CIM Exploration

Writing a Pegasus CIM Provider

Document status: Preliminary
Document number: ATI-2002.103
Issue E
Issue date: 22nd October 2002
Security status: Unrestricted Distribution
Authors: Ying Zeng, Chris Hobbs, Tom Chmara
Electronic File: ~cwlh/newOpc/docs/cim

Copyright © 2002 Nortel Networks
This document is the property of Nortel Networks and is licenced for use under the terms and conditions of the Pegasus Licensing Agreement (see "Pegasus Licencing Agreement" on page 40).

History

Draft A **23rd September 2002**
Draft release for comment.

Draft E **29th October 2002**
First public release.

Approvals

NAME	SIGNATURE	DATE
Ying Zeng		
Thomas Chmara		
Chris Hobbs		

Chapter 1: Introduction 6

1.1 The Common Information Model (CIM) 6
1.2 CIM-Speak 6
1.3 The Pegasus Implementation 7

Chapter 2: Running Pegasus 9

2.1 Obtaining Pegasus 9
2.2 Compiling Pegasus 9
2.3 Pegasus Environment Variables 10
2.4 The Repository 10
2.5 Running the CIMOM Server 11
2.6 Compiling mof Code 12
2.7 Pegasus Utilities 12
2.7.1 cimprovider 12
2.7.2 cimconfig 13

Chapter 3: Example 14

Chapter 4: Steps in Writing a Provider 15

4.1 Defining the Structure 15
4.2 Handling mof Files 17
4.2.1 Writing the mof Files 17
4.2.2 mof Makefile 24
4.2.3 Compiling the mof Files 24
4.3 Handling the Provider Code 25
4.3.1 Writing the Provider Code 25
4.3.2 Provider Makefile 32

References 35

Scope

Note that all comments in this document relate to Pegasus version 2.0RC4. At the time of writing there is a significant amount of development in progress and many of the points made herein may no longer be relevant.

This document has been written by a team which has coded a CIM provider using the Pegasus CIM implementation. This is the document that would have saved us a lot of time had it been available earlier—we have written it for those who come after us.

Since the documentation distributed with the Pegasus code is limited, much of what is included in this document has been found by a combination of reverse-engineering and trial-and-error. While being confident that the method described herein works, the authors are not sure that the method is the best way of writing a provider.

This document is in three parts:

- a brief summary of CIM and the Pegasus system: see "Introduction" on page 6
- a worked example of a very simple provider: see "Example" on page 14 and "Steps in Writing a Provider" on page 15
- some miscellaneous tips, hints and wrinkles that we found useful while producing a provider: see "Useful Tips" on page 36. In particular some necessary bug fixes are described in "Bug Fixes" on page 36.

Chapter 1: Introduction

1.1 The Common Information Model (CIM)

The DMTF[1] has standardised an OAM methodology comprising:

- a language (mof) in which the structure of a device or service to be managed can be described: reference (1).
- a definition of the rôle of a management entity called a CIMOM which provides the interface between operators, a repository of static management information and programs known as providers which act as proxies for the real managed objects.
- a definition of the interface (currently XML/HTTP) between the operator and the CIMOM.
- definitions in mof of generic systems: a base system, a network, a user, etc. These are sufficiently general to correspond to any system being managed and companies are expected to add the specific details of their products into these generic definitions.

The interface between the CIMOM and the providers has not yet been formally defined: this is work in progress.

1.2 CIM-Speak

Within the CIM specification, terms are used with very specific meanings. Without an understanding of these terms, it is difficult to write a provider. A few of the more important terms are described here.

- **Class** and **Instance.** These terms are used in the normal object-oriented sense as a generic description of a set of items and a particular element of that set.
- **Indication.** This is CIM-speak for a trigger or event. An indication is also a class and so may have methods and properties.
- **Association.** An association is also a class (and can therefore have methods and properties) and represents a relationship between two or more classes. Note that the naming of some of the standard associations in the CIM reference model is strange. Outside of CIM, associations are normally labelled with verbs or verbal phrases: “loves”, “hates”, “is the parent of”. This happens for some of the CIM associations (e.g. “realises”, “manages”) but most are labelled with nouns which makes them very hard

1. Distributed Management Task Force (see www.dmtf.org)

to read. For example, PhysicalElementLocation is an association between a PhysicalElement and a PhysicalLocation. The relationship effectively says that *this* PhysicalElement is in *that* PhysicalLocation. In this case the relationship should presumably be entitled something like isIn (verbal phrase) rather than PhysicalElementLocation (noun).

- **Reference.** This is the term used to describe how a class behaves in an association. For example, the references for the manages class described above might be manager at one end and employee at the other.
- **Qualifier.** This term is used for a description of an item (classes, methods, properties, etc). The CIM standard defines some qualifiers (for example, Association is a qualifier of a class) and users may define others (effectively by creating an enum). The Pegasus system, in particular, does not enforce user-defined qualifiers.

Another aspect of terminology which is difficult to handle arises from the fact that the major application of CIM appears to be in the storage and computer industries. Much storage and computer terminology and thinking appears in the standard models. This means that many of the models appear very naive to a telecommunications eye (e.g. description of a field-replaceable unit), others appear to be ludicrously over-the-top (e.g. authentication of a user by DNA sampling or by an EEG trace) whereas others are frankly incomprehensible (e.g the VideoBIOSFeature class).

1.3 The Pegasus Implementation

Pegasus, from OpenPegasus (part of the Open Group), is a C++ implementation of the CIM standards. A Java implementation has been produced by the SNIA but has not been used by the authors of this document.

Figure 1: The CIM RunTime Components

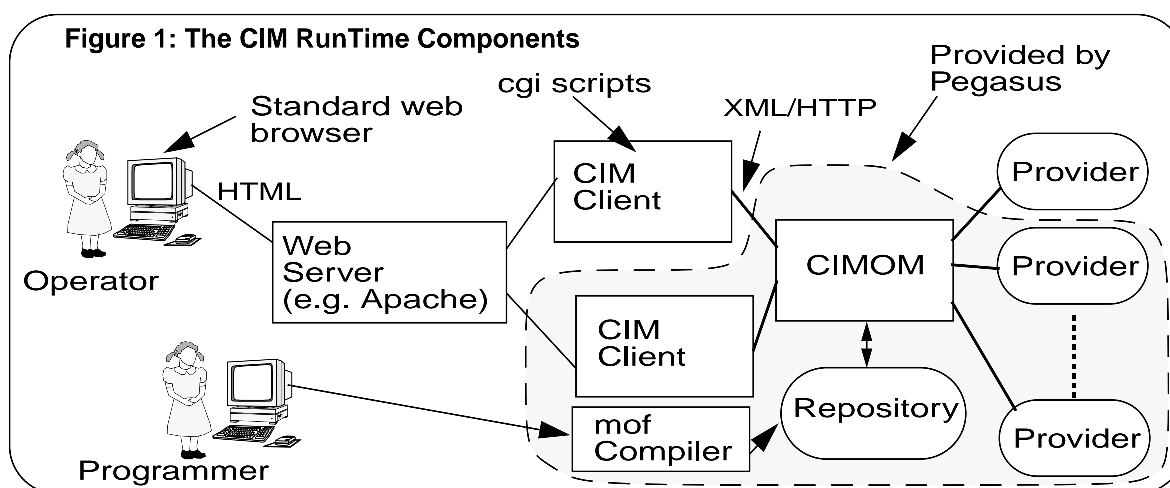


Figure 1 illustrates the major components of the Pegasus implementation of the CIM standards. The components are as follows:

- a mof compiler which accepts mof input and populates the repository.
- a CIMOM which is the heart of the system and which provides the following interfaces:
  — a mapping to and from XML/HTTP for use as an operator interface. This interface is standardised by the DMTF.
  — an interface to persistent storage to hold the less volatile management information.
  — an interface to “providers” (effectively drivers which stand proxy for the real items being managed). This interface is not yet defined by the DMTF.
- a simple and extremely inefficient repository comprising a file of XML for each object. While this is too cumbersome for production, it makes debugging extremely easy.
- a CIM client for accessing the repository and Operating System providers in a consistent manner. The actual format of the displayed information is useful for debugging purposes but it not suitable for end-users.
- a number of sample providers for such objects as the Linux Operating System.

Note that, although the DMTF has standardised the interface between an operator and the CIMOM, it has not yet standardised the interface between the CIMON and providers. In order to produce an implementation, the OpenPegasus group has defined an interim standard for this interface. This document assumed that standard.

Pegasus is open-source. The code can be obtained (as described in "Making CGIClient" on page 36) from the www.openpegasus.org web site. The code is issued under the terms of the terms of the Pegasus License Agreement which is based on the MIT Open-Source licence (see "Pegasus Licencing Agreement" on page 40).

Chapter 2: Running Pegasus

2.1 Obtaining Pegasus

Pegasus can be checked out of the CVS[1] system on the Pegasus web site (www.openpegasus.org). Anonymous access for read is with the name and password "anon" as follows:

```
CVSROOT=:pserver:anon@cvs.opengroup.org:/cvs/MSB
```

The source tree is in the directory `pegasus`. To check out the complete Pegasus source tree type:

```
cvs co pegasus
```

A Pegasus directory will be created under the current directory (`$PEGASUS_ROOT` in what follows) and populated with the complete source tree and documentation. To get the latest updates after a checkout just type the following from `$PEGASUS_ROOT`:

```
cvs update -d
```

2.2 Compiling Pegasus

The environment variables described in "Pegasus Environment Variables" on page 10 must be set before attempting to compile Pegasus.

To compile the system, type `make` in `$PEGASUS_ROOT`.

The base classes are then loaded into the repository on disc by typing `make repository` in the directory `$PEGASUS_ROOT/Schemas/Pegasus`.

To build the linux providers enter the directory `$PEGASUS_ROOT/src/Providers/linux` and type `make`.

To put the linux classes into the repository, type `make repository` in the directory `$PEGASUS_ROOT/src/Providers/linux/load`. It is then necessary to `make register` to push the linux providers into the `PG_InterOp` namespace. It is then necessary to restart the CIMOM.

1. See, for example, reference (2)

2.3 Pegasus Environment Variables

The following environment variables need to be set before working with Pegasus (values are those suitable for the CIM Exploration project, other projects may need to modify $PEGASUS_PLATFORM as appropriate):

```
HOME=<root directory into which Pegasus was built>
export PEGASUS_HOME=$HOME
export PEGASUS_ROOT=$HOME/CIMOM/pegasus
export PEGASUS_PLATFORM=LINUX_IX86_GNU
export PATH=$PATH:$PEGASUS_HOME/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PEGASUS_HOME/lib
```

$PEGASUS_HOME is thus set to point to the parent directory for the generated bin, lib, etc directories. $PEGASUS_ROOT is set to point to the parent directory for the source code (as checked out of the OpenPegasus CVS).

2.4 The Repository

Pegasus provides a very simple default repository. This consists of a directory of files under $PEGASUS_HOME/repository. Under that directory, each namespace within the repository is held as a subdirectory. Each namespace then has subdirectories for classes, instances and qualifiers. Each class within a namespace is held in XML in a separate file within the classes subdirectory. The use of a separate file for each class and the XML encoding makes this type of repository extremely simple for debugging but very inefficient for use in product. This is acknowledged by the Pegasus team and the repository can be replaced by a standard database.

One advantage of the storage of the repository in this manner is that it is persistent across loads and unloads of the CIMOM. This is also a disadvantage if new instances of objects are to be created during development since the old instance will still be in the repository and, currently, a running CIMOM is not able to replace one instance with another, returning the error message:

```
Warning: the instance already exists.
In this implementation, that means it cannot be changed.
```

Getting rid of an existing repository is difficult as caching occurs in three places: within the files of the repository, within the CIMOM itself and within any web browser being used to access the CIMOM. Running

```
make repository
```

from $PEGASUS_ROOT says that the repository is up-to-date even if the entire repository is deleted! In order to remove the repository and cause it to be recompiled, the following commands need to be run from the $PEGASUS_ROOT/Schemas/Pegasus directory:

```
make clean
make repository
```

This cleans out the files but does not restart the CIMOM which is also caching the classes. The CIMOM must therefore be restarted by shutting it down

```
cimserver -s
```

(or `kill`) and then restarting it with

```
cimserver
```

Note that the `cimserver -s` command can only be given by root and it may therefore be necessary to use the `kill` command.

Within the repository, there are three namespaces of importance:

- `root/cimv2` contains static class specifications (note that the slash in `root/cimv2` does not represent a hierarchy as it would when describing, for example, a UNIX file: the name is simply "root-slash-cimv2") that are maintained in the repository
- anything that requires dynamic execution (especially loading and execution of a provider) is found in `root/PG_InterOp` unless the provider is an inherent part of the CIMOM
- if the provider is an inherent part of the CIMOM then it is found in `root/PG_Internal`

To avoid conflict with UNIX filenames, these are converted to `root#cimv2`,etc for storage within the `$PEGASUS_ROOT/repository` directory.

2.5 Running the CIMOM Server

The Pegasus version of the CIMOM is called `cimserver`. It is started by:

```
cimserver [ [ options ] | [ configProperty=value, ... ] ]
```

where the options are:

- `-v` to display CIM Server version number
- `-h` to print a help message
- `-s` to shut down CIM Server
- `-D [home]` to set Pegasus home directory
- `configProperty=value` to set a CIM Server configuration property (for a list of the keywords which may be used here, run `cimconfig -l`)

2.6 Compiling mof Code

`mof` code is compiled by using the `mof` compiler:

```
cimmof -n<namespace> <program>.mof
```

For example:

```
cimmof -nroot/cimv2 UNI-C.mof
```

The action of the compiler is somewhat strange: it parses the code and displays any errors in the manner of any normal compiler. It also puts the generated code into the repository and then causes the CIMOM Server to load the newly compiled objects. This means that, for complete success in compiling, the CIMOM Server must be running before the compilation is started. With the current Pegasus implementation, there is no way to remove objects from the CIMOM Server and so, if the `mof` code is changed, the CIMOM Server needs to be stopped and restarted before the compilation is carried out.

More information about the compiler is given in "Compiling mof Code" on page 12.

2.7 Pegasus Utilities

2.7.1 cimprovider

This utility allows the user to manipulate the providers in the repository.

```
cimprovider -l {-s}
```

simply lists the providers in the repository (with a status if the -s flag is given).

A provider may be disabled by the command

```
cimprovider -d -m <name>
```

and may be reënabled by the command

```
cimprovider -e -m <name>
```

An attempt to access a disabled provider through the operator interface results in the error message CIM_ERR_ACCESS_DENIED.

It should be possible to delete a provider permanently from the repository files with the command:

```
cimprovider -r -m <name>
```

but, although this gives a good message after execution, nothing appears to have been deleted from the repository.

2.7.2 cimconfig

When the CIMOM server starts, it gets its configuration from the file `$PEGASUS_ROOT/cimserver_planned.conf`. While running, the current configuration is placed into the file `$PEGASUS_ROOT/cimserver_current.conf` which is created if it does not exist.

The contents of `cimserver_planned.conf` can be changed by running the utility

```
cimconfig -s <name>=<value> -p
```

and the value of the current parameters can be changed dynamically by running

```
cimconfig -s <name>=<value> -c
```

`cimconfig` can also be used to determine the current values of parameters by using

```
cimconfig -l -{c|p}
```

where c indicates the current and p the planned values.

Other options for cimconfig are:

```
cimconfig -u <name> -{c|p}
```

which resets the named parameter to its default value and

```
cimconfig -g <name> -{c|p|d}
```

which prints the current (c), planned (p) or default (d) value of the named parameter.

Note that commands relating to the planned parameters can be run whether or not the CIMOM is running (they affect only the configuration file). Commands relating to the current parameters can only be executed if the CIMOM is running.

Chapter 3: Example

Throughout this document, a very simple example is used. This example consists of the management of a system which establishes, lists and removes transport connexions known as lightpaths.

Since the version of Pegasus being used does not handle instance creation and deletion well, the example was coded with explicit `createLightPath`, `deleteLightPath` and `getNextLightPath` methods.

Chapter 4: Steps in Writing a Provider

4.1 Defining the Structure

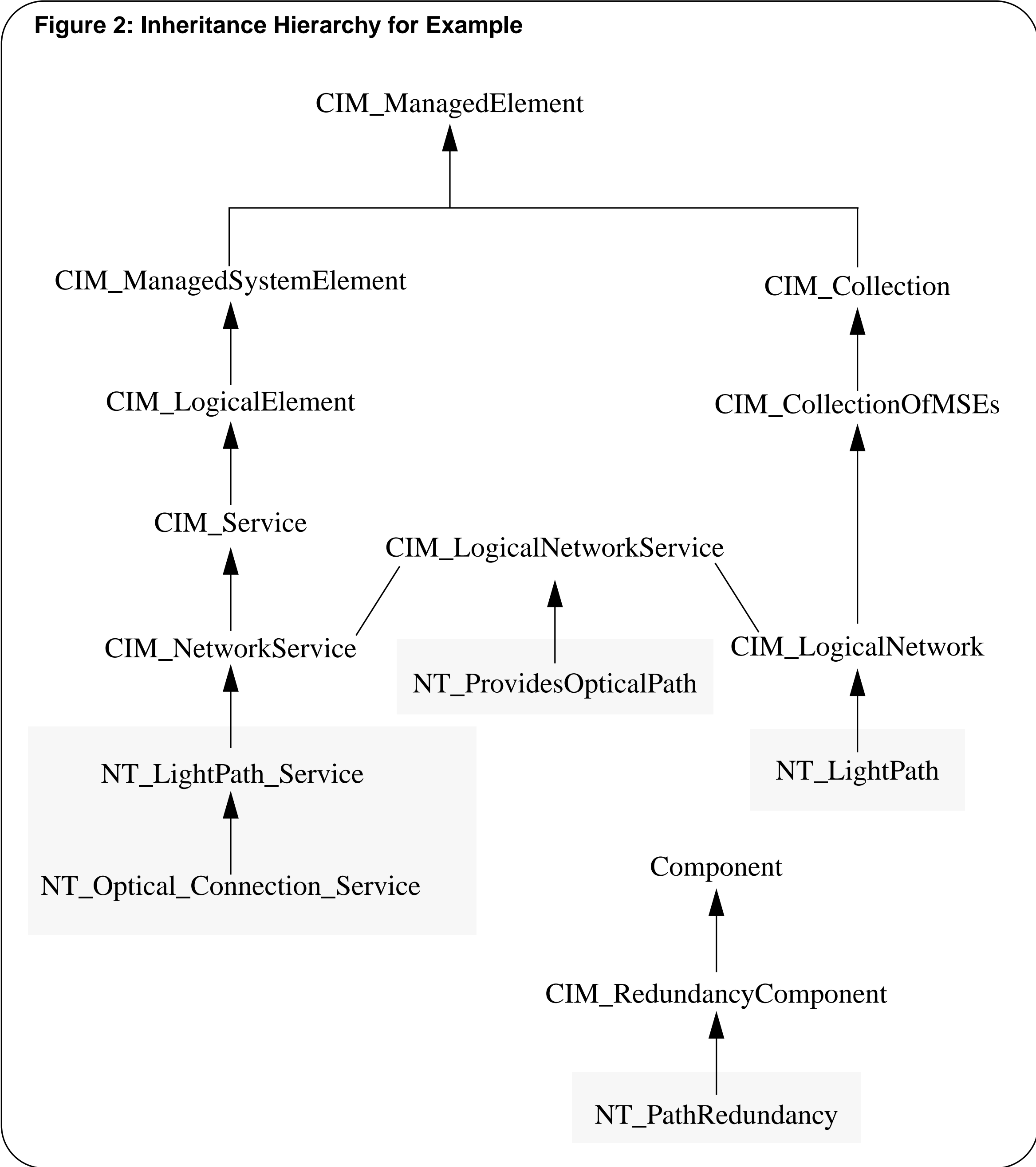
The DMTF defines a number of standard structures (e.g. core model, network model, user model). The appropriate position in those hierarchies must be chosen for the new entity. Note that the relationship in the hierarchies is “is-a” rather than the “has-a” more normal in SNMP and Passport CAS.

In the case of the example described above, the following classes were defined:

- `NT_Lightpath_Service` which inherits from the `CIM_NetworkService` class which is part of the DMTF definition
- `NT_Optical_Connection_Service` which inherits from `NT_Lightpath_Service`
- `NT_Lightpath` which inherits from the `CIM_LogicalNetwork` class which is part of the DMTF definition
- `NT_ProvidesOpticalPath` which is an association (with a verbal phrase as name) which inherits from `CIM_LogicalNetworkService` (which is a noun and part of the DMTF definition)
- `NT_PathRedundancy` which is an association (OK, so even we don’t get the names correct all the time) which inherits from `CIM_RedundancyComponent` which is part of the DMTF definition

The choice of where these classes should be added to the standard hierarchy as defined by the DMTF appears to be a black art. The positioning of the above classes for example was a learning exercise and should not be treated as perfect (or even good)—they are, however, illustrated in figure 2.

Figure 2: Inheritance Hierarchy for Example



4.2 Handling mof Files

4.2.1 Writing the mof Files

Once the position in the hierarchy has been decided, the classes must be described in `mof`. The `mof` language is defined in reference (1).

Two mof files are required for the system described in "Example" on page 14. One of these (`UNI-C.mof`) simply describes the application. This is as follows:

```
// *********************************************************
//    module      UNI-C.mof
//    purpose    define the structure of the xxxxxx demo
//    written    september 2002
//    author     fred bloggs
//    history
//               02/10/02 Changed by joe to incorporate
//                changes agreed at a meeting on
//                01/10/02
// *********************************************************

#pragma Locale ("en_GB")

// =========================================================
//
// Lightpath_Service
//
// Currently defined as "NT_Lightpath_Service", as it's
// not a formal component of the CIM.
//
// Units of "sigType" are "const short" in CORBA; uint16 in XML.
//
// =========================================================

[Abstract, Version ("1.0.0"), Description (
"This is an abstract base class, derived from NetworkService."
"It serves as the root for lightpath (i.e. Layer 0, Layer 1)"
"connection services, as distinct from packet services.")]

class NT_Lightpath_Service : CIM_NetworkService
    {
    [ Key ]
    uint16  keyValue;

    [Description ("This field defines the 'signallingtype',"
      "i.e. the bandwidth capacity of the lightpath."),
    Valuemap { "5", "6" },
    Values { "STS1_SPE", "STS3C_SPE" } ]
    uint16sigType;
    };

// =========================================================
//
// Optical_Connection_Service
//
```

```
// Currently defined as "NT_Optical_Connection_Service",
// as it's not a formal component of the CIM.
//
// Units of "encoding" are "const short" in CORBA;
// uint16 in XML.
//
// ==============================================================

[Version ("1.0.0"), Description (
"This is a instantiable class, derived from Lightpath_Service, "
"used to create SONET and SDH optical connections.")]

class NT_Optical_Connection_Service : NT_Lightpath_Service
    {
    [ Key,
      Description("Name of this lightpath instance")]
    String  lightPathName;

    [ Description("IP Address of the lightpath source")]
    uint32  sourceAddress;

    [ Description("IP Address of the lightpath destination")]
    uint32  destinationAddress;

    [ Description("Set TRUE if the lightpath is "
                  "bi-directional")]
    boolean biDirectional;

    [ Description("Indication of lightpath speed"),
      Valuemap { "5", "6" },
      Values { "STS1_SPE", "STS3C_SPE" }]
    uint16 sigType;

    [ Description("Indication of lightpath type:"
                " Synchronous Digital Hierarchy,"
              " SONET, Gigabit Ethernet or"
              " Fibre Channel"),
      Valuemap { "5", "6", "10", "11" },
      Values { "SDH", "SONET", "GE", "FC" }]
    uint16 encoding;

    [ Description("Level of protection for the lightpath"),
      Valuemap { "16", "4" ,"49", "50", "0", "2" },
      Values { "ONE_FOR_ONE", "ONE_FOR_N",
              "REVERTIVE", "PREEMPTIVE",
              "ANY", "UNPROTECTED" }]
    uint16 classOfService;

    [ Description("First timeslot within the frame")]
    uint16 firstTimeSlot;

    [ Description("Port number within the bundle")]
    uint16 portIndex;

    [ Description("Number of components")]
    uint16 numberComponents;

    [ Description("Concatentation type: only STANDARD"
              " supported")]
    uint16 concatenationType;
```

```
// ====================================================
// method: deleteLightPath
// purpose:delete the lightpath with the given name.
// input:name of the lightpath to be deleted
// output:TRUE if lightpath deleted, FALSE otherwise
// note:will not be needed when deleteInstance
//                works properly
// ====================================================

[Description("Delete a lightpath with the given name")]

boolean deleteLightPath([IN] String name);

// ====================================================
// method: createLightPath
// purpose:create a new lightpath with the given
//                parameters
// input:parameters (is there a way to define a
//                structure in mof?)
// output:TRUE if lightpath created, FALSE otherwise
// note:will not be needed when createInstance
//                works properly
// ====================================================

[Description("Create a lightpath with the given "
                              "parameters")]

boolean createLightPath([IN] String name,
                        [IN] uint32 source,
                        [IN] uint32 destination,
                        [IN] uint16 biDirectional,
                        [IN] uint16 signallingType,
                        [IN] uint16 encodingType,
                        [IN] uint16 classOfService,
                        [IN] uint16 firstTimeSlot,
                        [IN] uint16 portIndex,
                        [IN] uint16 numberComponents,
                        [IN] uint16 concatenationType);

// ====================================================
// method: getNextLightPath
// purpose:return the details of a lightpath
// input:index. If 0 this means that the first
//                lightpath's details should be
//                returned. Otherwise this should
//                be the index returned by the
//                previous call to getNextLightPath
//                to get the next lightpath's
//                details.
// output: index. If this is -1 then it indicates
//                that there was no "next"
//                lightpath to return. Otherwise
//                the parameters of a lightpath
//                are returned in the oher
//                parameters.
// ====================================================

[Description("Get details of a lightpath")]

boolean getNextLightPath([IN] sint32 index,
                        [OUT] String name,
```

```
                              [OUT] uint32 source,
                              [OUT] uint32 destination,
                              [OUT] uint16 biDirectional,
                              [OUT] uint16 signallingType,
                              [OUT] uint16 encodingType,
                              [OUT] uint16 classOfService,
                              [OUT] uint16 firstTimeSlot,
                              [OUT] uint16 portIndex,
                              [OUT] uint16 numberComponents,
                              [OUT] uint16 concatenationType);
    };

// ==========================================================
//
// Lightpath
//
// Currently defined as "NT_Lightpath", as it's
// not a formal component of the CIM.
//
// Units of "upLabel" are "unsigned long" in CORBA;
//             uint32 in XML.
//
// ==========================================================

[Version ("1.0.0"), Description (
"This is an instantiable class, derived from "
"LogicalNetwork, used to aggregate those protocol"
"endpoints that form the lightpath. "
"It may be that this structure needs to be extended or "
"subclassed; there may not be (e.g.) a timeslot if in fact"
"it's a lambda path.")]

class NT_Lightpath : CIM_LogicalNetwork
    {
    [ Description ("Bidirectional is TRUE if the lightpath"
                  "carries traffic in both directions;"
                  "FALSE if it is a unidirectional
                  "path." )]
    boolean Bidirectional;
    [ Description ("upLabel specifies the first of the "
                  "SONET/SDH timeslots used to carry"
                  "this Lightpath.")]
    uint32 upLabel;
    };

// ==========================================================
//
// ProvidesOpticalPath
//
// Currently defined as "NT_ProvidesOpticalPath",  as it's not
// a formal component of the CIM.
//
// ==========================================================

[Association, Version ("1.0.0"), Description (
"This is an instantiable association, which relates the"
"service to the lightpath(s) it has created.  The"
"CIM_LogicalNetworkService is used to 'identify services"
"that are provided by particular network devices', so I"
"don't know if it's a direct match but it should give "
"us the relationship we seek. We may also not need it,"
```

```
"unless we have parameters that distinguish it from"
"CIM_LogicalNetworkService" ) ]

class NT_ProvidesOpticalPath : CIM_LogicalNetworkService
    {
    };

// ============================================================
//
// PathRedundancy
//
// Currently defined as "NT_PathRedundancy", as it's
// not a formal component of the CIM.
//
// Units of "cos" are "const short" in CORBA; uint16 in XML.
//
// ============================================================

[Association, Version ("1.0.0"), Description (
"This association allows description of the redundancy"
"characteristics of a Lightpath." ) ]

class NT_PathRedundancy : CIM_RedundancyComponent
    {
    [ Description ("This field allows specification of"
                      "the nature of the redundancy in the"
                      "Lightpath; it must be 1:n or 1:1 "
      "(if it is not redundant, then this association is not "
      "instantiated" ),
    Valuemap { "16", "4", "49", "50", "0", "2" },
    Values { "ONE_FOR_ONE",
                "ONE_FOR_N",
                "REVERTIVE",
                "PREEMPTIVE",
                "ANY",
                "UNPROTECTED" }]
    uint16 cos;
    };
```

Note that the parameters in the calls to `createLightPath` and `getNextLightPath` are clumsy—there appears to be no way in which to pass a structure.

The second `mof` file `(UNI_CR.mof)` ties the example into the repository as a particular instance of a `PG_Provider`: i.e. a provider which will be dynamically loaded and tied into the repository the first time that it is accessed.

```
//
// Modelled after PG_UnixProcess20R.mof...
//
// PG_ProviderModule is used to identify the shared-library
// module from which we load our providers; there can be
// more than one provider in a module.
// IBM refers to this as the "physical provider" details.
//
//  Library name:  "Location"
//  Name: ?? probably used to tie these together with the
//        PG-Provider and Capabilities fields.
```

```
//
// We are using "NT_UNI" for the UNI-C demo.
//
instance of PG_ProviderModule
    {
    Name = "NT_UNI_ProviderModule";
    Location = "NT_UNI_Provider";
                   // should become libNT_UNI_Provider.so
    Vendor = "Nortel Networks Ltd.";
    Version = "1.0.0";
    InterfaceType = "C++Default";
    InterfaceVersion = "1.0.0";
    };

//
// Provider for OCS-related instances
// The "Name" is the provider name (the object?), not the
// class name(s).
// IBM refers to this as the "logical provider" details.
//
instance of PG_Provider
    {
     // I believe the ModuleName correlates with the
     // above section
     // Again, "Name" is ???
     //
    ProviderModuleName = "NT_UNI_ProviderModule";
                               // within the above .so
     Name = "NT_UNI_Provider";  // name of the provider class
    };

//
// IBM refers to the following as the "logical provider
// capabilities".
//

//
// Class NT_Optical_Connection_Service
//
instance of PG_ProviderCapabilities
    {
    ProviderModuleName = "NT_UNI_ProviderModule";
    ProviderName = "NT_UNI_Provider";
    CapabilityID = "1";
    ClassName = "NT_Optical_Connection_Service";
    Namespaces = {"root/cimv2"};
    ProviderType = { 2, 5 };
                   // Instance provider AND method provider
    SupportedProperties = NULL; // All properties
    SupportedMethods = NULL; // All methods
    };

//
// Class NT_Lightpath
//
instance of PG_ProviderCapabilities
    {
    //
    //
    ProviderModuleName = "NT_UNI_ProviderModule";
    ProviderName = "NT_UNI_Provider";
```

```
    CapabilityID = "2";
    ClassName = "NT_Lightpath";
    Namespaces = {"root/cimv2"};
    ProviderType = { 2 }; // Instance provider
    SupportedProperties = NULL; // All properties
    SupportedMethods = NULL; // All methods
    };

//
// Class ProvidesOpticalPath (Association)
//
instance of PG_ProviderCapabilities
    {
    //
    //
    ProviderModuleName = "NT_UNI_ProviderModule";
    ProviderName = "NT_UNI_Provider";
    CapabilityID = "3";
    ClassName = "NT_ProvidesOpticalPath";
    Namespaces = {"root/cimv2"};
    ProviderType = { 2 }; // Instance provider
    SupportedProperties = NULL; // All properties
    SupportedMethods = NULL; // All methods
    };

//
// Class PathRedundancy (Association)
//
instance of PG_ProviderCapabilities
    {
    //
    //
    ProviderModuleName = "NT_UNI_ProviderModule";
    ProviderName = "NT_UNI_Provider";
    CapabilityID = "4";
    ClassName = "NT_PathRedundancy";
    Namespaces = {"root/cimv2"};
    ProviderType = { 2 }; // Instance provider
    SupportedProperties = NULL; // All properties
    SupportedMethods = NULL; // All methods
    };
```

`PG_ProviderModule` can be thought of as the details of the physical provider: its location, naming, etc. Each corresponds to a shared library which is loaded on demand.

`PG_Provider` can be thought of as the logical provider; there can be more than one of these in each shared library.

`PG_ProviderCapabilities` specifies the capabilities of each logical provider. The `CapabilityID` field appears to be a monotonically increasing field for each instance described (it is possible that they simply need to be different and the monotonic increasing property is just a simple way of ensuring this). The `ProviderType` field can have one or more values corresponding to the interfaces that it exports: 2 for instances, 5 for methods. A provider which supports both instance and method invocations would be coded as

```
ProviderType = { 2, 5 };
```

4.2.2 mof Makefile

The following Makefile was found to work for compiling the example mof:

```
all:    uni-c uni-cr

uni-c:  UNI-C.mof Makefile
        cimmof -nroot/cimv2 UNI-C.mof

uni-cr: UNI-CR.mof Makefile
        cimmof -nroot/PG_InterOp UNI-CR.mof
```

4.2.3 Compiling the mof Files

The above Makefile executes the mof compiler (`cimmof`) which takes the following parameters:

- `-h`, `--help` -- show this help.
- `-E` -- syntax check only.
- `-w` -- suppress warnings.
- `-Rrepository` -- specify the repository path (`cimmofl`) or hostname:portnumber (`cimmof`)[1]
- `--CIMRepository=repository` -- specify repository path or hostname:portnumber.
- `-Ipath` -- specify an include path.
- `-ffile` -- specify file containing a list of MOFs to compile.
- `--file=file` -- specify file containing list of MOFs.
- `-npath` -- override the default `CIMRepository` namespace.
- `--namespace=path` -- override default `CIMRepository` namespace.
- `--xml` -- output XML only, to stdout. Do not update repository.
- `--trace` or `--trace=ttracefile` -- trace to file (default to stdout).

`cimmof` not only compiles the code but, unless `-E` or `--xml` are given as options, it also loads the resulting models into the repository. For this to happen the CIMOM server has to be running. If the classes are already in the repository, then a warning message is given:

```
Warning:  Class XXXXX already exists in the repository
```

1. In the current version of Pegasus, `cimmof` and `cimmofl` are identical in all respects.

Similarly, if the instances already exist then the message:

```
Warning: the instance already exists.
In this implementation, that means it cannot be changed.
```

To clean the repository, see the instructions in "The Repository" on page 10.

4.3 Handling the Provider Code

4.3.1 Writing the Provider Code

Two C++ files are required:

- `NT_UNI_ProviderMain.cpp` which simply creates an instance of the `NT_UNI_Provider` class
- `NT_UNI_Provider.cpp` which implements the inherited CIMInstanceProvider methods

The following header file is included in both the C++ files.

```
#ifndef Pegasus_NT_UNI_Provider_h
#define Pegasus_NT_UNI_Provider_h

// Includes for the network address information.
//
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#include <Pegasus/Common/Config.h>
#include <Pegasus/Common/System.h>
#include <Pegasus/Common/CIMObjectPath.h>
#include <Pegasus/Provider/CIMInstanceProvider.h>
#include <Pegasus/Provider/CIMMethodProvider.h>
#include <Pegasus/Common/OperationContext.h>
#include <Pegasus/Provider/ProviderException.h>

#include <Pegasus/Common/Tracer.h>

#include <orbsvcs/orbsvcs/CosNamingC.h>

// Includes for the UNI-C stuff

#include <corbaMessageHandlerC.h>

PEGASUS_USING_PEGASUS;
PEGASUS_USING_STD;

// ======================================================
//  Constants relating to the provider
// ======================================================

......deleted........
```

```
// ========================================================
//  Text names of the classes for which we are provider.
// ========================================================

.......deleted........

// ========================================================
//  Text name of the UNI-C Object
// ========================================================

const char * const UNI_C_Name = "%s/UNIC/0";

class NT_UNI_Provider : public CIMInstanceProvider,
                            public CIMMethodProvider
    {
    public:
        //
        // The constructor and destructor are mandatory,
          // and mandatorily empty.
          NT_UNI_Provider(void);
          ~NT_UNI_Provider(void);

         // From the CIMBaseProvider class; these must
        // be provided.
        // A number simply won't do anything (as per
        // IPRouteProvider).

        void initialize(CIMOMHandle & cimom);
        void terminate(void);

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

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

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

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

```
        void createInstance(
                const OperationContext & context,
                const CIMObjectPath & ref,
                const CIMInstance & obj,
                ObjectPathResponseHandler & handler);

        void deleteInstance(
                const OperationContext & context,
                const CIMObjectPath & ref,
                ResponseHandler & handler);

        //
        // From CIMMethodProvider:
        //

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

private:
        ..... lots of local stuff deleted
    };
#endif
```

It can be seen from this header file that `NT_UNI_Provider` is-a `CIMInstanceProvider` and a `CIMMethodProvider`.

Notice that `NT_UNI_Provider` is given as the implementation of `NT_Optical_Connection_Service` and other classes in the `UNI_CR.mof` file.

The methods defined include:

- `initialize`

  Perform any setup required before normal operation. 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 CIMOM invokes any other function of the provider, other than `terminate`.

  `initialize` is inherited from `CIMBaseProvider`.

- `getInstance`

  This function is called to return the properties of a particular instance. Note that, in the current version of the Pegasus code, it is also invoked to return a specific property—the `getProperty` method of `CIMPropertyProvider` not being used.

  The parameters of this function are:

— `const OperationContext& context`. This contains the identity of the user making the request, thus allowing the provider to provide a small amount of security.

— `const CIMObjectPath& ref`. This object provides the full name of the particular instance including hostname (`getHost()`), namespace name (`getNameSpace()`), classname (`getClassName()`) and instance label (`getKeyBindings().getName()` on the "Name" component).

— `const Boolean includeQualifiers`. This is set to FALSE in the current implementation.

— `const Boolean includeClassOrigin`. This is set to FALSE in the current implementation.

— `const CIMPropertyList& propertyList`. This appears not to be used when the system is invoked from the web browser application.

– `InstanceResponseHandler& handler`. For a `getInstance` request, this is used to return the result:

```
handler.processing();

handler.deliver(instance);

handler.complete();
```

- `invokeMethod`

This is the method which is actually called when a client makes a method call (i.e. a call to `invokeMethod` on the CIMOM). The parameters to the call are passed in an array of `CIMParamValue` as illustrated below. There appears, unfortunately, to be no automatic checking of parameter types (or even names) against the mof definition. Perhaps this will be improved in later releases of the Pegasus code.

At the client end, a call to `deleteLightPath` would be made as follows:

```
Array<CIMParamValue> inParams;
Array<CIMParamValue> outParams;
inParams.append(CIMParamValue("name",
                        CIMValue(String(connexionName))));

retValue = client->invokeMethod(nameSpace,
                              CIMObjectPath(instanceName),
                                    "deleteLightPath",
                                    inParams,
                                      outparams);

// all done: let's check the return code and
// print a message for the user

Boolean returnCode;
retValue.get(returnCode);
```

The parameters are thereby passed as name/value pairs. In the provider, the function `invokeMethod` is called and the actual function being invoked (here `deleteLightPath`) has to be extracted from the incoming parameters as follows:

```
void NT_UNI_Provider::invokeMethod(
     const OperationContext& context,
     const CIMObjectPath& objectReference,
     const CIMName& methodCName,
     const Array<CIMParamValue>& inParameters,
     MethodResultResponseHandler& handler)
{
String methodName = methodCName;

//-- so, what class are we working with, anyway?

if ( String::equalNoCase( methodName, DELETEPATH ))
     {
     //
     // Delete the given LightPath, if it exists.
     //
      NT_UNI_Provider::deletePath( inParameters, handler );
     }
```

Once the actual method being invoked has been discovered, the parameters have to be extracted from `inParameters` (and placed, where appropriate, into `outParameters`). Extracting the parameters from `inParameters` can be done as follows:

```
if (inParameters.size() != n_deletePathParameters)
     {
     int nparms = inParameters.size();
     cout << "Wrong number of parameters; expected " <<
             n_deletePathParameters << " got " << nparms <<
             " instead!" << endl;
     throw PEGASUS_CIM_EXCEPTION( CIM_ERR_INVALID_PARAMETER,
                 "Wrong number of input parameters" );
     }

// Get the input parameter values

for (Uint32 i = 0; i < inParameters.size(); i++)
     {
      String parmName = inParameters[i].getParameterName();

     if (String::equalNoCase(parmName, "name")) {
          //
          // get the name of the lightPath to delete
          //
             inParameters[i].getValue().get(lightPathName);
          }
     else
          {
           throw PEGASUS_CIM_EXCEPTION(
                   CIM_ERR_INVALID_PARAMETER,
                   "Input parameters are not valid.");
          }
```

```
}
```

One of puzzling aspects of this interface was determining what type of field the parameter value would turn up in. The call to `getValue().get(x)` fails if x is of the wrong type (e.g. `uint32` instead of `sint32`). In the end, we found the only practical way of determining how to decode the input parameters was to change the code to print the type of each parameter (by using `getType()` instead of `getValue()`), printing these and then going back and using the type which Pegasus was using for the call to `getValue().get()`. There must be a better way of doing this but we didn't find it.

If returned parameters are defined in the `mof`, then these can be returned from the provider by encoding them as keyword/value pairs and making calls similar to the following:

```
handler.processing();
String nameString = services[serviceIndex].instance.name;
handler.deliverParamValue( CIMParamValue( "name",
                              CIMValue( nameString )));
handler.deliverParamValue( CIMParamValue( "source",
                              CIMValue( 45 ));
.....do all the parameters....
handler.complete();
```

These values can then be extracted from the `outParams` parameter of the client's call to `invokeMethod()` using a similar technique to that shewn for the provider above:

```
for (Uint32 i = 0; i < outParams.size(); i++)
      {
      String parmName = outParams[i].getParameterName();
      CIMValue   val = outParams[i].getValue();
      }
```

- `enumerateInstanceNames`

  This is invoked to extract a list of instance names from the provider. The parameters are `const OperationContext& context`, `const CIMObjectPath& ref` and `ObjectPathResponseHandler& handler` which are as described above. The list of instance names is returned by a call to `handler.deliver()`.

- `modifyInstance`

  This method is called to modify not only an instance but, in the current version of the code, also when `setProperty` is invoked to change a single property. The call to `setProperty` is intercepted by `ProviderManagerService` and turned into a call to `modifyInstance`. This will presumably change in later releases.

  Most of the parameters to this function are as described above. The properties to be changed, however, are held in the `CIMInstance& instanceObject` parameter and can be obtained by using the

getPropertyCount() to find out how many there are and then by iterating through instanceObject.getProperty(i).getValue() and instanceObject.getProperty(i).getName() functions.

- terminate

  Perform any cleanup required before termination. The terminate function allows the provider to conduct the necessary preparations to prepare for termination. This function may be called by the CIMOM 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 (this may be done by setting a global flag)

  — kill subprocesses

  If the provider instance was created on the heap with new in PegasusCreateProvider, then it must be deleted in terminate. terminate is inherited from CIMBaseProvider.

When a provider is first invoked, PegasusCreateProvider in the appropriate library (in the case of the example, ${PEGASUS_HOME}/lib/libNT_UNI_Provider.so) is called with a string argument containing the provider to be created. For the example, this function provides the following code which simply checks that the argument is NT_UNI_Provider and, if so, creates an instance of the NT_UNI_Provider class.

```
#include <Pegasus/Common/Config.h>
#include <Pegasus/Common/String.h>

#include "NT_UNI_Provider.h"

extern "C" PEGASUS_EXPORT CIMBaseProvider *
    PegasusCreateProvider(const String & providerName)
    {
     if (String::equalNoCase(providerName, "NT_UNI_Provider"))
        {
        return(new NT_UNI_Provider());
        }
    cout << "Attempt to start up unknown provider:" <<
                providerName << endl;
    return(0);
    }
```

This code is included in the NT_UNI_ProviderMain.cpp file.

4.3.2 Provider Makefile

The following Makefile was found to work for the example provider. Note that this program also makes use of the Tao Orb and many of the libraries and include directories are related to that software rather than to Pegasus. Note also that line-wrapping has occurred in copying the Makefile into this document—beware if you are re-using it.

```
UNI_ROOT = ${PEGASUS_ROOT}/../../UNI-C

ROOT = $(PEGASUS_ROOT)

TARGET = ${PEGASUS_HOME}/lib/libNT_UNI_Provider.so

INSTANCE_PROVIDER_OBJECTS = NT_UNI_Provider.o \
                                NT_UNI_ProviderMain.o \
                                corbaMessageHandlerC.o

TAO_ROOT=/usr/local/src/ACE/ACE_wrappers/TAO
ACE_ROOT=/usr/local/src/ACE/ACE_wrappers

LIBRARY_FLAGS = -L${PEGASUS_HOME}/lib

PROVIDERROOT = $(PEGASUS_ROOT)/../providers

DIR = Providers/ManagedSystem/OperatingSystem

COMPILE_FLAGS = -W -Wall -Wpointer-arith -pipe -O3 -g \
                -Wno-uninitialized \
                -fno-implicit-templates \
                -D_POSIX_THREADS \
                -D_POSIX_THREAD_SAFE_FUNCTIONS \
                -D_REENTRANT \
                -DACE_HAS_AIO_CALLS \
                -DACE_HAS_EXCEPTIONS \
                -Wno-unused  \
                -D_GNU_SOURCE  \
                -DTHREAD_SAFE  \
                -DPEGASUS_PLATFORM_LINUX_IX86_GNU

LINK_FLAGS = -W -Wall -Wpointer-arith -pipe -O3 -g \
                -Wno-uninitialized \
                -fno-implicit-templates \
                -D_POSIX_THREADS \
                -D_POSIX_THREAD_SAFE_FUNCTIONS \
                -D_REENTRANT \
                -DACE_HAS_AIO_CALLS \
                -DACE_HAS_EXCEPTIONS \
                -L${TAO_ROOT}/tao \
                -L${ACE_ROOT}/ace \
                -L${TAO_ROOT}/orbsvcs/orbsvcs \
                -L./ \
                -shared

EXTRA_INCLUDES = -I$(PROVIDERROOT) -I${ACE_ROOT} \
                     -I${TAO_ROOT} -I${PEGASUS_ROOT}/src \
                     -I${UNI_ROOT}
```

```
SOURCES = \
        NT_UNI_Provider.cpp \
        NT_UNI_ProviderMain.cpp

LIBRARIES = \
        pegcommon \
        pegprovider

DYNAMIC_LIBRARIES = \
        -lpegcommon \
        -lpegprovider \
        -lTAO_PortableServer \
        -lTAO -lACE -ldl -lpthread -lrt -lTAO_CosNaming

# specific rules for making the library

${TARGET} : ${INSTANCE_PROVIDER_OBJECTS} Makefile
        ${CC} ${LINK_FLAGS} ${LIBRARY_FLAGS} -o ${TARGET} \
          ${INSTANCE_PROVIDER_OBJECTS} ${DYNAMIC_LIBRARIES}

# specific rules for making the .o files

NT_UNI_Provider.o : NT_UNI_Provider.cpp Makefile \
                                  NT_UNI_Provider.h
        ${CC} ${COMPILE_FLAGS} ${EXTRA_INCLUDES} -c \
                NT_UNI_Provider.cpp

NT_UNI_ProviderMain.o   : NT_UNI_ProviderMain.cpp Makefile \
                            NT_UNI_Provider.h
       ${CC} ${COMPILE_FLAGS} ${EXTRA_INCLUDES} -c \
                            NT_UNI_ProviderMain.cpp

corbaMessageHandlerC.o : ${UNI_ROOT}/corbaMessageHandlerC.cpp\
          Makefile ${UNI_ROOT}/corbaMessageHandler.idl \
         ${CC} ${COMPILE_FLAGS} ${EXTRA_INCLUDES} \
                -c ${UNI_ROOT}/corbaMessageHandlerC.cpp
```

Appendix A:References

A.1 Industry Standards and Drafts

(1) **Common Information Model (CIM) Specification, Document DSP004, Distributed Management Task Force (version 2.2, June 14th 1999 is the latest version at the time of writing)**

A.2 Related Documents

(2) **Open Source Development with CVS, 2nd Edition, Fogel and Bar, ISBN 1-58880-173-X**

Appendix B: Useful Tips

This appendix contains some useful tips and wrinkles which we found empirically while writing a provider. They may not represent the best practices but they worked for us.

B.1 Making CGIClient

The `CGIClient` which is invoked by the Apache server when using the HTML interface needs to be placed into `$PEGASUS_HOME/cgi/cgi-bin`.

Note that when `make` is typed in the `$PEGASUS_ROOT/src/Clients/CGIClient` directory, the object file is placed only in `$PEGASUS_HOME/bin` directory. In order to get it into the correct directory, it is necessary to run `make` from the `$PEGASUS_ROOT/cgi` directory.

B.2 Tracing

The trace level and trace file can be set in the configuration files (see "cimconfig" on page 13). Trace level 4 effectively traces everything (debug-level) and trace levels down to 1 are supposed to trace progressively less and less information.

Pegasus provides a very useful pair of macros:

```
PEG_METHOD_ENTER(integer,string);

PEG_METHOD_EXIT();
```

The first of these can be placed as the first line in a method and it causes a useful trace message, including the string, to be logged. The integer needs unfortunately to come from the enum `TRACE_COMPONENT_ID` in `${PEGASUS_ROOT}/src/Pegasus/Common/TraceComponents.h` and there appears to be no “other” element in this enum.

The second of these is then included immediately before the `return` statement in the method.

B.3 Bug Fixes

Note that these bug fixes relate only to the version of Pegasus described in "Introduction" on page 6.

$PEGASUS_ROOT/cgi/htdocs/setProperty.htm, line 18, alter ClassName to InstanceName.

$PEGASUS_ROOT/cgi/htdocs/setProperty.htm, line 13, alter GetProperty to SetProperty.

$PEGASUS_ROOT/cgi/htdocs/getProperty.html, line 17, alter ClassName to InstanceName.

$PEGASUS_ROOT/src/Pegasus/ProviderManager/ProviderManagerService.cpp, line 1574, remove initialisation of the exception class to "not implemented". Line should simply read CIMException(),.

$PEGASUS_ROOT/src/Pegasus/ProviderManager/ProviderManagerService.cpp, line 1610, remove comment symbol. Line should read String propertyName = request->propertyName;.

$PEGASUS_ROOT/src/Pegasus/ProviderManager/ProviderManagerService.cpp, line 1611, remove comment symbol and change the line to read CIMValue propertyValue = request->newValue;.

B.4 Bug Non-Fixes

In addition to those listed above, another bug was found which was eventually worked around rather than being fixed.

The bug manifested itself in the client making calls to the invokeMethod method of the CIMOM:

```
try
    {
    Array<CIMParamValue> outParams;
    Array<CIMParamValue> inParams;
    CIMClient *client;

    client = new CIMClient();
    client->connect(cimomAddress,
                            String::EMPTY, String::EMPTY);

    inParams.append(CIMParamValue("index",
                      CIMValue(Sint32(startPoint))));

    retValue = client->invokeMethod(nameSpace,
                      CIMObjectPath(instanceName),
                      "getNextLightPath",
                       inParams,
                       outParams);

    // note: the following disconnexion does not seem
    //       to be necessary because the destructor
```

```
//       for CIMClient calls the destructor for
//       the embedded CIMClientRep and this
//       seems to do a disconnect. The documentation,
//       however, says that calling disconnect
//       twice cannot do any harm.

client->disconnect();

cout << "Entering destructor" << endl;
delete(client);
cout << "Back from destructor" << endl;
.....etc.....
```

This code crashes in the `delete(client)` call with a segmentation violation. When the destructor of the `CIMClient` is called it calls various other destructors ending up in a routine called `AsyncOpNode.cpp`. This routine maintains a free list of `AsyncOpNode` blocks. Something appears to be corrupting the _parent pointer in one of these blocks *while it is on the free list*.

We therefore set a watchpoint on that location and got the following ddd trace.

The _parent pointer is correctly set to 0x8059408 and then is overwritten with the value 0x5, which is certainly not a valid pointer (on one occasion it was set to the value 0x4D3C0A3E which is also not a valid pointer). This is corrected by the call to `AsyncOpNode::delete` but eventually nemesis catches up with us and we die in a horrible mess.

For the purposes of getting a demonstration running, the code was changed to create and connect a CIMClient only once. This “worked” but presumably just moved the problem elsewhere.

```
(gdb) watch node->_parent
Watchpoint 2: node->_parent
(gdb) cont
Watchpoint 2: node->_parent

Old value = (AsyncOpNode *) 0x8059408
New value = (AsyncOpNode *) 0x5
0x40101f7b in Pegasus::Mutex::unlock () from /users/guest/tpc/
demo/felix/demo-Phase-I/lib/libpegcommon.so
(gdb) cont
Watchpoint 2: node->_parent

Old value = (AsyncOpNode *) 0x5
New value = (AsyncOpNode *) 0x8059408
0x40101f80 in Pegasus::Mutex::unlock () from /users/guest/tpc/
demo/felix/demo-Phase-I/lib/libpegcommon.so
(gdb) cont
Watchpoint 2: node->_parent

Old value = (AsyncOpNode *) 0x8059408
New value = (AsyncOpNode *) 0x5
0x400c6850 in Pegasus::AsyncOpNode::operator new (size=264) at
AsyncOpNode.cpp:65
(gdb) print *node
```

```
$4 = {static _headOfFreeList = 0x8059408, static BLOCK_SIZE =
200, static _alloc_mut = {_mutex = {mut = {__m_reserved = 0,
__m_count = 0, __m_owner = 0x0, __m_kind = 0, __m_lock =
{__status = 0, __spinlock = 0}}, mutatt = {__mutexkind = 0},
owner = 0}}, _client_sem = {_semaphore = {sem = {__sem_lock =
{__status = 112, __spinlock = 7}, __sem_value = 1, __sem_waiting
= 0x1200000}, owner = 0}, _count = 0}, _mut = {_mutex = {mut =
{__m_reserved = 0, __m_count = 0, __m_owner = 0x5, __m_kind =
134637312, __m_lock = {__status = 0, __spinlock = 0}}, mutatt =
{__mutexkind = 0}, owner = 0}}, _request = {_rep = 0x0, _next =
0x0, _prev = 0x0, _cur = 0x1, _isHead = 88, _count = 1075092568,
_vptr. = 0x0}, _response = {_rep = 0x1, _next = 0xffffff01,
_prev = 0x0, _cur = 0x0, _isHead = false, _count = 1392508928,
_vptr. = 0x61}, _operation_list = {_rep = 0x404596a8, _vptr. =
0x404596a8}, _state = 0, _flags = 0, _offered_count = 0,
_total_ops = 0, _completed_ops = 0, _user_data = 1,
_completion_code = 778398825, _op_dest = 0x39, _start = {tv_sec
= 134639192, tv_usec = 1078302424},_lifetime = {tv_sec =
1769235301, tv_usec = 28271}, _updated = {tv_sec = 0, tv_usec =
33}, _timeout_interval = {tv_sec = 134569472, tv_usec =
1078302400},_parent = 0x5, _children = {_rep = 0x8066668, _next
= 0x0, _prev = 0x0, _cur = 0x60, _isHead = 48, _count = 0, _vptr.
= 0x0}, _async_callback = 0,__async_callback = 0,
_callback_node = 0x1, _callback_response_q = 0x1, _callback_ptr
= 0x0, _callback_parameter = 0x0, _callback_handle =
0x706d6f63,_callback_notify = 0x7461, _callback_request_q =
0x0, _service_ptr = 0x19, _thread_ptr = 0x80667b0,
_source_queue = 134637440}
(gdb) cont

Watchpoint 2 deleted because the program has left the block in
which its expression is valid.
0x400c6852 in Pegasus::AsyncOpNode::operator new
(size=134572720) at AsyncOpNode.cpp:65
/users/guest/tpc/demo/felix/demo-Phase-I/CIMOM/pegasus/src/Pe-
gasus/Common/AsyncOpNode.cpp:65:2430:beg:0x400c6852
(gdb) bt
#0  0x400c6852 in Pegasus::AsyncOpNode::operator new
(size=134572720) at AsyncOpNode.cpp:65
#1  0x400df9a0 in Pegasus::cimom::_shutdown_routed_queue ()
from /users/guest/tpc/demo/felix/demo-Phase-I/lib/libpegcom-
mon.so
#2  0x401067f3 in Pegasus::MessageQueueService::~Message-
QueueService () from /users/guest/tpc/demo/felix/demo-Phase-I/
lib/libpegcommon.so
#3  0x4010016f in Pegasus::HTTPConnector::~HTTPConnector ()
from /users/guest/tpc/demo/felix/demo-Phase-I/lib/libpegcom-
mon.so
#4  0x40257a8e in Pegasus::CIMClientRep::~CIMClientRep () from
/users/guest/tpc/demo/felix/demo-Phase-I/lib/libpegclient.so
#5  0x4025e4b6 in Pegasus::CIMClient::~CIMClient () from /us-
ers/guest/tpc/demo/felix/demo-Phase-I/lib/libpegclient.so
#6  0x0804e59e in getNextLightPath (lp=0xbffffa10, start-
Point=0) at DemoCGIClient.cpp:634
#7  0x0804ac76 in listAllLightPaths (qs=@0xbffffa78) at
DemoCGIClient.cpp:174
#8  0x0804f536 in main (argc=1, argv=0xbffffae4) at DemoCGICli-
ent.cpp:869
```

Appendix C: Pegasus Licencing Agreement

BEGIN LICENSE

Copyright (c) 2000, 2001, 2002 BMC Software, Hewlett-Packard Company, IBM, The Open Group, Tivoli Systems, Nortel Networks

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE ABOVE COPYRIGHT NOTICE AND THIS PERMISSION NOTICE SHALL BE INCLUDED IN ALL COPIES OR SUBSTANTIAL PORTIONS OF THE SOFTWARE. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

END LICENSE

CIM Exploration

Writing a Pegasus CIM Provider

© Copyright 2002 Nortel Networks

This document is the property of Nortel Networks and is licenced for use under the terms and conditions of the Pegasus Licensing Agreement (see "Pegasus Licencing Agreement" on page 40).

29 October 2002