VMware Smart Assurance Dynamic Modeling Tutorial

VMware Smart Assurance 10.0.0

vmware®

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2019 VMware, Inc. All rights reserved. Copyright and trademark information.

Contents

1 Introduction 5

Availability, features, and licensing 5

Differences between MODEL and Dynamic Model 6

Restrictions of dynamic modeling 7

Developing dynamic models 9

Location and loading of dynamic model files 13

2 Exercise 1 – Adding a New Printer Class 16

Objectives 16

Exercise 17

3 Exercise 2 - Adding an Attribute to the Router Class 31

Objectives 31

Exercise 32

4 Exercise 3 - Polling for Values of Additional Attributes 46

Objectives 46

Exercise 47

5 Guidelines and Syntax for Dynamic Model Files 61

Guidelines and requirements 61

Adding a class to the model 62

Updating an existing class in the model 63

6 The dynmodel Command Syntax and Options 69

Syntax and options 69

Pragmas 72

7 Methods of Creating Classes and Populating Attributes 73

Overview 73

Using dmctl 73

Populating attributes by using an SDK API 78

Discovery customization 79

8 Updating the Global Manager 85

Loading the dynamic model enhancements 85

Transferring topology and events 86

9 Troubleshooting 90

Overview 90

Loading errors 90

Discovery errors 95

10 MODEL and Dynamic Models 97

The ICIM Reference 97

Representing systems with MODEL 98

Differences between MODEL and Dynamic Model 99

Introduction

1

This chapter includes the following topics:

- Availability, features, and licensing
- Differences between MODEL and Dynamic Model
- Restrictions of dynamic modeling
- Developing dynamic models
- Location and loading of dynamic model files

Availability, features, and licensing

Dynamic Model is an extension of the Managed Object Definition Language (MODEL). MODEL is integral to the VMware Smart Assurance SDK application development process. Its purpose is to render the schema and semantics of the VMware Smart Assurance ICIM Reference specification. These libraries are loaded by a Domain Manager and are operated on by SDK applications. Foran SDK application, all of the classes, relationships, attributes, and events are defined at compile time.

Dynamic modeling enables you to add new classes, and refine classes that are already defined in the data model libraries. These additions and refinements occur at runtime when the dynamic models are loaded into a Domain Manager.

This section describes the different products that currently support Dynamic Model. It also gives an overview of the features that are available.

Availability of dynamic modeling

Dynamic modeling is supported on all Domain Managers.

Features of dynamic modeling

The following features are provided by Dynamic Model:

- Classes may be made known to the system at runtime, when they are loaded into a Manager as a dynamic model.
- A dynamic model compiler that produces a file with a .ldm extension that can be loaded by a Manager.

- You can load a dynamic model when the Domain Manager is started or after the Domain Manager is running.
- When you load a dynamic model, the existing model class dependencies and the new dynamic model class dependencies are resolved.
- A new class may be defined that derives from any existing ICIM class that is part of the ICIM_ManagedElement and ICIM_MetaObject trees. The VMware Smart Assurance ICIM Reference provides information for determining which classes fall under the ICIM_ManagedElement and ICIM_MetaObject trees.
- An existing ICIM class may be extended by defining new attributes, events, aggregates, relationships, and operations for the class.
- An existing ICIM class may also be extended, by defining new problems and symptoms.
- You can refine relationships, events, aggregates, operations, and attributes in an existing ICIM class.
- You can refine problems and symptoms in an existing ICIM class.

Differences between MODEL and Dynamic Model

Although MODEL and Dynamic Model are used independently and involve different steps and tools, the dynamic model is fully derived from MODEL so many of the concepts that apply to MODEL also apply to the dynamic model. Familiarity with the VMware Smart Assurance MODEL Reference Guide is useful when working with dynamic models.

Differences Between MODEL and Dynamic Model presents a summary of the differences between using MODEL to create a custom solution and using Dynamic Model to enhance an existing solution.

Table 1-1. Differences Between MODEL and Dynamic Model

Feature	MODEL	Dynamic Model
Define new classes	3	3
Use “refine interface” to extend an existing ICIM class	No	3
Use “refine interface” to extend a subclass of a new subclass	No	3
Define new problems	3	3
Define new relationships	3	3
Refine existing relationships, events, aggregates, operations, and attributes	3	3
Refine existing problems and symptoms	3	3
Define new symptoms	3	3
Define new events	3	3
Define new stored attributes	3	3
Define new propagated attributes	3	3
Define new instrumented attributes	3	3
Define new computed attributes	3	3

Table 1-1. Differences Between MODEL and Dynamic Model (Continued)

Feature	MODEL	Dynamic Model
Define new aggregates	3	3
Define new operations	3	3
Delete existing ICIM classes	No	No

The refine interface keyword

Dynamic model uses the **refine interface** keyword to indicate that any declarations that follow will modify an existing ICIM class. In MODEL, you cannot refine an existing class.

The refine keyword

In both Dynamic Model and MODEL, the **refine** keyword is used to refine existing inherited properties of a class, such as attributes, events, and so on. You can refine inherited class relationships, attributes and events when you create a new subclass. These class properties may be inherited from the parent class or any of the other classes up the inheritance tree to the base class.

Restrictions of dynamic modeling

This section describes the restrictions that you need to be aware of when using dynamic modeling.

Preprocessor macros

Preprocessor macros, including the preprocessor statement "*#include*", may not be used in Dynamic Model source code.

Pragmas

Only those pragmas listed in Pragmas, can be used with Dynamic Model. Pragmas that produce output in C++ source files may not be used, since Dynamic Model does not generate C++ source files.

User-defined source code

User-defined source code that produces C++ stubs may not be included in Dynamic Model source files.

ICIM_ManagedDomain

ICIM_ManagedDomain and any of its subclasses cannot be subclassed or refined.

Single class refinement

All refinements to a class *must* be contained in a single

refine interface <*classname*> declaration. For example, you cannot have a

refine interface Router declaration in the mymodel1.mdl file, and then another **refine interface Router** in the mymodel2.mdl. Likewise, you cannot have more than one **refine interface Router** declaration in the mymodel1.mdl file.

MODEL keywords and data types not available in the Service Assurance Manager

There are several MODEL keywords and data types that can be used in the dynamic model files of the IP Manager, but not in Service Assurance Manager.

- binary
- byte
- external

  Avoid using the binary or byte data type, or the **external** keyword in any dynamic model files of the IP Manager that you will be copying over to the Service Assurance Manager.

The binary data type

The binary data type represents binary data and can be used in IP Manager dynamic model files. However, the binary data type is not available in the Service Assurance Manager dynamic model files.

After copying over dynamic model files from the IP Manager to the Service Assurance Manager, you must remove any instances of the binary data type before compiling.

byte data type

The byte data type represents an unsigned 8-bit integer ranging from 0 - 255 in IP Manager dynamic model files. However, the byte data type is not available in the Service Assurance Manager dynamic model files.

After copying over dynamic model files from the IP Manager to the Service Assurance Manager, you must replace any instances of the byte data type with the int data type before compiling.

external keyword

The **external** keyword can be used in IP Manager dynamic model files.The **external** keyword indicates whether the literal text specified as the initial value of a string attribute will be externalized. The **external** keyword implies both static and readonly. When including the **external** keyword in a declaration, do not include the **static** and **readonly** keywords.

The **external** keyword is not available in Service Assurance Manager dynamic model files.

After copying over dynamic model files from the IP Manager to the Service Assurance Manager, you must remove any instances of the **external** keyword before compiling.

Developing dynamic models

This section describes the guidelines that should be used when using dynamic models. It also describes the steps that are involved in the development cycle when extending a model of a Domain Manager.

Guidelines

Follow these guidelines when using dynamic models:

- Create and verify dynamic models on a test Manager prior to deploying them into any production environment. The test Manager may be started and stopped several times during the process to verify the results of each step.
- Create and edit the dynamic model files (those with extension .mdl) by using **sm_edit**.

  Note Only the .mdl files are created and edited by using **sm_edit**. The .ldm file generated by the **dynmodel** compiler *must* not be edited.

- Classes that are related to each other should be contained in the same dynamic model file.
- Dynamic model files can be loaded when a Manager is started or after the Domain Manager is already running. Location and loading of dynamic model files provides additional information.
- Perform a back up on the repository of the Manager before using dynamic modeling in a production environment.
- Precede any new attributes, classes, aggregates or events that are defined with a unique tag, such as “USR_”. You should not use "SM_" as this is already used by SDK.
- Avoid using the binary or byte data type, or the **external** keyword in any dynamic model files of the IP Manager that you will be copying over to the Service Assurance Manager.

  Note Changes made by using dynamic modeling may impact the performance and load of the Domain Manager.

Requirements

The following requirements must be followed when using dynamic model:

- Dynamic model filenames may contain ASCII letters (A..Z,a..z), ASCII numbers (0...9), and the underscore character (_), but may contain no other characters. The first character must be a letter.
- The dynamic model filename must be unique within the first 23 characters.
- The dynamic model file that you edit must have a .mdl filename extension.
- Dynamic model files must follow the syntax rules described in Adding a class to the model and Updating an existing class in the model.
- All refinements to a class *must* be contained in a single
- **refine interface** <*classname*> declaration. For example, you cannot have a

- **refine interface Router** declaration in the mymodel1.mdl file, and then another **refine interface Router** in the mymodel2.mdl. Likewise, you cannot have more than one **refine interface Router** declaration in the mymodel1.mdl file.
- Dynamic model files (those with extension .ldm) must be located in either BASEDIR/smarts/modelorBASEDIR/smarts/model/<*config*>. The <*config*> refers to the configuration directory that is specified on the command line when the Domain Manager is started. Loading dynamic models during Manager startup provides further details.

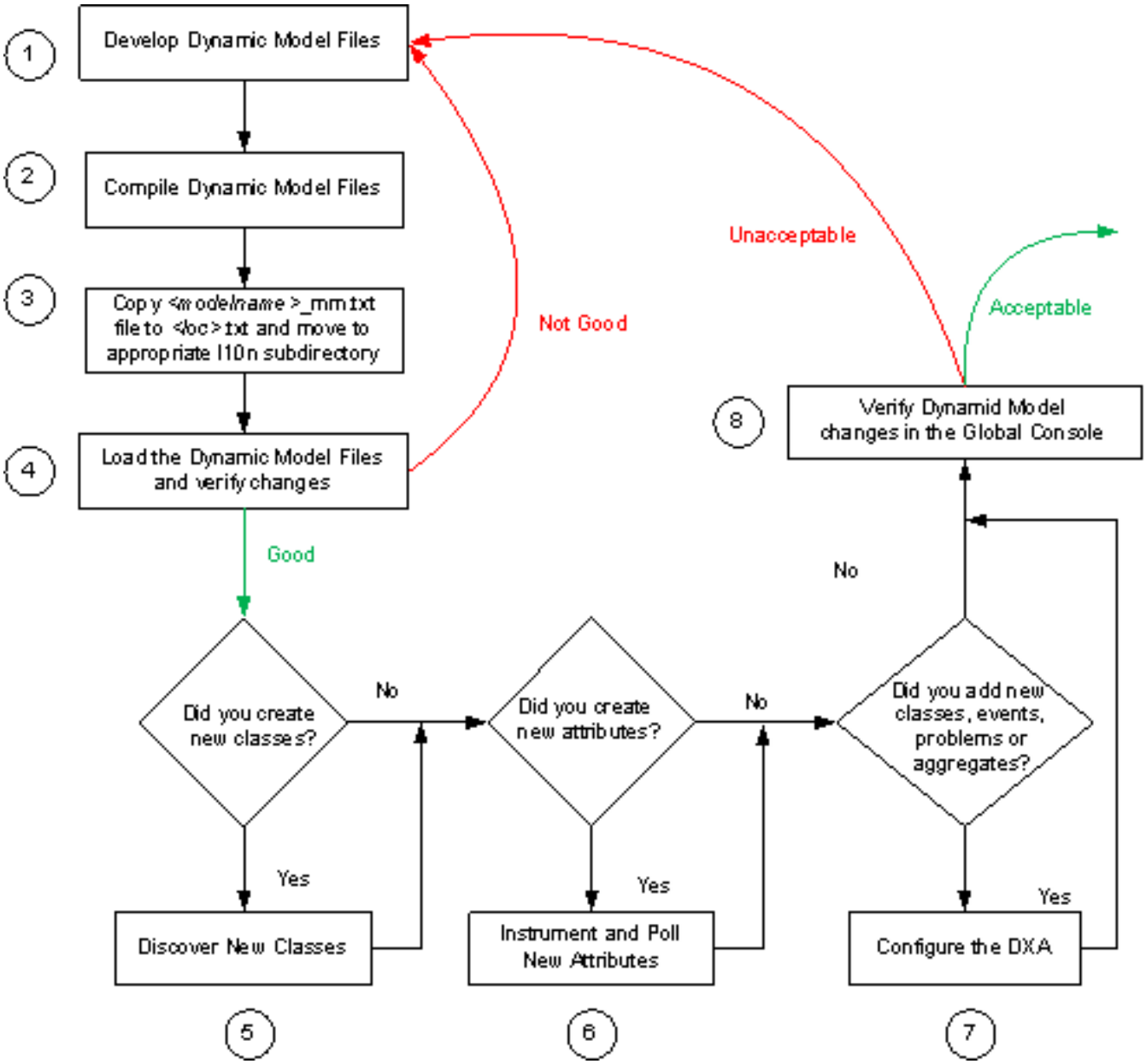
  Note It is recommended to have both the .mdl and .ldm versions of a dynamic model file located together in either the BASEDIR/smarts/model or BASEDIR/smarts/model/<*config*> directory.

- The dynamic model files (those with extension .ldm) must be loaded into each Manager that requires knowledge of the new topology and events.

Development cycle

Several steps are involved in the development cycle of extending a model of a Domain Manager, by using dynamic modeling. Steps involved in developing dynamic models illustrates the development cycle.

Figure 1-1. Steps involved in developing dynamic models



The first five steps are common to all dynamic modeling development.

1 Add the new model elements to either a new dynamic model text file (.mdl) or an existing one. Chapter 5 Guidelines and Syntax for Dynamic Model Files contains additional information.

2 Compile the dynamic model text file into a file (those with extension .ldm) that can be loaded into a Manager. The command used to compile the dynamic model file is described in Chapter 6 The dynmodel Command Syntax and Options

3 If you require the string literals in the dynamic model text file to be of a language other than the language in the dynamic model text file:

4 Copy the *<modelname>*_mm.txt file to *<loc>*.txt, and move it to the appropriate l10n subdirectory. The *<loc>* refers to a locale such as, fr_FR for French as spoken in France.

5 Edit <*loc*>.txt. Change the “root {“ to “<*loc*> {“ in the first line. Translate the string literals as required. Return to step 2 on page 22 .

6 From the *BASEDIR/*smarts/model directory, copy the following files to the *SM_SITEMOD*/l10n directory:

- Example1_mmbe.dat
  - Example1_mmle.dat

7 Load the compiled dynamic model file into a Domain Manager. Verify that the new model elements have been added to the list of classes or the list of properties by using the Global Console.

If you encounter errors, go back to step 1 on page 22 and make the appropriate modifications to your .mdl file. Loading errors provides additional troubleshooting information.

8 If you have defined new classes, create or discover instances of those classes. *Appendix C, “Methods of Creating Classes and Populating Attributes,”* provides additional information.

9 If you have defined new attributes, populate the values of those attributes, either manually or by using the discovery mechanism. *Appendix C, “Methods of Creating Classes and Populating Attributes,”* provides additional information.

10 If you want to view the new classes, attributes, problems, aggregates or events in the Global Manager, then configure the appropriate DXA. Chapter 8 Updating the Global Manager describes DXA configuration.

11 If you added new events, then add a subscription to the event in the appropriate bootstrap.conf files.

12 Verify the topology in the Global Console.

If you encounter errors, go back to step 1 on page 22 and make the appropriate modifications. Chapter 9 Troubleshooting provides additional information.

Model enhancement examples

This tutorial presents three different exercises:

- Chapter 2 Exercise 1 – Adding a New Printer Class, adds a new printer class to the model.
- Chapter 3 Exercise 2 - Adding an Attribute to the Router Class, adds a new attribute to an existing ICIM class.
- Chapter 4 Exercise 3 - Polling for Values of Additional Attributes, adds an event whose state is determined by a new attribute. The exercise also presents how to set up polling of the value for the new attribute.

The source files for the exercises are located in the following directory:

```
BASEDIR/smarts/examples/dynamic-model
```

Location and loading of dynamic model files

Dynamic model files can be located in either the BASEDIR/smarts/local/model or BASEDIR/smarts/local/model/<*config*> directories. The <*config*> refers to the configuration directory that was specified on the command line when the Domain Manager was started.

Dynamic models can be loaded into a Manager during startup or while it is running.

The examples in this guide use the **sm_server** command to load the dynamic models into a Manager during startup. If you use **sm_service** to start your Manager, edit the command string used by **sm_service** to include the appropriate option to load your dynamic models. The VMware Smart Assurance System Administration Guide provides additional information.

Loading dynamic models into a running Manager

The loadModel operation in the SM-System instance of the SM_System class is used to load a model into a running Manager. All dynamic model files must be located in the BASEDIR/smarts/model directory. The **dmctl** utility is used to invoke the loadModel operation. Using dmctl provides additional information. The syntax for the command line is as follows:

```
t#

         BASEDIR/smarts/bin/dmctl -s <domain_manager> [-b <broker>] invoke SM_System::SM-System
loadModel <modelfilename> SM_DYNAMIC_LIBRARYs
```

where:

- <*domain_manager*> is the name of the Domain Manager.
- (Optional) <*broker*> is the name of the Broker to which the Domain Manager has been registered. The default is localhost:426.
- <*modelfilename*> is the name of the model file without the extension.

  For example, you would type the following to load the Example3.ldm dynamic model file into the TESTDM Domain Manager attached to the broker on localhost:426.

  ```
  t#

                 BASEDIR
                 /smarts/bin/dmctl -s TESTDM -b localhost:426 invoke SM_System::SM-System loadModel
  Example3 SM_DYNAMIC_LIBRARYs
  ```

  Verify that the model file was loaded. If you started your Domain Manager with the

  -- output option, you can check the log file. Your output should be similar to the following:

  ```
  [July 10, 2009 12:25:29 PM EDT +789ms] t@1120213312 LDM file processing #1
  DYN-W-FILE_OPEN_OK- Reading
      '/opt/InCharge8/IP/smarts/local/model/Example3.ldm'
  [July 10, 2009 12:25:44 PM EDT +141ms] t@1120213312 LDM file processing #1
  ```

```
MRM-*-MRM_PROP_INIT-Property TS_OverrunThreshold initial value ='10'
[July 10, 2009 12:25:44 PM EDT +142ms] t@1120213312 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of TS_OverrunThreshold to 10
[July 10, 2009 12:25:44 PM EDT +143ms] t@1120213312 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for TS_OverrunThreshold = 10
[July 10, 2009 12:25:44 PM EDT +144ms] t@1120213312 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface NetworkAdapter_Performance_Setting
    refined from Example3.mdl:1
[July 10, 2009 12:25:44 PM EDT +145ms] t@1120213312 LDM file processing #1
MRM-*-MRM_PROP_INIT-Property TS_OverrunThreshold initial value ='10'
[July 10, 2009 12:25:44 PM EDT +146ms] t@1120213312 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of TS_OverrunThreshold to 10
[July 10, 2009 12:25:44 PM EDT +146ms] t@1120213312 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for TS_OverrunThreshold = 10
[July 10, 2009 12:25:44 PM EDT +147ms] t@1120213312 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface Interface_Performance_CiscoRouter
    refined from Example3.mdl:8
```

Loading dynamic models during Manager startup

A Manager can be started either without or with the **--**dynamic option. The use of the **--**dynamic option determines which .ldm files are loaded into the Manager.

Starting a Manager without the --dynamic option

If you start a Manager without the **--**dynamicoption, the following .ldm files will be loaded:

- *BASEDIR/*smarts/model */<config>*

  The *<config>* refers to the configuration directory that is specified on the command line when the Domain Manager is started.

  In this example, a Global Manager is started without the **--**dynamic option. This example loads all of the .ldm files in *BASEDIR/*smarts/model */*ics since ics is the configuration directory specified in the command line.

  ```
  sm_server -n TEST-SA -c ics --output
  ```

Starting a Manager with the --dynamic option

Including the **--**dynamic option on the command line, when you start up a Manager, loads all of the .ldm files in the following directories:

- BASEDIR/smarts/model
- BASEDIR/smarts/model / *<config>*

  The *<config>* refers to the configuration directory that is specified on the command line when the Domain Manager is started.

In this example, all of the .ldm files in the BASEDIR/smarts/model and the BASEDIR/smarts/model/icf-r directories are loaded into the TEST_NR server:

```
sm_server -n TEST_NR -c icf-r --output --dynamic
```

Exercise 1 – Adding a New Printer Class

2

This chapter includes the following topics:

- Objectives
- Exercise

Objectives

This exercise describes how to use Dynamic Model to add a new printer class without attributes or other properties to the following:

- Network-Resource Manager
- IP-Perspective Manager
- IP-Analysis Manager
- Global Manager

This exercise also demonstrates how new instances can be classified in the new class, and then discovered. Instances of the new class will be discovered automatically, based on the device's MIB object ID (OID) when the Network-Resource Manager is started.

Note This exercise should be performed on test versions of a Broker, Network-Resource Manager, IP-Perspective Manager, IP-Analysis Manager, and Global Manager. Before starting this exercise, back up the repository on the test Managers.

This exercise includes the following tasks:

- Create and compile a dynamic model file.
- Add a new printer class to the models in a Network-Resource Manager, IP-Perspective Manager, IP-Analysis Manager, and Global Manager.
- View the new printer class by using the Global Console.
- Set up the printer OID for discovery.
- Reload the dynamic model file into the Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager.
- Set up topology sources for the IP-Perspective Manager, and the IP-Analysis Manager.

- Discover instances of the new class by using the Network-Resource Manager.
- Add the new printer class into the Global Manager model.

Exercise

After your network has been discovered, you may want to differentiate certain devices, such as personal computers or printers, from hosts in your topology. This exercise extends the model by adding a class to represent a printer. The printer class is declared without any additional properties, other than those attributes, events, and relationships that are inherited from its parent class.

A new class is declared as a subclass of an existing ICIM class. The key is to determine which class should be the parent class. This determination is based on several factors:

- How you want to search and report on the new class
- Functional similarities to existing elements in topology
- The events, attributes, and relationships the new class needs to inherit

  A printer is a physical element, on the same level as a router or a host. In this exercise, its parent class is the UnitaryComputerSystem, which is also the parent class of the router and host. A printer, router, and host all inherit the same attributes, events, and relationships from the UnitaryComputerSystem class.

  The IP Manager counts the instances of the ICIM_UnitaryComputerSystem. When your dynamic model classes inherit from ICIM_UnitaryComputerSystem, instances of that class will take advantage and participate on the full predefined IP Management Suite root cause analysis model.

  If the instances of your class do not need to participate on the full root cause analysis, you can inherit from either ICIM_System or ICIM_ComputerSystem. In addition, if you inherit from either ICIM_System or ICIM_ComputerSystem you will have to add your new class into the appropriate dxa to transfer the topology to the Global Manager. Chapter 3 Exercise 2 - Adding an Attribute to the Router Class provides information about adding a class and its associated attributes to dxa files.

  In the following procedures, the syntax is the same as that used to declare any MODEL class. Adding the new class provides more detail.

  Note Use a unique prefix with new class names and other properties. The name must differ from existing ICIM classes and properties. “SM_” is used as a prefix in this exercise. Do not use "SM_" as this is already used by SDK.

  Unless otherwise noted, BASEDIR refers to the directory for the IP Manager.

Creating and compiling the dynamic model

In this exercise, you create and compile a new dynamic model.

Creating the dynamic model

In this example, the same dynamic model is loaded into several Domain Managers, each starting with a different **--**conf directory. Instead of copying the same compiled dynamic model file (the file with extension .ldm) to each BASEDIR/smarts/local/model/<*conf*> directory, one single copy is created and compiled in BASEDIR/smarts/local/model. Each of the Domain Managers is then started with the **--**dynamic command line option to load the compiled dynamic model file from BASEDIR/smarts/local/model.

To create the dynamic model:

1 Create and edit a new file in the *BASEDIR/*smarts/modeldirectory. Type the following at a command prompt:

```
BASEDIR
/smarts/bin/sm_edit --create model/Example1.mdl
```

2 To declare the new printer class as a subclass of the UnitaryComputerSystem class, type the following lines into the Example1.mdl file.

```
interface SM_Printer:UnitaryComputerSystem
"Create a printer class inheriting from UCS"
{
}
```

3 Save and close the Example1.mdl file.

The file is saved in the *BASEDIR*/smarts/local/modeldirectory.

Compiling the dynamic model

From theBASEDIR/smarts/local/model directory, run the command to compile the Example1.mdl model file:

```
#

cd
BASEDIR
/smarts/local/model
#

BASEDIR
/smarts/bin/dynmodel Example1.mdl
```

If it is successful, the **dynmodel** command does not produce any output to the command-line console. In addition, the following files are created in the BASEDIR/smarts/local/model directory:

- Example1.ldm
- Example1_mmbe.dat
- Example1_mmle.dat
- Example1_mm.txt

The following subdirectory is also created in the BASEDIR/smarts/model directory:

- Example1_mm

Compilation errors on page 111 provides additional information if you encounter errors.

Copying files for localization

From the BASEDIR/smarts/model directory, copy the following files to the SM_SITEMOD/l10n directory:

- Example1_mmbe.dat
- Example1_mmle.dat

Loading and viewing the new class in the Domain Manager s

Once the dynamic model has been compiled, you must load it into the appropriate Domain Managers. The Printer class appears in the topology of the Network-Resource Manager, the IP-Perspective Manager, and the IP-Analysis Manager. The dynamic model is loaded into all three of these Domain Managers.

Loading the dynamic model into the Network-Resource Manager

In this procedure, TEST_SERVER_RM is the name of the test Network-Resource Manager. The Network-Resource Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note If the test Network-Resource Manager is already running, stop it first.

To load the dynamic model:

Note Two dash (**--**) characters precede the output and dynamic options of the **sm_server** command.

1 Load the compiled dynamic model file into the test Network-Resource Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_RM -c icf-r
```

```
                --broker=localhost:426 --bootstrap=bootstrap.conf --output

                --dynamics
```

2 Open the log file.

If the new SM_Printer class has been processed successfully, the output from the **sm_server** command appears similar to:

```
[May, 27, 2009 12:23:33 PM EDT +999ms] t@4097444752 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading
   'BASEDIR/smarts/local/model/Example1.ldm'
[May, 27, 2009 12:23:47 PM EDT +200ms] t@4097444752 LDM file processing #1
DYN-W-CLASS_DEFINITION_CREATED- Interface SM_Printer created from
   Example1.mdl:3
```

Verifying that the new class is present in the Network-Resource Manager

Before trying to discover instances of the new printer class, verify that the SM_Printer class has been added to the data model of TEST_SERVER_RM.

To verify the presence of the new class:

1 Open a Global Console. For this example, attach to TEST_SERVER_RM.

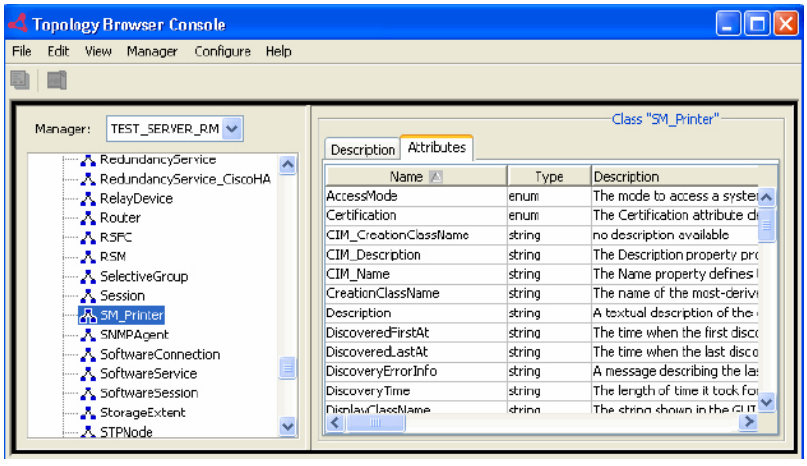
2 Right-click within the Topology Browser Console topology tree and select **Select** .

3 Select **Show classes with no instances in current domain**, and click **OK**.

4 In the Topology Browser Console, verify that the SM_Printer class has been added as shown in SM_Printer class in topology of TEST_SERVER_RM.

Note You can also use **dmctl**to verify whether the class is present in the topology. Command-line modeprovides further details.

Figure 2-1. SM_Printer class in topology of TEST_SERVER_RM



Loading the dynamic model into the IP-Perspective Manager

In this procedure, TEST_SERVER_P is the name of the test IP-Perspective Manager. The IP-Perspective Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note If the test IP-Perspective Manager is already running, stop it first.

To load the dynamic models:

Note Two dash (--) characters precede the output and dynamic options of the **sm_server** command.

1 Load the compiled dynamic model file into the test IP-Perspective Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_P -c icf-p

                 --broker=localhost:426 --bootstrap=bootstrap.conf --output

                 --dynamics
```

2 Open the log file. If the new SM_Printer class has been processed successfully, the output from the **sm_server** command appears similar to:

```
[May, 27, 2009 12:23:33 PM EDT +999ms] t@4097444752 LDM file processing
#1
DYN-W-FILE_OPEN_OK-Reading
   'BASEDIR/smarts/local/model/Example1.ldm'
[May, 27, 2009 12:23:47 PM EDT +200ms] t@4097444752 LDM file processing
#1
DYN-W-CLASS_DEFINITION_CREATED- Interface SM_Printer
created from
   Example1.mdl:3
```

Verifying that the new class is present in the IP-Perspective Manager

In the Topology Browser Console, attach to the TEST_SERVER_P Domain Manager. Verify that the SM_Printer class has been added to the data model of TEST_SERVER_P.

To verify the presence of the new class:

1 In the Topology Browser Console menu bar, select **Manager** > **Attach**.

2 In the **Attach Manager** dialog box, select **TEST_SERVER_P** in the **Manager** drop-down list. Type the appropriate username and password, and click **OK**.

3 In the Topology Browser Console, select **TEST_SERVER_P** from the **Manager** drop-down list.

4 Right-click within the Topology Browser Console topology tree and click **Select** . Select **Show classes with no instances in current domain**, and click **OK**.

5 In the Topology Browser Console, verify that the SM_Printer class has been added.

Note You can also use **dmctl** to verify whether the class is present in the topology. Command-line mode provides further details.

.

Loading the dynamic model into the IP-Analysis Manager

In this procedure, TEST_SERVER_A is the name of the test IP-Analysis Manager. The IP-Analysis Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note If the test IP-Analysis Manager is already running, stop it first.

To load the dynamic model:

Note Two dash (--) characters precede the output and dynamic options of **sm_server**.

1 Load the compiled dynamic model file into the test IP-Analysis Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_A -c icf-a

                 --broker=localhost:426 --bootstrap=bootstrap.conf --output

                 --dynamics
```

2 Open the log file. If the new SM_Printer class has been processed successfully, the output from the **sm_server** command appears similar to:

```
[May, 27, 2009 12:23:33 PM EDT +999ms] t@4097444752 LDM file processing
#1
DYN-W-FILE_OPEN_OK-Reading
   'BASEDIR/smarts/local/model/Example1.ldm'
[May, 27, 2009 12:23:47 PM EDT +200ms] t@4097444752 LDM file processing
#1
DYN-W-CLASS_DEFINITION_CREATED- Interface SM_Printer
created from
   Example1.mdl:3
```

Verifying that the new class is present in the IP-Analysis Manager

In the Topology Browser Console, attach to the TEST_SERVER_A Domain Manager. Verify that the SM_Printer class has been added to the data model of TEST_SERVER_A.

To verify the presence of the new class:

1 In the Topology Browser Console menu bar, select **Manager** > **Attach**.

2 In the **Attach Manager** dialog box, select **TEST_SERVER_A** in the **Manager** drop-down list, and type the password. Click **OK**.

3 In the Topology Browser Console, select **TEST_SERVER_A** from the **Manager** drop-down list.

4 Right-click within the Topology Browser Console topology tree and select **Select** . Select **Show classes with no instances in current domain**, and click **OK**.

5 In the Topology Browser Console, verify that the SM_Printer class has been added.

Note You can also use **dmctl** to verify whether the class is present in the topology. Command-line mode provides further details.

Setting up the printer OID for discovery

Printers are network devices, but they are classified generically as a host by the SDK discovery process. You can, however, add such devices to a field certification file so that SDK:

- Properly recognizes the devices
- Classifies the devices as instances of the new SM_Printer class

Determining if the printer is already defined

You need to determine if the printer is already defined generically (TYPE = Host), in one of the existing SDK certification files. Certification files are named oid2type_*.conf and are found in the BASEDIR/smarts/conf/discovery directory.

If the printer OID is already present in one of the existing certification files, use **sm_edit** to edit the particular oid2type file to comment out the entry.

In this procedure, the printer OID, .1.3.6.1.4.1.253.8.62.1.19.3.1.3, is also defined in BASEDIR/smarts/conf/discovery/oid2type_Misc.conf.

To remove the generic definition:

1 Edit the oid2type_Misc.conf file, by using **sm_edit**.

```
#

BASEDIR
/smarts/bin/sm_edit conf/discovery/oid2type_Misc.conf
```

2 Comment out the entry for the duplicate OID by preceding the lines with a pound sign (#):

```
#.1.3.6.1.4.1.253.8.62.1.19.3.1.3 {
#  TYPE = Host
#  VENDOR = HP
#  CERTIFICATION = VALIDATED
```

```
#
                #INSTRUMENTATION:
#  Interface-Fault   = MIB2
#  Interface-Performance  = MIB2
#}
```

3 Save and close the file.

The edited copy is saved in the *BASEDIR*/smarts/local/conf/discovery/ directory.

Adding the printer OID to the field certification file

To add the printer OID to the field certification file:

1 Open the oid2type_Field.conf file by using the **sm_edit** utility in the *BASEDIR*/smarts/bin directory.

```
#

cd
BASEDIR
/smarts/bin
#

sm_edit conf/discovery/oid2type_Field.conf
```

2 Type the following lines at the bottom of the oid2type_Field.conf file.

These lines create an instance of the SM_Printer class each time this model printer, designated by the OID (.1.3.6.1.4.1.253.8.62.1.19.3.1.3), is discovered in the network. The TYPE attribute designates this OID as a member of the SM_Printer class.

```
.1.3.6.1.4.1.253.8.62.1.19.3.1.3 {
     TYPE = SM_Printer
      VENDOR = XEROX
      MODEL = Xerox Phaser 5500
      CERTIFICATION = TEMPLATE
 INSTRUMENTATION:
      Interface-Fault                   = MIB2
      Interface-Performance             = MIB2
 }
```

3 Save and close the oid2type_Field.conf file.

The modified version of the file is saved to the *BASEDIR*/smarts/local/conf/discovery directory.

Reloading the dynamic model

Because the oid2type_Field.conf file is processed when a Domain Manager is first started, TEST_SERVER_RM, TEST_SERVER_P, and TEST_SERVER_A must be stopped and restarted for any changes to take effect. The new class must be loaded into all three Domain Managers.

The Domain Managers are started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

To reload the dynamic model:

1 Stop TEST_SERVER_RM.

2 Reload the compiled dynamic model file into TEST_SERVER_RM.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_RM -c icf-r

                --broker=localhost:426 --bootstrap=bootstrap.conf --output

                --dynamics
```

3 Stop TEST_SERVER_P.

4 Reload the compiled dynamic model file into the test Domain Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_P -c icf-p

                --broker=localhost:426 --bootstrap=bootstrap.conf --output

                --dynamics
```

5 Stop TEST_SERVER_A.

6 Reload the compiled dynamic model file into TEST_SERVER_A.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_A -c icf-a

                --broker=localhost:426 --bootstrap=bootstrap.conf --output

                --dynamics
```

Setting up topology sources

The Network-Resource Manager is added as a source of topology to both the IP-Perspective Manager and the IP-Analysis Manager. In addition, the IP-Perspective Manager is also added as a topology source to the IP-Analysis Manager.

Adding the Network-Resource Manager as a source to the IP-Perspective Manager

To add the Network-Resource Manager as a source:

1 In the Topology Browser Console, select **Configure> Domain Manager Administrative Console**.

2 In the Domain Manager Administrative Console, select **TEST_SERVER_P** in the **Manager** drop-down list.

3 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

4 In the **Add Source** dialog box, type **TEST_SERVER_RM** in the **Element Name** field. Keep **NETWORK-RESOURCE-TYPE** as the **Element Type.** Click **OK**.

Adding the Network-Resource Manager as a source to the IP-Analysis Manager

To add the Network-Resource Manager as a source:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_A** in the **Manager** drop-down list.

2 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

3 In the **Add Source** dialog box, type **TEST_SERVER_RM** in the **Element Name** field. Select **NETWORK-RESOURCE-TYPE** as the **Element Type**. Click **OK**.

Adding the IP-Perspective Manager as a source to the IP-Analysis Manager

To add the IP-Perspective Manager as a source:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_A** in the **Manager** drop-down list.

2 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

3 In the **Add Source** dialog box, select **IP-PERSPECTIVE-TYPE** in the **Element Type** drop-down list. Type **TEST_SERVER_P** in the **Element Name** field. Click **OK**.

Discovering an instance of the SM_Printer class

To discover a printer instance:

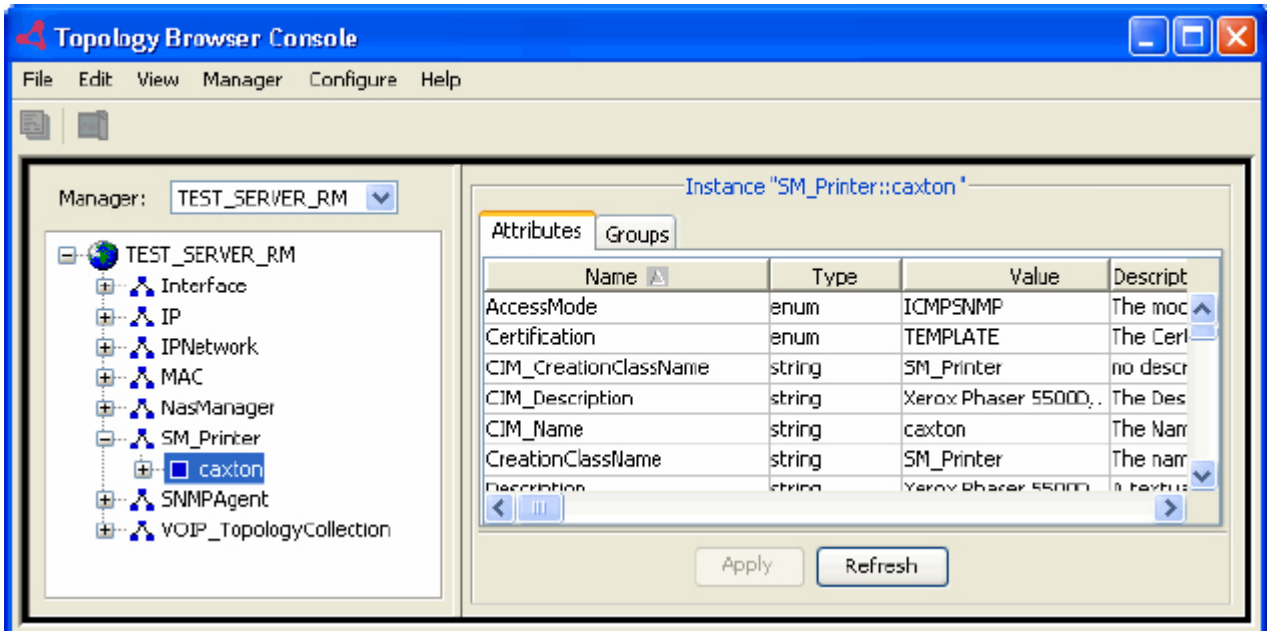
1 In the Domain Manager Administrative Console, select **TEST_SERVER_RM** in the **Manager** drop-down list.

2 Initiate discovery. One method is to select **Topology > Add Agent** *,* and type either the name or IP address of the printer.

The VMware Smart Assurance IP Management Suite Discovery Guide describes various methods of discovery.

3 Verify that the printer has been added to the topology as an instance of the new SM_Printer class as shown in Printer discovered as an instance of SM_Printer.

Figure 2-2. Printer discovered as an instance of SM_Printer



4 Select **TEST_SERVER_P** in the **Manager** drop-down list and verify that the printer instance has been added to the topology.

5 Select **TEST_SERVER_A** in the **Manager** drop-down list and verify that the printer instance has been added to the topology.

Adding the new class to the Global Manager

The new SM_Printer class has only been loaded into the test Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager. The test Global Manager does not contain the new class in its model. Therefore, the new class cannot be viewed in the Global Manager. This section explains:

- Compiling and loading the dynamic model into the Global Manager.
- Transferring the topology information from the Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager to the Global Manager.

Note In this section,BASEDIR is the directory for the Service Assurance Manager, unless otherwise noted.

Compiling and loading the dynamic model into the Global Manager

Note This exercise should be performed on a test version of a Global Manager. Before starting this exercise, back up the repository on the test Global Manager.

The dynamic model enhancements must be loaded into the Global Manager. Otherwise, the Global Manager has no awareness of the new SM_Printer class.

To compile and load the dynamic model into the Global Manager:

1 Copy Example1.mdlfrom the IP Manager's *BASEDIR*/smarts/local/modeldirectoryto the Service Assurance Manager's *BASEDIR*/smarts/local/modeldirectory.

2 From the Service Assurance Manager's *BASEDIR*/smarts/local/model directory, run the command to compile the Example1.mdl model file:

```
#

cd
BASEDIR
/smarts/local/model
#

BASEDIR
/smarts/bin/dynmodel Example1.mdl
```

3 Load the compiled dynamic model file into the test Global Manager.

```
t#

BASEDIR
/smarts/bin/sm_server -n TEST_SA -c ics

 --broker=localhost:426 --output --dynamics
```

This example registers TEST_SA with the Broker that is running on the host named "localhost" and port 426. Substitute the host and port of your Broker if they are different.

Adding the Network-Resource Manager as a source to the Global Manager

To add the Network-Resource Manager as a source:

1 In the Topology Browser Console menu bar, select **Manager** > **Attach**

2 In the **Attach Manager** dialog box, select **TEST_SA** in the **Manager** drop-down list. Type the appropriate username and password, and click **OK**.

3 In the Topology Browser Console, select **Configure> Global Manager Administrative Console**.

4 In the Global Manager Administrative Console, select **TEST_SA** in the **Manager** drop-down list.

5 In the Global Manager Administrative Console, select **Edit** > **New Domain**.

6 In the **Domain Creation Wizard** dialog box, type **TEST_SERVER_RM** in the **Domain Name** field. Select **Network-Resource** in the **Copy Existing** drop-down list. Click **Finish**.

Adding the IP-Perspective Manager as a source to the Global Manager

To add the IP-Perspective Manager as a source:

1 In the Global Manager Administrative Console, select **Edit** > **New Domain**.

2 In the **Domain Creation Wizard** dialog box, type **TEST_SERVER_P** in the **Domain Name** field. Select **IP-Perspective** in the **Copy Existing** drop-down list. Click **Finish**.

Adding the IP-Analysis Manager as a source to the Global Manager

To add a IP-Analysis Manager as a source:

1 In the Global Manager Administrative Console, select **Edit** > **New Domain**.

2 In the **Domain Creation Wizard** dialog box, type **TEST_SERVER_A** in the **Domain Name** field. Select **IP-Analysis** in the **Copy Existing** drop-down list. Click **Finish**.

Reconfiguring the Global Manager

To reconfigure:

1 In the Global Manager Administrative Console, select **Configure** > **Reconfigure Server**.

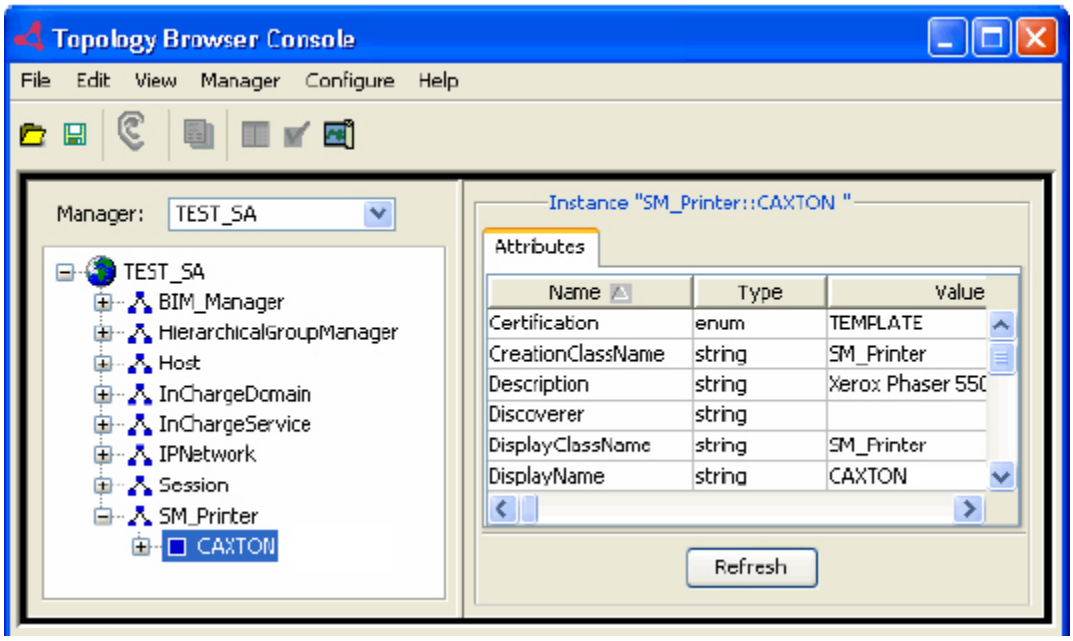
2 Click **Close** at the prompt.

Verifying the presence of the SM_Printer class instance in the Global Manager

To verify the presence of the SM_Printer class:

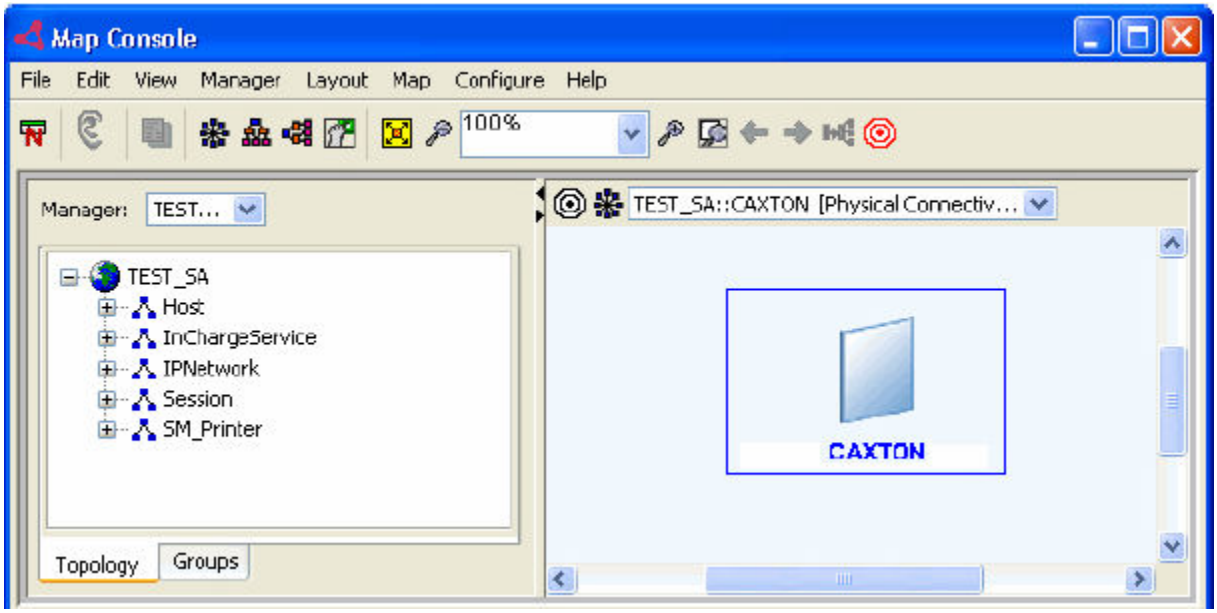
1 In the Topology Browser Console, verify that the SM_Printer class has been added, as shown in SM_Printer class in the TEST_SA Global Manager.

Figure 2-3. SM_Printer class in the TEST_SA Global Manager



2 Right-click the printerinstance (**CAXTON** in this procedure) and select **Show Map.** The printer instance is displayed in a Map Console, as shown in Map view of the SM_Printer instance.

Figure 2-4. Map view of the SM_Printer instance



3 If you have a custom icon to represent a printer, you can modify the icon used to display an instance of SM_Printer. The VMware Smart Assurance Service Assurance Manager Configuration Guide has more information on custom icons for map nodes.

Exercise 2 - Adding an Attribute to the Router Class

3

This chapter includes the following topics:

- Objectives
- Exercise

Objectives

In this exercise, you add a new attribute to an existing ICIM class in the following:

- Network-Resource Manager
- IP-Perspective Manager
- IP-Analysis Manager
- Global Manager

This exercise also demonstrates how the value of the new attribute can be populated.

Note This exercise should be performed on test versions of a Broker, Network-Resource Manager, IP-Perspective Manager, IP-Analysis Manager, and Global Manager. Before starting this exercise, back up the repository on the test Managers.

This exercise includes the following tasks:

- Create and compile a dynamic model file.
- Add an attribute to the existing Router class in the models of the Network-Resource Manager, IP-Perspective Manager, and the IP-Analysis Manager.
- View the new attribute in the Router class by using the Global Console.
- Set up topology sources for the IP-Perspective Manager, and the IP-Analysis Manager.
- Discover an instance of the Router class.
- Populate the value of the new attribute in the Network-Resource Manager, IP-Perspective Manager, and the IP-Analysis Manager.
- Compile and load the new Router attribute into the Global Manager model.
- Transfer the value of the new Router attribute to the Global Manager.

Exercise

This exercise describes how to add an attribute for the serial number to the existing Router class in order to extend the following models:

- Network-Resource Manager
- IP-Perspective Manager
- IP-Analysis Manager
- Global Manager

The syntax is the same as that used to declare any attribute in a MODEL class.

Note Use a unique prefix with new class names and other properties. The name must differ from existing ICIM classes and properties. “SM_” is used as a prefix in this exercise. Do not use "SM_" as this is already used by SDK.

Unless otherwise noted, BASEDIR refers to the directory for the IP Manager.

Creating and compiling the dynamic model

In this exercise, you create and compile a new dynamic model.

Creating the dynamic model

In this example, the same dynamic model is loaded into several Domain Managers, each starting with a different **--**conf directory. Instead of copying the same compiled dynamic model file (the file with extension .ldm) to each BASEDIR/smarts/local/model/<*conf*> directory, one single copy is created and compiled in BASEDIR/smarts/local/model. Each of the Domain Managers is then started with the **--**dynamic command line option to load the compiled dynamic model file from BASEDIR/smarts/local/model.

To create the dynamic model:

1 Create and edit a new file in the *BASEDIR*/smarts/local/modeldirectory. Type the following at a command prompt:

```
BASEDIR
/smarts/bin/sm_edit --create model/Example2.mdl
```

2 To add the serial number attribute to the Router class, type the following lines into the Example2.mdl file.

```
refine interface Router
{
 stored attribute string SM_SerialNumber
 "Custom serial number"
 = "NotAvailable";
}
```

3 Save and close the Example2.mdl file.

The file is saved in the *BASEDIR*/smarts/local/modeldirectory.

Compiling the dynamic model

From theBASEDIR/smarts/local/model directory, run the command to compile the Example2.mdl model file.

To compile:

```
#

cd
BASEDIR
/smarts/local/model
#

BASEDIR
/smarts/bin/dynmodel Example2.mdl
```

If it is successful, the **dynmodel** command does not produce any output to the command-line console. In addition, the following files and subdirectory are created in the BASEDIR/smarts/local/model directory:

- Example2.ldm
- Example2_mmbe.dat
- Example2_mmle.dat
- Example2_mm.txt

The following subdirectory is also created in the BASEDIR/smarts/local/model directory:

- Example2_mm

Compilation errors on page 111 provides additional information if you encounter errors.

Copying files for localization

From the BASEDIR/smarts/local/model directory, copy the following files to the SM_SITEMOD/l10n directory:

- Example2_mmbe.dat

- Example2_mmle.dat

Transferring the new attribute to the other Domain Manager s

Whether or not the new attribute is transferred from the Network-Resource Manager to the other Domain Managers depends on the purpose of the attribute. Several scenarios for updating dxa files between Domain Managers presents a few scenarios and describes which inter-Domain Manager dxa file needs to be updated with the new attribute.

Table 3-1. Several scenarios for updating dxa files between Domain Manager s

Scenario	Inter-Domain Manager dxa file that is modified
Variable is discovered in the Network-Resource Manager and only needs to appear in the Global Manager	None **Note** The attribute does need to be added to the Network-Resource Manager to Global Manager dxa.
Variable is made available to satellite Domain Managers through the IP-Analysis Manager	Network-Resource Manager to IP-Analysis Manager
Variable is to be used for postprocessing	Network-Resource Manager to IP-Perspective Manager

However, so that you may see all of the files that are involved, this example shows the steps to transfer the new attribute from the Network-Resource Manager to both the IP-Perspective Manager and the IP-Analysis Manager.

Transferring the new attribute from the Network-Resource Manager to the IP-Perspective Manager

The dxa-iptm-from-iprm.conf file provides a listing of the classes, and attributes of those classes, that will be transferred from the Network-Resource Manager to the IP-Perspective Manager.

To add the Router attribute to the dxa-iptm-from-iprm.conf file:

1 Edit the dxa-iptm-from-iprm.conf file by using **sm_edit**:

```
                #
        BASEDIR/smarts/bin/sm_edit conf/icf-p/dxa-iptm-from-iprm.conf
```

2 Type the following two lines for the Router class (shown in **bold**) before the entry for the UnitaryComputerSystem as follows:

```
# DXA file for transferring topology from IP Resource Manager
# to IP Perspective (topology) manager.
#
prehook    icf-p/iptm-iprm-dxa-prehook.asl
class Router
               attr  SM_SerialNumber
```

```
class UnitaryComputerSystem
attr  DisplayName
attr  IsManaged   IsManagedActive
attr  UseBridgingRelationship
```

3 Save and close the dxa-iptm-from-iprm.conf file.

The modified version of the file is saved to the *BASEDIR*/smarts/local/conf/icf-p directory.

Transferring the new attribute from the Network-Resource Manager to the IP-Analysis Manager

The dxa-ipam-from-iprm.conf file provides a listing of what classes, and attributes of those classes, will be transferred from the Network-Resource Manager to the IP-Analysis Manager.

To add the Router attribute to the dxa-ipam-from-iprm.conf file:

1 Edit the dxa-ipam-from-iprm.conf file by using **sm_edit**:

```
                #
        BASEDIR/smarts/bin/sm_edit conf/icf-a/dxa-ipam-from-iprm.conf
```

2 Type the following two lines for the Router class (shown in **bold**) before the entry for the UnitaryComputerSystem, as follows:

```
# DXA file for transferring topology from Network Resource Manager
# to IP Analysis Manager.
#
prehook    icf-a/ipam-iprm-dxa-prehook.asl
class Router
               attr  SM_SerialNumber
class UnitaryComputerSystem
peek  IsManaged   TRUE
attr  DisplayName
attr  IsManaged   IsManagedActive
attr  UseBridgingRelationship
```

3 Save and close the dxa-ipam-from-iprm.conf file.

The modified version of the file is saved to the *BASEDIR*/smarts/local/conf/icf-a directory.

Loading and viewing the new attribute in the Domain Manager s

Once the dynamic model has been compiled, you must load it into the appropriate Domain Managers. The Router class appears in the topology of the Network-Resource Manager, the IP-Perspective Manager, and the IP-Analysis Manager. The dynamic model is loaded into all three of these Domain Managers.

Loading the dynamic model into the Network-Resource Manager

In this procedure, TEST_SERVER_RM is the name of the test Network-Resource Manager. The Network-Resource Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note If the test Network-Resource Manager is already running, stop it first.

To load the dynamic model:

Note Two dash (--) characters precede the output and dynamic options of the **sm_server** command.

1 Load the compiled dynamic model file into the test Network-Resource Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_RM -c icf-r

                 --broker=localhost:426 --bootstrap=bootstrap.conf --output

                 --dynamics
```

2 Open the log file. If the new SM_SerialNumber attribute has been processed successfully, the output from the **sm_server** command appears similar to:

```
[September 11, 2009 4:27:08 PM EDT +263ms] t@1084463424 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading '/opt/InCharge8/IP/smarts/local/model/Example2.ldm'
[September 11, 2009 4:27:15 PM EDT +401ms] t@1084463424 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of SM_SerialNumber to "NotAvailable"
[September 11, 2009 4:27:15 PM EDT +401ms] t@1084463424 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for SM_SerialNumber = "NotAvailable"
[September 11, 2009 4:27:15 PM EDT +405ms] t@1084463424 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface Router refined from Example2.mdl:1
```

Verifying that the new attribute is present in the Network-Resource Manager

Before trying to populate the value of the new attribute, verify that the SM_SerialNumber attribute has been added to the Router class in the data model of TEST_SERVER_RM.

To verify the presence of the new attribute:

1 Open a Global Console. For this example, attach to TEST_SERVER_RM.

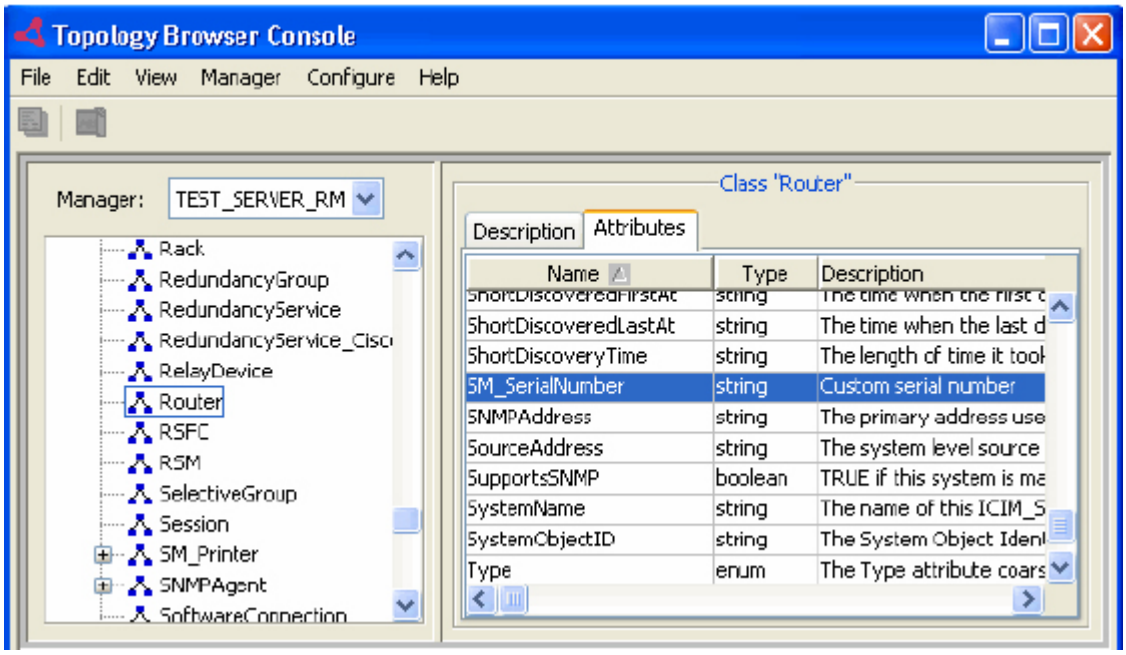
2 Right-click within the Topology Browser Console topology tree and select **Select** .

3 Select **Show classes with no instances in current domain**, and click **OK**.

4 Select the **Router** class in the topology tree to display the attributes in the **Attributes** tab.

The SM_SerialNumber attribute appears, as shown in Serial number attribute added to the Router class in TEST_SERVER_RM.

Figure 3-1. Serial number attribute added to the Router class in TEST_SERVER_RM



Loading the dynamic model into the IP-Perspective Manager

In this procedure, TEST_SERVER_P is the name of the test IP-Perspective Manager. The IP-Perspective Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note If the test IP-Perspective Manager is already running, stop it first.

To load the dynamic model:

Note Two dash (--) characters precede the output and dynamic options of the **sm_server** command.

1 Load the compiled dynamic model file into the test IP-Perspective Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_P -c icf-p

                 --broker=localhost:426 --bootstrap=bootstrap.conf --output

                 --dynamics
```

2 Open the log file. If the new SM_SerialNumber attribute has been processed successfully, the output from the **sm_server** command appears similar to:

```
[September 11, 2009 4:27:08 PM EDT +263ms] t@1084463424 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading '/opt/InCharge8/IP/smarts/local/model/Example2.ldm'
[September 11, 2009 4:27:15 PM EDT +401ms] t@1084463424 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of SM_SerialNumber to "NotAvailable"
[September 11, 2009 4:27:15 PM EDT +401ms] t@1084463424 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for SM_SerialNumber = "NotAvailable"
[September 11, 2009 4:27:15 PM EDT +405ms] t@1084463424 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface Router refined from Example2.mdl:1
```

Verifying that the new attribute is present in the IP-Perspective Manager

In the Topology Browser Console, attach to the TEST_SERVER_P Domain Manager. Verify that the SM_SerialNumber attribute has been added to the Router class in the data model of TEST_SERVER_P.

To verify the presence of the new attribute:

1 In the Topology Browser Console menu bar, select **Manager** > **Attach**.

2 In the **Attach Manager** dialog box, select **TEST_SERVER_P** in the **Manager** drop-down list. Type the appropriate username and password, and click **OK**.

3 In the Topology Browser Console, select **TEST_SERVER_P** from the **Manager** drop-down list.

4 Right-click within the Topology Browser Console topology tree and select **Select** . Select **Show classes with no instances in current domain**, and click **OK**.

5 In the Topology Browser Console, verify that the **SM_SerialNumber** attribute has been added to the Router class.

.

Loading the dynamic model into the IP-Analysis Manager

In this procedure, TEST_SERVER_A is the name of the test IP-Analysis Manager. The IP-Analysis Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note If the test IP-Analysis Manager is already running, stop it first.

To load the dynamic model:

Note Two dash (--) characters precede the output and dynamic options of **sm_server**.

1 Load the compiled dynamic model file into the test IP-Analysis Manager.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_A -c icf-a
```

```
                --broker=localhost:426 --bootstrap=bootstrap.conf --output

                --dynamics
```

2 Open the log file. If the new SM_SerialNumber attribute has been processed successfully, the output from the **sm_server** command appears similar to:

```
[September 11, 2009 4:27:08 PM EDT +263ms] t@1084463424 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading '/opt/InCharge8/IP/smarts/local/model/Example2.ldm'
[September 11, 2009 4:27:15 PM EDT +401ms] t@1084463424 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of SM_SerialNumber to "NotAvailable"
[September 11, 2009 4:27:15 PM EDT +401ms] t@1084463424 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for SM_SerialNumber = "NotAvailable"
[September 11, 2009 4:27:15 PM EDT +405ms] t@1084463424 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface Router refined from Example2.mdl:1
```

Verifying that the new attribute is present in the IP-Analysis Manager

In the Topology Browser Console, attach to the TEST_SERVER_A Domain Manager. Verify that the SM_SerialNumber attribute has been added to the Router class in the data model of TEST_SERVER_A.

To verify the presence of the new attribute:

1 In the Topology Browser Console menu bar, select **Manager** > **Attach**.

2 In the **Attach Manager** dialog box, select **TEST_SERVER_A** in the **Manager** drop-down list. Type the appropriate username and password. Click **OK**.

3 In the Topology Browser Console, select **TEST_SERVER_A** from the **Manager** drop-down list.

4 Right-click within the Topology Browser Console topology tree and select **Select** . Select **Show classes with no instances in current domain**, and click **OK**.

5 In the Topology Browser Console, verify that the SM_SerialNumber attribute has been added to the Router class.

Setting up topology sources

The Network-Resource Manager is added as a source of topology to both the IP-Perspective Manager and the IP-Analysis Manager. In addition, the IP-Perspective Manager is also added as a topology source to the IP-Analysis Manager.

Adding the Network-Resource Manager as a source to the IP-Perspective Manager

To add the Network-Resource Manager as a source of topology:

1 In the Topology Browser Console, select **Configure> Domain Manager Administrative Console**.

2 In the Domain Manager Administrative Console, select **TEST_SERVER_P** in the **Manager** drop-down list.

3 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

4 In the **Add Source** dialog box, type **TEST_SERVER_RM** in the **Element Name** field. Keep **NETWORK-RESOURCE-TYPE** as the **Element Type**. Click **OK**.

Adding the Network-Resource Manager as a source to the IP-Analysis Manager

To add the Network-Resource Manager as a source of topology:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_A** in the **Manager** drop-down list.

2 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

3 In the **Add Source** dialog box, type **TEST_SERVER_RM** in the **Element Name** field. Select **NETWORK-RESOURCE-TYPE** as the **Element Type**. Click **OK**.

Adding the IP-Perspective Manager as a source to the IP-Analysis Manager

To add the IP-Perspective Manager as a source of topology:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_A** in the **Manager** drop-down list.

2 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

3 In the **Add Source** dialog box, select **IP-PERSPECTIVE-TYPE** in the **Element Type** drop-down list. Type **TEST_SERVER_P** in the **Element Name** field. Click **OK**.

Discovering an instance of the Router class

1 In the Domain Manager Administrative Console, select **TEST_SERVER_RM** in the **Manager** drop-down list.

2 Initiate discovery.

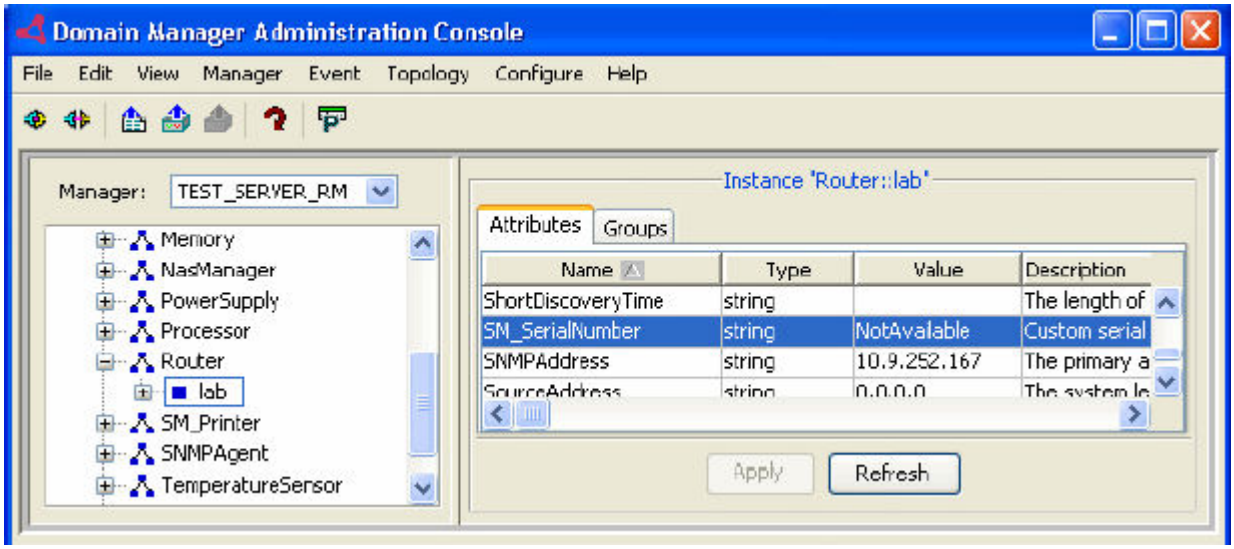
3 Select **Topology > Add Agent.**

4 Type either the name or IP address of the router.

The VMware Smart Assurance IP Management Suite Discovery Guide describes various methods of discovery.

5 Verify that the router has been added to the topology as an instance of the Router class. The initial value of the new SM_SerialNumber attribute will be “NotAvailable” as shown in Router discovered with the SM_SerialNumber attribute in TEST_SERVER_RM.

Figure 3-2. Router discovered with the SM_SerialNumber attribute in TEST_SERVER_RM



6 Select **TEST_SERVER_P** in the **Manager** drop-down list and verify that the router has been added to the topology.

7 Select **TEST_SERVER_A** in the **Manager** drop-down list and verify that the router has been added to the topology.

Populating the value of SM_SerialNumber

To populate the values of the new Router attribute, use any of the methods described in *Appendix C, “Methods of Creating Classes and Populating Attributes.”*

After populating the value, rediscover the Router instance. The rediscovery transfers the new value of the SM_SerialNumber property from the Network-Resource Manager to the IP-Perspective Manager and IP-Analysis Manager.

To rediscover the Router:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_RM** in the **Manager** drop-down list.

2 Right-click the Router instance, and then select **Rediscover**.

Forwarding the new attribute to the Global Manager

Note In this section,BASEDIR refers to the base directory for the Service Assurance Manager.

The BASEDIR/smarts/conf/ics/ics-default.xml file is the configuration file for the Global Manager. This file designates which dxa files are used to transfer events and topology from the Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager into the Global Manager. The ics-default.xml file will contain a separate section for each of the three Domain Managers. Each section indicates the dxa file that is used for that particular Domain Manager.

In this example, the models of the Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager have all been enhanced with the new Router attribute. However, only one Domain Manager should transfer that attribute to the Global Manager. In this example, only the Network-Resource Manager will transfer the new attribute to the Global Manager.

Updating the Network-Resource Manager dxa file

The ics-default.xml defines which topology and events to transfer from a Domain Manager to the Global Manager. In this example, the ics-default.xml file sets dxa-net-rm-conn-perf.conf as the dxa file that defines which topology and events to transfer from a Network-Resource-Suite to the Global Manager.

TEST_SERVER_RM is of the type Network-Resource-Suite. The new Router class attribute must be added to the dxa-net-rm-conn-perf.conf file.

```
<domaintype
     name="Network-Resource-Suite"
     dxaConfFile="dxa-net-rm-conn-perf.conf"
     MinimumCertainty="0.01"
     SmoothingInterval="65">
     <description>Type for Network Resource Connectivity and Performance Suite</description>
     <hook enable="False">ics/dxa-sample-hook.asl</hook>
     <domainname>Network-Resource</domainname>
 </domaintype>
```

To add the Router attribute to the dxa-net-rm-conn-perf.conf file:

1 Edit the dxa-net-rm-conn-perf.conf file by using **sm_edit**:

```
#

BASEDIR
/smarts/bin/sm_edit conf/ics/dxa-net-rm-conn-perf.conf
```

2 Type the following two lines for the Router class (shown in **bold**) before the entry for the ICIM_UnitaryComputerSystem, as follows:

```
# Remote Name of Class/Property        Mapped Name of Class/Property
   class   Router
                  attr    SM_SerialNumber
   class   ICIM_UnitaryComputerSystem          Node
   attr    Certification
   attr    DisplayName
   attr    Description
   attr    Location
```

```
attr    Model
attr    PrimaryOwnerContact
attr    Type
attr    Vendor
rel     ConnectedVia
```

3 Save and close the dxa-net-rm-conn-perf.conf file.

The modified version of the file is saved to the *BASEDIR*/smarts/local/conf/ics directory.

Compiling and loading the dynamic model into the Global Manager

Note This exercise should be performed on a test version of a Global Manager. Before starting this exercise, back up the repository on the test Global Manager.

The dynamic model enhancements must be loaded into the Global Manager. Otherwise, the Global Manager has no awareness of the new SM_SerialNumber attribute of the Router class.

Typically, the model that is loaded into a Global Manager contains only those attributes that are needed to identify the object in the topology. The values of these types of attributes, such as model number and serial number, are not polled by the underlying Domain Manager.

In addition, the Global Manager model does not contain any events or problems. If your dynamic model contains events or problems, remove the events after you copy the original .mdl file to the Service Assurance Manager's BASEDIR/smarts/local/modeldirectory. Then recompile the edited .mdl file (by using dynmodel) to create the .ldm file.

In this example, only an attribute is added to the Router class. You do not have to remove the events or problems before recompiling the file.

To compile and load the dynamic model into the Global Manager:

1 Copy Example2.mdlfrom the test IP Manager's *BASEDIR*/smarts/local/modeldirectoryto the test Service Assurance Manager's *BASEDIR*/smarts/local/modeldirectory.

2 From the Service Assurance Manager's *BASEDIR*/smarts/local/model directory, run the following commands to compile the Example2.mdl model file:

```
#

cd
BASEDIR
/smarts/local/model
#

BASEDIR
/smarts/bin/dynmodel Example2.mdl
```

3 Load the compiled dynamic model file into the test Global Manager.

```
t#

                BASEDIR
                /smarts/bin/sm_server -n TEST_SA -c ics

                --broker=localhost:426 --output --dynamics
```

This example registers TEST_SA with the Broker that is running on the host named “localhost” and port 426. Substitute the host and port of your Broker if they are different.

Transferring the value of the new attribute to the Global Manager

The new attribute for the Router class has been loaded into the model of the Global Manager. To transfer the value of the new attribute, the Network-Resource Manager, IP-Perspective Manager, and the IP-Analysis Manager must be added as sources to the Global Manager.

Adding the Network-Resource Manager as a source to the Global Manager

To add the Network-Resource manager as a source:

1 In the Topology Browser Console menu bar, select **Manager** > **Attach**

2 In the **Attach Manager** dialog box, select **TEST_SA** in the **Manager** drop-down list. Type the appropriate username and password, and click **OK**.

3 In the Topology Browser Console, select **Configure> Global Manager Administrative Console**.

4 In the Global Manager Administrative Console, select **TEST_SA** in the **Manager** drop-down list.

5 In the Global Manager Administrative Console, select **Edit** > **New Domain**.

6 In the **Domain Creation Wizard** dialog box, type **TEST_SERVER_RM** in the **Domain Name** field. Select **Network-Resource** in the **Copy Existing** drop-down list. Click **Finish**.

Adding the IP-Perspective Manager as a source to the Global Manager

To add the IP-Perspective manager as a source:

1 In the Global Manager Administrative Console, select **Edit** > **New Domain**.

2 In the **Domain Creation Wizard** dialog box, type **TEST_SERVER_P** in the **Domain Name** field. Select **IP-Perspective** in the **Copy Existing** drop-down list. Click **Finish**.

Adding the IP-Analysis Manager as a source to the Global Manager

To add the IP-Analysis manager as a source:

1 In the Global Manager Administrative Console, select **Edit** > **New Domain**.

2 In the **Domain Creation Wizard** dialog box, type **TEST_SERVER_A** in the **Domain Name** field. Select **IP-Analysis** in the **Copy Existing** drop-down list. Click **Finish**.

Reconfiguring the Global Manager

To reconfigure the Global Manager:

1 In the Global Manager Administrative Console, select **Configure** > **Reconfigure Server**.

2 Click **Yes** to confirm the reconfiguration.

3 Click **Close** at the prompt.

Verifying the presence of SM_SerialNumber in the Global Manager

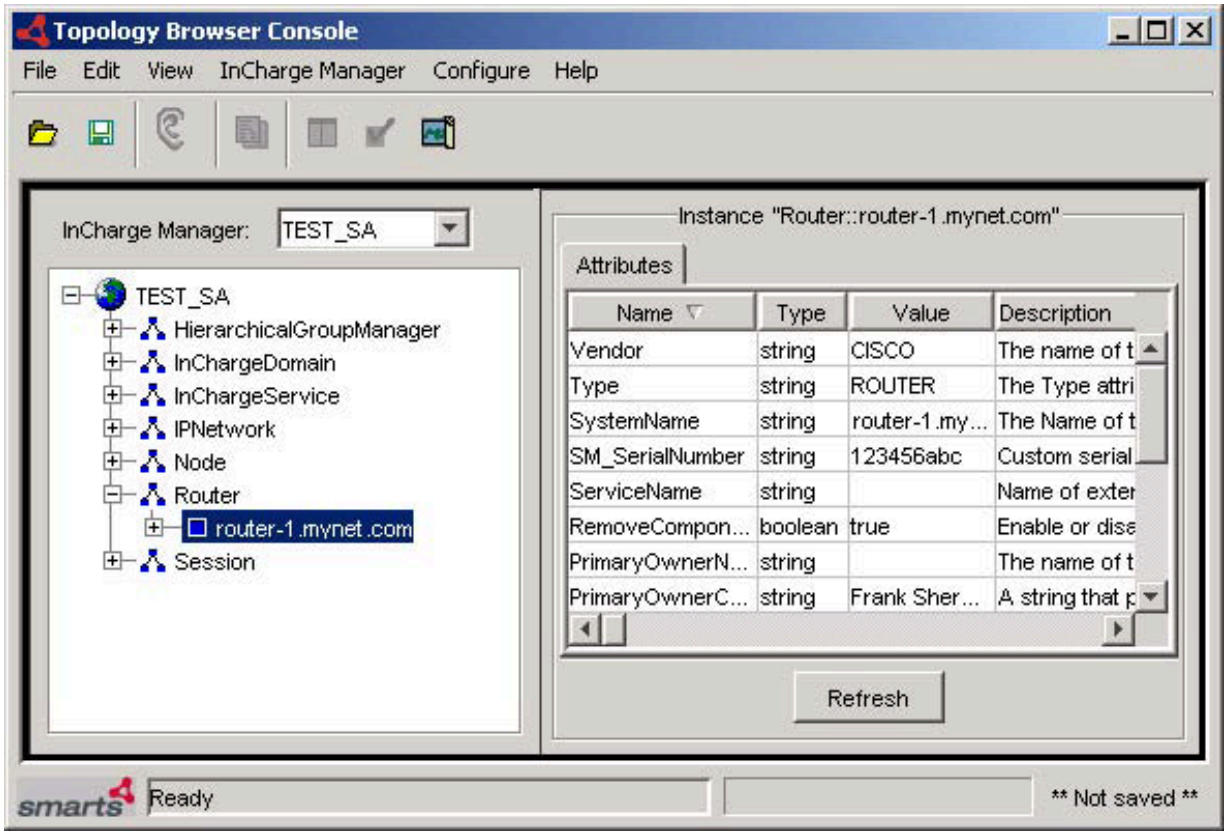
To verify the presence of SM_SerialNumber:

1 In the Global Console, attach to the TEST_SA Global Manager.

2 In the Topology Browser Console, verify that the SM_SerialNumber attribute has been added to the Router class as shown in SM_SerialNumber in the TEST_SA Global Manager.

Note If you did not populate the value of the attribute (refer to Populating the value of SM_SerialNumber), then the value of SM_SerialNumber will still be “NotAvailable.”

Figure 3-3. SM_SerialNumber in the TEST_SA Global Manager



Exercise 3 - Polling for Values of Additional Attributes

4

This chapter includes the following topics:

- Objectives
- Exercise

Objectives

This exercise is a complex example of using Dynamic Model. This example enhances the following Domain Managers:

- Network-Resource Manager
- IP-Perspective Manager
- IP-Analysis Manager
- Global Manager

The purpose of the exercise includes:

- Polling for a new attribute.
- Generating an event based on the value of the attribute.
- Forwarding the new event to a Global Manager so that it can be displayed as a notification.

Note This exercise should be performed on test versions of a Broker, Network-Resource Manager, IP-Perspective Manager, IP-Analysis Manager, and Global Manager. Before starting this exercise, back up the repository on the test Managers.

This exercise includes the following tasks:

- Determine the instrumentation class that will be refined to poll for the new attribute.
- Create and compile a dynamic model file:
  - Define an additional MIB variable that can be polled by the Network-Resource Manager.
  - Define a setting class for the new attribute.
  - Refine an instrumentation class that will define the MIB OID to poll.

- Subscribe to the event in the Network-Resource Manager.
- Add a new threshold setting to adapter-setting.asl.
- Add the polled event to the Global Manager.
- Load the dynamic model into the Network-Resource Manager, IP-Perspective Manager, and the IP-Analysis Manager.
- Set up topology sources.
- Discover an instance of the Router class.
- Verify variable in Polling and Threshold dialog box.
- Poll the values of the new attributes in real-time.
- Generate an event based on the values of this polled attribute.
- Have the Global Manager subscribe to and display notifications associated with the event based on the value of this polled attribute.

Exercise

In this exercise, the Network-Resource Manager polls for a new attribute and generates an event. The Global Manager subscribes to the event, and when the event is active the event notification is displayed in the Notification Log Console.

The new TS_HighOverrunRate event is triggered if the rate of increase of the locIfInOverrun attribute of the interface, expressed as a percentage of the inputPacketRate, is over a user defined threshold.

You do not need to copy and load the dynamic model enhancements (the file with extension .ldm) into the Global Manager. This exercise polls for a value and creates an event based on the polled value. However, the Global Manager model does not contain events. Instead, it contains only those attributes that are required to identify the device in topology.

Note Use a unique prefix with new class names and other properties. The name must differ from existing ICIM classes and properties. The prefix “TS_” is used in this example. Do not use "SM_" as this is already used by SDK.

Unless otherwise noted, BASEDIR refers to the base directory for the IP Manager.

Determining the instrumentation class

In this exercise, the Network-Resource Manager polls an additional attribute that measures the performance of the Ethernet connection with a Router.

The Network-Resource Manager uses instances of designated instrumentation classes to hold the polled attributes, in order to abstract the instrumentation details between device models. These instances are in relationship with the object representing the device, which is where the event logic is defined. The event logic may depend on comparison with thresholds, which are stored in setting objects.

The instrumentation class that you need to refine can be determined from the oid2type_*.conf files. In this example, the OID for a Router appears in the oid2type_Cisco.conf file.

The TS_locIfInOverrun variable is instrumenting the performance of an Ethernet connection. To determine the instrumentation class, locate **Interface-Ethernet-Performance** under **INSTRUMENTATION**.

```
.1.3.6.1.4.1.9.1.469 {
    TYPE = Router
    VENDOR = Cisco
    MODEL = 2621XM
    CERTIFICATION = CERTIFIED
    CONT = Cisco-Router
    NEIGHBOR = Cisco-Cdp
    HEALTH = Cisco-Router
INSTRUMENTATION:
    Card-Fault                          = OldCiscoChassis
    Environment                         = CiscoEnvMon:DeviceID
    CPU/Memory                          = CiscoRouter:DeviceID
    Interface-Fault                     = MIB2
    Interface-Performance               = CiscoRouter
    Interface-Ethernet-Performance      = CiscoRouter_Ethernet
}
```

The name of the instrumentation class to refine is obtained by concatenating “Interface_Performance_” with the string associated with the Interface-Ethernet-Performance instrumentation type. The string CiscoRouter_Ethernet is associated with the Interface-Ethernet-Performance instrumentation type. Therefore, the name of the instrumentation class is Interface_Performance_CiscoRouter_Ethernet.

Creating and compiling the dynamic model

In this exercise, you create and compile a new dynamic model.

Creating the dynamic model

In this example, the same dynamic model is loaded into several Domain Managers, each starting with a different **--**conf directory. Instead of copying the same compiled dynamic model file (.ldm) to each BASEDIR/smarts/local/model/<*conf*> directory, one single copy is created and compiled in BASEDIR/smarts/local/model. Each of the Domain Managers is then started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

To create the dynamic model:

1 Create and edit a new file in the *BASEDIR*/smarts/local/model directory. Type the following at a command prompt:

```
#

BASEDIR
/smarts/bin/sm_edit --create model/Example3.mdl
```

2 Refine the setting class that sets the default value for the alarm threshold. Type the following lines into the Example3.mdl file:

```
refine interface NetworkAdapter_Performance_Setting
{
        attribute float [0 .. 100] TS_OverrunThreshold
        "Acceptable upper threshold for Overruned packets, "
        "expressed as a percentage of the total number of input packets."
        = 10;
}
```

3 Refine the Interface_Performance_CiscoRouter_Ethernet instrumentation class that will be used to poll for the attribute. Type the following lines into the Example3.mdl file:

```
refine interface Interface_Performance_CiscoRouter_Ethernet
{
attribute float TS_OverrunThreshold
"The upper threshold for Overrun packet errors expressed as a percentage"
"of the total number of output packets."
= 10;
instrument SNMP {
TS_locIfInOverrun = "1.3.6.1.4.1.9.2.2.1.1.14"
};
readonly instrumented attribute int TS_locIfInOverrun
"The number of overrun errors.";
readonly computed attribute float TS_locIfInOverrunRate
= rate(TS_locIfInOverrun, PollingInterval);
computed attribute float TS_OverrunErrorRate
"The rate of ignored Overrun packets expressed in packets"
"per second."
= TS_locIfInOverrunRate;
#pragma Uses Propagation
readonly computed attribute float TS_OverrunErrorPct
"The rate of Overruned input packets expressed as a percentage"
"of total input packets."
= (TS_OverrunErrorRate / (InputPacketRate + 0.001)) * 100;
event TS_HighOverrunRate
"Indicates that the percentage of packets with Overrun errors exceeds
OverrunErrorThreshold"
= if NetworkAdapterOperational && (TS_OverrunThreshold < 100) && (MaxSpeed > 0)
check (TS_OverrunErrorPct > TS_OverrunThreshold);
export TS_HighOverrunRate;
}
TS_OverrunErrorPct > TS_OverrunThreshold
if NetworkAdapterOperational and (TS_OverrunThreshold < 100) and (MaxSpeed > 0)
```

4 Save the Example3.mdl file.

The file is saved in the *BASEDIR*/smarts/local/model directory.

Compiling the dynamic model

To compile the Example3.mdl model file, type the following commands:

```
#

cd
BASEDIR
/smarts/local/model
#

BASEDIR
/smarts/bin/dynmodel Example3.mdl
```

If it is successful, the **dynmodel** command does not produce any output to the command-line console.

In addition, the following files are created in the BASEDIR/smarts/local/model directory:

- Example3.ldm
- Example3_mmbe.dat
- Example3_mmle.dat
- Example3_mm.txt

  The following directory is also created in the BASEDIR/smarts/local/model directory:

- Example3_mm

  Compilation errors on page 111 provides additional information if you encounter errors.

Copying files for localization

From the BASEDIR/smarts/local/model directory, copy the following files to the SM_SITEMOD/l10n directory:

- Example3_mmbe.dat
- Example3_mmle.dat

Automatically subscribing to the event in a Domain Manager

One of the strengths of SDK technology is that expressions defined in an event are not evaluated and polled unless there is a subscription to that event. In order for the Network-Resource Manager to automatically begin polling and monitoring for the new event, add a subscription the Network-Resource Manager bootstrap file.

1 Edit the bootstrap file by using **sm_edit**:

```
#

BASEDIR
/smarts/bin/sm_edit conf/icf-r/bootstrap.conf
```

2 Type the following line into the bootstrap file in the **GA_SubscriberProfile::PROFILE-default** section:

```
{"Interface_Performace_CiscoRouter_Ethernet",".*","TS_HighOverrunRate",77879 }
{ <class>, <instance>, <event>, <subscription-level> },
```

3 Save and close the bootstrap file.

The file will be saved in the *BASEDIR*/smarts/local/conf/icf-rdirectory.

Adding the new threshold setting to the adapter-setting.asl file

The new threshold setting attribute must be added to the BASEDIR/smarts/rules/health/adapter-setting.asl file.

To add the threshold setting:

1 Edit the BASEDIR/smarts/rules/health/adapter-setting.asl file by using **sm_edit**:

```
#

BASEDIR
/smarts/bin/sm_edit rules/health/adapter-setting.asl
```

2 Look for the **APPLY_SETTING()** section in the adapter-setting.asl file. Type the following line, at the following location, in the adapter-setting.asl file.

```
if (!setObj->isNull()) {
 REMOVE_PERF_INSTRUMENTATION(obj);
 continue;
}
threshold_map += TS_OverrunThreshold;
if (obj->CreationClassName == "Port") {
```

3 Save and close the adapter-setting.asl file.

The modified version of the file is saved to the *BASEDIR*/smarts/local/rules/health/ directory.

Adding the polled event to the Global Manager

Note In this section,BASEDIR refers to the directory for the Service Assurance Manager .

The BASEDIR/smarts/conf/ics/ics-default.xml file is the configuration file for the Global Manager. This file designates which dxa files are used to transfer events and topology into the Global Manager from the Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager. The ics-default.xml file contains a separate section for each of the three Domain Managers. Each section indicates the dxa file that is used for that particular Domain Manager.

In this example, the event is generated in the Network-Resource Manager. The event is not used in any analysis in the IP-Analysis Manager. Therefore, the event subsciption needs to be added to only the dxa file that defines which topology and events are transferred from the Network-Resource Manager to the Global Manager.

The ics-default.xml file sets dxa-net-rm-conn-perf.conf as the dxa file that defines which topology and events to transfer from a Network-Resource-Suite to the Global Manager. This is shown in the following section of the ics-default.xml file.

```
<domaintype
     name="Network-Resource-Suite"
IP-Analysis Manager     dxaConfFile="dxa-net-rm-conn-perf.conf"
     MinimumCertainty="0.01"
     SmoothingInterval="65">
     <description>Type for Network Resource Connectivity and Performance Suite</description>
     <hook enable="False">ics/dxa-sample-hook.asl</hook>
     <domainname>Network-Resource</domainname>
 </domaintype>
```

TEST_SERVER_RM is of the type Network-Resource-Suite. Therefore, the subscription to the new event must be added to the dxa-net-rm-conn-perf.conf file.

Event and notification subscriptions in the Global Manager are configured within dxa configuration files by using the **sub** keyword. Patterns (regular expressions) separated by "::" match the class name, instance name, and event name in the Domain Manager.

The syntax is as follows:

sub <*classPattern*>::<*instancePattern*>::<*eventPattern*>/<*modifier*s>

where:

- <*classPattern*> is the pattern to match for the class name.
- <*instancePattern*> is the pattern to match for the name of the instance.
- <*eventPattern*> is the pattern to match for the name of the problem or event.

- *<modifiers>* can have the following values:
  - p = problem
  - e = event
  - a = aggregates
  - n = notifications

  To add the event:

  1. Edit the dxa-net-rm-conn-perf.conf file by using **sm_edit**:

     ```
     #

     BASEDIR
     /smarts/bin/sm_edit conf/ics/dxa-net-rm-conn-perf.conf
     ```

  2. Type the following lines in the subscription section of the file.

     ```
     # Added for Dynamic Model
      sub
     Interface_Performance_CiscoRouter_Ethernet::.*::TS_HighOverrunRate/e
     ```

  3. (Optional) Set the severity and category of the notification that is created for this event. Setting the notification characteristics of the event provides more information.
  4. Save and close the dxa-net-rm-conn-perf.conf file.

     The modified version of the file is saved to the *BASEDIR*/smarts/local/conf/ics directory.

Loading the dynamic model into the Domain Manager s

Once the dynamic model has been compiled, you must load it into the Domain Managers.

Note In this section,BASEDIR is the directory for the IP Manager, unless otherwise noted.

Loading the dynamic model into the Network-Resource Manager

In this procedure, TEST_SERVER_RM is the name of the test Network-Resource Manager. The Network-Resource Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note Two dash (--) characters precede the dynamic option of the **sm_server** command.

To load the dynamic model:

1 Type the following command:

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER_RM -c icf-r

                 --broker=localhost:426 --bootstrap=bootstrap.conf --output

                 --dynamics
```

2 Open the log file. If the dynamic model has been processed successfully, the CREATED FILES section of the output from the **sm_server** command appears similar to the following:

```
[July 10, 2009 12:25:29 PM EDT +789ms] t@1120213312 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading
    '/opt/InCharge8/IP/smarts/local/model/Example3.ldm'
[July 10, 2009 12:25:44 PM EDT +141ms] t@1120213312 LDM file processing #1
MRM-*-MRM_PROP_INIT-Property TS_OverrunThreshold initial value ='10'
[July 10, 2009 12:25:44 PM EDT +142ms] t@1120213312 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of TS_OverrunThreshold to 10
[July 10, 2009 12:25:44 PM EDT +143ms] t@1120213312 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for TS_OverrunThreshold = 10
[July 10, 2009 12:25:44 PM EDT +144ms] t@1120213312 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface NetworkAdapter_Performance_Setting
    refined from Example3.mdl:1
[July 10, 2009 12:25:44 PM EDT +145ms] t@1120213312 LDM file processing #1
MRM-*-MRM_PROP_INIT-Property TS_OverrunThreshold initial value ='10'
[July 10, 2009 12:25:44 PM EDT +146ms] t@1120213312 LDM file processing #1
MRM-*-MRM_SET_PROP-Setting initial property of TS_OverrunThreshold to 10
[July 10, 2009 12:25:44 PM EDT +146ms] t@1120213312 LDM file processing #1
MRM-*-MRM_ACT_PROP-Actual initial value for TS_OverrunThreshold = 10
[July 10, 2009 12:25:44 PM EDT +147ms] t@1120213312 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface Interface_Performance_CiscoRouter
    refined from Example3.mdl:8
```

Loading the dynamic model into the IP-Perspective Manager

In this procedure, TEST_SERVER_P is the name of the test IP-Perspective Manager. The IP-Perspective Manager is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note Two dash (--) characters precede the dynamic option of the **sm_server** command.

To load the dynamic model, type the following command:

```
t #

         BASEDIR
         /smarts/bin/sm_server -n TEST_SERVER_P -c icf-p
```

```
--broker=localhost:426 --bootstrap=bootstrap.conf --output

--dynamics
```

Loading the dynamic model into the IP-Analysis Manager

In this procedure, TEST_SERVER_A is the name of the test IP-Analysis Manager. The IP-Analysis Managers is started with the **--**dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

Note Two dash (--) characters precede the dynamic option of the **sm_server** command.

To load the dynamic model, type the following command:

```
t #

        BASEDIR
        /smarts/bin/sm_server -n TEST_SERVER_A -c icf-a

         --broker=localhost:426 --bootstrap=bootstrap.conf --output

         --dynamics
```

Setting up topology sources

The Network-Resource Manager is added as a source of topology to both the IP-Perspective Manager and the IP-Analysis Manager. In addition, the IP-Perspective Manager is also added as a topology source to the IP-Analysis Manager.

Opening a Global Console and attaching to the Domain Manager s

To open a Global Console and attach to the Domain Managers:

1 Open a Global Console. For this example, attach to TEST_SERVER_RM.

2 In the Topology Browser Console menu bar, select **Manager** > **Attach**.

3 In the **Attach Manager** dialog box, select **TEST_SERVER_P** in the **Manager** drop-down list. Type the appropriate username and password, and click **OK**.

4 In the Topology Browser Console menu bar, select **Manager** > **Attach**.

5 In the **Attach Manager** dialog box, select **TEST_SERVER_A** in the **Manager** drop-down list. Type the appropriate username and password, and click **OK**.

Adding the Network-Resource Manager as a source to the IP-Perspective Manager

To add the Network-Resource Manager as a source of topology:

1 In the Topology Browser Console, select **Configure> Domain Manager Administrative Console**.

2 In the Domain Manager Administrative Console, select **TEST_SERVER_P** in the **Manager** drop-down list.

3 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

4 In the **Add Source** dialog box, type **TEST_SERVER_RM** in the **Element Name** field. Keep **NETWORK-RESOURCE-TYPE** as the **Element Type**. Click **OK**.

Adding the Network-Resource Manager as a source to the IP-Analysis Manager

To add the Network-Resource Manager as a source of topology:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_A** in the **Manager** drop-down list.

2 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

3 In the **Add Source** dialog box, type **TEST_SERVER_RM** in the **Element Name** field. Select **NETWORK-RESOURCE-TYPE** as the **Element Type**. Click **OK**.

Adding the IP-Perspective Manager as a source to the IP-Analysis Manager

To add the IP-Perspective Manager as a source of topology:

1 In the Domain Manager Administrative Console, select **TEST_SERVER_A** in the **Manager** drop-down list.

2 In the Domain Manager Administrative Console, select **Topology** > **Add Source**.

3 In the **Add Source** dialog box, select **IP-PERSPECTIVE-TYPE** in the **Element Type** drop-down list. Type **TEST_SERVER_P** in the **Element Name** field. Click **OK**.

Discovering an instance of the Router class

To discover an instance of the Router class:

1 Initiate discovery.

2 Select **Topology > Add Agent.**

3 Type either the name or IP address of the router.

The VMware Smart Assurance IP Management Suite Discovery Guide describes various methods of discovery.

4 Verify that the router and interface have been added to the topology.

5 Click within the Topology Browser Console topology tree .

6 Type **Ctrl-Alt-m**.

7 Right-click within the Topology Browser Console topology tree and select **Refresh tree** .

8 Verify that the following attributes have been added to the Interface_Performance_CiscoRouter_Ethernet instance:

9 TS_locIfInOverrun

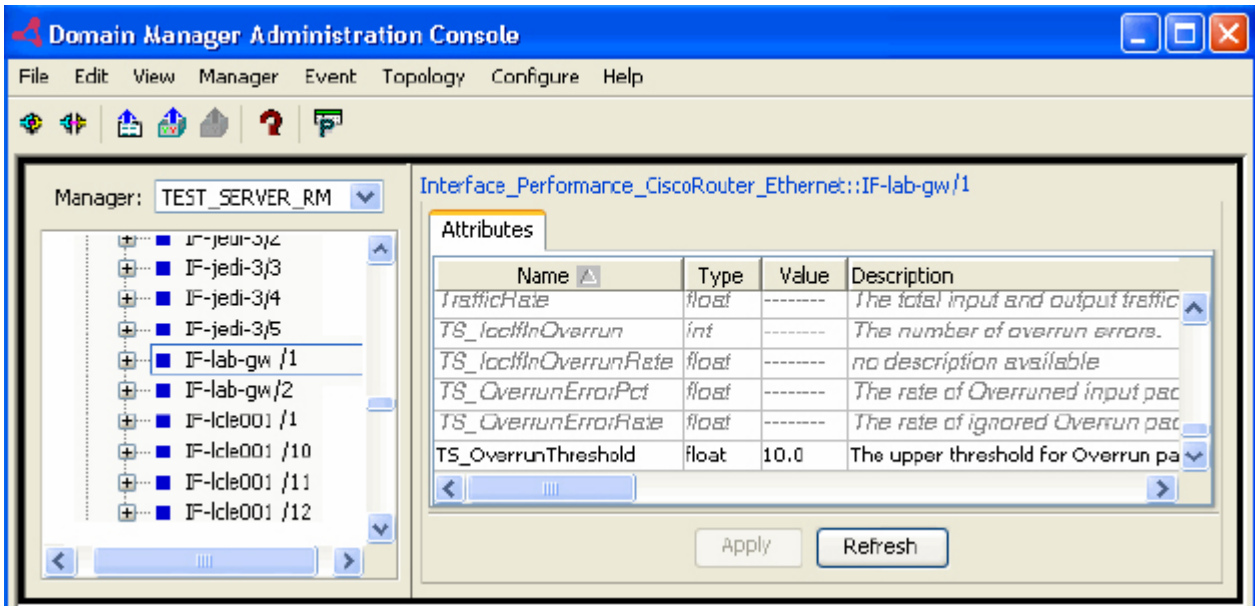
10 TS_locIfInOverrunRate

11 TS_OverrunErrorRate

12 TS_OverrunErrorPct

Use the **Attributes** tab as shown in Attributes tab with the new attributes before polling.

Figure 4-1. Attributes tab with the new attributes before polling



Verifying the variable in the Polling and Settings Threshold dialog box

In this example, the TS_OverrunThreshold attribute was added by

refining the NetworkAdapter_Performance_Setting class. The NetworkAdapter_Performance_Setting class defines parameters that are used

for monitoring performance characteristics of a network adapter. These include basic parameters common to all network adapters, such as utilization, errors, broadcast, and packet drops.

To verify the TS_OverrunThreshold attribute:

1 In the Domain Manager Administration Console, select **Configure > Polling and Thresholds**. The **Polling and Thresholds** console opens.

2 In the left panel, select the **Thresholds** tab.

3 In the left panel, select **Interface Groups.**

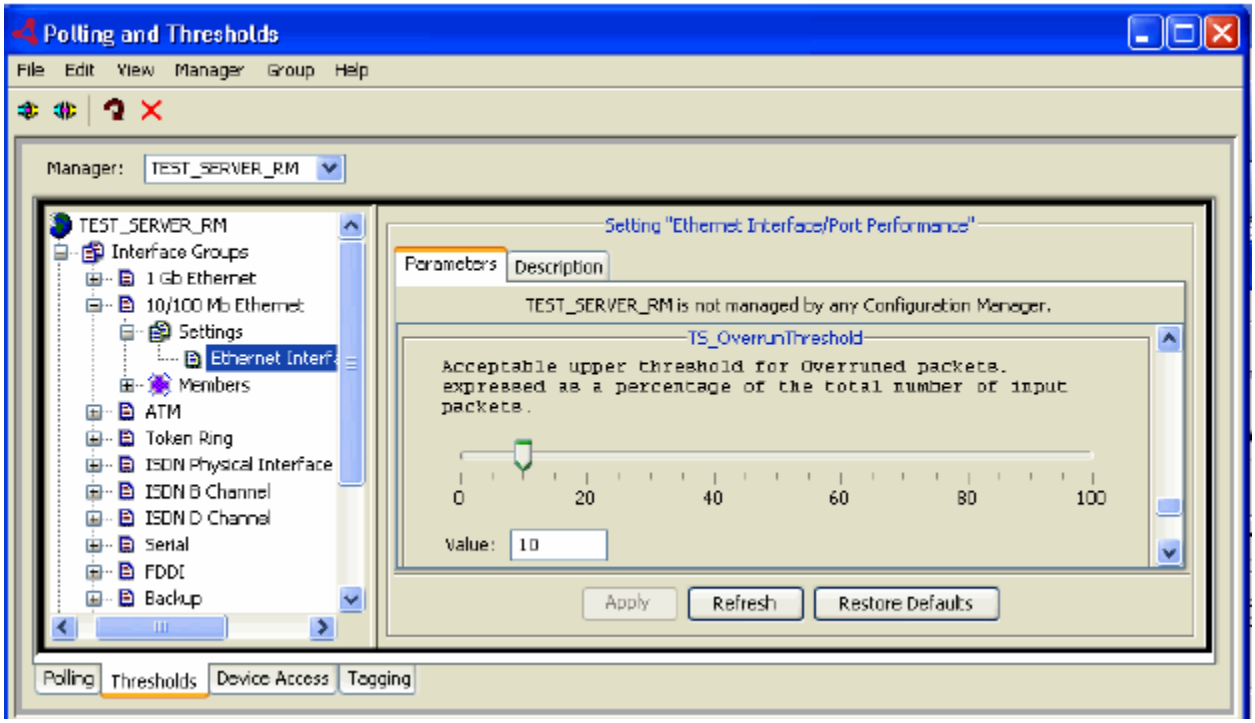
4 Find the interface group to which your router interfaces are members.

5 In the left panel, under the appropriate interface group, select **Settings > Ethernet Interface/Port Performance**.

Note If your router interfaces are members of **Other Interfaces**, then select **Settings > Generic Interface/Port Performance**.

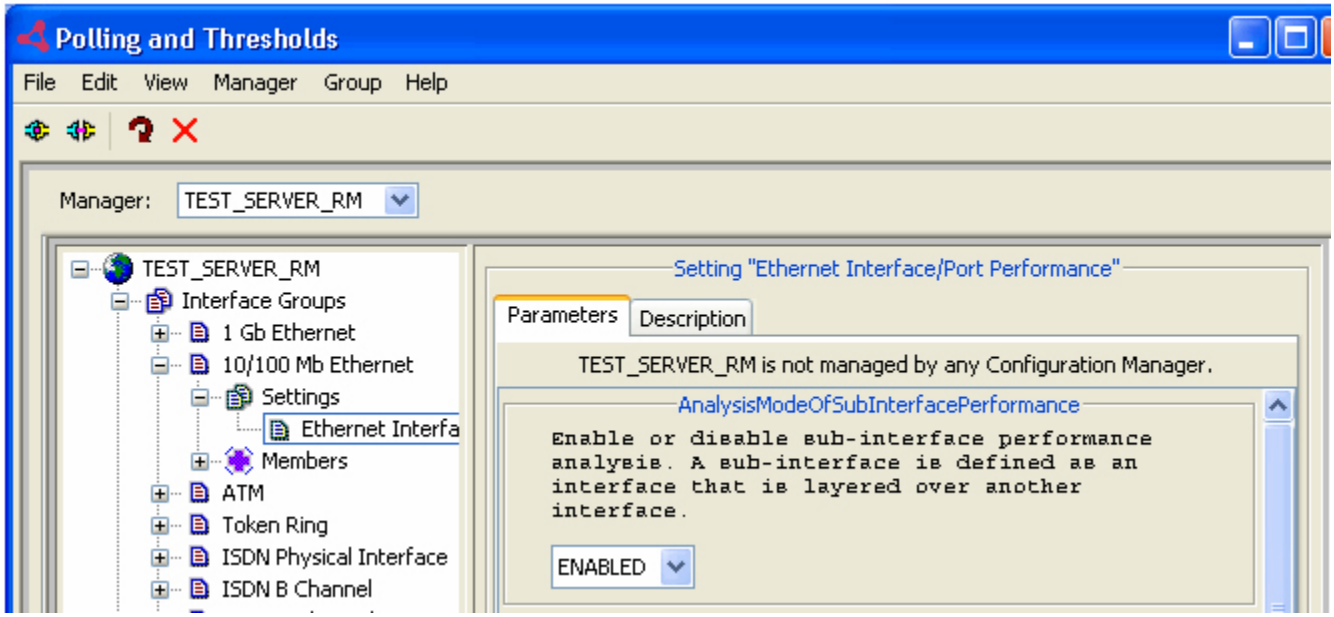
6 Verify that the value of the TS_OverrunThreshold attribute can be set as shown in TS_OverrunThreshold attribute in the Thresholds tab.

Figure 4-2. TS_OverrunThreshold attribute in the Thresholds tab



7 In this example, subinterface performance analysis must be enabled. Scroll to the top of the **Parameters** tab. Enable subinterface performance analysis if disabled as shown in Enabled subinterface performance analysis.

Figure 4-3. Enabled subinterface performance analysis



Polling the new attributes

The expressions defined in an event are not evaluated and polled unless there is a subscription to that event. The previous edit of the dxa-net-rm-conn-perf.conf file added a subscription to the TS_HighOverrunRate event from the Global Manager. The Global Manager must be started for the subscription to take effect and for polling of the new variables to commence.

Note In this section, BASEDIR refers to the directory for the Service Assurance Manager.

To start polling the new attributes:

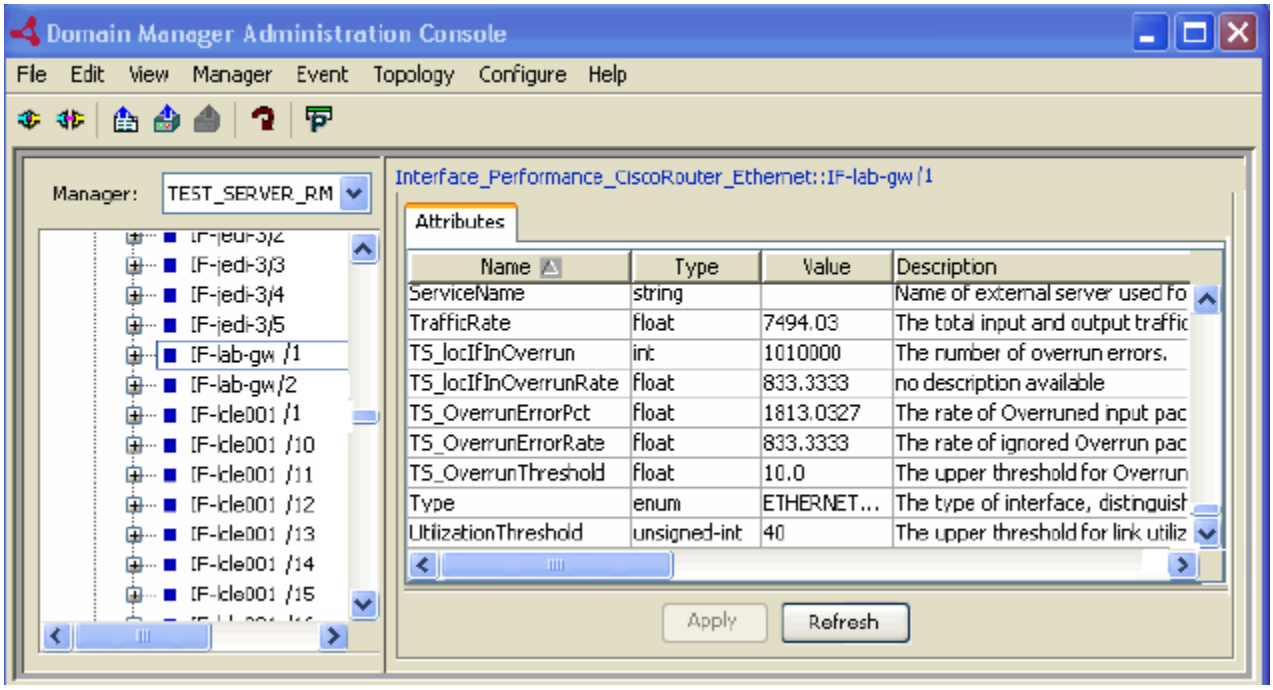
1 Type the following command to start the Global Manager:

```
t
                    #


                BASEDIR
                /smarts/bin/sm_server -n TEST_SA -c ics --broker=localhost:426 --outputs
```

2 In the Global Console, attach to the TEST_SA Global Manager.

Once polling has started, the values of TS_locIfOverrun, TS_locIfInOverrunRate, TS_OverrunErrorRate, and TS_OverrunErrorPct update and will appear similar to Attributes tab with the new attributes after polling.

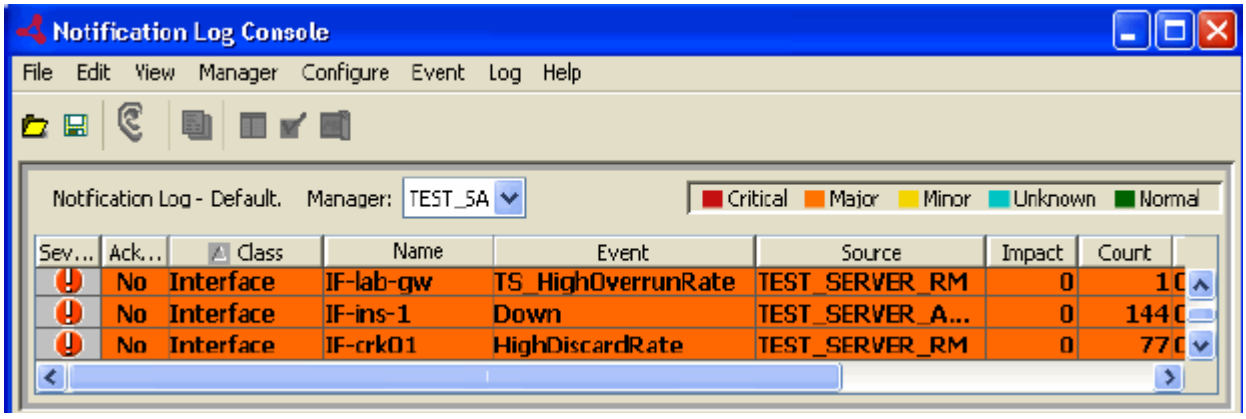
Figure 4-4. Attributes tab with the new attributes after polling



TS_HighOverrunRate event in the Global Manager

When the TS_HighOverrunRate event is triggered by the TS_OverrunErrorPct value, the TS_HighOverrunRate event is transferred to the Global Manager and displayed in the Global Console, as shown in TS_HighOverrunRate event in the Global Manager.

Figure 4-5. TS_HighOverrunRate event in the Global Manager



Guidelines and Syntax for Dynamic Model Files

5

This chapter includes the following topics:

- Guidelines and requirements
- Adding a class to the model
- Updating an existing class in the model

Guidelines and requirements

A dynamic model file describes the model elements that extend the basic VMware Smart Assurance data model. The following guidelines apply to dynamic model files:

- Typically, those classes that are related to each other are contained in the same dynamic model file.
- Dynamic model filenames may contain ASCII letters (A..Z,a..z), ASCII numbers (0...9), and the underscore character (_), but may contain no other characters. The first character must be a letter.
- The dynamic model filename must be unique within the first 23 characters.
- The dynamic model file must have a .mdl filename extension.
- Create and edit the dynamic model files (those with extension .mdl) by using **sm_edit**.
- Dynamic model files must follow the syntax rules described in Adding a class to the model and Updating an existing class in the model.
- All refinements to a class *must* be contained in a single
- **refine interface** <*classname*> declaration. For example, you cannot have a
- **refine interface Router** declaration in the mymodel1.mdl file, and then another **refine interface Router** in the mymodel2.mdl. Likewise, you cannot have more than one **refine interface Router** declaration in the mymodel1.mdl file.
- Precede any new attributes, classes, aggregates or events that are defined with a unique tag, such as “USR_”. You should not use "SM_" as this is already used by SDK.
- Avoid using the binary or byte data type, or the **external** keyword in any dynamic model files of the IP Manager that you will be copying over to the Service Assurance Manager.

- Dynamic model files must be located in either BASEDIR/smarts/modelorBASEDIR/smarts/model/<*config*>. The <*config*> refers to the configuration directory that is specified on the command line when the Domain Manager is started.
- The dynamic model files (those with extension .ldm) must be loaded into each Manager that requires knowledge of the new topology and events. The loading of the dynamic model files incorporates them into the runtime data model.
- When the option **--**dynamic is not added to the **sm_server** command, the Manager will load **all** dynamic model files that can be loaded into a Manager (those with extension .ldm) located in:

  BASEDIR/smarts/model/<*config*>

  The <*config*> refers to the configuration directory that is specified on the **sm_server** command line when the Domain Manager is started.

  Chapter 6 The dynmodel Command Syntax and Options provides information on creating dynamic model files that can be loaded into a Manager.
- When the **--**dynamic option is specified on the **sm_server** command line, the Manager will load **all** dynamic model files that can be loaded into a Manager (those with extension .ldm) located in:
  - BASEDIR/smarts/model
  - BASEDIR/smarts/model/<*config*>.

    The <*config*> refers to the configuration directory that is specified on the **sm_server** command line when the Domain Manager is started.

    Note Two dash (--) characters precede the dynamic option of **sm_server**.

    If a Manager is started by using **sm_service**, edit the **sm_service** command to include the **--**dynamic option.

Adding a class to the model

First, the class to be added needs to be declared, as described in Adding the new class. Next, it is compiled into a form that can be loaded into a Manager. Once the model that contains the new class is loaded into the Manager, any operations and facilities available for common VMware Smart Assurance classes in the Manager are now available for the new class. However, additional steps are needed to actually discover and instantiate instances of the class.

Determining the parent of the new class

A new class is declared as a subclass of an existing ICIM class. The key is to determine which class should be the parent class. This determination is based on several factors:

- How you want to search and report on the new class
- Functional similarities to existing elements in topology

- The events, attributes, and relationships the new class needs to inherit

  For example, suppose a class that represents a printer needs to be added to the model. A printer is a physical element, similar to a router, switch, or host. Therefore, the printer class can be defined as a subclass of UnitaryComputerSystem, which is the parent class of routers, switches, and hosts. The printer class inherits all of the attributes, relationships, and events of the UnitaryComputerClass class.

  Note The IP Manager counts the instances of the ICIM_UnitaryComputerSystem. If this is a problem for you, you can inherit from either ICIM_System or ICIM_ComputerSystem. However, if you inherit from either ICIM_System or ICIM_ComputerSystem you will have to add your new class into the appropriate dxa to transfer the topology to the Global Manager. Chapter 3 Exercise 2 - Adding an Attribute to the Router Class provides information about adding a class and its associated attributes to dxa files.

Adding the new class

To add a new class to a model, use the **interface** keyword. The **interface** keyword indicates that any declarations that follow define a new class.

The syntax is as follows:

```
interface <class_name>:<parent_class>
{
<declarations>
}
```

In this example the printer class inherits from the UnitaryComputerSystem class, and is named SM_Printer.

```
interface SM_Printer:UnitaryComputerSystem
"Create a printer class inheriting from UCS"
{

 stored attribute string SM_PrtrLocation
 "Location of Printer"="";

}
```

The new class may include a description.

The VMware Smart Assurance MODEL Reference Guide describes the <*declarations*> syntax. In dynamic modeling, declarations include attributes, relationships, operations, problems, symptoms, aggregates, and events.

Updating an existing class in the model

You may also make changes to the ICIM classes that already exist in the model installed on a Domain Manager. An existing class can be *extended* by adding any of the following:

- Attribute

- Relationship
- Event
- Aggregate
- Problem
- Symptom
- Operation

However, when a Manager, such as a Network-Resource Manager, is started, its model is loaded from a library file. Dynamic modeling allows you to *extend* one of the classes in the library file by adding new properties, such as attributes and events. However, you cannot delete any class properties in the delivered library file.

In this example, several properties are declared in a class in the library file:

```
interface Host : ICIM_System
 "Host is a computer system that represents a workstation or server."
{
    export Down,
    Unresponsive;

    relationship SystemPackagedIn, Chassis, PackagesSystems
    problem Down =>
   Unresponsive,
   Unresponsive explains;
    event Unresponsive = IsUnresponsive;
    stored attribute boolean IsUnresponsive = FALSE;
    stored attribute boolean IsAgentUnresponsive = FALSE;
}
```

Dynamic modeling allows a new property to be added or one of the existing properties to be modified. However, the existing Unresponsive event, Down problem, SystemPackagedIn relationship, and the IsUnresponsive and IsAgentUnresponsive attributes cannot be deleted.

refine interface keyword

To modify an existing class, use the **refine interface** keyword. The **refine interface** keyword indicates that any declarations that follow will modify an existing ICIM class.

All refinements to a class *must* be contained in a single **refine interface** <*classname*> declaration. For example, you cannot have a **refine interface Router** declaration in the mymodel1.mdl file, and then another **refine interface Router** in the mymodel2.mdl. Likewise, you cannot have more than one **refine interface Router** declaration in the mymodel1.mdl file.

In this example, the stored attribute SM_RtrChassisType is added to the Chassis class. The Chassis class is one of the ICIM classes.

```
refine interface Chassis
{
 stored attribute string SM_RtrChassisType;
}
```

Refining inherited class properties

You can refine inherited attributes, events, problems, symptoms, operations or relationships when you create a new subclass. These class properties may be inherited from the parent class or any of the other classes up the inheritance tree to the base class. The syntax for refining an inherited attribute, event, problem, symptom, operation or relationship is:

```
interface <child_class> : <parent_class>
{
 refine <property declarations>
}
```

The **refine** syntax for each type of declaration such as an attribute, is the same as that described in the VMware Smart Assurance MODEL Reference Guide.

Adding an attribute

The syntax for adding a new attribute to an existing ICIM class is:

```
refine interface <ICIM_class_name>
{
<attribute declarations>
}
```

The following example adds the stored attribute SM_SerialNumber to the existing Router class.

```
refine interface Router
{
 stored attribute string SM_SerialNumber
 “Custom serial number stored in the Cisco”
 “chassis-id MIB (.1.3.6.1.4.1.9.3.6.3)”
 = “NotAvailable”;
}
```

For more information about attribute declarations and valid data types, see the VMware Smart Assurance MODEL Reference Guide.

Adding a relationship

A new relationship (one-to-one or one-to-many) can be added to an existing ICIM class. The syntax for adding a new relationship to an existing ICIM class is:

```
refine interface <ICIM_class_name>
{
<relationship declarations>
}
```

The following example adds a relationship, OSInstalled, to the existing ICIM class, UnitaryComputerSystem.

```
refine interface UnitaryComputerSystem
{
 relationshipset OSInstalled, SM_OperatingSystem, OSRunningOn;
}
```

For more information about relationship declarations, see the VMware Smart Assurance MODEL Reference Guide.

Adding an event

A new event can be added to an existing ICIM class. The syntax for adding a new event to an existing ICIM class is:

```
refine interface <ICIM_class_name>
{
<event declarations>
}
```

The following example adds the IPForwardingOn event to the existing ICIM class, Host.

```
refine interface Host
 {
  propagate attribute int min SM_ipForwarding
  "Value of ipFrowarding from SM_ipForwardingInstrument"
  <=SM_ipForwardingInstrument, InstrumentedBy, SM_ipForwarding;

  event IPForwardingOn "A Host has IPForwarding turned
   on" = SM_ipForwarding==1;

  export IPForwardingOn;
 }
```

For more information about event declarations, see the VMware Smart Assurance MODEL Reference Guide.

Adding an aggregate

A new aggregate can be added to an existing class. The syntax for adding a new aggregate to an existing class is

```
refine interface <class_name>
{
<aggregate declarations>
}
```

The following example adds the DegradedWirelessConnectivity aggregate to the existing class, SM_WirelessAccessPoint.

```
refine interface SM_WirelessAccessPoint
{
 stored attribute int MaxNumberIntended =20;
 propagate attribute int max SM_SignalDegradation
 <=SM_APInstrumentation, InstrumentedBy,
 SM_SignalDegradation;
 propagate attribute int max SM_NumberOfMACS
 <=SM_APInstrumentation, InstrumentedBy,
 SM_NumberofMACS;
 event PoorConnectivity “AP MIB returns loss in
 SignalPower (dB)” SM_SignalDegradation >10;
 event MaxUsersReached “Number of users on a signal
 AP device has been exceeded”
 SM_NumberOfMACS>MaxNumberIntended;
 aggregate DegradedWirelessConnectivity “Wireless
 connectivity for the AP is degraded” <= PoorConnectivity,
 MaxUsersReached;
 export DegradedWirelessConnectivity;
}
```

For more information about aggregate declarations, see the VMware Smart Assurance MODEL Reference Guide.

Adding a problem

A new problem can be added to an existing ICIM class. The syntax for adding a new problem to an existing ICIM class is:

```
refine interface <ICIM_class_name>
{
<problem declarations>
}
```

The following example adds the IPForwarded problem to the existing ICIM class, Host.

```
refine interface Host
 {
  propagate attribute int min SM_ipForwarding
  "Value of ipFrowarding from SM_ipForwardingInstrument"
  <=SM_ipForwardingInstrument, InstrumentedBy, SM_ipForwarding;

  event IPForwardingOn "A Host has IPForwarding turned
   on" = SM_ipForwarding==1;

  problem IPForwarded =>
   IPForwardingOn,
   IPForwardingOn explains;
  export IPForwardingOn;
  export IPForwarded;
 }
```

For more information about problem declarations, see the
VMware Smart Assurance MODEL Reference Guide.

Adding a symptom

A new symptom can be added to an existing ICIM class. The syntax for adding a new symptom to an existing ICIM class is:

```
refine interface <ICIM_class_name>
{
<symptom declarations>
}
```

For more information about symptom declarations, see the
VMware Smart Assurance MODEL Reference Guide.

Adding an operation

A new operation can be added to an existing ICIM class. The syntax for adding a new operation to an existing ICIM class is:

```
refine interface <ICIM_class_name>
{
<operation declaration>
}
```

For more information about operation declarations, see the
VMware Smart Assurance MODEL Reference Guide.

The dynmodel Command Syntax and Options

6

This chapter includes the following topics:

- Syntax and options
- Pragmas

Syntax and options

After you create dynamic model files, the class and property declarations must be compiled into a form that can be loaded by a Manager. The dynamic model compiler, **dynmodel**, is used to compile the modified model files.

The **dynmodel** compiler is located in the BASEDIR/smarts/bin/ directory.

The **dynmodel** command is used to invoke the dynamic model compiler on one or more files that contain model class definitions. The compiler checks the MODEL syntax but does not verify compatibility with other models, including name resolution. The dynamic model compiler generates a corresponding model file, for each input file, that can be loaded into a Manager. The generated files can be loaded by a Manager as part of the data model.

The syntax of the **dynmodel** command is:

```
dynmodel [<options>] <mdl-files>
```

where:

- <*options*> may be in any order, but must precede the .mdl files. Options are defined in Command line options for the dynmodel command.
- <*mdl-files*> specify one or more files, separated by a space.

Table 6-1. Command line options for the dynmodel command

option	Description
-w	Suppress generation of warning messages produced by the compiler.
-Wb,ProblemMode	Compile dynamic model files that have added or refined problems and symptoms.
--help	Prints the compiler's command line options and then exits.
--version	Prints the compiler's version information and then exits.

Table 6-1. Command line options for the dynmodel command (Continued)

option	Description
--l10nsrc	The base directory that holds locale source files.The default is l10nsrc. When **dynmodel** generates Resource Packages it incorporates files for each model based on *<dir>/<pkg_id>* where: *<pkg_id>* is the model filename less its extension, plus "_mm", and then truncated at 23 characters. This option *must* be used in conjuction with the **--**locales option. **Note** This option is not available in the Service Assurance Manager.
--locales	A comma-delimited list of locales which correspond to localized text files to be incorporated into the Resource Package for model files. This option *must* be used in conjuction with the **--**locales option. **Note** This option is not available in the Service Assurance Manager.

In this example, the command line suppresses the generation of warning messages produced when compiling the mymodel.mdl file.

```
dynmodel –w mymodel.mdl
```

The **dynmodel** command creates the following files in the BASEDIR/smarts/model directory, where *<modelname>* is the name of a dynamic model. Files and directories generated by the dynmodel compiler describes the set of files that is created for each of the dynamic model files included on the command line.

Table 6-2. Files and directories generated by the dynmodel compiler

Generated file or directory	Description
<modelname>.ldm	The version of the dynamic model that is loaded into a Domain Manager.
*<modelname>*_mmbe.dat	The Big Endian message file for the MODEL interface, attribute, and event descriptions
*<modelname>*_mmle.dat	The Little Endian message file for the MODEL interface, attribute, and event descriptions
*<modelname>*_mm.txt	The Resource Package source that contains MODEL text.
*<modelname>*_mm	A directory created for holding temporary files needed during the dynamic model compilation process. **Note** Do not remove this directory.

For example, the following commands compile the Asset.mdl and AssetUCS.mdl dynamic model files:

```
#

cd
BASEDIR
/smarts/local/model
```

```
            #


          BASEDIR
          /smarts/bin/dynmodel Asset.mdl AssetUCS.mdl
```

When the Asset.mdl and AssetUCS.mdl dynamic model files are compiled, the following set of files and directories are produced:

- Asset.ldm
- Asset_mmbe.dat
- Asset_mmle.dat
- Asset_mm.txt
- Asset_mm
- AssetUCS.ldm
- AssetUCS_mmbe.dat
- AssetUCS_mmle.dat
- AssetUCS_mm.txt
- AssetUCS_mm

```
Asset.ldm is the version of the dynamic model file that can be loaded by a Manager. Likewise,
AssetUCS.ldm is the version of the dynamic model file AssetUCS.mdlthat can be loaded by a Manager.
```

Using the --l10nsrc and --locales options

The --l10nsrc and --locales options cannot be used independently. You must either include both options on the command line or exclude both options from the command line.

The two options are applied uniformly to all model files on the command line. If you want to compile two (or more) models that have a different list of locales or with different l10nsrc base directories, then you need to use two (or more) different **dynmodel** calls.

When **dynmodel** generates Resource Packages, it will incorporate files for each model based on<*dir*>/<*pkg_id*> (where <*pkgId*> is the model file name less its extension, plus "_mm", then truncated at 23 characters.

In this example, **dynmodel** compiles two models, model1 and model2. The **dynmodel** compiler incorporates ./mysource/model1_mm/ja.txt and ./mysource/model1_mm/de.txt into model1's .dat file. The **dynmodel** compiler also incorporates ./mysource/model2_mm/ja.txt and ./mysource/model2_mm/de.txt into model2's .dat file.

```
dynmodel model1 model2 --l10nsrc=mysource --locales=ja,de
```

If you do not include the --l10nsrc and --locales options then **dynmodel** will not look for any localization .txt files, and the .dat file for the model will only support the native language of the .mdl file.

Pragmas

Pragmas can be used in dynamic model files to convey SNMP MIB qualities, and other directives, to the dynamic model compiler. Pragmas used in dynamic modeling lists the available pragmas.

Table 6-3. Pragmas used in dynamic modeling

Pragma	Description
#pragma DotNotation	Specifies that an SNMP instrumented attribute is to be converted from dotted notation when retrieved from an SNMP agent.
#pragma HexNotation	Specifies that an SNMP instrumented attribute is to be converted from hex notation when retrieved from an SNMP agent.
#pragma Local Operation	Locks only the object on which the operation is invoked. Normally, the dynamic model compiler generates code so that invoking an operation implicitly locks the entire repository.
#pragma ObjectID	Appears before an instrumented attribute declaration of type string whose value is the OID of an SNMP device. This pragma tells the repository to convert the OID type returned by the SNMP agent into a MODEL string.
#pragma Uses Propagation	Must appear before the attribute declaration to which it applies. This pragma tells the dynamic model compiler that access to this attribute may require access to other repository instances. Use this pragma before any attribute that can be refined as propagated or as computed with an expression that refers to other propagated attributes. This pragma locks the entire repository, however, the lock type (read/write) will be according to the type of operation called on the attribute. For example, if you are setting the value of an attribute, it will be a write lock.
#pragma WrapCounter	Specifies that an SNMP instrumented attribute is to be treated as a wrapping counter, which affects comparisons of previous and current values.

Methods of Creating Classes and Populating Attributes

7

This chapter includes the following topics:

- Overview
- Using dmctl
- Populating attributes by using an SDK API
- Discovery customization

Overview

For new classes added by dynamic modeling, instances of the class must be discovered and added to the repository. In addition, new attributes need to be populated with values.

There are several methods that may be used to create instances of new classes or to populate values of new attributes:

- The **dmctl** command
- Text file accessed by one of the SDK APIs (PERL, C, Java)
- ASL scripts
- Working with scripts that are invoked as part of the postprocessing phase ofthe discovery mechanism

Using dmctl

The **dmctl** command is a command line tool for interacting with a Manager. It may be used to create instances of new classes or manually populate attributes in a Manager. It can either execute commands typed at the command line, execute commands read from a batch file, or interactively read commands that the you type.

If the Global Manager is already running, and you use **dmctl** to modify class properties that it subscribes to, you need to re-synchronize the topology. Synchronizing topology changes with the Global Manager provides additional information.

Command-line mode

To run **dmctl** in command-line mode, connect to the Manager and type the command by using the following syntax:

```
#

BASEDIR
/smarts/bin/dmctl -s
<manager_name> [--broker=
<host>
:
<port>] <command_name>
<command_params>
```

where:

- *<manager_name>* is the name of the Manager.
- *<host>* is the host on which the Broker resides.
- *<port>* is the port on the host from which the Broker can be accessed.
- *<command_name>* is the name of the command.
- *<command_params>* is the list of the parameters for the command.

Note The **--**broker option is optional. The default value is localhost:426.

The following command, starts **dmctl** interacting with the TEST_SERVER Manager and creates an instance of the SM_Printer class, named HP_Printer1:

```
#

BASEDIR
/smarts/bin/dmctl -s TEST_SERVER create SM_Printer::HP_Printer1
```

You can also use **dmctl** to verify that a class has been added. For example:

```
#
BASEDIR/smarts

/bin/dmctl -n TEST_SERVER_RM getInstances <classname>
```

where:

- *<classname>* is the name of the class you want to verify.

  If the class is not loaded, **dmctl** will display output similar to:

  ```
  MR-E-CLASS_NOT_FOUND_EX-Cannot find class '<classname>'
  ```

  If the class has been added, **dmctl** will output a (possibly empty) list of instances.

Interactively

To run **dmctl** interactively, connect to the Manager as follows:

```
#

BASEDIR
/smarts/bin/dmctl -s
<server_name> [--broker=
<host>
:
<port>]
```

where:

- *<server_name>* is the name of the Manager.
- *<host>* is the host on which the Broker resides.
- *<port>* is the port on the host from which the Broker can be accessed.

Note The **--**broker option is optional. The default value is localhost:426.

The following command, starts **dmctl** interacting with the TEST_SERVER Manager registered with the broker running on the local host and port 426. At the prompts, type the username and password.

```
#

BASEDIR
/smarts/bin/dmctl -s TEST_SERVER --broker=localhost:426
```

The output appears similar to the following:

```
>dmctl -s TEST_SERVER --broker=localhost:426
Server TEST_SERVER User: admin
admin's Password: XXXXXXX
Domain Manager Control Program (v9.0.0.1) -- Type 'help' for a
list of commands.
Attached to 'TEST_SERVER'
dmctl>
```

Once attached to the Manager, use the **create** command to create an instance of a class:

```
create <class_name>::<instance_name>
```

where:

- *<class_name>* is the name of the class.
- *<instance_name>* is the name of the instance of the class.

For example, type the following command to create an instance of the SM_Printer class, named HP_Printer1:

```
dmctl>create SM_Printer::HP_Printer1
```

The **getClasses** command lists all of the classes in the repository.

You can also populate the value of an attribute by using the **put** command:

```
put <class_name>::<instance_name>::<property_name> <value>
```

where:

- *<class_name>* is the name of the class.
- *<instance_name>* is the name of the instance of the class.
- *<property_name>* is the name of the property in the class instance.
- *<value>* is the value of the property.

For example, type the following command to populate the value of the SM_SerialNumber attribute associated with router-1.mynet.com (an instance of the Router class):

```
dmctl>put Router::router-1.mynet.com::SM_SerialNumber 123456abc
```

Batch mode

The **dmctl** command may also be run in batch mode by passing it a filename. The batch file is created by using a text editor and consists of a series of **dmctl** commands, with one command per line.

The following batch file creates two instances of the Printer class, and sets the value of the SM_SerialNumber attribute for an instance of the Router class:

```
create SM_Printer::HP_Printer1
create SM_Printer::HP_Printer2
put Router::lab-gw::SM_SerialNumber 123456abc
```

Before running **dmctl** in batch mode, change the directory to the location of the batch file. The following example runs **dmctl** in batch mode:

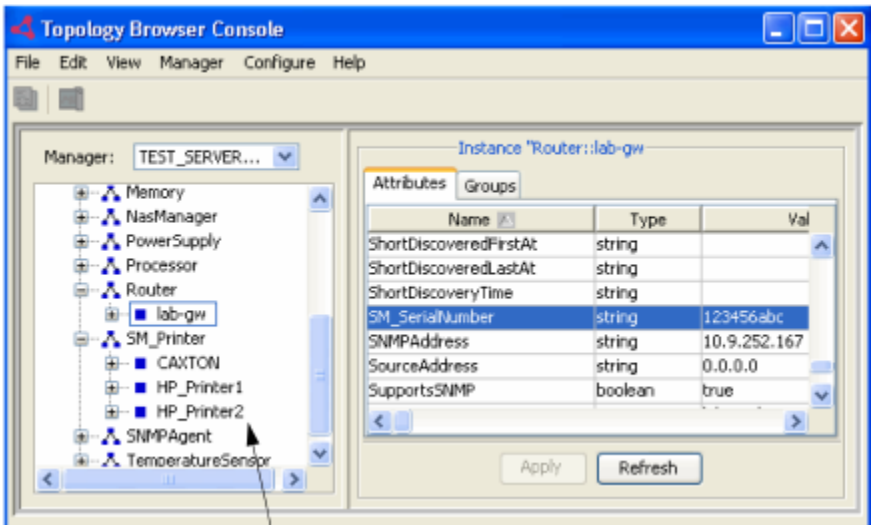
```
t#

        BASEDIR
        /smarts/bin/dmctl --server=TEST_SERVER_RM

        --broker=localhost:426 --file=test_dmctl_batch.txts
```

Resulting topology from dmctl batch file “test_dmctl_batch.txt” shows the two new printers and the new value of the Router serial number in topology.

1 New Instances of SM_Printer Class

New Value of SM_SerialNumber

Figure 7-1. Resulting topology from dmctl batch file “test_dmctl_batch.txt”



Synchronizing topology changes with the Global Manager

If the Global Manager is already running, and you modify class properties that it subscribes to, you need to resynchronize the topology in order for Global Manager to reflect the changes. From the command line, type the following:

```
t#

        BASEDIR
        /smarts/bin/dmctl -s <GLOBAL_MANAGER> invoke GA_Driver::<
           Domain Manager
        >_Topo-Driver starts
```

where:

- *<GLOBAL_MANAGER>* is the name of the Global Manager.

- *<Domain Manager>* is the name of the underlying Domain Manager that is being synchronized.

  For example:

```
t#

                BASEDIR
                /smarts/bin/dmctl -s TEST_SA invoke GA_Driver::TEST_SERVER_Topo-Driver starts
```

Populating attributes by using an SDK API

This section describes how to populate attributes with the SDK API.

PERL API

The following is an example of a PERL script that can be used to populate the value of the SM_SerialNumber attribute in the Router class. The VMware Smart Assurance Perl Reference Guide provides additional information.

```
use InCharge::session;
#Establish a connection to the VMware Smart Assurance Server
$session=InCharge::session->new( server=>"TEST_SERVER_EX2",
            broker =>"localhost:426",
            user=>"admin",
            password=>"ad1");

# Create a handle to the object router-1.mynet.com
$deviceHandle=$session->object("Router","router-1.mynet.com");
# Retrieve value of current Serial Number
$currentSerial=$deviceHandle->{SM_SerialNumber};
print "Current Serial Number: $currentSerial \n";
#Assign a value to the attribute SM_SerialNumber of device router-1.mynet.com
$deviceHandle->{SM_SerialNumber}="123456abc";
# Print the new value
print "New Serial is: $deviceHandle->{SM_SerialNumber}";
```

ASL

The following is an example of an ASL script that can be used to populate the value of the SM_SerialNumber attribute in the Router class. The VMware Smart Assurance ASL Reference Guide provides additional information.

```
START{
 ..eol
 }
 do
 {
  /* Retrieve existing value */
  objHandle=object("Router","router-1.mynet.com");
  print("Current serial: ".objHandle->SM_SerialNumber);
```

```
  /* Assign a new value */
  objHandle->SM_SerialNumber="123456abc";
  print("New serial: ".objHandle->SM_SerialNumber);

  stop();
 }
```

Discovery customization

This example populates the value of SM_SerialNumber (an attribute in the Router class) by querying a MIB. The postprocessing scripts can be used to query a MIB during discovery.

First, a driver is created that sets up the process of querying the MIB for the value of SM_SerialNumber. A driver is an instance of the GA_Driver class, and in this example it is defined in an import file. The import file is loaded into the test Network-Resource Manager when it is started with the **sm_server** command.

During a discovery, the custom-end-system.asl script is automatically run for each device after Phase III (discovery probing). Additional rules are added to this file. Within these rules, if the object is a router, it launches the driver that was created when the import file was read into memory. The driver executes and retrieves the values in the appropriate OID. The results are then forwarded to an ASL script that parses the input and populates the value of the SM_SerialNumber attribute.

This process is summarized in Figure 15 on page 98.

1 1

2 2

3 3

4 4

5 5

DISCOVERY_serialNumber.import

custom-end-system.asl

The import file is loaded into the Manager memory at runtime. An instance of GA_Driver is created that defines the input source, how the input is parsed, and which repository the attribute value is stored in.

This discovery postprocessing script is executed during the discovery process.

This discovery postprocessing script invokes the GA_Driver that was loaded into memory when the Manager was started.

The GA_Driver performs an SNMPWalk to retrieve the value from the MIB

The input that is retrieved is parsed by this ASL script (defined by the GA_Driver) and assigned to the appropriate instance and attribute.

dynmodel-sn.asl

Figure 7-2. Using the discovery postprocessing mechanism

Creating the driver

First, you create a driver that sets up the process of querying the MIB for the value of SM_SerialNumber.

A driver is an instance of the GA_Driver class, and is composed of three parts:

- Front End — The source of the input (defined by the ReadsInputFrom attribute of the GA_Driver class).
- Rule Set — The ASL script that parses through the input string to determine the values of the attributes (defined by the ReadsRulesFrom attribute of the GA_Driver class).
- Back End — The repository where the attribute values that are parsed from the input string are stored.

  In this example, the back end is the Manager that is started with the **sm_server** command and loads the import file. The value for the root OID in this example is for a Cisco router, Cisco chassis-id MIB (3).

  To create an import file:

  a Start editing the import file.

```
t #

                    BASEDIR
                    /smarts/bin/sm_edit --create

                     conf/discovery/DISCOVERY_serialNumber.imports
```

  b Type the following lines into the DISCOVERY_serialNumber.importfile:

```
                     GA_Driver::DynModel-SerialNumber-Driver {
                      ReadsInputFrom = SNMP_WalkFE::DynModel-SN-FrontEnd
                      {
                       retries =5
                       rootOIDs = {
                         { 10, ".1.3.6.1.4.1.9.3.6.3" }
                       }
                       trace=FALSE
                      }

                      ReadsRulesFrom = GA_RuleSet::DynModel-SN-RuleSet
                      { fileName = "discovery/dynmodel-sn.asl"
                       trace=FALSE}


                     waitForCompletion =TRUE
                    }
```

c Save and close the DISCOVERY_serialNumber.import file.

The DISCOVERY_serialNumber.import file is saved into the *BASEDIR*/smarts/local/conf/discovery directory.

In this import file, an instance of the GA_Driver class is created, named DynModel-SerialNumber-Driver. Since an OID is queried for its value, a SNMP_WalkFE front end is used. The SNMP_WalkFE instance, DynModel-SN-FrontEnd:

- Retrieves all of the MIB objects under a set of root OIDs.
- Converts the values to ASCII and markers.
- Passes them to the ASL file that is defined to parse the input..

In this import file, the ASL file is defined by the ReadRulesFrom attribute as discovery/dynmodel-sn.asl.

Editing custom-end-system.asl

During a discovery, the custom-end-system.asl script is automatically run for each device after Phase III (discovery probing). At the end of the custom-end-system.asl script is a section where you can add your own processing for each device, as shown in Modifying custom-end-system.asl.

START {

/*

* Add Parsing rules here.

*/

.. eol

} do {

/*

* Add processing rules here.

*/

stop();

}

New processing rules are added here.

Figure 7-3. Modifying custom-end-system.asl

To add custom processing:

1 Start editing the custom-end-system.asl file, by using **sm_edit**:

```
#

cd
BASEDIR
```

```
/smarts/bin
#

sm_edit rules/discovery/custom/custom-end-system.asl
```

2 Type the following lines after **Add processing rules here** in the START rule.

```
/* Check first if the object is a router since you have only added
* serial number of the router class */
if (nodeObj->CreationClassName =="Router")
{
 /* Setup a handle to the internal GA_Driver's
 GA_Parameters instance for custom-end-system.asl */
   cesObj=this->findMyParameters();

 /* This statement creates the object handle to the
 * driver defined in our import file
 * DISCOVERY_serialNumber.import. */
 DynModelSerialNumberDriverObj = object("DynModel-SerialNumber-Driver");
 /* Now you kick off the GA _Driver that you have defined
 * in the import file whose purpose is to poll the device
 * for some MIB values and return it.   You start the driver
 * by calling the operation startWithParamters and pass as an
 * argument the parameters that are available through cesOBJ.
   *
   * Note, since the driver in the import file uses
 * SNMPWalkFE as the frontend, the results from this
 * front end are in the form of OID|Value
 * for example, a sample output is:
 * .1.3.6.1.2.1.25.1.1.0|6297154|;
 * therefore the ASL script dynmodel-sn.asl
 * defined by the driver in the import file has to
 * parse the output in this format.
 */
 print("Starting DynModel-SerialNumber-Driver...");
 DynModelSerialNumberDriverObj->startWithParameters(cesObj);
}
```

3 Save and close the custom-end-system.asl file.

The custom-end-system.asl file is saved into the *BASEDIR*/smarts/local/rules/discovery/custom directory.

Creating the script to parse the value of SM_SerialNumber

This script parses the results of the SNMPWalk to retrieve the value of SM_SerialNumber.

To create the script:

1 Start editing the ASL file.

```
#

cd
BASEDIR
/smarts/bin
#

sm_edit --create rules/discovery/dynmodel-sn.asl
```

2 Type the following lines into the dynmodel-sn.asl file:

```
/* Create a handle to the SNMP Agent object */
 agentObj=object(AgentName);

 /* Test that the results are not null */
 if (agentObj->isNull()  ) {
  print("***********Error in dynmodelsn.asl********");
  stop();
  }
 /* DEBUG= TRUE; */


 /* Get a handle to the actual device by using the getSystem

 operation available through the agent object handle */
 nodeObj=agentObj->getSystem();
 print("NODE:".nodeObj->DisplayClassName);
 START{
  SERIAL
 }
 /* fs is need as field separators in this parsing.*/
 /* This OID matches what was defined in the import file */
 serialNumberOID {
  ".1.3.6.1.4.1.9.3.6.3"
 }
 SERIAL{
  oid:{ serialNumberOID.".".integer }  fs
  serial:word fs
  }
  do{
   /* As the snmpwalk returns a value without knowing the

  type, you need to use hexToString to convert the
    * result from hex to ASCII string */
   serial=hexToString(serial);

  /* Assign the results to the new SM_SerialNumber

  attribute */
```

```
                 nodeObj->SM_SerialNumber=serial;

                stop();
                }
```

3 Save and close the dynmodel-sn.asl file.

The dynmodel-sn.asl file is saved into the *BASEDIR*/smarts/local/rules/discovery directory.

Restarting the Network-Resource Manager

Once the driver has been created, and the custom-end-system.asl and dynmodel-sn.asl files have been edited, the Network-Resource Manager needs to be restarted.

To restart the Network-Resource Manager:

1 Stop the test Network-Resource Manager.

2 Reload the compiled dynamic model file into the test Network-Resource Manager.

```
                #

                cd
                BASEDIR
                /smarts/local/model
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER -c icf-r

                 --broker=localhost:426 --dynamics
```

Updating the Global Manager

8

This chapter includes the following topics:

- Loading the dynamic model enhancements
- Transferring topology and events

Loading the dynamic model enhancements

If the new topology needs to be accessible by the Global Manager, then the dynamic model files must be loaded into the Global Manager. In addition, the Global Manager configuration file must be modified to subscribe to new events.

Note BASEDIR is the directory for the Service Assurance Manager.

To load the dynamic model enhancements into the Global Manager:

1 Copy the appropriate dynamic model files (.mdl files) to the Service Assurance Manager's *BASEDIR*/smarts/local/modeldirectory.

   All subsequent steps refer to the copies of the dynamic model files in the Service Assurance Manager's *BASEDIR*/smarts/local/modeldirectory.

2 Remove any instances of the binary data type from the dynamic model files.

3 Replace any instances of the byte data type with the int data type in the dynamic model files.

4 Remove any instances of the **external** keyword in the dynamic model files.

5 Remove any problems or events in the dynamic model files.

6 Compile the dynamic model files.

   For example, to compile the Example2.mdl dynamic model file:

```
#

cd
BASEDIR
/smarts/local/model
#
```

```
                BASEDIR
                /smarts/bin/dynmodel Example2.mdl
```

7 Load the compiled dynamic model files into the Global Manager.

For example:

```
t#

                BASEDIR
                /smarts/bin/sm_server -n TEST_SA -c ics

                 --broker=localhost:426 --output --dynamics
```

Transferring topology and events

The BASEDIR/smarts/conf/ics/ics-default.xml file is the configuration file for the Global Manager. This file designates which dxa files are used to transfer events and topology from the Network-Resource Manager, IP-Perspective Manager, and IP-Analysis Manager into the Global Manager. The ics-default.xml file contains a separate section for each of the three Domain Managers. Each section indicates the dxa file that is used for that particular Domain Manager.

In this example, the ics-default.xml file sets dxa-net-rm-conn-perf.conf as the dxa file that defines which topology and events to transfer from a Network-Resource-Suite to the Global Manager.

```
<domaintype
     name="Network-Resource-Suite"
     dxaConfFile="dxa-net-rm-conn-perf.conf"
     MinimumCertainty="0.01"
     SmoothingInterval="65">
     <description>Type for Network Resource Connectivity and Performance Suite</description>
     <hook enable="False">ics/dxa-sample-hook.asl</hook>
     <domainname>Network-Resource</domainname>
 </domaintype>
```

Transferring topology

The first section of the dxa configurationfile provides a listing of what classes, and attributes of those classes, will be transferred from the Manager to the Global Manager. If an attribute has been added to an existing ICIM class, and you want that attribute to appear in the Global Manager, then the class and attribute must be added to this listing.

Note Use **sm_edit** to edit the dxa configuration file.

For example, the attribute SM_SerialNumber is added to the Router class. To add the SM_SerialNumber attribute in the Global Manager, the following lines would be typed (shown in **bold**) before the entry for the ICIM_UnitaryComputerSystem, as illustrated:

```
# Remote Name of Class/Property        Mapped Name of Class/Property
server   Network-Resource
class   Router
         attr    SM_SerialNumber
class   ICIM_UnitaryComputerSystem           Node
attr    Certification
attr    DisplayName
attr    Description
attr    Location
attr    Model
attr    PrimaryOwnerContact
attr    Type
attr    Vendor
rel     ConnectedVia
```

The addition of these two lines configures the Global Manager to subscribe to topology changes for the new SM_SerialNumber attribute of the Router class.

Transferring events

In order for the event that is generated by one of the IP Manager Domain Managers to appear in the Global Manager, you need to edit the appropriate dxa configuration file to subscribe to the new event.

Events are based on expressions that are created based on attribute values. These attribute values are not polled for unless there is a subscription to the event that utilizes this value.

Note Use **sm_edit** to edit the dxa configuration file.

Subscribing to the event

Event and notification subscriptions in the Global Manager are configured within dxa configuration files by using the **sub** keyword. Patterns (regular expressions) separated by "::" match the class name, instance name, and event name in the Domain Manager.

The syntax is as follows:

sub <*classPattern*>**::**<*instancePattern*>**::**<*eventPattern*>*I*<*modifier*s>

where:

- <*classPattern*> is the pattern to match for the class name.
- <*instancePattern*> is the pattern to match for the name of the instance.
- <*eventPattern*> is the pattern to match for the name of the problem or event.
- <*modifiers*> can have the following values:
  - p = problem

- e = event
- a = aggregates
- n = notifications

  For example:

  ```
  # Subscriptions
  .
  .
  sub     ICIM_UnitaryComputerSystem::.*::DiscoveryError/e
  sub     ICIM_UnitaryComputerSystem::.*::Unresponsive/e
  .
  .
  ```

  To subscribe to a new event, add a line to this Subscriptions section. For example, to subscribe to a new IPForwardingOn event in the Host class, the following line is added:

  ```
  sub  Host::.*::IPForwardingOn/e
  ```

Setting the notification characteristics of the event

The next section of the dxa file defines the category and severity associated with each of the subscribed events.

```
#COMPUTER SYSTEMS
sev  ExternalPowerSupply  Down               Operational 1
sev  Host      Unresponsive       Operational 1
sev  Host       AgentUnresponsive  Operational 2
sev  Application       Unresponsive       Operational 2
```

The syntax is:

```
sev
<Class_Name>
<Problem_Event_Name>
<category>
<severity_code>
```

where *<severity_code>* can have one of the following values:

- 1 = red, critical.
- 2 = orange, major.
- 3 - yellow, minor.
- 4 = cyan, unknown.

- 5 = green, normal.

  The
  VMware Smart Assurance IP Performance Manager and Server Performance Manager User Guide and VMware Smart Assurance IP Availability Manager User Guide provide additional information concerning the valid values for <*category*>.

  To set the category and severity of an event that you have created, add a new line to this section. For example, to define the severity and category for the new IPForwardingOn event, add the following line:

  ```
  sev  Host    IPForwardingOn       Operational 3
  ```

Troubleshooting

9

This chapter includes the following topics:

- Overview
- Loading errors
- Discovery errors

Overview

This appendix describes common problems and their resolutions. The problems are categorized as:

- Compilation — Errors that may occur while compiling dynamic models.
- Loading — Errors that may occur while loading dynamic models into a Manager.
- Discovery — Errors that may occur when discovering new classes and populating attributes.

Loading errors

This section explains some common errors that you may encounter while loading dynamic models into an Manager.

Note If you include the **--**output option with the **sm_server** command used to load dynamic models into an Manager, these error messages appear in the log file. Otherwise, these error messages are displayed on the command console.

Missing ICU files

The following error is encountered when the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files have not been copied over to the SM_SITEMOD/l10n directory, after the model files have been compiled using **dynmodel**.

```
[September 28, 2009 3:30:19 PM EDT +834ms] t@1119086912 LDM file processing #1
DYN-W-CANT_LOAD_DAT-Can't load ICU dat file 'Example1_mm' version 4ac10ddd (pkgname
'Example1_mm4ac10ddd')
 CI-REGISTER_PKG-Error occurred while registering package 'Example1_mm4ac10ddd'; in file
"/work/blackcurrent/DMT-9.0.0.X/1189/smarts/clsapi/ci_i18n.c" at line 290
 I18N-EACCES_PKGFILE-Cannot access resource bundle file
```

Solution

From the BASEDIR/smarts/model directory, copy the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files to the SM_SITEMOD/l10n directory.

For example, copy the following files to the SM_SITEMOD/l10n directory:

- Example1_mmbe.dat
- Example1_mmle.dat

> **Note** You must copy the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files to the SM_SITEMOD/l10n directory every time you compile the <*modelname*>.mdl file using the **dynmodel** compiler.

Duplicate OID entries

The following error is encountered when an OID is defined in more than one oid2type file. In this example, the printer OID was added to BASEDIR/smarts/conf/discovery/oid2type_Field.conf so that this printer would be added to topology as an instance of SM_Printer. However, this OID is also defined in another oid2type file.

```
[November 3, 2009 12:35:08 PM EST +668ms] t@1150445888 InCharge Framework
ASL_MSG-*-ASLP-discovery/discovery-init.asl:
DSCV-W-While loading oid2type files, the following entries are duplicate
[November 3, 2009 12:35:08 PM EST +668ms] t@1150445888 InCharge Framework
ASL_MSG-*-ASLP-discovery/discovery-init.asl: .1.3.6.1.4.1.11.2.3.9.1 in file:
BASEDIR/smarts/local/conf/discovery/oid2type_Field.conf
```

Solution

Determine which oid2type_*.conf file contains the duplicate entry. Use **sm_edit** to edit the oid2type file that contains the duplicate entry and comment out the entry.

For example, the printer OID in this error is also defined in BASEDIR/smarts/conf/discovery/oid2type_Misc.conf.

1 Edit the file by using **sm_edit**.

```
BASEDIR/smarts/bin/sm_edit conf/discovery/oid2type_Misc.conf
```

2 Comment out the entry for the duplicate OID, by preceding the lines with a pound sign (#):

```
#.1.3.6.1.4.1.11.2.3.9.1 {
#  TYPE = Host
#  VENDOR = HP
#  CERTIFICATION = VALIDATED
#
```

```
#INSTRUMENTATION:
#  Interface-Fault   = MIB2
#  Interface-Performance  = MIB2
#}
```

3 Save and close the file. The edited copy is saved in the *BASEDIR*/smarts/local/conf/discovery/ directory.

4 Restart the Manager with the **--**dynamic option.

Manager cannot restore topology

An error, similar to the following, is encountered when you try to start a Manager

```
[November 3, 2009 12:11:00 PM EST +288ms] t@1141098816 InCharge Framework
ICF-E-EINIT-While initializing the InCharge application
 ICF-ERESTOREREPOS-While attempting to restore topology from file
BASEDIR/smarts/local/repos/icf/TEST_SERVER_RM.rps
 MR-CANTCREATE_INSTANCE-'SM_Printer::CAXTON' could not be created.  ; in file
"/work/blackcurrent/DMT-9.0.0.X/1201/smarts/repos/mr/dyn_acc.c" at line 2829
 MR-CLASS_NOT_FOUND-Class of given name 'SM_Printer' not found; in file
"/work/blackcurrent/DMT-9.0.0.X/1201/smarts/repos/mr/dyn_acc.c" at line 1709
[November 3, 2009 12:11:00 PM EST +430ms] t@1141098816 InCharge Framework
SVIF-N-SHUTDOWN-Shutting down...
```

This error is occurring because you have an instance of a class in the repository, but the class is not defined in the data model of the Manager. Similar errors occur for attributes, events, operations, problems, symptoms, or aggregates that are in the repository, but not defined in the data model of the Manager.

For example, a dynamic model file was previously loaded into the Manager. The dynamic model file added a new class called, SM_Printer. An instance of the SM_Printer class was discovered and added to the repository of the Manager.

Now, when you invoke the **sm_server** command the appropriate dynamic model file must be in the proper directory. The directory is determined by whether or not you are including the **--**dynamic option on the **sm_server** command line. If you are not loading the dynamic model enhancements into the Manager's data model, then the SM_Printer class does not exist in the Manager's data model. The instance of SM_Printer still exists in the repository, and the system tries to restore the instance of SM_Printer that was previously discovered. However, the class does not exist.

Solution

If you want the dynamic model enhancements and the previously discovered topology to be restored in the Manager, then place your dynamic model in the appropriate directory. The directory is based on your use of the **--**dynamic option when you invoke **sm_server.**

Including the **--**dynamic option on the command line when you start up a Manager, loads all of the .ldm files in the following directories:

- *BASEDIR*/smarts/model

- *BASEDIR*/smarts/model / *<config>*

  The *<config>* refers to the configuration directory that is specified on the command line when the Domain Manager is started.

  If you start a Manager without the **--**dynamicoption, then only the following .ldm files will be loaded:

- *BASEDIR*/smarts/local/model */<config>*

  The *<config>* refers to the configuration directory that is specified on the command line when the Domain Manager is started.

  If you want to discard the dynamic model enhancements, including any topology changes that were discovered for the dynamic model enhancements, then invoke the **sm_server** command with the **--**ignore-restore-errors option.

```
t #

                BASEDIR
                /smarts/bin/sm_server -n TEST_SERVER -c icf-r

                 --broker=localhost:426 --bootstrap=bootstrap-am.conf

                 --ignore-restore-errorss
```

Unable to open <ldmfilename>

The .ldm file, *<ldmfilename>*, cannot be opened by the Manager. BASEDIR/smarts/model directory.

Solution

Verify that *<ldmfilename>* has the correct read permissions. This error should be pretty rare, since most Managers are started by root. They have administrative privileges.

DYN-E-CLASS_NAME_ALREADY_EXIST

The name of a class declared in the .mdl source file, at the given line number, conflicts with an existing class name in the configuration already loaded.

Solution

You must choose a different class name.

1. Edit the .mdl file.
2. Run **dynmodel**.
3. Copy the *<modelname>*_mmbe.dat and *<modelname>*_mmle.dat files to the SM_SITEMOD/l10n directory.
4. Restart the Manager with the **--**dynamic option.

DYN-E-STRUCT_NAME_ALREADY_EXISTS

The name of a structure declared in the .mdl source file, at the given line number, conflicts with an existing structure name in the configuration already loaded.

Solution

You must choose a different structure name.

1 Edit the .mdl file.

2 Run **dynmodel**.

3 Copy the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files to the SM_SITEMOD/l10n directory.

4 Restart the Manager with the **--**dynamic option.

DYN-E-ENUMERATION_NAME_ALREADY_EXISTS

The name of an enumeration declared in the .mdl source file, at the given line number, conflicts with an existing enumeration name in the configuration already loaded.

Solution

You must choose a different enumeration name.

1 Edit the .mdl file.

2 Run **dynmodel**.

3 Copy the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files to the SM_SITEMOD/l10n directory.

4 Restart the Manager with the **--**dynamic option.

DYN-E-ATTRIBUTE_NAME_ALREADY_EXISTS

The name of an attribute declared in the .mdl source file, at the given line number, conflicts with an existing attribute name in the configuration already loaded.

Solution

You must choose a different attribute name.

1 Edit the .mdl file.

2 Run **dynmodel**.

3 Copy the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files to the SM_SITEMOD/l10n directory.

4 Restart the Manager with the **--**dynamic option.

DYN-E-EVENT_NAME_ALREADY_EXISTS

The name of an event declared in the .mdl source file, at the given line number, conflicts with an existing name in the configuration already loaded.

Solution

You must choose a different event name.

1 Edit the .mdl file.

2 Run **dynmodel**.

3 Copy the <*modelname*>_mmbe.dat and <*modelname*>_mmle.dat files to the SM_SITEMOD/l10n directory.

4 Restart the Manager with the **--**dynamic option.

DYN-E-NO_AGGREG_OP

A property that is propagated across a relationshipset needs an aggregation operator. The Dynamic MODEL compiler cannot tell whether a name is a relation or a relationshipset until runtime, so it does not require an aggregation operator. This message is followed by the class name, propagated property name, and relationshipset name.

```
<classname>:<property>:<relationshipset>
```

Solution

Find the .mdl source file which declares the propagated property and add the aggregation operator to the propagated property declaration.

Discovery errors

This section explains some of the errors that you may encounter while discovering classes and populating attributes.

Class property changes do not appear in Service Assurance

You modified class properties, such as attribute values, in the underlying Domain Manager, but you do not see those changes reflected in the Global Manager. The problem is that the topology in the underlying Domain Manager and the Global Manager are no longer synchronized.

Solution

You can synchronize the topology between an underlying Domain Manager and the Global Manager by:

- Initiating a discovery.

- Using the **dmctl** command. Synchronizing topology changes with the Global Manager provides additional information.

Running the ASL scripts

ASL-ERROR_PUT

The following error usually occurs when there is a misspelling or missing capital letters. For example, the following error was generated because filename was used instead of fileName:

```
# ASL-ERROR_PUT-While attempting to set property 'filename' of object
#     'GA_RuleSet::DynModel-OS-RuleSet'
#CI-EGENERIC-filename
# MR-PROPERTY_NOT_FOUND-Specified property not found in class
```

ASL-ERROR_CREATE

The following error was generated due to a syntax error. For example, GA_RulesSet was used instead of the correct GA_RulesSet

```
# ASL-ERROR_CREATE-While attempting to create the object
#    'GA_RulesSet::DynModel-OS-RuleSet'
# CI-EGENERIC-GA_RulesSet
# MR-CLASS_NOT_FOUND-Class of given name not found
```

MODEL and Dynamic Models

10

This chapter includes the following topics:

- The ICIM Reference
- Representing systems with MODEL
- Differences between MODEL and Dynamic Model

The ICIM Reference

Managed Object Definition Language (MODEL) is integral to the SDK application development process. Its purpose is to render the schema and semantics of the VMware Smart Assurance ICIM Reference specification. MODEL is compiled, producing data model libraries that describe the model. These libraries are loaded by a Domain Manager and are operated on by SDK applications. For an SDK application, all of the classes, relationships, attributes, events, and so on are defined at compile time.

Dynamic Modeling is an extension of the existing MODEL language, building on the same core implementation and principles. It enables you to add new classes, and refine classes that are already defined in the data model libraries, at runtime when the Dynamic Models are loaded into an Manager.

The ICIM represents logical and physical components of managed systems across networked technology domains. By using an object-oriented representation, it is a conceptual view of the problem domain that provides Domain Managers with semantics that support root-cause analysis.

SDK models are object-oriented.

- The notion of *class* provides a mechanism for encapsulating a managed element's state or properties. Two semantic properties especially useful for root-cause analysis are *relationship* and *event*. Relationship expresses the correlation dependencies between classes. Events are those actions that cause transitions within the model.
- The object-oriented *inheritance* construct allows new classes to be derived from existing classes. This is a mechanism for specialization within the model, while preserving model semantics for the new class. New classes *inherit* not only the properties of the base class but its semantics, as well.

- The object-oriented *instance* construct provides a mechanism for separating the model representation of a managed element from its actual implementation. This supports desirable modeling goals, where the model is conceptual and generalized, and independent of any particular implementation within the problem domain. With respect to ICIM, this means that the correlation model, represented by *classes*, is separated from the domain topology, represented by *instances*. So ICIM is a true model, in the sense that it can accurately conceptualize, or model, particular implementations without prior knowledge.

Representing systems with MODEL

An ICIM class is actually defined by describing it as an interface. This is consistent with the programming practice of information hiding, where program elements are known and fully described by their interface and not by their implementation. To accomplish this, a language is needed to establish syntactic rules for the interface itself and for programs that use the interface. For SDK, the Managed Object Definition Language (MODEL) was developed to describe ICIM classes.

The MODEL is a textual notation with a syntax that supports a common way of representing heterogeneous elements. This also allows tools, such as compilers and analysis programs, to be applied to the model. The MODEL also defines semantic constructs that support the correlation model.

The example shows a simple MODEL interface declaration.

```
interface MyExample : ICIM_ManagedSystemElement
“An interface to generate an event when value > threshold.”
{
 attribute int MyValue = 0;
 attribute int MyThreshold = 50;
 event MyValueOverThreshold = MyValue > MyThreshold;
 export MyValueOverThreshold;
};
```

The interface has a name identifier, MyExample, and inherits from a base class, ICIM_ManagedSystemElement. The interface may also include a comment that describes the interface.

Constructs that support the correlation model include:

- Typed attributes
- The attribute int
- Events that are computed by and exported from the interface: event and export

  The MODEL compiler generates the C++ source code needed to access an interface’s attributes and to monitor the event conditions and generate an event notification. The generated source code is linked as an executable model library, which is loaded by a Manager whose analysis programs operate on the loaded model.

Add your model source file to the makefile used to build the system. Executing the appropriate makefileinvokes the model compiler for your model file, producing C++ source code and header files. The C++ compiler is then called to produce an object file for linking. (The filename extension of the executable model is specific to your operating system.) Your model object file is linked with VMware Smart Assurance model libraries and produces an executable model of the domain, which includes your application.

To execute your application, start a Manager by using **sm_server**. Specify the model executable to load on the **sm_server** command line.

Differences between MODEL and Dynamic Model

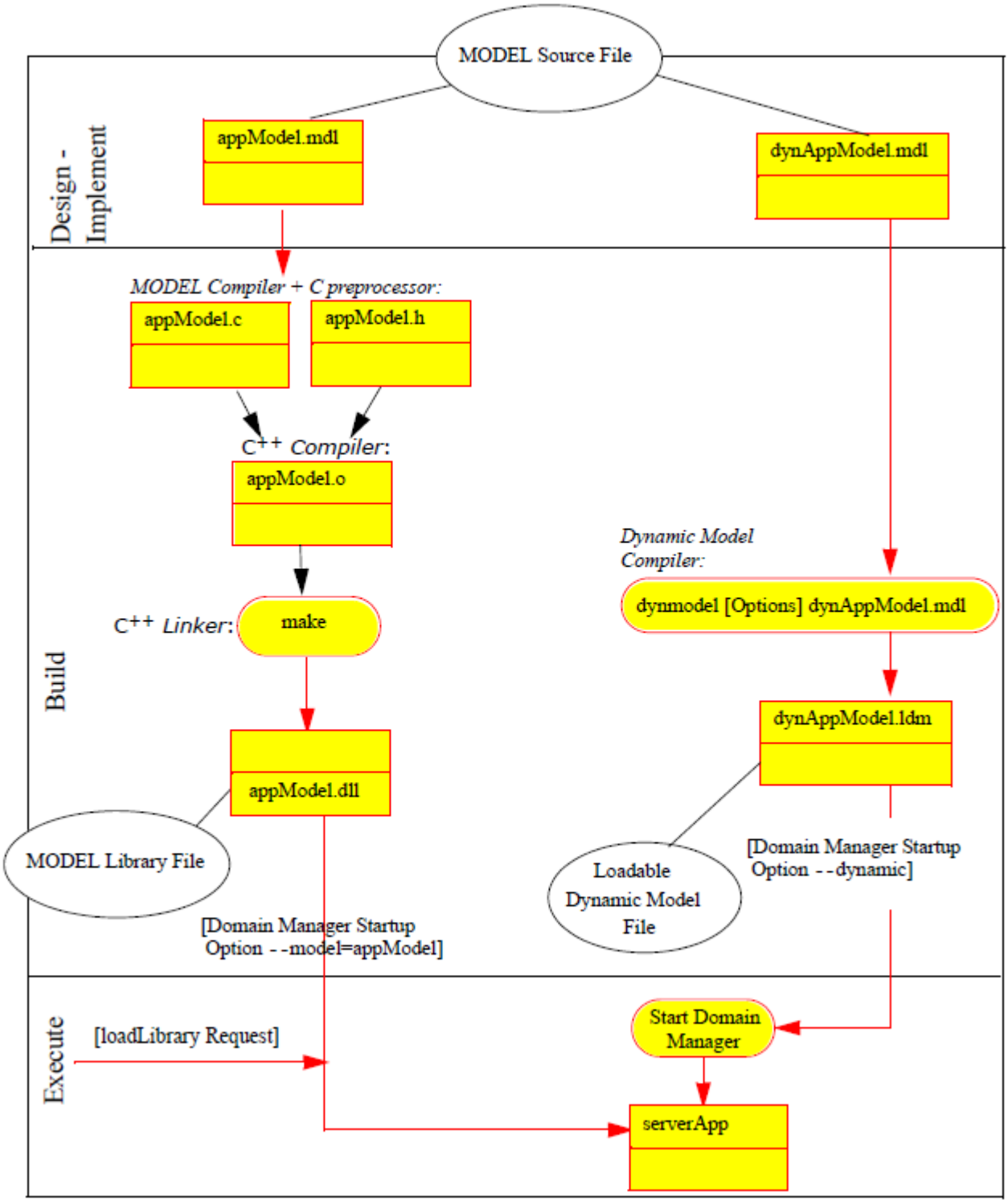
ICIM models are bound to programs that operate on the models at compile time. This is a characteristic of both the MODEL language and of the methodology for developing models.

This imposes a limitation on applications that are not aware of all of an interface's properties at compile time. This typically occurs because VMware Smart Assurance models tend to be general enough to support a large customer base. A particular customer, for whom more interface properties would be useful, must, however, limit the application programs to only what is made available by the ICIM model.

Dynamic modeling is a step toward solving this problem. It achieves this by allowing delays when binding occurs between the model and the programs that operate on the model. Instead of binding at compile time, as with the traditional MODEL, interfaces may now effectively be made known to the system at runtime, when they are loaded into a Manager as a dynamic model.

Simplified MODEL and Dynamic Model methodology comparison shows the differences between using MODEL and Dynamic Model. The obvious differences between the process of creating a traditional model executable and creating a dynamic model executable is that intermediate C++ source code is not generated and the dynamic model is not linked into the system during the build phase. Instead, the dynamic model compiler is invoked to directly produce a file with a .ldm extension that can be loaded by a Manager. When the Manager is started, and the dynamic model is loaded, the existing model class dependencies and the new dynamic model class dependencies are resolved.

Figure 10-1. Simplified MODEL and Dynamic Model methodology comparison



Note the following:

There are other files that are produced by the MODEL compiler and by the Dynamic Model compiler. Simplified MODEL and Dynamic Model methodology comparison is a simplified view to highlight the major differences. A more detailed explanation of the MODEL compiler is available in the VMware Smart Assurance MODEL Reference Guide.

Dynamic model files can be loaded when a Manager is started or after the Manager is already running.

Although MODEL and Dynamic Model are used independently and involve different steps and tools, the dynamic model is fully derived from MODEL. Therefore, many of the concepts that apply to MODEL also apply to the dynamic model. Familiarity with the VMware Smart Assurance MODEL Reference Guide is useful when working with Dynamic Model.