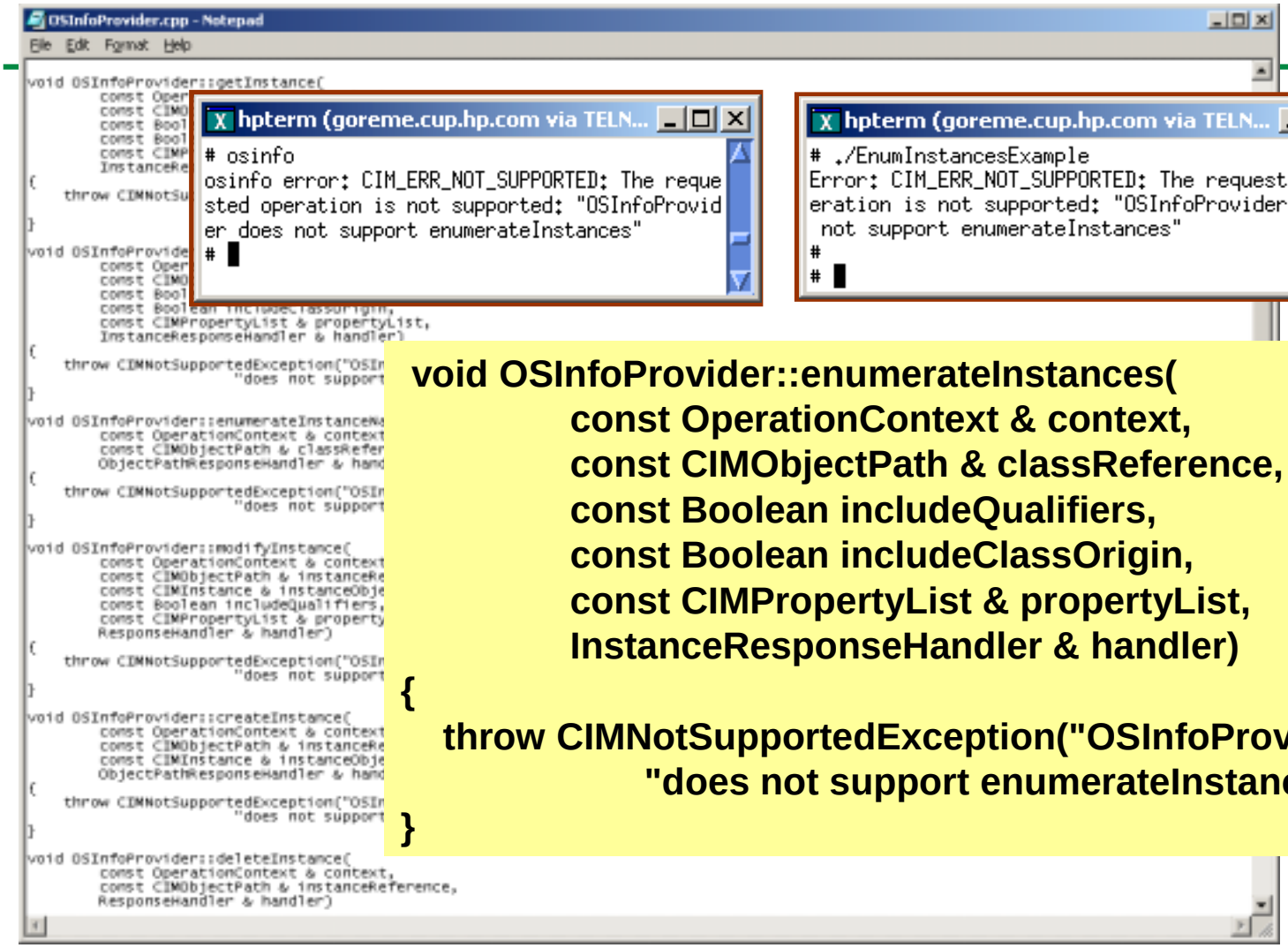
Build Instructions

Hack Alert: In this module, we'll temporarily replace the packaged Operating System Provider with our own implementation.

hpterm (goreme.cup.hp.com via TELNET)

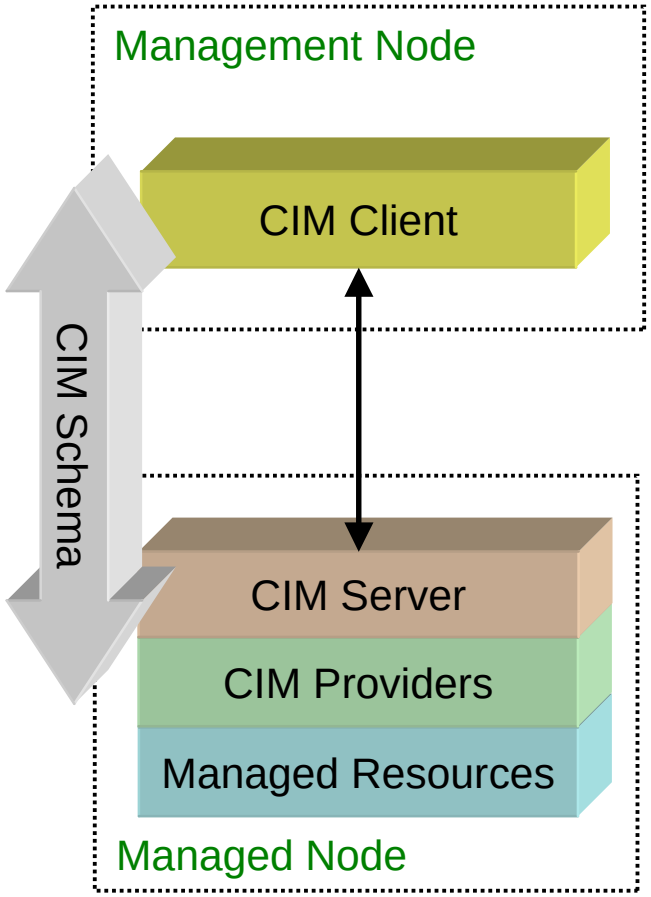
```
# pwd
/opt/wbem/sample/ClassProviders/NotSupportedOSProvider
# make build
        make disableOSProvider
        cimprovider -d -m OperatingSystemModule
Disabling provider module...
Provider module disabled successfully.
        make
        aCC -c -o OSInfoProviderMain.o  +DD64 -AA -mt +Z -I/opt/wbem/include -DPEG
ASUS_PLATFORM_HPUX_IA64_ACC -DHPUX_IA64_NATIVE_COMPILER -DPEGASUS_TEMP_HARDCODED_I
ND_DELIVERY -DINDICATION_DIR=\"/var/opt/wbem\"  OSInfoProviderMain.cpp
        aCC -c -o OSInfoProvider.o  +DD64 -AA -mt +Z -I/opt
_PLATFORM_HPUX_IA64_ACC -DHPUX_IA64_NATIVE_COMPILER -DPEGAS
ELIVERY -DINDICATION_DIR=\"/var/opt/wbem\"  OSInfoProvider.cpp
        aCC +DD64 -AA -mt +Z -b -Wl,+s -Wl,+b/opt/wbem/lib -L/opt/wbem/lib -DPEGAS
US_PLATFORM_HPUX_IA64_ACC -DHPUX_IA64_NATIVE_COMPILER -DPEGASUS_TEMP_HARDCODED_IND
_DELIVERY -DINDICATION_DIR=\"/var/opt/wbem\"  -olibOSProvider.so OSInfoProviderMai
n.o OSInfoProvider.o -lpthread -lrt -lpegcommon -lpegclient
        make -i unlink
        rm -f /opt/wbem/providers/lib/libOSProvider.so
        ln -f -s /opt/wbem/sample/ClassProviders/NotSupportedOSProvider/libOSProvi
der.so /opt/wbem/providers/lib/libOSProvider.so
        make enableOSProvider
        cimprovider -e -m OperatingSystemModule
Enabling provider module...
Provider module enabled successfully.
#
```

1. **Disable and Unload OS Provider**
2. **Build New Provider**
3. **Replace OS Provider.**
4. **Enable OS Provider**




```
void OSInfoProvider::enumerateInstances(
        const OperationContext & context,
        const CIMObjectPath & classReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
   throw CIMNotSupportedException("OSInfoProvider "
                "does not support enumerateInstances");
}
```

Provider Implementation



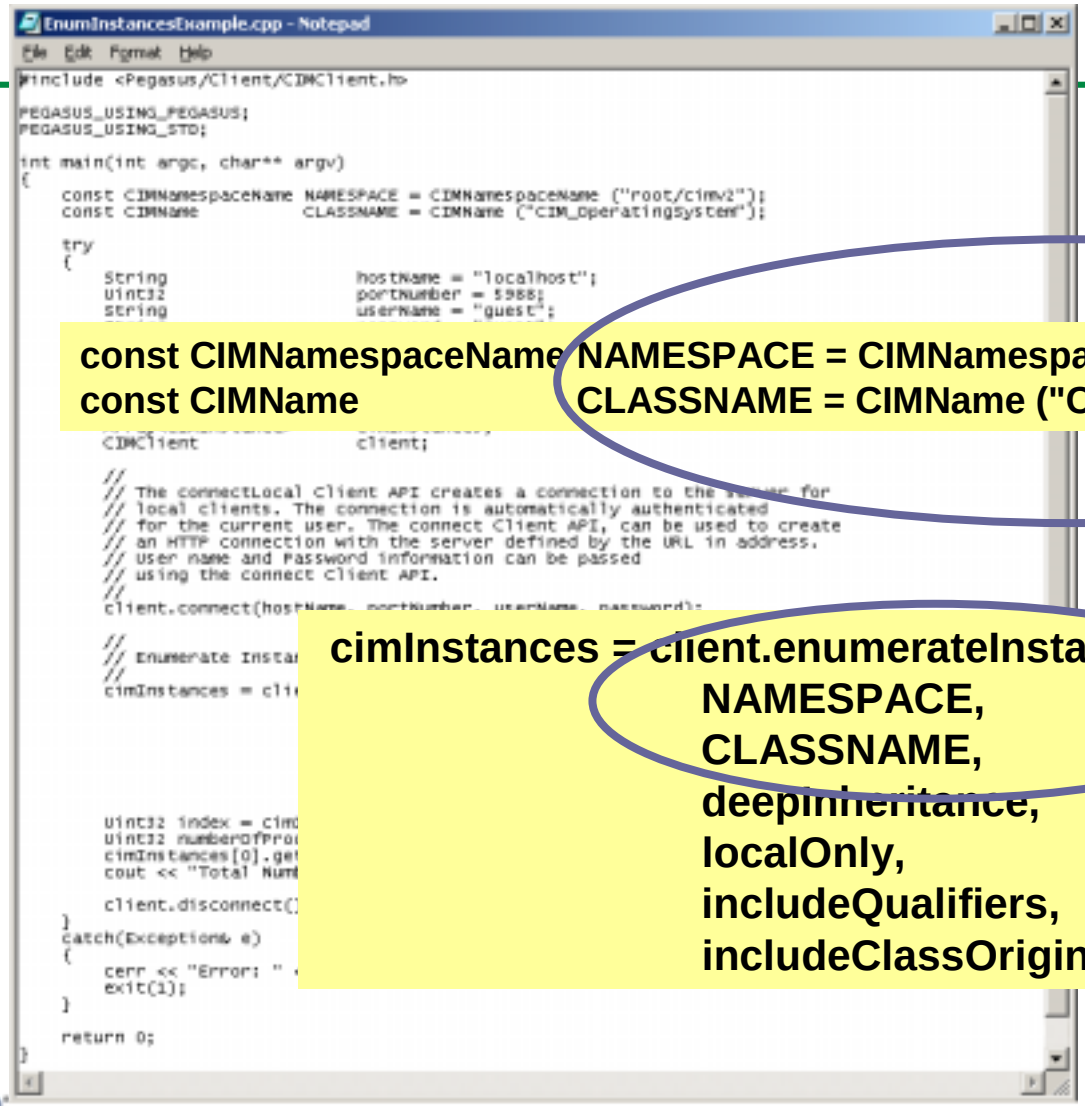
Now let's try something slightly more complicated, but not really more useful.

☺

Instead of returning NOT SUPPORTED. The new implementation will return the value 12345 for NumberOfProcesses.

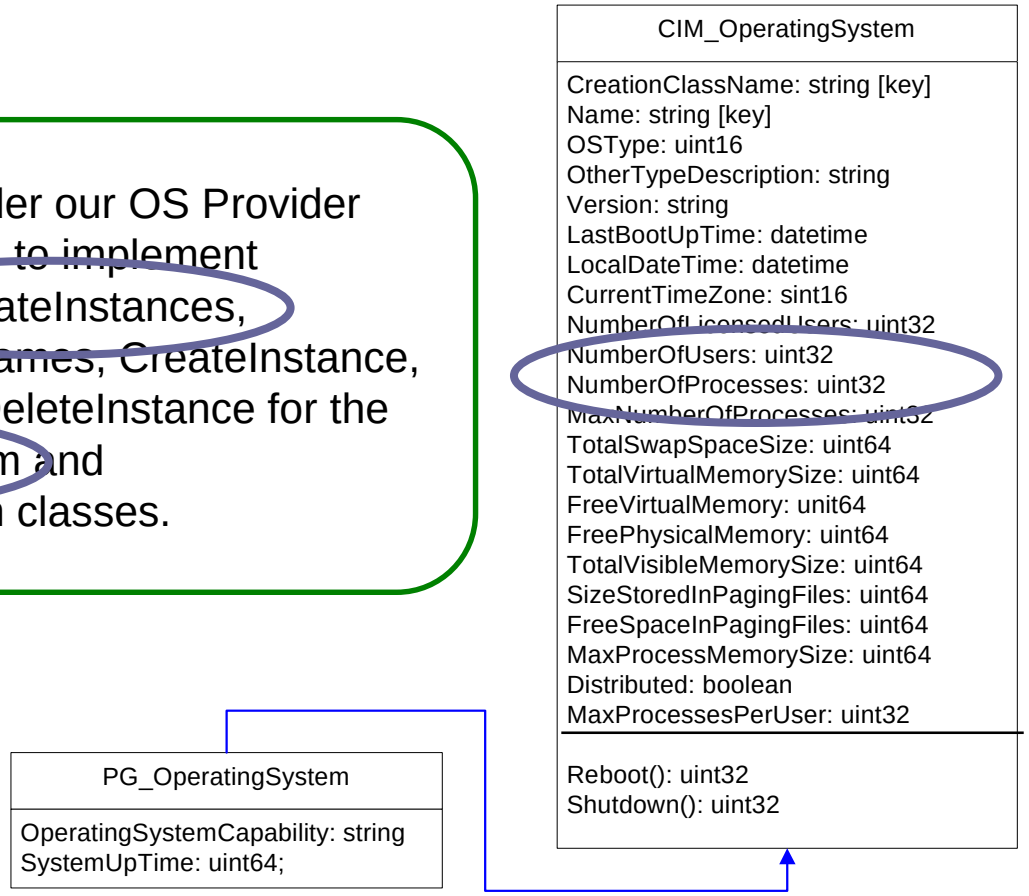
What is the value of CIM_OperatingSystem.NumberOfProcesses?

12345

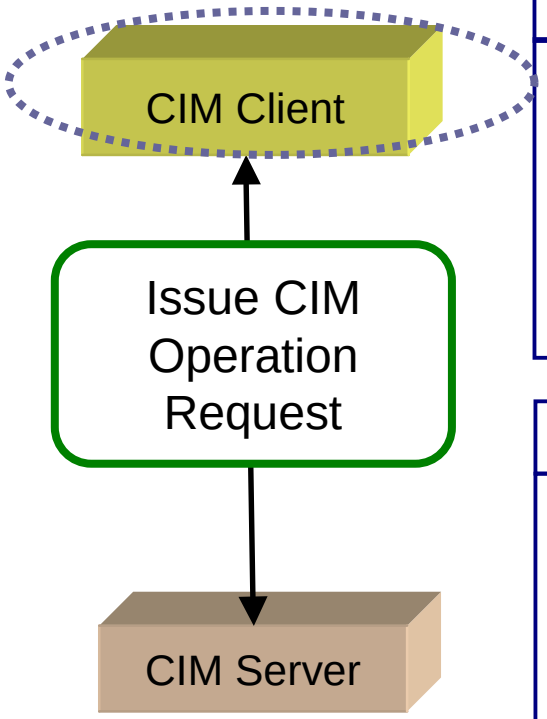


Operating System Provider

As an Instance Provider our OS Provider replacement will need to implement GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, ModifyInstance and DeleteInstance for the CIM_OperatingSystem and PG_OperatingSystem classes.



EnumerateInstance



CIM Operation

EnumerateInstances

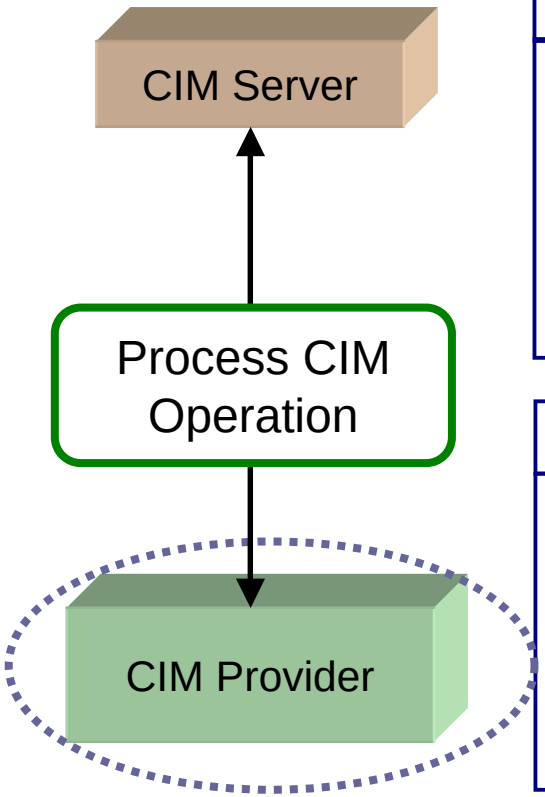
```
<namedInstance>*EnumerateInstances (
        [IN] <className> ClassName,
        [IN,OPTIONAL] boolean LocalOnly = true,
        [IN,OPTIONAL] boolean DeepInheritance = true,
        [IN,OPTIONAL] boolean IncludeQualifiers = false,
        [IN,OPTIONAL] boolean IncludeClassOrigin = false,
        [IN,OPTIONAL,NULL] string PropertyList [] = NULL
)
```

OpenPegasus C++ Client API

enumerateInstances

```
Array<CIMInstance> enumerateInstances (
        const CIMNamespaceName& nameSpace,
        const CIMName& className,
        Boolean deepInheritance = true,
        Boolean localOnly = true,
        Boolean includeQualifiers = false,
        Boolean includeClassOrigin = false,
        const CIMPropertyList& propertyList = CIMPropertyList()
);
```

EnumerateInstance



CIM Operation

EnumerateInstances

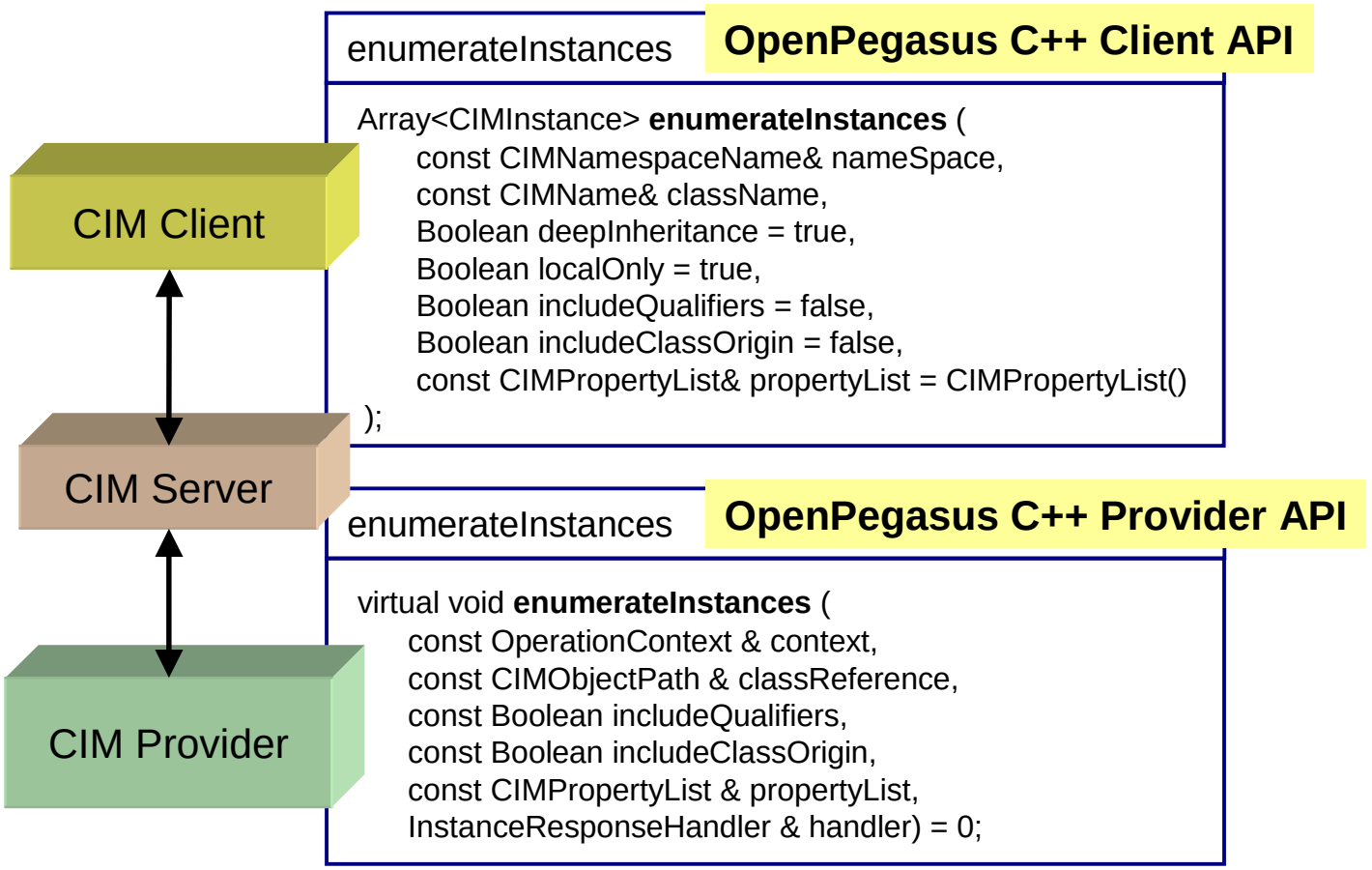
```
<namedInstance>*EnumerateInstances (
        [IN] <className> ClassName,
        [IN,OPTIONAL] boolean LocalOnly = true,
        [IN,OPTIONAL] boolean DeepInheritance = true,
        [IN,OPTIONAL] boolean IncludeQualifiers = false,
        [IN,OPTIONAL] boolean IncludeClassOrigin = false,
        [IN,OPTIONAL,NULL] string PropertyList [] = NULL
 )
```

OpenPegasus C++ Provider API

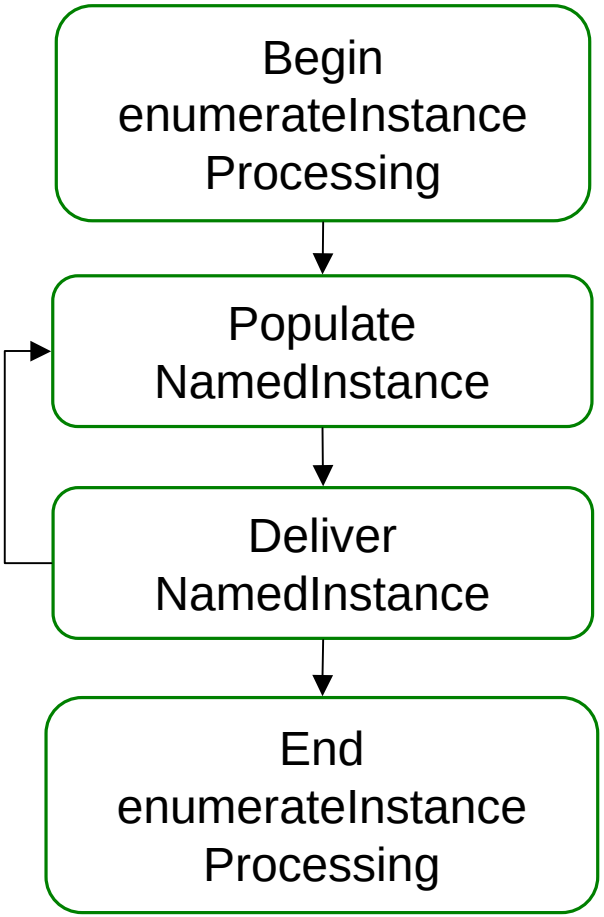
enumerateInstances

```
virtual void enumerateInstances (
      const OperationContext & context,
      const CIMObjectPath & classReference,
      const Boolean includeQualifiers,
      const Boolean includeClassOrigin,
      const CIMPropertyList & propertyList,
      InstanceResponseHandler & handler) = 0;
```

CIM Operation



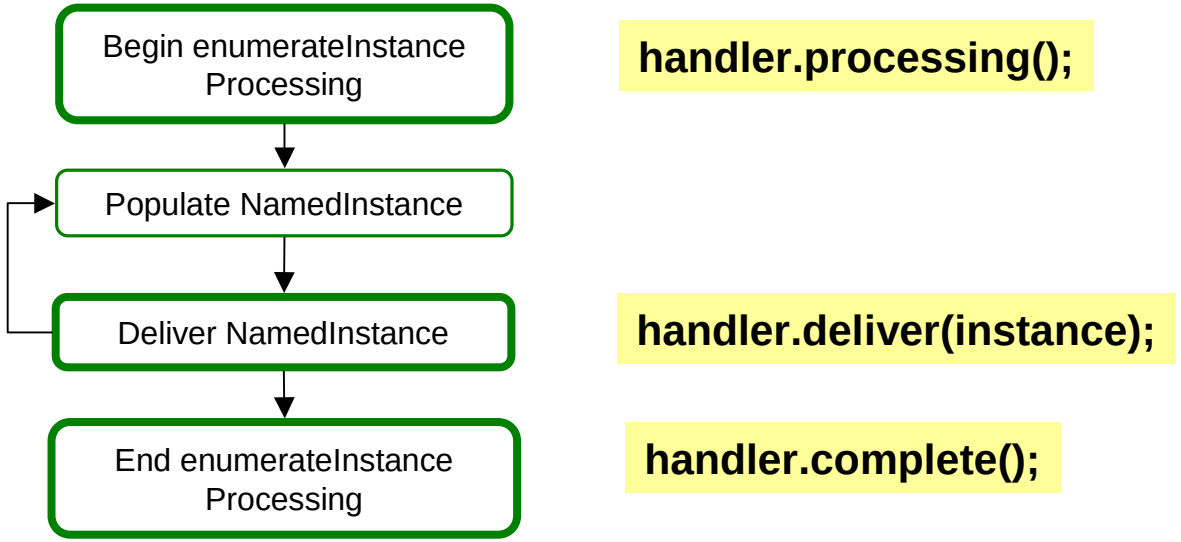
CIM Provider Logic



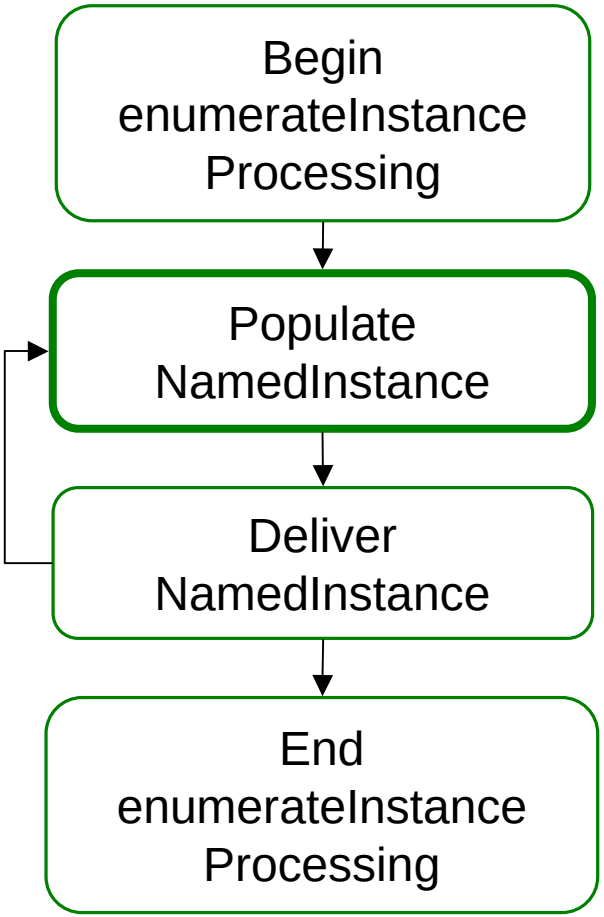
CIM Provider Logic

enumerateInstances **OpenPegasus C++ Provider API**

```
virtual void enumerateInstances (
    const OperationContext & context,
    const CIMObjectPath & classReference,
    const Boolean includeQualifiers,
    const Boolean includeClassOrigin,
    const CIMPropertyList & propertyList,
    InstanceResponseHandler & handler) = 0;
```



CIM Provider Logic



1. **Initialize instance with key properties; set object path.**
2. **Add non-key properties to instance.**

OSInfoProvider.cpp - Notepad

File Edit Format Help

```
}

CIMInstance
OSInfoProvider::_initializeNamedInstance(const CIMObjectPath& classReference,
                                         const String OSName)
{
    Array<CIMKeyBinding> keys;
    CIMInstance instance(classReference.getClassName());

    keys.append(CIMKeyBinding("CSCreationClassName",
        CSCREATIONCLASSNAME.getString(), CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding("CSName", CSNAME,
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding("CreationClassName",
        CREATIONCLASSNAME.getString(), CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding("Name", OSName,
                              CIMKeyBinding::STRING));
    instance.setPath(CIMObjectPath(CSNAME, classReference.getNameSpace(),
                CSCREATIONCLASSNAME.getString(), keys));

    instance.addProperty(CIMProperty("CSCreationClassName",
                             CSCREATIONCLASSNAME.getString()));
    instance.addProperty(CIMProperty("CSName", CSNAME));
    instance.addProperty(CIMProperty("CreationClassName",
                             CREATIONCLASSNAME.getString()));
    instance.addProperty(CIMProperty("Name", OSName));

    instance.setPath(CIMObjectPath(CSNAME, "root/cimv2",
                CSCREATIONCLASSNAME.getString(), keys));

    return(instance);
}

void OSInfoProvider::getInstance(
        const OperationContext & context,
        const CIMObjectPath & instanceReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
```

1. **Initialize instance with key properties**
2. **Set object path.**

OSInfoProvider.cpp - Notepad

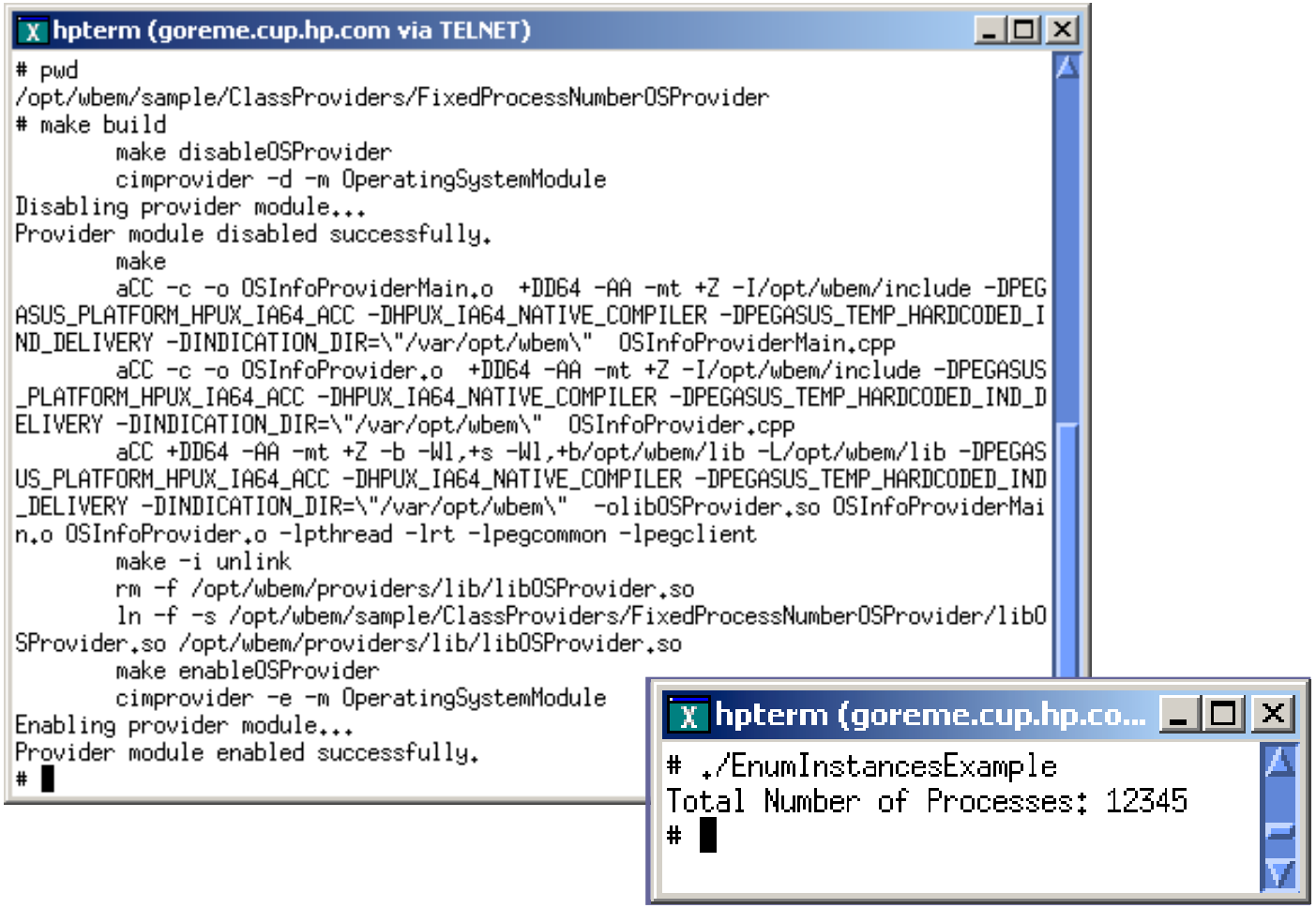
File Edit Format Help

```
{
    throw CIMNotSupportedException("OSInfoProvider "
                        "does not support getInstance");
}

void OSInfoProvider::enumerateInstances(
        const OperationContext & context,
        const CIMObjectPath & classReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
    CIMInstance instance;
    CIMName className = classReference.getClassName();
    CIMObjectPath objectPath;

    if ((className.equal ("PG_OperatingSystem")) ||
        (className.equal ("CIM_OperatingSystem")))
    {
        handler.processing();
        instance = _initializeNamedInstance(classReference, "MyOS");

        Uint32 NumberOfProcesses = 12345;
        instance.addProperty(CIMProperty("NumberOfProcesses",
                                         NumberOfProcesses));

        handler.deliver(instance);
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("OSInfoProvider "
}
vo
```

3. Add non-key properties to instance.

```
Uint32 NumberOfProcesses = 12345;
instance.addProperty(CIMProperty("NumberOfProcesses",
                                 NumberOfProcesses));
```

Build New Version of Provider



More Things to Consider …

enumerateInstances

OpenPegasus C++ Provider API

```
virtual void enumerateInstances (
    const OperationContext & context,
    const CIMObjectPath & classReference,
    const Boolean includeQualifiers,
    const Boolean includeClassOrigin,
    const CIMPropertyList & propertyList,
    InstanceResponseHandler & handler) = 0;
```

const OperationContext & context

const Boolean includeQualifiers

const Boolean includeClassOrigin

const CIMPropertyList & propertyList

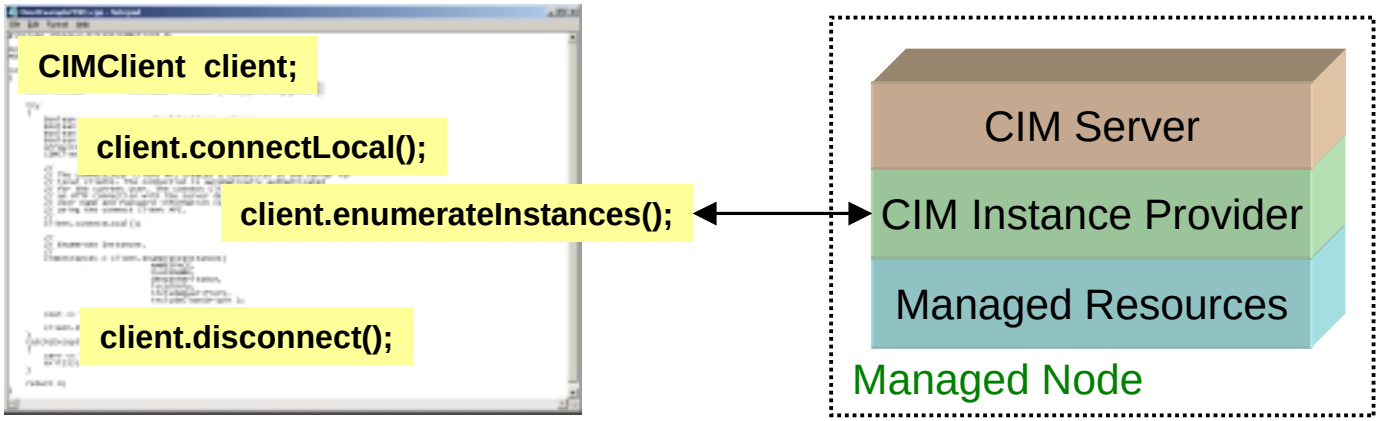
Module Content

C++ Provider Overview

- Concept Overview
- Provider Example
- **Instance Provider API**
- Method Provider API

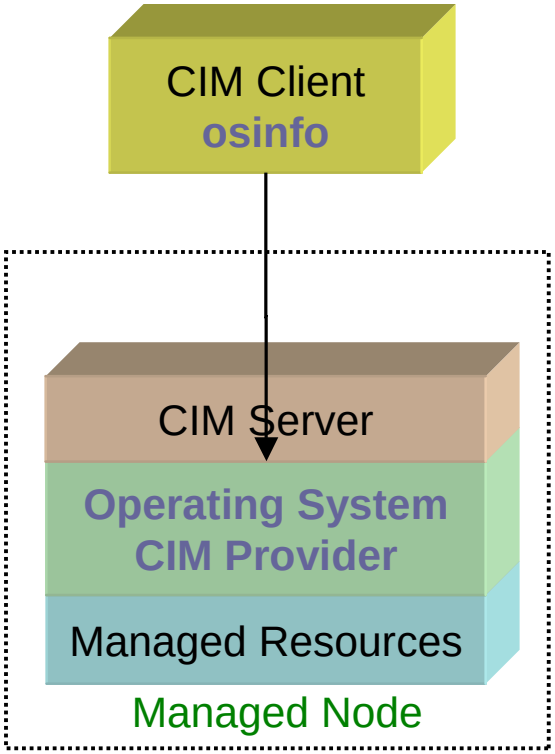
Instance Provider

CIM Operation	Implementation Owner
GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, ModifyInstance, DeleteInstance	**Instance Provider**



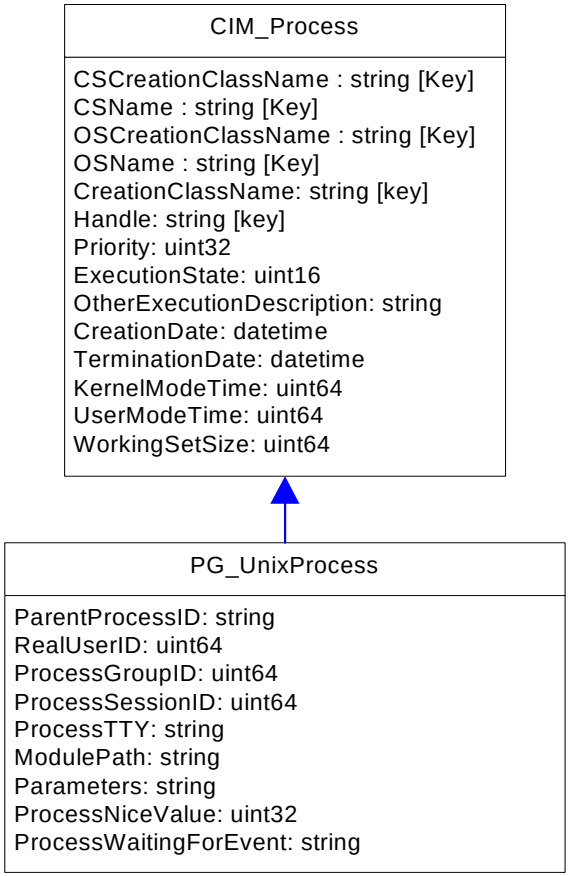
Instance Provider Example

```
bodie
# osinfo
OperatingSystem Information
  Host: bodie.cup.hp.com
  Name: HP-UX
  Version: B.11.00
  UserLicense: Unlimited user license
  OSCapability: 32 bit
  LastBootTime: May 12, 2003  8:54:59 (-0700)
  LocalDateTime: Jun 14, 2003  10:46:5 (-0700)
  SystemUpTime: 2857866 seconds = 33 days, 1 hr, 51 mins, 6 secs
#
```



Packaged with CIM Server is an **Operating System Provider** that serves as an **Instance Provider** for the **CIM_OperatingSystem** and **PG_OperatingSystem** classes in the **root/cimv2** namespace.

Process Provider



In this module, we'll replace the packaged **Process Provider** with our own, significantly less useful, implementation.

Process Provider Registration

cimmof –nroot/PG_InterOp PG_UnixProcess20R.mof

PG_OperatingSystem20R.mof

PG_ProviderModule

```
instance of PG_ProviderModule
{
   Name = "ProcessModule";
   Location = "ProcessProvider";
   Vendor = "Pegasus Community";
   Version = "2.0.0";
   InterfaceType = "C++Default";
   InterfaceVersion = "2.1.0";
};

// Provider for PG_UnixProcess

instance of PG_Provider
{
  ProviderModuleName = "ProcessModule";
  Name = "ProcessProvider";
};
```

PG_Provider

PG_ProviderCapabilities

```
instance of PG_ProviderCapabilities
{
  ProviderModuleName = "ProcessModule";
  ProviderName = "ProcessProvider";
  CapabilityID = "1";
  ClassName = "CIM_Process";
  Namespaces = { "root/cimv2" };
  ProviderType = { 2 }; // Instance
  SupportedProperties = NULL; // All properties
  SupportedMethods = NULL; // All methods
};

instance of PG_ProviderCapabilities
{
  ProviderModuleName = "ProcessModule";
  ProviderName = "ProcessProvider";
  CapabilityID = "2";
  ClassName = "PG_UnixProcess";
  Namespaces = { "root/cimv2" };
  ProviderType = { 2 }; // Instance
  SupportedProperties = NULL; // All properties
  SupportedMethods = NULL; // All methods
};
```

Build New Version of Provider

hpterm (goreme.cup.hp.com via TELNET)

```
# pwd
/opt/wbem/sample/ClassProviders/ProcessProvider1
# make build
        make disableProcessProvider
        cimprovider -d -m ProcessModule
Disabling provider module...
Provider module disabled successfully.
        make
        aCC -c -o ProcessInfoProvider.o  +DD64 -AA -mt +Z -I/opt/wbem/include -DPEGASUS_PLA
HPUX_IA64_ACC -DHPUX_IA64_NATIVE_COMPILER -DPEGASUS_TEMP_HARDCODED_IND_DELIVERY -DINDICATIO
\"/var/opt/wbem\"  ProcessInfoProvider.cpp
        aCC +DD64 -AA -mt +Z -b -Wl,+s -Wl,+b/opt/wbem/lib -L/opt/wbem/lib -DPEGASUS_PLATFO
X_IA64_ACC -DHPUX_IA64_NATIVE_COMPILER -DPEGASUS_TEMP_HARDCODED_IND_DELIVERY -DINDICATION_D
var/opt/wbem\"  -olibProcessProvider.so ProcessInfoProviderMain.o ProcessInfoProvider.o -lp
 -lrt -lpegcommon -lpegclient
        make -i unlink
        rm -f /opt/wbem/providers/lib/libProcessProvider.so
        ln -f -s /opt/wbem/sample/ClassProviders/ProcessProvider1/libProcessProvider.so /op
/providers/lib/libProcessProvider.so
        make enableProcessProvider
        cimprovider -e -m ProcessModule
Enabling provider module...
Provider module enabled successfully.
#
```

getInstance

CIM Operation
GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, ModifyInstance, DeleteInstance

getInstance — **OpenPegasus C++ Client API**

```
CIMInstance getInstance(
    const CIMNamespaceName& nameSpace,
    const CIMObjectPath& instanceName,
    Boolean localOnly = true,
    Boolean includeQualifiers = false,
    Boolean includeClassOrigin = false,
    const CIMPropertyList& propertyList = CIMPropertyList()
  );
```

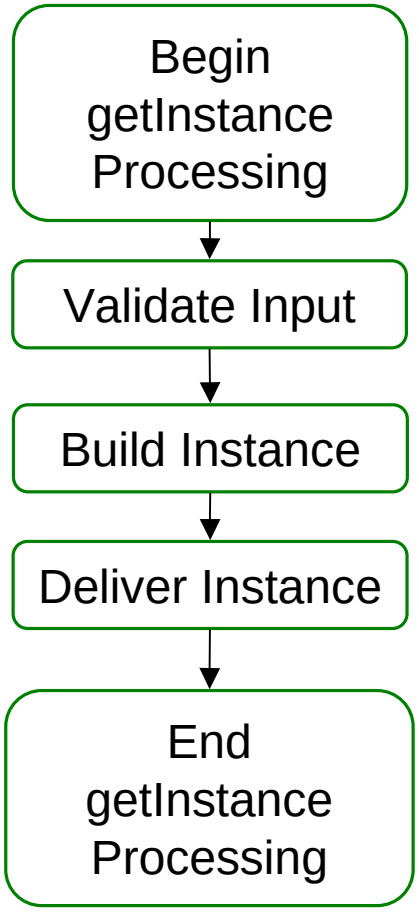
CIM Client

getInstance — **OpenPegasus C++ Provider API**

```
void getInstance(
    const OperationContext & context,
    const CIMObjectPath & instanceReference,
    const Boolean includeQualifiers,
    const Boolean includeClassOrigin,
    const CIMPropertyList & propertyList,
    InstanceResponseHandler & handler) = 0;
```

CIM Provider

CIM Provider Logic



ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::getInstance(
        const OperationContext & context,
        const CIMObjectPath & instanceReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
    CIMInstance instance;
    CIMName className = instanceReference.getClassName();
    CIMObjectPath objectPath;

    if ((className.equal ("PG_UnixProcess")) ||
        (className.equal ("CIM_Process")))
    {
        Array<CIMKeyBinding> keyBindings = instanceReference.getKeyBindings();
        Uint32 numberOfKeys = keyBindings.size();
        Uint32 index = 0;
        Uint32 pid = 0;
        CIMKeyBinding keyBinding;
        CIMName propertyName;

        while (index < numberOfKeys)
        {
            keyBinding = keyBindings[index++];
            propertyName = keyBinding.getName();

            if (propertyName.equal(HANDLEPROPERTYNAME))
            {
                pid = atoi(keyBinding.getValue().getCString());
                break;
            }
        }

        if ((pid > NUMBEROFRUNNINGPROCESSES) || (pid <= 0))
        {
            throw CIMInvalidParameterException("Invalid Handle");
        }

        handler.processing();
        instance = _initializeNamedInstanceKeys(instanceReference,
        _initializeNamedInstanceNonKeys(instance, pid);
        handler.deliver(instance);
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider "
                "does not support class " + className.getString());
    }
}
```

Validate Input

Build Instance

Deliver Instance

ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::getInstance(
        const OperationContext & context,
        const CIMObjectPath & instanceReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
    CIMInstance instance;
    CIMName className = instanceReference.getClassName();
    CIMObjectPath objectPath;

    if ((className.equal ("PG_UnixProcess")) ||
        (className.equal ("CIM_Process")))
    {
        Array<CIMKeyBinding> keyBindings = instanceReference.getKeyBindings();
        Uint32 numberOfKeys = keyBindings.size();
        Uint32 index = 0;
        Uint32 pid = 0;
        CIMKeyBinding keyBinding;
        CIMName propertyName;

        while (index < numberOfKeys)
        {
           keyBinding = keyBindings[index++];
           propertyName = keyBinding.getName();

           if (propertyName.equal(HANDLEPROPERTYNAME))
           {
              pid = atoi(keyBinding.getValue().getCString());
              break;
           }
        }

        if ((pid > NUMBEROFRUNNINGPROCESSES) || (pid <= 0))
        {
           throw CIMInvalidParameterException("Invalid Handle");
        }

        handler.processing();
        instance = _initializeNamedInstanceKeys(instanceReference, pid);
        _initializeNamedInstanceNonKeys(instance, pid);
        handler.deliver(instance);
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider "
                "does not support class " + className.getString());
    }
}
```

Validate Input

Build Instance
_initializeNamedInstanceKeys()
_initializeNamedInstanceNonKeys()

ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
CIMInstance ProcessInfoProvider::_initializeNamedInstanceKe
    const CIMObjectPath& classReference,
    Uint32 pid)
{
    Array<CIMKeyBinding> keys;
    CIMName className = classReference.getClassName();
    CIMInstance instance(className);
    char pidValue[32];
    sprintf(pidValue, "%u", pid);

    keys.append(CIMKeyBinding(CSCREATIONCLASSPROPERTYNAME,
                              CSCREATIONCLASSNAME.getString(),
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(CSNAMEPROPERTYNAME,
                              CSNAME, CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(OSCREATIONCLASSPROPERTYNAME,
                              OSCREATIONCLASSNAME.getString(),
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(OSNAMEPROPERTYNAME,
                              OSNAME, CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(CREATIONCLASSPROPERTYNAME,
                              CREATIONCLASSNAME.getString(),
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(HANDLEPROPERTYNAME, String(pidValue),
                              CIMKeyBinding::STRING));

    instance.setPath(CIMObjectPath(CSNAME, classReference.getNameSpace(),
                     className, keys));

    instance.addProperty(CIMProperty(CSCREATIONCLASSPROPERTYNAME,
                                     CSCREATIONCLASSNAME.getString()));
    instance.addProperty(CIMProperty(CSNAMEPROPERTYNAME, CSNAME));
    instance.addProperty(CIMProperty(OSCREATIONCLASSPROPERTYNAME,
                                     OSCREATIONCLASSNAME.getString()));
    instance.addProperty(CIMProperty(OSNAMEPROPERTYNAME, OSNAME));
    instance.addProperty(CIMProperty(CREATIONCLASSPROPERTYNAME,
                                     CREATIONCLASSNAME.getString()));
    instance.addProperty(CIMProperty(HANDLEPROPERTYNAME, String(pidValue)));

    instance.setPath(CIMObjectPath(CSNAME, "root/cimv2",
                     CSCREATIONCLASSNAME.getString(), keys));

    return(instance);
}
```

Build Instance
_initializeNamedInstanceKeys()

getInstance

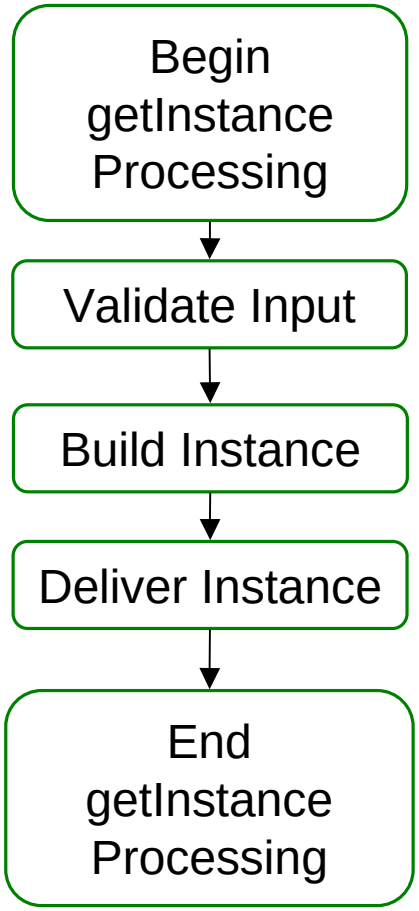
Build Instance
_initializeNamedInstanceKeys()
_initializeNamedInstanceNonKeys()

ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::_initializeNamedInstanceNonKeys(
    CIMInstance & instance,
    Uint32 pid)
{
   // Note:  This example uses contrived values for Process properties.
   // In the real world, these values would come from the underlying system.

   // For this example, the Priority of a process is computed using
   // the formula pid % 3 + 1.
   Uint32 priority = pid % 3 + 1;
   instance.addProperty(CIMProperty(PRIORITYPROPERTYNAME, priority));

   // For this example, the Name of a process is constructed as
   // the concatenation of the word "Process" followed by the
   // value of the PID.
   char name[32];
   sprintf(name, "Process %u", pid);
   instance.addProperty(CIMProperty(NAMEPROPERTYNAME, String(name)));
}
```

CIM Provider Logic



ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::getInstance(
        const OperationContext & context,
        const CIMObjectPath & instanceReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
    CIMInstance instance;
    CIMName className = instanceReference.getClassName();
    CIMObjectPath objectPath;

    if ((className.equal ("PG_UnixProcess")) ||
        (className.equal ("CIM_Process")))
    {
        Array<CIMKeyBinding> keyBindings = instanceReference.getKeyBindings();
        Uint32 numberOfKeys = keyBindings.size();
        Uint32 index = 0;
        Uint32 pid = 0;
        CIMKeyBinding keyBinding;
        CIMName propertyName;

        while (index < numberOfKeys)
        {
            keyBinding = keyBindings[index++];
            propertyName = keyBinding.getName();

            if (propertyName.equal(HANDLEPROPERTYNAME))
            {
                pid = atoi(keyBinding.getValue().getCString());
                break;
            }
        }

        if ((pid > NUMBEROFRUNNINGPROCESSES) || (pid <= 0))
        {
            throw CIMInvalidParameterException("Invalid Handle");
        }

        handler.processing();
        instance = _initializeNamedInstanceKeys(instanceReference, pid);
        _initializeNamedInstanceNonKeys(instance, pid);
        handler.deliver(instance);
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider "
                "does not support class " + className.getString());
    }
}
```

Deliver Instance

enumerateInstances

CIM Operation
GetInstance, **EnumerateInstances,** EnumerateInstanceNames, CreateInstance, ModifyInstance, DeleteInstance

enumerateInstances **OpenPegasus C++ Client API**

```
Array<CIMInstance> enumerateInstances (
        const CIMNamespaceName& nameSpace,
        const CIMName& className,
        Boolean deepInheritance = true,
        Boolean localOnly = true,
        Boolean includeQualifiers = false,
        Boolean includeClassOrigin = false,
        const CIMPropertyList& propertyList = CIMPropertyList()
 );
```

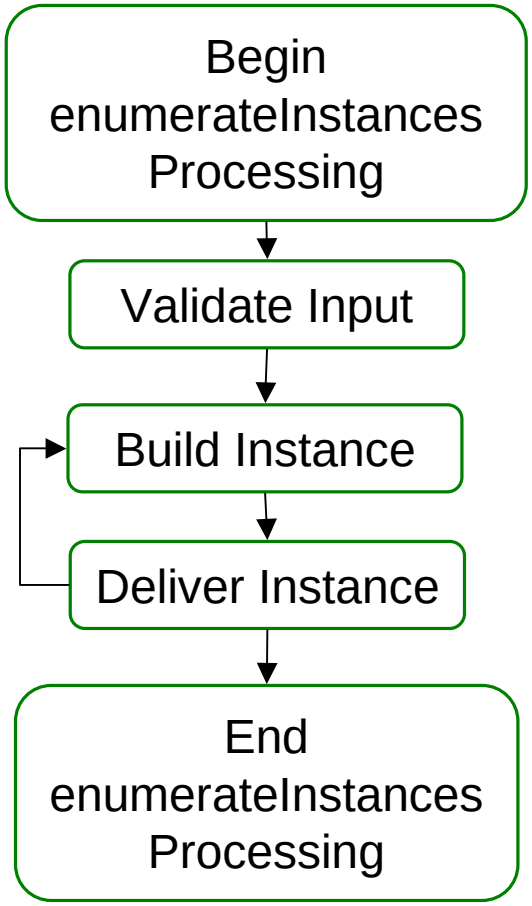
CIM Client

enumerateInstances **OpenPegasus C++ Provider API**

```
virtual void enumerateInstances (
        const OperationContext & context,
        const CIMObjectPath & classReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler) = 0;
```

CIM Provider

CIM Provider Logic



ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::enumerateInstances(
        const OperationContext & context,
        const CIMObjectPath & classReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
    CIMInstance instance;
    CIMName className = classReference.getClassName();
    CIMObjectPath objectPath;

    // Note: we need to be careful that we don't
    // enumerate instances for both the CIM_Process
    // class and the PG_UnixProcess class.  We
    // need to avoid creating duplicates.

    if (className.equal ("PG_UnixProcess"))
    {
        handler.processing();

        for (Uint32 i = 1; i <= NUMBEROFRUNNINGPROCESSES; i++)
        {
           instance = _initializeNamedInstanceKeys(classReference, i);
           _initializeNamedInstanceNonKeys(instance, i);
           handler.deliver(instance);
        }
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider "
                "does not support class " + className.getString());
    }
}
```

Validate Input

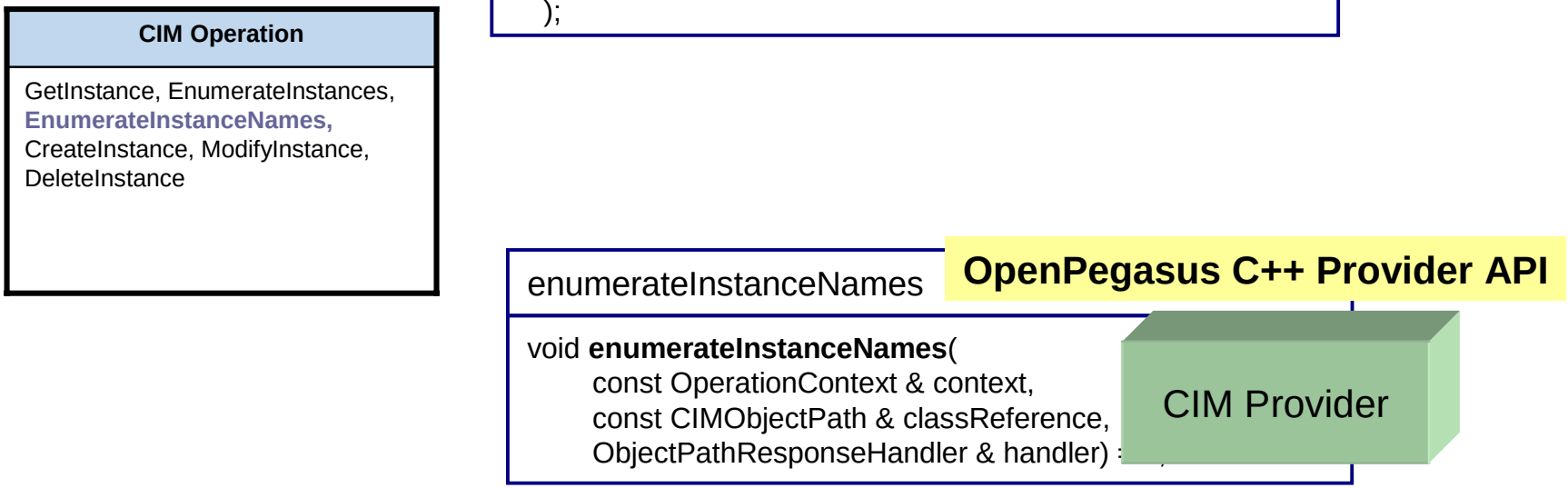
Build and Deliver Instances

ProcessInfoProvider.cpp - Notepad

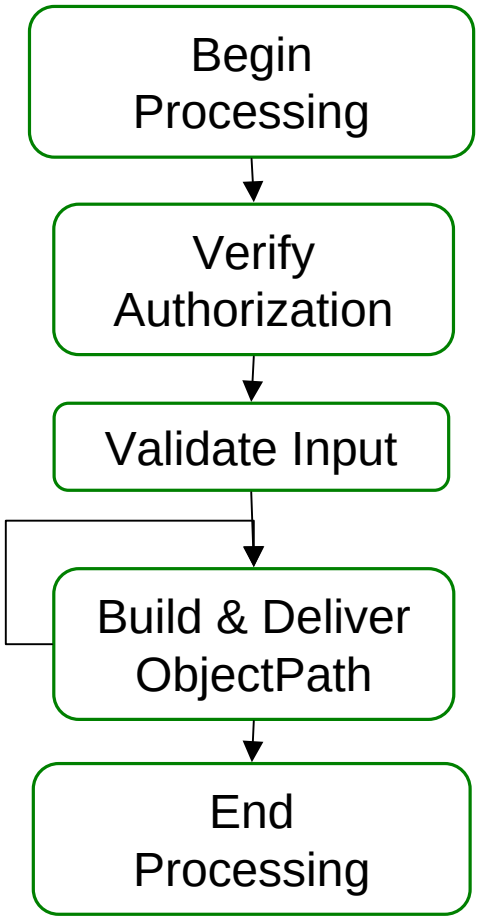
File Edit Format Help

```
void ProcessInfoProvider::enumerateInstances(
        const OperationContext & context,
        const CIMObjectPath & classReference,
        const Boolean includeQualifiers,
        const Boolean includeClassOrigin,
        const CIMPropertyList & propertyList,
        InstanceResponseHandler & handler)
{
    CIMInstance instance;
    CIMName className = classReference.getClassName();
    CIMObjectPath objectPath;

    // Note: we need to be careful that we don't
    // enumerate instances for both the CIM_Process
    // class and the PG_UnixProcess class.  We
    // need to avoid creating duplicates.

    if (className.equal ("PG_UnixProcess"))
    {
        handler.processing();

        for (Uint32 i = 1; i <= NUMBEROFRUNNINGPROCESSES; i++)
        {
           instance = _initializeNamedInstanceKeys(classReference, i);
           _initializeNamedInstanceNonKeys(instance, i);
           handler.deliver(instance);
        }
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider "
                "does not support class " + className.getString());
    }
}
```

enumerateInstanceNames



CIM Provider Logic



ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::enumerateInstanceNames(
        const OperationContext & context,
        const CIMObjectPath & classReference,
        ObjectPathResponseHandler & handler)
{
    _verifyAuthorization(context);

    CIMObjectPath objectPath;
    CIMName className = classReference.getClassName();

    // Note: we need to be careful that we don't
    // enumerate instances for both the CIM_Process
    // class and the PG_UnixProcess class.  We
    // need to avoid creating duplicates.

    if (className.equal ("PG_UnixProcess"))
    {
        handler.processing();

        for (Uint32 i = 1; i <= NUMBEROFRUNNINGPROCESSES; i++)
        {
           objectPath = _initializeObjectPath(classReference, i);
           handler.deliver(objectPath);
        }
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider
                "does not support class " + className.getString());
    }
}
```

Verify Authorization

Validate Input

Build and Deliver ObjectPaths

enumerateInstanceNames

ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::enumerateInstanceNames(
        const OperationContext & context,
        const CIMObjectPath & classReference,
        ObjectPathResponseHandler & handler)
{
    _verifyAuthorization(context);

    CIMObjectPath objectPath;
    CIMName className = classReference.getClassName();

    // Note: we need to be careful that we don't
    // enumerate instances for both the CIM_Process
    // class and the PG_UnixProcess class.  We
    // need to avoid creating duplicates.

    if (className.equal ("PG_UnixProcess"))
    {
        handler.processing();

        for (Uint32 i = 1; i <= NUMBEROFRUNNINGPROCESSES; i++)
        {
           objectPath = _initializeObjectPath(classReference, i);
           handler.deliver(objectPath);
        }
        handler.complete();
    }
    else
    {
        throw CIMNotSupportedException("ProcessInfoProvider "
                "does not support class " + className.getString());
    }
}
```

Verify Authorization

Validate Input

Build and Deliver ObjectPaths

enumerateInstanceNames

Verify Authorization

ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::_verifyAuthorization(
        const OperationContext & context)
{
   String userName;
   try
   {
      IdentityContainer container = context.get(IdentityContainer::NAME);
      userName = container.getUserName();
   }
   catch (...)
   {
      throw CIMAccessDeniedException(
          "Must be a valid system user to do this CIM Operation.");
   }
}
```

enumerateInstanceNames

ProcessInfoProvider.cpp - Notepad

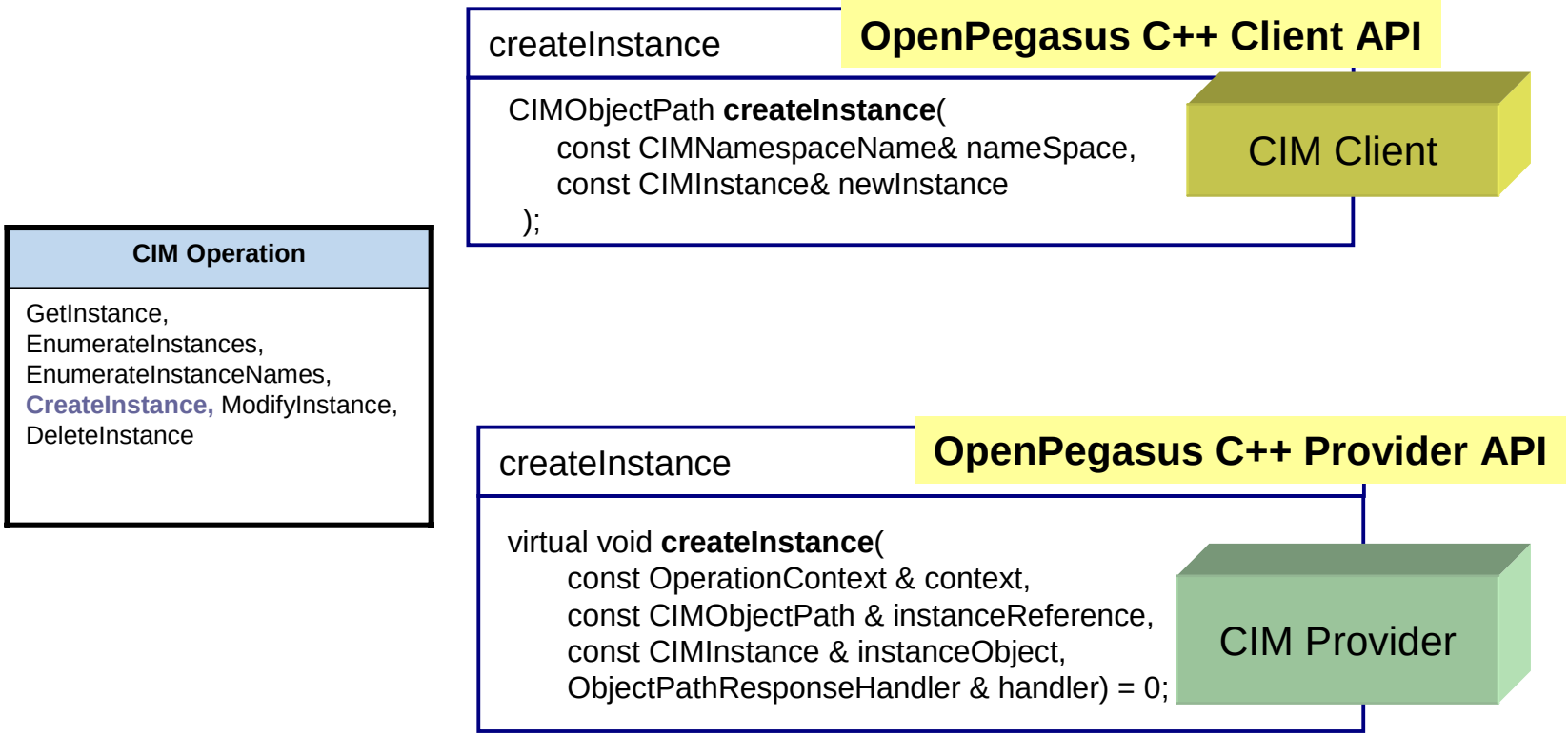
File Edit Format Help

```
CIMObjectPath ProcessInfoProvider::_initializeObjectPath(
    const CIMObjectPath& classReference,
    Uint32 pid)
{
    Array<CIMKeyBinding> keys;
    CIMName className = classReference.getClassName();
    char pidValue[32];
    sprintf(pidValue, "%u", pid);

    keys.append(CIMKeyBinding(CSCREATIONCLASSPROPERTYNAME,
                              CSCREATIONCLASSNAME.getString(),
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(CSNAMEPROPERTYNAME,
                              CSNAME, CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(OSCREATIONCLASSPROPERTYNAME,
                              OSCREATIONCLASSNAME.getString(),
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(OSNAMEPROPERTYNAME,
                              OSNAME, CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(CREATIONCLASSPROPERTYNAME,
                              CREATIONCLASSNAME.getString(),
                              CIMKeyBinding::STRING));
    keys.append(CIMKeyBinding(HANDLEPROPERTYNAME, String(pidValue),
                              CIMKeyBinding::STRING));

    return(CIMObjectPath(CSNAME, classReference.getNameSpace(),
                         className, keys));
}
```

Build and Deliver ObjectPaths

_initializeObjectPath

createInstance



modifyInstance

CIM Operation
GetInstance, EnumerateInstances, EnumerateInstanceNames, CreateInstance, **ModifyInstance,** DeleteInstance

modifyInstances

OpenPegasus C++ Client API

CIM Client

```
void modifyInstance(
        const CIMNamespaceName& nameSpace,
        const CIMInstance& modifiedInstance,
        Boolean includeQualifiers = true,
        const CIMPropertyList& propertyList = CIMPropertyList()
    );
```

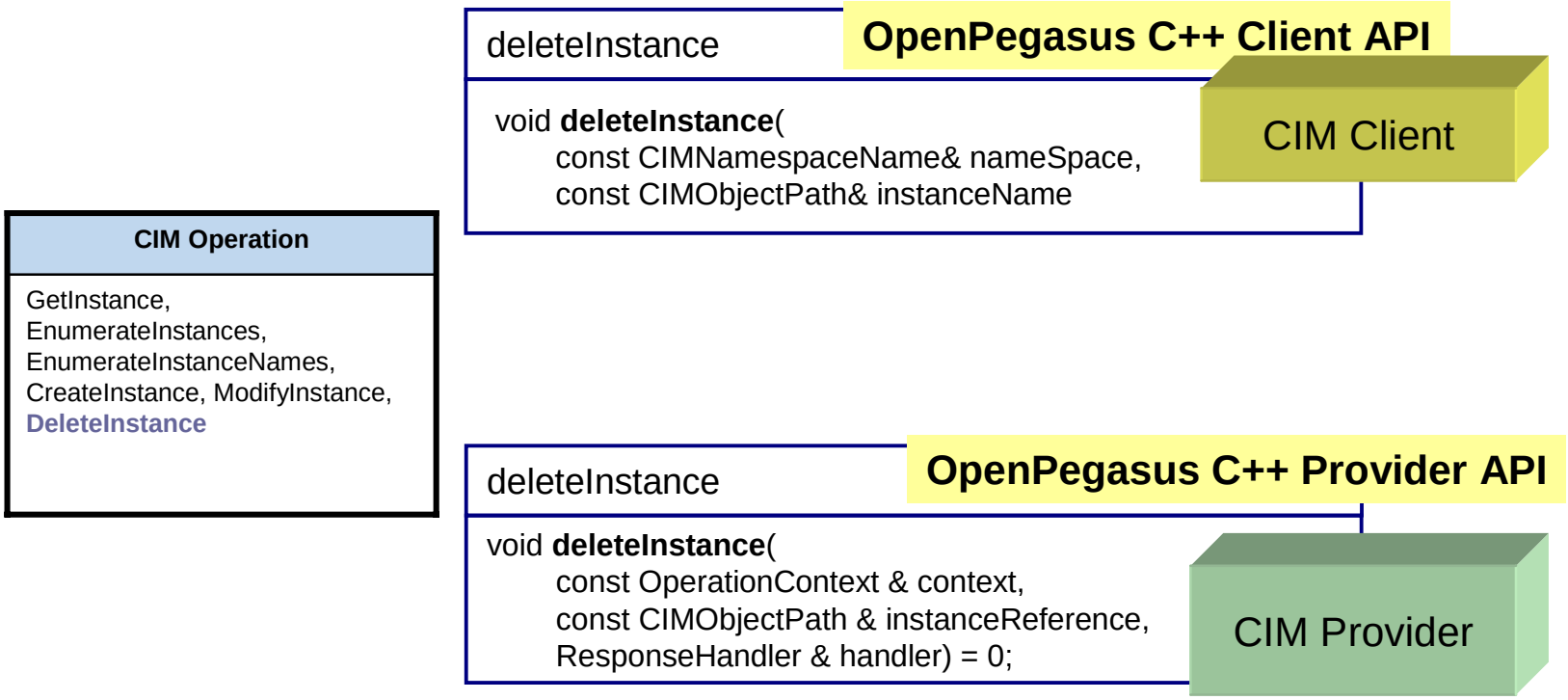
modifyInstance

OpenPegasus C++ Provider API

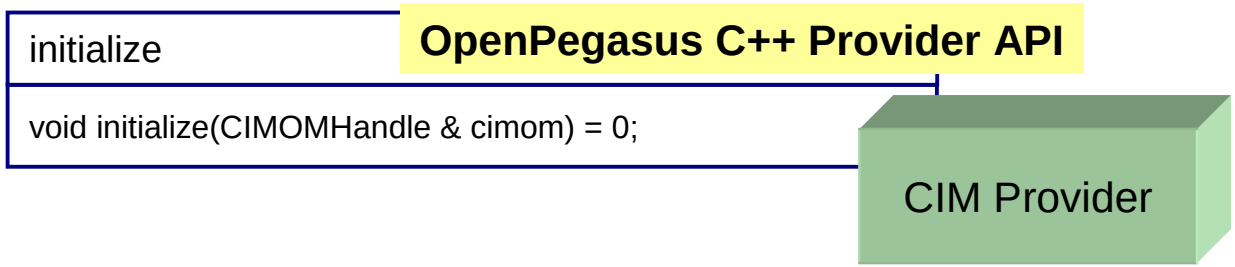
CIM Provider

```
virtual void modifyInstance(
        const OperationContext & context,
        const CIMObjectPath & instanceReference,
        const CIMInstance & instanceObject,
        const Boolean includeQualifiers,
        const CIMPropertyList & propertyList,
        ResponseHandler & handler) = 0;
```

deleteInstance

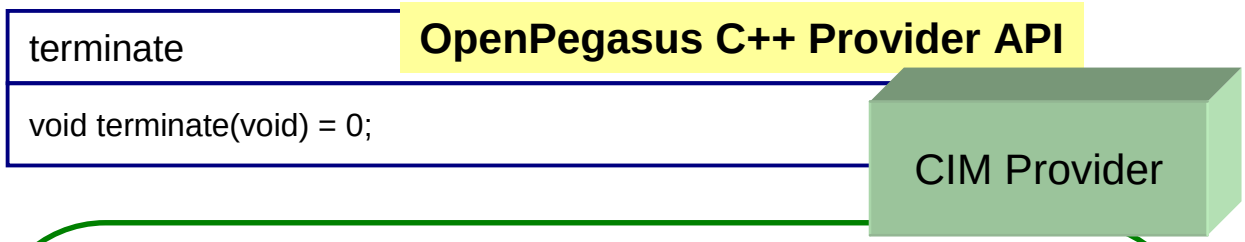


initialize



The ***initialize*** function allows the provider to conduct the necessary preparations to handle requests. It is called only once during the lifetime of the provider. This function must complete before the CIM Server invokes any other function of the provider, other than terminate.

terminate

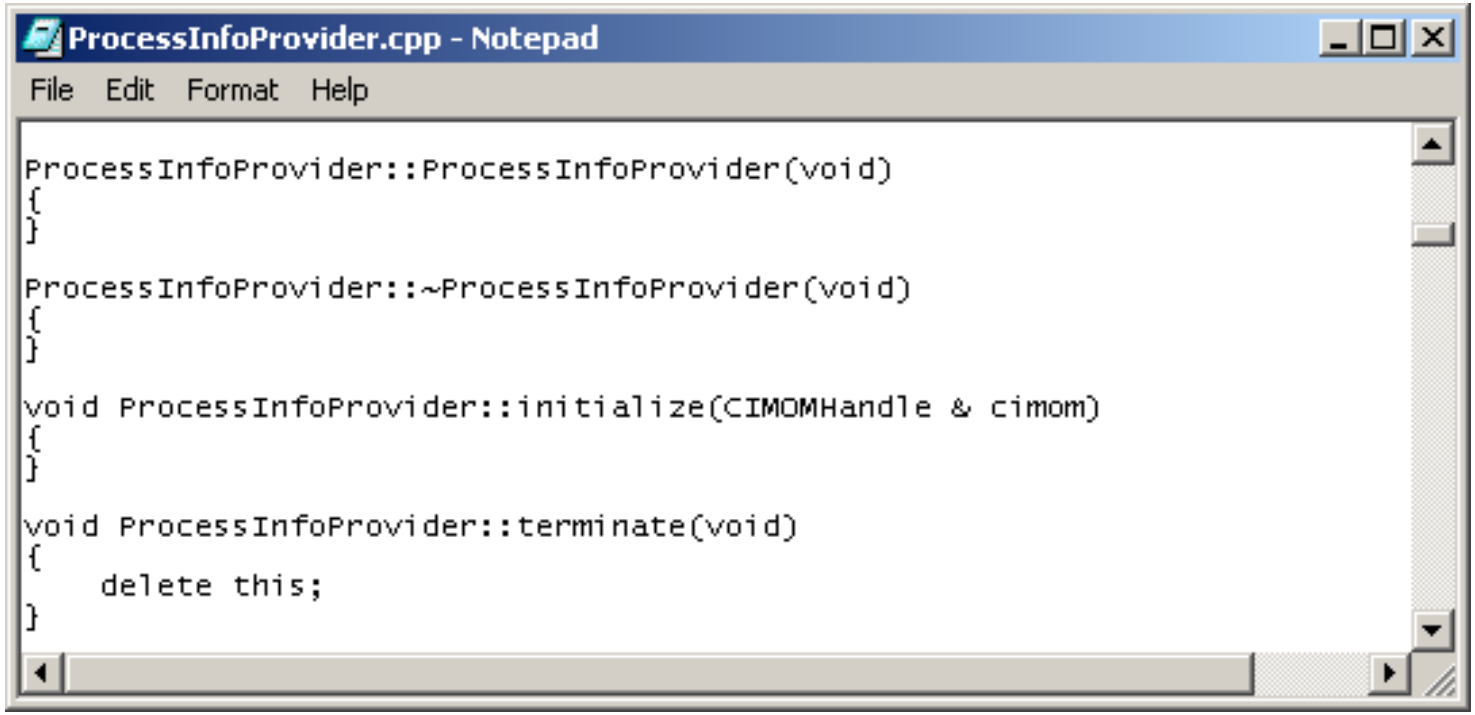


The ***terminate*** function allows the provider to conduct the necessary preparations for termination. This function may be called by the CIM Server at any time, including initialization. Once invoked, no other provider functions are invoked until after an eventual call to initialize.

The provider may, for example, do the following in the ***terminate*** function:

- close files or I/O streams
- release resources such as shared memory
- inform concurrently executing requests to complete immediately
- kill subprocesses
- etc.

Initialize and Terminate



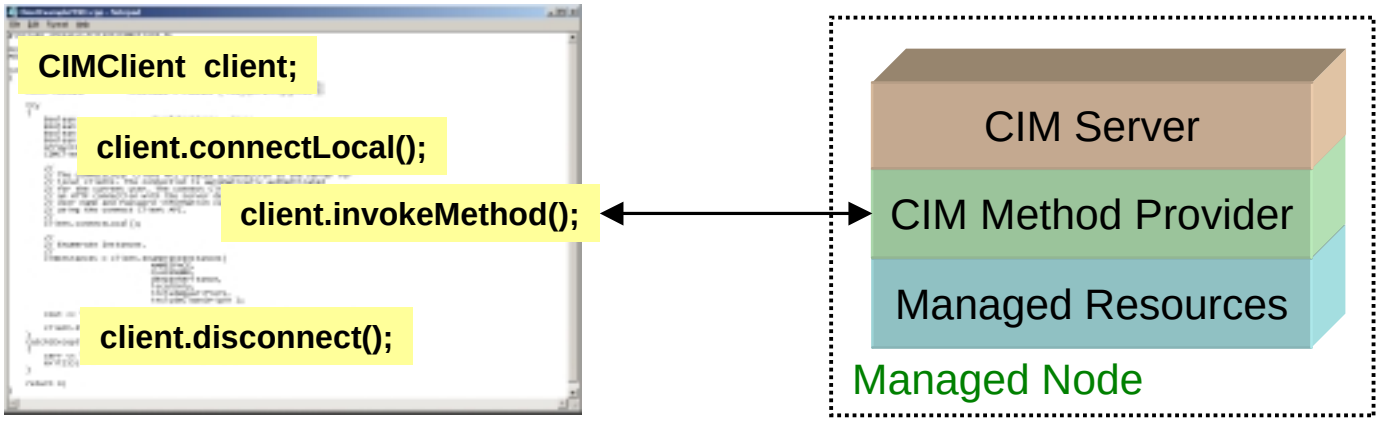
Module Content

C++ Provider Overview

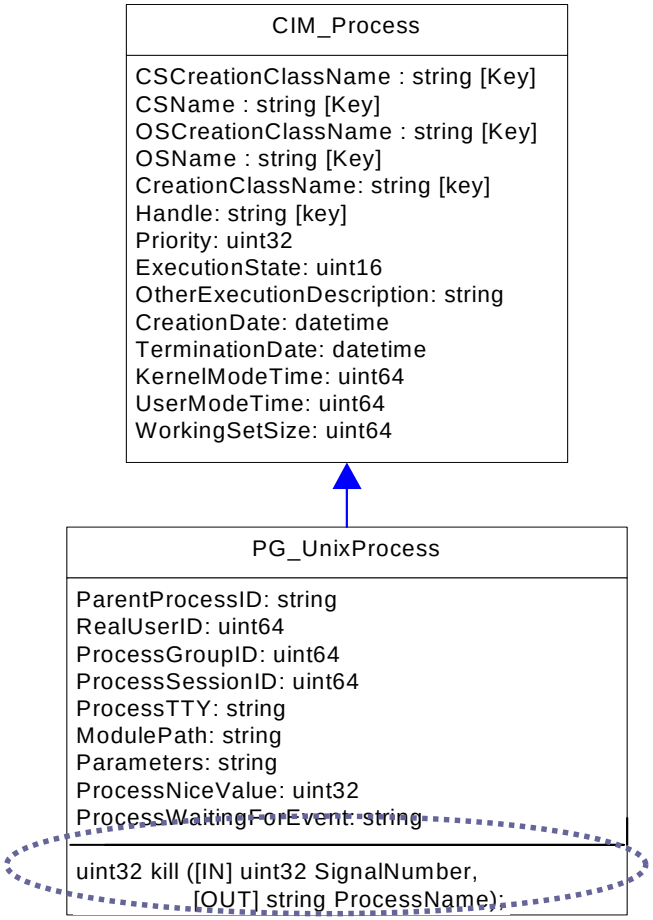
- Concept Overview
- Provider Example
- Instance Provider API
- **Method Provider API**

Method Provider

CIM Operation	Implementation Owner
InvokeMethod	**Method Provider**



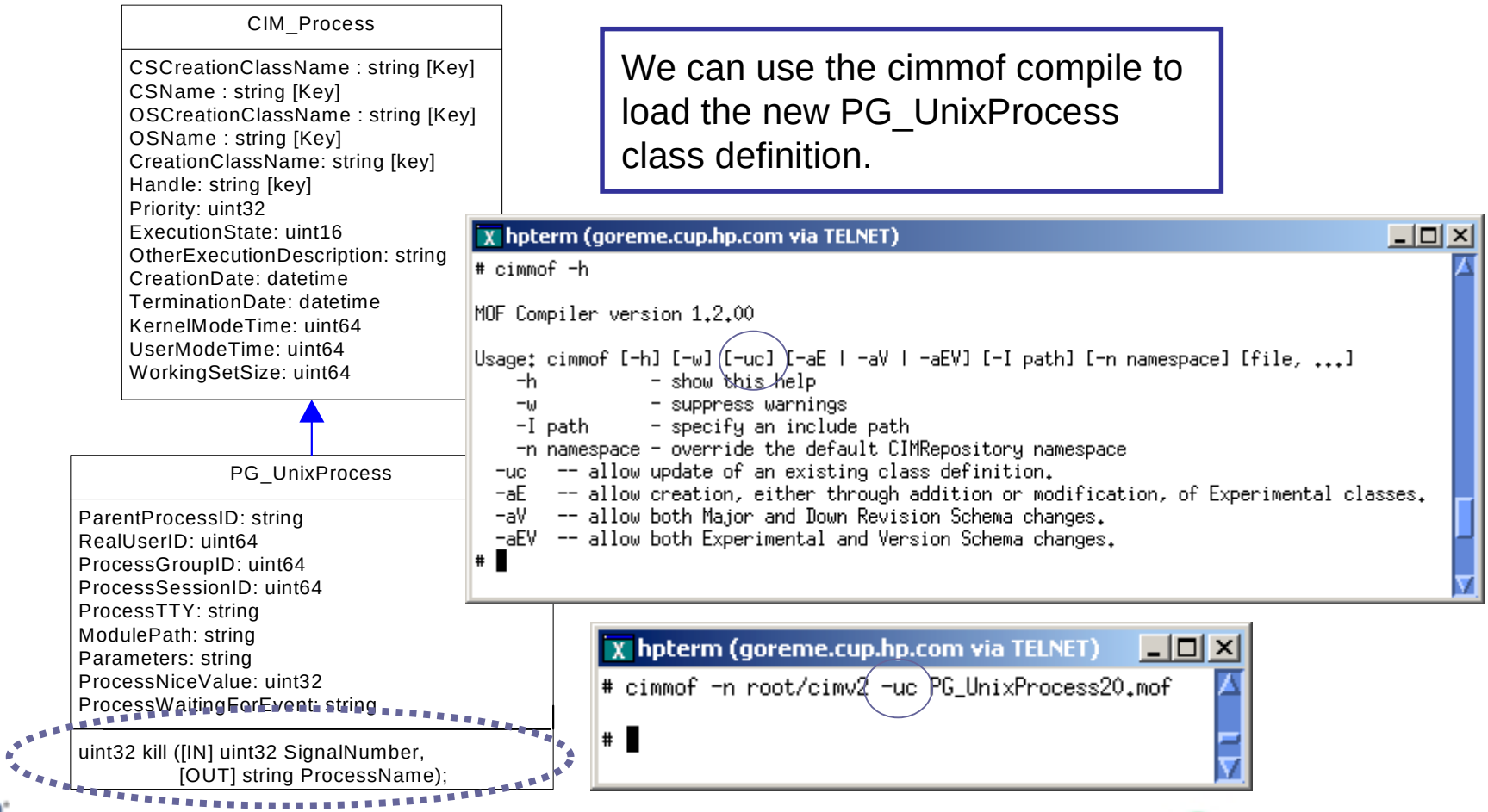
Process Provider



In this module, let's add a "kill" method to the **Process Provider** we developed in the previous module.

To do this, we will need to add the definition of a "kill" method to the PG_UnixProcess class.

PG_UnixProcess Extension



invokeMethod

CIM Operation
invokeMethod

invokeMethod — **OpenPegasus C++ Client API**

```
CIMValue invokeMethod(
        const CIMNamespaceName& nameSpace,
        const CIMObjectPath& instanceName,
        const CIMName& methodName,
        const Array<CIMParamValue>& inParameters,
        Array<CIMParamValue>& outParameters
    );
```

CIM Client

invokeMethod — **OpenPegasus C++ Provider API**

```
void invokeMethod(
        const OperationContext & context,
        const CIMObjectPath & objectReference,
        const CIMName & methodName,
        const Array<CIMParamValue> & inParameters,
        MethodResultResponseHandler & handler) = 0;
```

CIM Provider

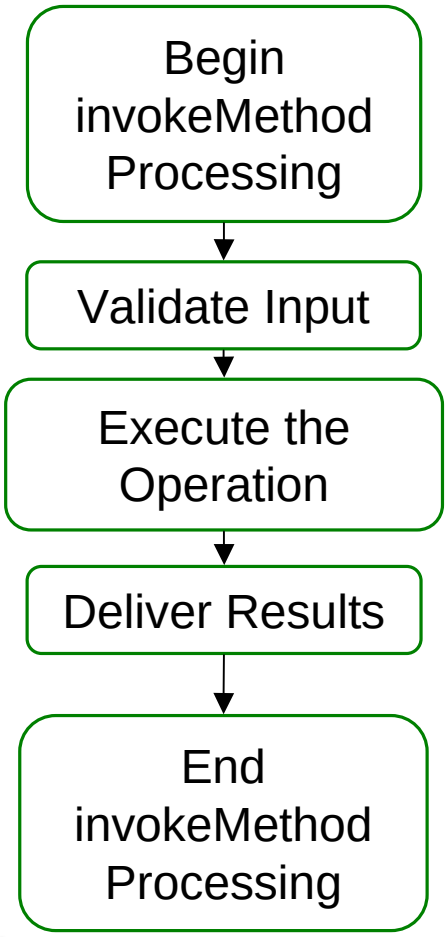
ProcessInfoProvider.h - Notepad

File Edit Format Help

```
#include <Pegasus/Provider/CIMInstanceProvider.h>
#include <Pegasus/Provider/CIMMethodProvider.h>

class ProcessInfoProvider :
        public CIMInstanceProvider, CIMMethodProvider
{
public:
        ProcessInfoProvider(void);
        virtual ~ProcessInfoProvider(void);

        // CIMProvider interface
        virtual void initialize(CIMOMHandle & cimom);
        virtual void terminate(void);

        // CIMInstanceProvider interface
        virtual void getInstance(
                const OperationContext & context,
                const CIMObjectPath & ref,
                const Boolean includeQualifiers,
                const Boolean includeClassOrigin,
                const CIMPropertyList & propertyList,
                InstanceResponseHandler & handler);

        virtual void enumerateInstances(
                const OperationContext & context,
                const CIMObjectPath & ref,
                const Boolean includeQualifiers,
                const Boolean includeClassOrigin,
                const CIMPropertyList & propertyList,
                InstanceResponseHandler & handler);

        virtual void enumerateInstanceNames(
                const OperationContext & context,
                const CIMObjectPath & ref,
                ObjectPathResponseHandler & handler);

        virtual void modifyInstance(
                const OperationContext & context,
                const CIMObjectPath & ref,
                const CIMInstance & obj,
                const Boolean includeQualifiers,
                const CIMPropertyList & propertyList,
                ResponseHandler & handler);

        virtual void createInstance(
                const OperationContext & context,
                const CIMObjectPath & ref,
                const CIMInstance & obj,
```

Note: We need to define the ProcessInfoProvider as both a CIMInstanceProvider and a CIMMethodProvider.

CIM Provider Logic



ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::invokeMethod(
        const OperationContext & context,
        const CIMObjectPath & objectReference,
        const CIMName & methodName,
        const Array<CIMParamValue> & inParameters,
        MethodResultResponseHandler & handler)
{
    // convert a fully qualified reference into a local reference
    // (class name and keys only).
    CIMObjectPath localReference = CIMObjectPath(
                String(),
                String(),
                objectReference.getClassName(),
                objectReference.getKeyBindings());

    handler.processing();

    if (objectReference.getClassName().equal ("PG_Process"))
    {
        if (methodName.equal ("kill"))
        {
            if ((inParameters.size() != 1) ||
                (inParameters[0].getParameterName() == "SignalNumber"))
            {
                throw CIMInvalidParameterException("Invalid signal");
            }
        }
        else
        {
            throw CIMInvalidParameterException("Invalid method name");
        }

        Array<CIMKeyBinding> keyBindings = objectReference.getKeyBindings();
        Uint32 pid = _getPID(keyBindings);

        Uint32 signalNumber;
        CIMValue paramVal = inParameters[0].getValue();
        paramVal.get(signalNumber);

        // execute kill -s signal pid

        // For this example, the Name of a process is constructed as
        // the concatenation of the word "Process" followed by the
        // value of the PID.
        char name[32];
        sprintf(name, "Process %u", pid);
        handler.deliverParamValue(CIMParamValue("ProcessName", String(name)));

        // Return error status of kill operation
        handler.deliver(CIMValue(Uint32(0)));
        handler.complete();
    }
}
```

Validate Input

Execute the operation

Deliver Results

ProcessInfoProvider.cpp - Notepad

File Edit Format Help

```
void ProcessInfoProvider::invokeMethod(
        const OperationContext & context,
        const CIMObjectPath & objectReference,
        const CIMName & methodName,
        const Array<CIMParamValue> & inParameters,
        MethodResultResponseHandler & handler)
{
    // convert a fully qualified reference into a local reference
    // (class name and keys only).
    CIMObjectPath localReference = CIMObjectPath(
                String(),
                String(),
                objectReference.getClassName(),
                objectReference.getKeyBindings());

    handler.processing();

    if (objectReference.getClassName().equal ("PG_Process"))
    {
       if (methodName.equal ("kill"))
       {
          if ((inParameters.size() != 1) ||
              (inParameters[0].getParameterName() == "SignalNumber"))
          {
             throw CIMInvalidParameterException("Invalid signal");
          }
       }
       else
       {
          throw CIMInvalidParameterException("Invalid method name");
       }

       Array<CIMKeyBinding> keyBindings = objectReference.getKeyBindings();
       Uint32 pid = _getPID(keyBindings);

       Uint32 signalNumber;
       CIMValue paramVal = inParameters[0].getValue();
       paramVal.get(signalNumber);

       // execute kill -s signal pid

       // For this example, the Name of a process is constructed as
       // the concatenation of the word "Process" followed by the
       // value of the PID.
       char name[32];
       sprintf(name, "Process %u", pid);
       handler.deliverParamValue(CIMParamValue("ProcessName", String(name)));

       // Return error status of kill operation
       handler.deliver(CIMValue(Uint32(0)));
       handler.complete();
    }
}
```

Validate Input

Execute the operation

Deliver Results

Provider Registation

PG_UnixProcess20R.mof - Notepad

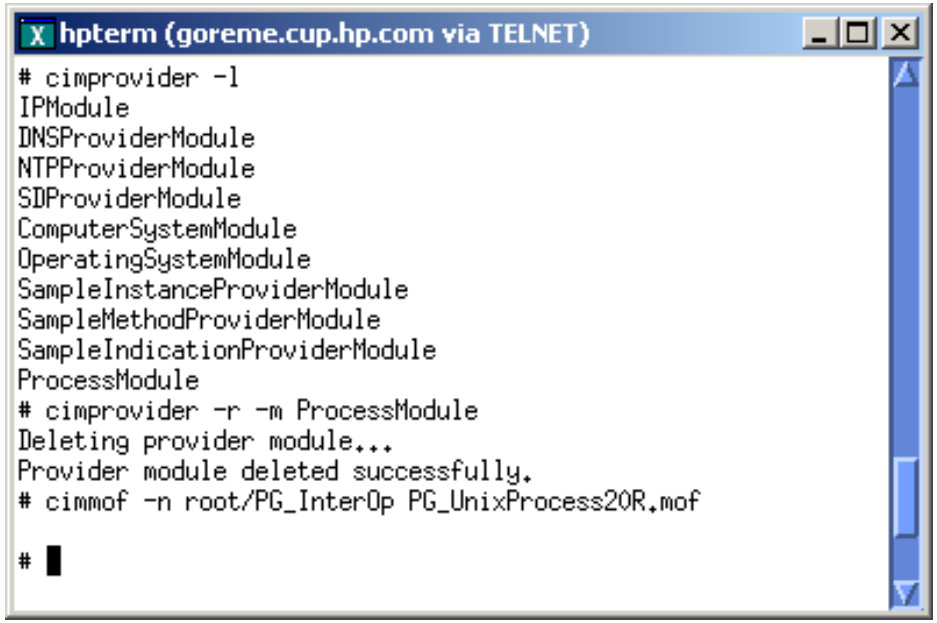
File Edit Format Help

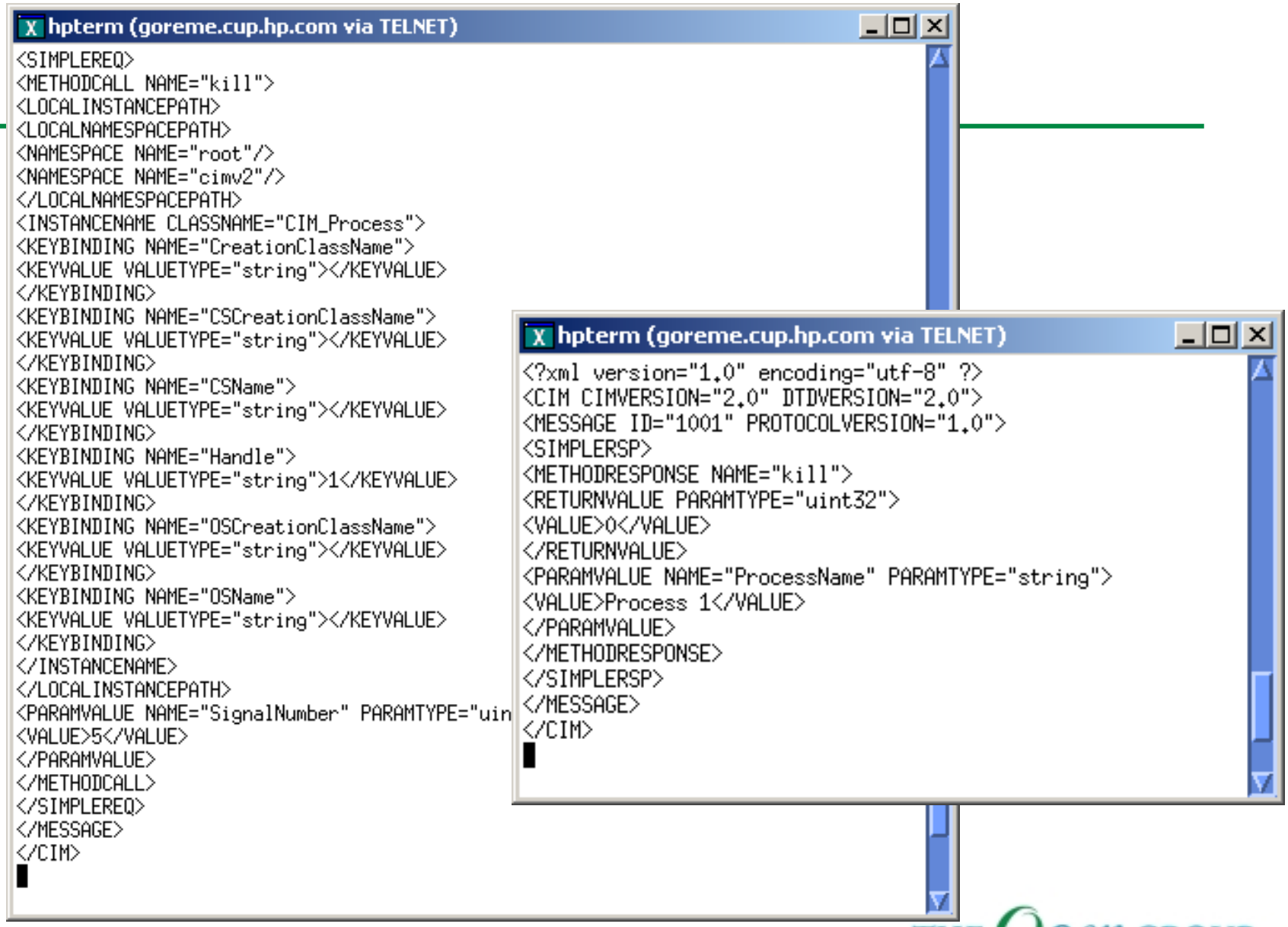
```
// We have one shared library module with two providers
// in it: Process and Processstat

instance of PG_ProviderModule
{
   Name = "ProcessModule";
   Location = "ProcessProvider";
   Vendor = "Pegasus Community";
   Version = "2.0.0";
   InterfaceType = "C++Default";
   Interfaceversion = "2.1.0";
};

// Provider for PG_UnixProcess

instance of PG_Provider
{
  ProviderModuleName = "ProcessModule";
  Name = "ProcessProvider";
};

instance of PG_ProviderCapabilities
{
  ProviderModuleName = "ProcessModule";
  ProviderName = "ProcessProvider";
  CapabilityID = "1";
  ClassName = "CIM_Process";
  Namespaces = { "root/cimv2" };
  ProviderType = { 2, 5 }; // Instance
  SupportedProperties = NULL; // All properties
  SupportedMethods = NULL; // All methods
};

instance of PG_ProviderCapabilities
{
  ProviderModuleName = "ProcessModule";
  ProviderName = "ProcessProvider";
  CapabilityID = "2";
  ClassName = "PG_UnixProcess";
  Namespaces = { "root/cimv2" };
  ProviderType = { 2, 5 }; // Instance
  SupportedProperties = NULL; // All properties
  SupportedMethods = { "kill" }; // All methods
};
```

Note: We still need to register the Process Provider as the "kill" method provider for the PG_UnixProcess and CIM_Process classes. First, we need to update the registration mof, PG_UnixProcess20R.mof.

Provider Registation

Note: Now we just need to re-register the Process Provider.





InvokeMethodExample.cpp - Notepad

File Edit Format Help

```
        Array<CIMKeyBinding> keyBindings;
        keyBindings.append(CIMKeyBinding(CSCREATIONCLASSPROPERTYNAME, String::EMPTY, CIMKeyBinding::S
        keyBindings.append(CIMKeyBinding(CSNAMEPROPERTYNAME, String::EMPTY, CIMKeyBinding::STRING));
        keyBindings.append(CIMKeyBinding(OSCREATIONCLASSPROPERTYNAME, String::EMPTY, CIMKeyBinding::S
        keyBindings.append(CIMKeyBinding(OSNAMEPROPERTYNAME, String::EMPTY, CIMKeyBinding::STRING));
        keyBindings.append(CIMKeyBinding(CREATIONCLASSPROPERTYNAME, String::EMPTY, CIMKeyBinding::STR
        keyBindings.append(CIMKeyBinding(HANDLEPROPERTYNAME, PID, CIMKeyBinding::STRING));

        CIMObjectPath instanceName = CIMObjectPath(String::EMPTY,
                                     NAMESPACE, CLASSNAME,
                                     keyBindings);

        Array<CIMParamValue> inParams;
        Array<CIMParamValue> outParams;

        inParams.append(CIMParamValue("SignalNumber", CIMValue(Uint32(5))));

        CIMValue retValue = client.invokeMethod(
                                NAMESPACE,
                                instanceName,
                                CIMName("kill"),
                                inParams,
                                outParams);
        Uint32 retCode;

        if ((retValue.getType() == CIMTYPE_UINT32) && (!retValue.isNull()))
        {
           retValue.get(retCode);
           cout << "retCode = " << retCode << endl;
        }
        else
        {
          cerr << "Error: invalid return value" << endl;
        }

        if (outParams.size() != 1)
        {
          cerr << "Error: invalid output parameter" << endl;
        }
        else
        {
           String ProcessName = String::EMPTY;
           CIMValue paramVal = outParams[0].getValue();
           paramVal.get(ProcessName);

           cout << "ProcessName = " << ProcessName << endl;
        }

        client.disconnect();
    }
    catch(Exception& e)
    {
        cerr << "Error: " << e.getMessage() << endl;
        exit(1);
    }

    return 0;
```