EMC® Smarts®

Foundation 9.2

Dynamic Modeling Tutorial

P/N 300-999-922

REV 01

EMC²

Copyright © 2004, 2006 - 2013 EMC Corporation. All rights reserved. Published in the USA.

Published March, 2013

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC^2, EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to the technical documentation and advisories section on EMC Online Support.

CONTENTS

Preface

Chapter 1 Introduction

Terminology 16
Availability, features, and licensing 16
Availability of dynamic modeling 16
Features of dynamic modeling 16
Licensing of dynamic modeling 17
Differences between MODEL and Dynamic Model 18
The refine interface keyword 18
The refine keyword 18
Restrictions of dynamic modeling 19
Preprocessor macros 19
Pragmas 19
User-defined source code 19
ICIM_ManagedDomain 19
Single class refinement 19
MODEL keywords and data types 19
Guidelines for developing dynamic models 20
Requirements for developing dynamic models 21
Development cycle 22
Model enhancement examples 24
Location and loading of dynamic model files 25
Loading dynamic models into a running Manager 25
Loading dynamic models during Manager startup 26

Chapter 2 Exercise 1 – Adding a New Printer Class

Objectives 28
Exercise 29
Creating the dynamic model 30
Compiling the dynamic model 31
Copying files for localization 31
Loading the new class into the IP Manager 32
Verifying the new class in the IP Manager 33
Setting up the printer OID for discovery 34
Determine if the printer is already defined 34
Add the printer OID to the field certification file 35
Reloading the dynamic model 36
Discovering an instance of the SM_Printer class 37
Adding the new class to the Global Manager 38
Compile and load the dynamic model into the Global Manager 38
Add the IP Manager as a source to the Global Manager 39
Verifying the presence of the SM_Printer class instance in the Global Manager. 41

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

Objectives 44

Exercise 44
Creating the dynamic model 45
Compiling the dynamic model 46
Copying files for localization 46
Loading the new attribute into the IP Manager 47
Verifying the new attribute in the IP Manager 48
Discovering an instance of the Router class 49
Populating the value of SM_SerialNumber 50
Adding the new attribute to the Global Manager 50
Compile and load the dynamic model into the Global Manager 50
Add the Router attribute to the IP Manager DXA file 51
Add the IP Manager as a source to the Global Manager 53
Verifying the presence of the SM_SerialNumber attribute in the Global Manager 54

Chapter 4 Exercise 3 - Polling for Values of Additional Attributes

Objectives 56
Exercise 57
Determining the instrumentation class 58
Creating the dynamic model 59
Compiling the dynamic model 61
Copying files for localization 61
Subscribing to the new event in the IP Manager 62
Adding the new threshold to the adapter-setting.asl file 63
Loading the dynamic model into the IP Manager 63
Discovering an instance of the Router class 65
Verifying the new attributes in the IP Manager 65
Verifying the new threshold in the IP Manager 67
Adding a new event subscription to the Global Manager 69
Add the event subscription to the IP Manager DXA file 69
Add the IP Manager as a source to the Global Manager 71
Polling the new attributes 73

Appendix A Guidelines and Syntax for Dynamic Model Files

Guidelines and requirements 76
Adding a class to the model 77
Determining the parent of the new class 77
Adding the new class 78
Updating an existing class in the model 79
refine interface keyword 79
Refining inherited class properties 80
Adding an attribute 80
Adding a relationship 81
Adding an event 81
Adding an aggregate 82
Adding a problem 82
Adding a symptom 83
Adding an operation 83

Appendix B The dynmodel Command Syntax and Options

Syntax and options 86
Using the --l10nsrc and --locales options 88
Pragmas 89

Appendix C Methods of Creating Classes and Populating Attributes

Overview 92
Using dmctl 92
Command-line mode 92
Interactively 93
Batch mode 94
Synchronizing topology changes with the Global Manager 95
Populating attributes by using an EMC Smarts API 95
PERL API 95
ASL 96
Discovery customization 96
Creating the driver 97
Editing custom-end-system.asl 99
Creating the script to parse the value of SM_SerialNumber 100
Restarting the IP Manager 101

Appendix D Updating the Global Manager

Loading the dynamic model enhancements 104
Transferring topology and events 105
Transferring topology 105
Transferring events 106

Appendix E Troubleshooting

Overview 110
Compilation errors 111
License server does not support compiling dynamic model files 111
License server does not support compiling dynamic model files with problems and symptoms 111
Loading errors 112
Missing ICU files 112
Duplicate OID entries 113
Manager cannot restore topology 114
Unable to open ‹ldmfilename› 115
DYN-E-CLASS_NAME_ALREADY_EXIST 115
DYN-E-STRUCT_NAME_ALREADY_EXISTS 115
DYN-E-ENUMERATION_NAME_ALREADY_EXISTS 116
DYN-E-ATTRIBUTE_NAME_ALREADY_EXISTS 116
DYN-E-EVENT_NAME_ALREADY_EXISTS 116
DYN-E-NO_AGGREG_OP 117
Discovery errors 117
Class property changes do not appear in the Global Manager 117
Running the ASL scripts 117

Appendix F MODEL and Dynamic Models

The ICIM Reference 120
Representing systems with MODEL 121
Differences between MODEL and Dynamic Model 122

Index

FIGURES

	Title	Page
1	Steps involved in developing dynamic models	22
2	SM_Printer class in topology of TEST_IP	33
3	Printer discovered as an instance of SM_Printer	37
4	Using the Domain Creation Wizard to create the TEST_IP domain for Exercise 1	40
5	SM_Printer class in the TEST_SA Global Manager	41
6	Map view of the SM_Printer instance	42
7	Serial number attribute added to the Router class in TEST_IP	48
8	Router discovered with the SM_SerialNumber attribute in TEST_IP	49
9	SM_SerialNumber in the TEST_SA Global Manager	54
10	Attributes tab with the new attributes before polling	66
11	TS_OverrunThreshold attribute in the Thresholds tab	67
12	Enabled subinterface performance analysis	68
13	Using the Domain Creation Wizard to create the TEST_IP domain for Exercise 3	72
14	Attributes tab with the new attributes after polling	73
15	Using the discovery postprocessing mechanism	97
16	Modifying custom-end-system.asl	99
17	Simplified MODEL and Dynamic Model methodology comparison	123

TABLES

	Title	Page
1	Differences between MODEL and Dynamic Model	18
2	Command line options for the dynmodel command	86
3	Files and directories generated by the dynmodel compiler	87
4	Pragmas used in dynamic modeling	89

PREFACE

As part of an effort to improve its product lines, EMC periodically releases revisions of its software and hardware. Therefore, some functions described in this document might not be supported by all versions of the software or hardware currently in use. The product release notes provide the most up-to-date information on product features.

Contact your EMC technical support professional if a product does not function properly or does not function as described in this document.

NOTICE

This document was accurate at publication time. Go to EMC Online Support (https://support.emc.com) to ensure that you are using the latest version of this document.

Purpose

This tutorial introduces EMC® Smarts® users to the concepts and methods of dynamic modeling.

Audience

This document is intended for those users who want to extend the EMC Smarts ICIM Common Information Model™ (ICIM), but for whom a C++ compiler is not available for model development. The ICIM can be extended by either:

- Augmenting existing ICIM classes
- Defining new classes

In addition, dynamic modeling is useful for extending an existing application, which is not possible even with a C++ compiler.

Prerequisites

It is beneficial to have an understanding of the basic EMC Smarts platform operations, functionality, and architecture, including ASL and Perl scripting languages. Courses for these scripting languages are available.

In particular, it is useful to have a fundamental understanding of the ICIM and how the repository is populated and instances are updated.

This guide focuses on dynamic modeling, which is an extension of Managed Object Definition Language (MODEL). The *EMC Smarts MODEL Reference Guide* should be used together with this document to understand basic modeling concepts and mechanisms.

You must have **admin** privileges to perform the exercises in this tutorial.

EMC Smarts installation directory

In this document, the term BASEDIR represents the location where EMC Smarts software is installed.

- For Windows, this location is C:\InCharge\‹*product*›.
- For UNIX, this location is /opt/InCharge/‹*product*›.

Optionally, you can specify the root of BASEDIR to be something other than:

- Windows: C:\InCharge
- UNIX: /opt/InCharge

However, you cannot change the ‹*product*› location under the root directory.

The *EMC Smarts System Administration Guide* provides more information about the directory structure.

Related documentation

In addition to this document, EMC Corporation provides a Help system for command line programs as well as product documentation.

Help for command line programs

Descriptions of command line programs are available as HTML pages. The index.html file, which provides an index to the various commands, is located in the BASEDIR/smarts/doc/html/usage directory.

EMC Smarts documentation

Readers of this document might find the following related documentation helpful. These documents are updated periodically. Electronic versions of the updated manuals are available on EMC Online Support:

- *EMC Smarts System Administration Guide*
- *EMC Smarts Common Information Model (ICIM) Reference for Service Assurance Manager*
- *EMC Smarts Common Information Model (ICIM) 1.11 Reference for Non-Service Assurance Manager Products*
- *EMC Smarts ASL Reference Guide*
- *EMC Smarts Perl Reference Guide*
- *EMC Smarts MODEL Reference Guide*
- *Smarts Foundation EMC Data Access API (EDAA) Programmer Guide*
- *FLEXnet Licensing End User Guide*
- *EMC Smarts Foundation Documentation Portfolio*

Conventions used in this document

EMC uses the following conventions for special notices:

NOTICE

NOTICE is used to address practices not related to personal injury.

Typographical conventions

EMC uses the following type style conventions in this document:

Bold	Use for names of interface elements, such as names of windows, dialog boxes, buttons, fields, tab names, key names, and menu paths (what the user specifically selects or clicks)
Italic	Use for full titles of publications referenced in text
`Monospace`	Use for: • System output, such as an error message or script • System code • Pathnames, filenames, prompts, and syntax • Commands and options
`Monospace italic`	Use for variables.
`Monospace bold`	Use for user input.
[ ]	Square brackets enclose optional values
|	Vertical bar indicates alternate selections — the bar means “or”
{ }	Braces enclose content that the user must specify, such as x or y or z
...	Ellipses indicate nonessential information omitted from the example

Where to get help

EMC support, product, and licensing information can be obtained as follows:

Product information — For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

https://support.emc.com

Technical support — Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

Your comments

Your suggestions will help us continue to improve the accuracy, organization, and overall quality of the user publications. Send your opinions of this document to:

`techpubcomments@emc.com`

CHAPTER 1
Introduction

This chapter consists of the following sections:

- Terminology 16
- Availability, features, and licensing 16
- Differences between MODEL and Dynamic Model 18
- Restrictions of dynamic modeling 19
- Guidelines for developing dynamic models 20
- Requirements for developing dynamic models 21
- Development cycle 22
- Model enhancement examples 24
- Location and loading of dynamic model files 25

Terminology

Throughout this chapter and the chapters that follow, the term Manager is used to refer to an EMC® Smarts® IP Manager or Global Manager. The term IP Manager is used to refer to any of the following EMC Smarts Domain Managers:

- EMC Smarts IP Availability Manager
- EMC Smarts IP Performance Manager
- EMC Smarts IP Availability Manager Extension for NAS

Availability, features, and licensing

Dynamic Model is an extension of the EMC Smarts Managed Object Definition Language (MODEL). MODEL is integral to the EMC Smarts application development process. Its purpose is to render the schema and semantics of the *EMC Smarts ICIM Reference* specification. These libraries are loaded by Managers and are operated on by EMC Smarts applications. For an EMC Smarts 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 Manager.

Availability of dynamic modeling

Dynamic modeling is supported on IP Manager s and Global Managers. All dynamic model examples in this tutorial pertain to the IP Manager and the Global Manager.

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 Manager is started or after the 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 *EMC Smarts 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, with a separate license, by defining new problems and symptoms.
- You can refine relationships, events, aggregates, operations, and attributes that are inherited from an existing ICIM class.
- You can refine problems and symptoms in an existing ICIM class with a separate license.

Licensing of dynamic modeling

Two licenses are required to run Dynamic Modeling:

- SMA-SDM-C

  A license to run the dynmodel compiler and generate .ldm files (the version of dynamic modeling files that can be loaded into a Manager). This license also allows you to load the .ldm files into a Manager.

- SMA-SDM-F

  A license to run the dynmodel compiler on dynamic model files that have refined classes by adding new problems and symptoms.

The *EMC Smarts System Administration Guide* contains information concerning license administration.

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 *EMC Smarts MODEL Reference Guide* is useful when working with dynamic models.

Table 1 on page 18 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 Differences between MODEL and Dynamic Model

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

Preprocessor macros

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

Pragmas

Only those pragmas that are listed in Table 4 on page 89, 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

The **binary** and **byte** MODEL keywords and data types can be used in any dynamic model file on any Manager. The binary data type represents binary data, and the byte data type represents an unsigned 8-bit integer ranging from 0 - 255.

The **external** MODEL keyword cannot be used in any dynamic model file on any Manager.

Guidelines for developing dynamic models

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 the sm_edit utility.

  NOTICE

  The .mdl files are created and edited by using sm_edit. The .ldm files that are 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 Manager is already running. "Location and loading of dynamic model files" on page 25 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 prefix is already used by EMC Smarts.
- Do not use the **external** keyword in any dynamic model file on any Manager.

Changes made by using dynamic modeling might impact the performance and load of the Manager.

Requirements for developing dynamic models

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" on page 77 and "Updating an existing class in the model" on page 79.
- 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/local/model or BASEDIR/smarts/local/model/‹*config*›. The ‹*config*› refers to the configuration directory that is specified on the command line when the Manager is started. "Loading dynamic models during Manager startup" on page 26 provides further details.

  NOTICE

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

- Dynamic model files (those with extension .ldm) must be loaded into each Manager that requires knowledge of the new topology and events.
- For Foundation 9.0- or 9.1-based Managers, dynamic model string files (‹*modelname*›_mmbe.dat, ‹*modelname*›_mmle.dat) must be copied to the SM_SITEMOD/l10n directory *even if you do no localization*.

  Note that:

  - SM_SITEMOD is an environment variable that is set to BASEDIR/smarts/local by default.
  - l10n stands for localization ("l" followed by 10 letters followed by "n") and represents country-specific settings such as the representation of numbers, dates, and money.

Development cycle

Several steps are involved in the development cycle of extending a model of a Manager, by using dynamic modeling. Figure 1 on page 22 illustrates the development cycle.

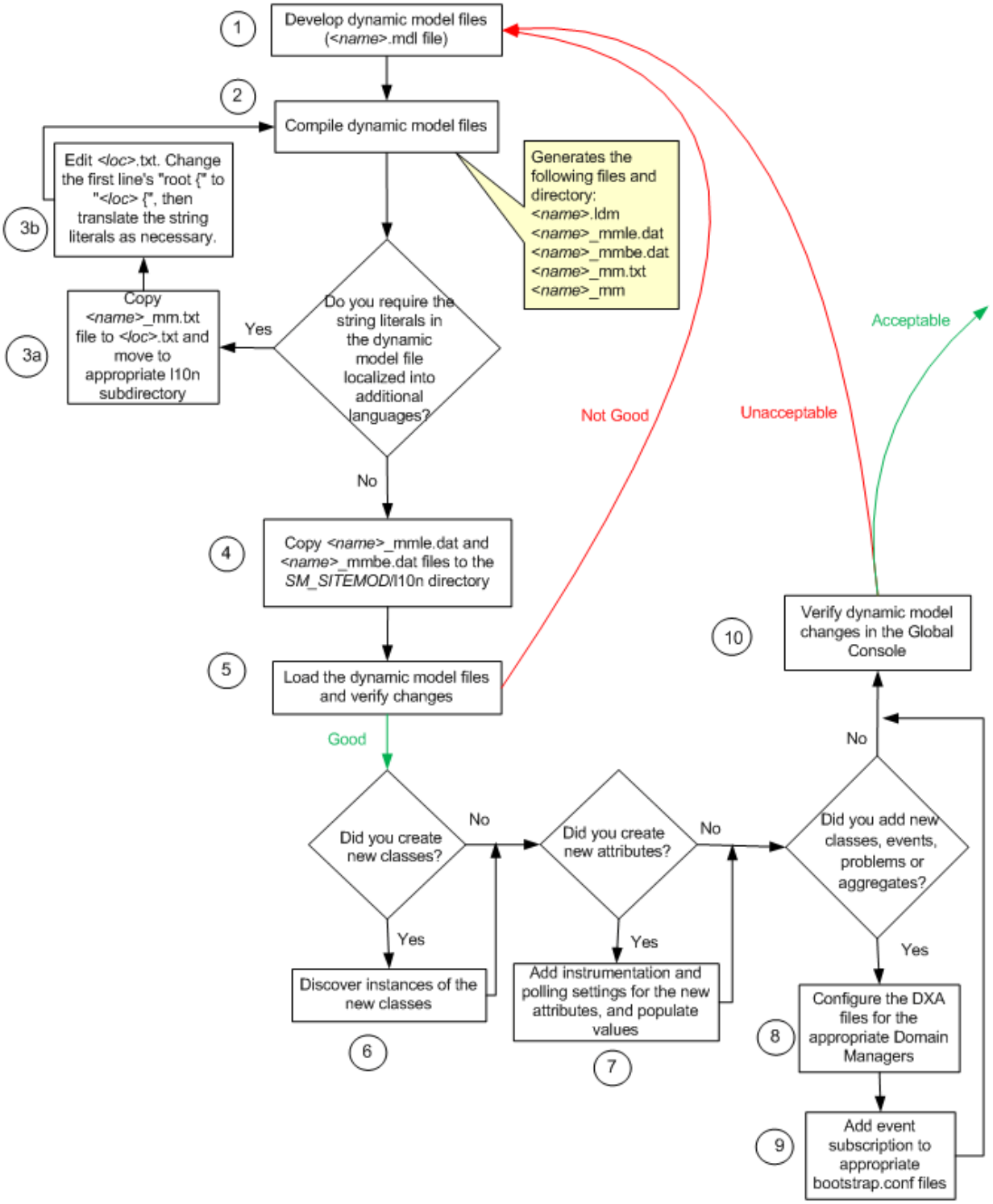

Figure 1 Steps involved in developing dynamic models

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. Appendix A, "Guidelines and Syntax for Dynamic Model Files," contains additional information.
2. Compile the dynamic model text file into a file (with extension .ldm) that can be loaded into a Manager. The command used to compile the dynamic model file is described in Appendix B, "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:
   a. 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.
   b. Edit *‹loc›*.txt. Change the "root {" to "*‹loc›* {" in the first line. Translate the string literals as required. Return to step 2 on page 23 .
4. From the BASEDIR/smarts/local/model directory, copy the following files to the SM_SITEMOD/l10n directory:
   - Example1_mmbe.dat
   - Example1_mmle.dat

   By default, SM_SITEMOD is set to BASEDIR/smarts/local.
5. Load the compiled dynamic model file into a 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 23 and make the appropriate modifications to your .mdl file. "Loading errors" on page 112 provides additional troubleshooting information.
6. 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.
7. 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.
8. If you want to view the new classes, attributes, problems, aggregates or events in the Manager, configure the appropriate data exchange adapter (DXA) configuration file. Appendix D, "Updating the Global Manager," describes DXA configuration.
9. If you added new events, add a subscription to the event in the appropriate bootstrap file.
10. Verify the topology in the Global Console.

    If you encounter errors, go back to step 1 on page 23 and make the appropriate modifications. Appendix E, "Troubleshooting," provides additional information.

Model enhancement examples

This tutorial presents three different exercises:

- "Exercise 1 – Adding a New Printer Class" on page 27, adds a new printer class to the model.
- "Exercise 2 - Adding an Attribute to the Router Class" on page 43, adds a new attribute to an existing ICIM class.
- "Exercise 3 - Polling for Values of Additional Attributes" on page 55, 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

The exercises are performed on test versions of an IP Manager named TEST_IP and a Global Manager named TEST_SA.

In Exercises 1 and 2, the test IP Manager is started with the bootstrap-am.conf file. Starting the test IP Manager with bootstrap-am.conf file causes the IP Manager to operate as an IP Availability Manager.

In Exercise 3, the test IP Manager is started with the bootstrap-am-pm.conf file. Starting the test IP Manager with bootstrap-am-pm.conf file causes the IP Manager to operate as an IP Availability Manager and an IP Performance Manager running as a single process (AM-PM).

NOTICE

The --config option (or just -c) in an IP Manager startup command points to the bootstrap files in the icf directory. The default bootstrap file is bootstrap.conf, which starts an IP Manager as an "AVAILABILITY_AND_PERFORMANCE_MANAGER" (AM-PM). The addition of the --bootstrap option in an IP Manager startup command points to a particular bootstrap file in the icf directory, such as bootstrap-am.conf, which starts an IP Manager as an "AVAILABILITY_MANAGER" (AM), or bootstrap-am-pm.conf, which also starts an IP Manager as an "AVAILABILITY_AND_PERFORMANCE_MANAGER" (AM-PM).

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 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 that is used by **sm_service** to include the appropriate option to load your dynamic models. The *EMC Smarts 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/local/model directory. The **dmctl** utility is used to invoke the loadModel operation. "Using dmctl" on page 92 provides additional information. The syntax for the command line is as follows:

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

where:

- *‹managername›* is the name of the Manager.
- (Optional) *‹broker›* is the name of the Broker to which the Manager has been registered. The default Broker location 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 TEST Manager that is attached to the Broker on localhost:426.

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

NOTICE

Do not include the .ldm file extension.

Verify that the model file was loaded. If you started your 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 --dynamic option, all of the .ldm files in the BASEDIR/smarts/local/model/‹*config*› directory will be loaded, where ‹*config*› refers to the configuration directory that is specified on the command line when the Manager is started.

In the following example, a Global Manager named TEST_SA is started *without* the --dynamic option:

```
sm_server -n TEST_SA -c ics --output
```

This example loads all of the .ldm files in the BASEDIR/smarts/local/model/ics directory because *ics* is the configuration directory that is specified in the command line.

Starting a Manager with the --dynamic option

If you start a Manager with the --dynamic option, all of the .ldm files in the following directories will be loaded:

- BASEDIR/smarts/local/model
- BASEDIR/smarts/local/model/‹*config*›

where ‹*config*› refers to the configuration directory that is specified on the command line when the Manager is started.

In the following example, an IP Manager is started *with* the --dynamic option:

```
sm_server -n TEST_IP -c icf --output --dynamic
```

This example loads all of the .ldm files in the BASEDIR/smarts/local/model directory and in the BASEDIR/smarts/local/model/icf directory because *dynamic* and *icf* are specified in the command line.

CHAPTER 2
Exercise 1 – Adding a New Printer Class

This chapter consists of the following sections:

- Objectives 28
- Exercise 29
- Creating the dynamic model 30
- Compiling the dynamic model 31
- Copying files for localization 31
- Loading the new class into the IP Manager 32
- Verifying the new class in the IP Manager 33
- Setting up the printer OID for discovery 34
- Reloading the dynamic model 36
- Discovering an instance of the SM_Printer class 37
- Adding the new class to the Global Manager 38
- Verifying the presence of the SM_Printer class instance in the Global Manager 41

Objectives

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

- IP 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 IP Manager is started.

NOTICE

This exercise should be performed on test versions of a Broker, IP 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 the IP Manager.
- Use the Global Console to view the new printer class.
- Set up the printer OID for discovery.
- Reload the dynamic model file into the IP Manager.
- Use the IP Manager to discover instances of the new class.
- Add the new printer class to the Global Manager.

Exercise

After your network has been discovered, you might 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 that 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 toward the system volume license. When your dynamic model classes inherit from ICIM_UnitaryComputerSystem, instances of that class will take advantage and participate on the full predefined IP Manager root cause analysis model, and therefore such instances will be counted towards the system volume license.

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 configuration file 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” on page 78 provides more detail.

NOTICE

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 prefix is already used by EMC Smarts.

Creating the dynamic model

In this example, an instance of a compiled dynamic model file named Example1.ldm is created in the BASEDIR/smarts/local/model/‹*conf*› directory of the IP Manager installation area. The IP Manager 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. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to create a new model file named Example1.mdl:

   ```
   sm_edit --create model/Example1.mdl
   ```

2. To declare the new printer class as a subclass of the UnitaryComputerSystem class, add the following lines to the Example1.mdl file:

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

3. Save and close the file.

   The newly created Example1.mdl file is saved to the BASEDIR/smarts/local/model directory.

Compiling the dynamic model

To compile the Example1.mdl model file:

1. Enter the following command to change to the BASEDIR/smarts/local/model directory in the IP Manager installation area:

   cd *BASEDIR*/**smarts/local/model**

2. From the BASEDIR/smarts/local/model directory, enter the following command to compile Example1.mdl:

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

If the compilation 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/local/model directory:

- Example1_mm

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

Copying files for localization

From the BASEDIR/smarts/local/model directory in the IP Manager installation area, copy the following files to the SM_SITEMOD/l10n directory:

- Example1_mmbe.dat
- Example1_mmle.dat

By default, SM_SITEMOD is set to BASEDIR/smarts/local.

Loading the new class into the IP Manager

Once the dynamic model has been compiled, you must load it into the IP Manager. The Printer class will appear in the topology of the IP Manager.

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

If the test IP Manager is already running, stop it.

To load the compiled dynamic model file into the test IP Manager:

Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to start the test IP Manager and load the compiled dynamic model file:

```
sm_server -n TEST_IP -c icf --broker=localhost:426
   --bootstrap=bootstrap-am.conf --output --dynamic
```

The startup code uses bootstrap-am.conf as the bootstrap file for the test IP Manager. The startup code also registers the test IP Manager as TEST_IP with the Broker that is running on port 426 of the local host. Substitute the bootstrap file and the Broker location with your bootstrap file and Broker location. As an example, you could substitute bootstrap-am.conf with bootstrap-am-pm.conf to start the IP Manager as an IP Availability Manager and an IP Performance Manager running as a single process (AM-PM).

In addition, all messages will be logged to a log file, such as BASEDIR/smarts/local/logs/TEST_IP_en_US_UTF-8.log.

To check that the compiled dynamic model file was loaded:

Go to the BASEDIR/smarts/local/logs directory and use any text editor to open the server 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 the new class in the IP 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 the test IP Manager.

To verify the presence of the new class:

1. Attach the Global Console to the test IP Manager (TEST_IP) and open a Topology Browser Console view of the Global Console.

   The *EMC Smarts Service Assurance Manager Operator Guide* provides information about starting the Global Console, logging in, and attaching to an EMC Smarts application. The default administrative account is username *admin* and password *changeme*.

2. In the topology tree panel of the Topology Browser Console, right-click an open space, and select **Select** from the pop-up menu to display the **Select tree content** dialog box.

3. In the dialog box, 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 Figure 2 on page 33.

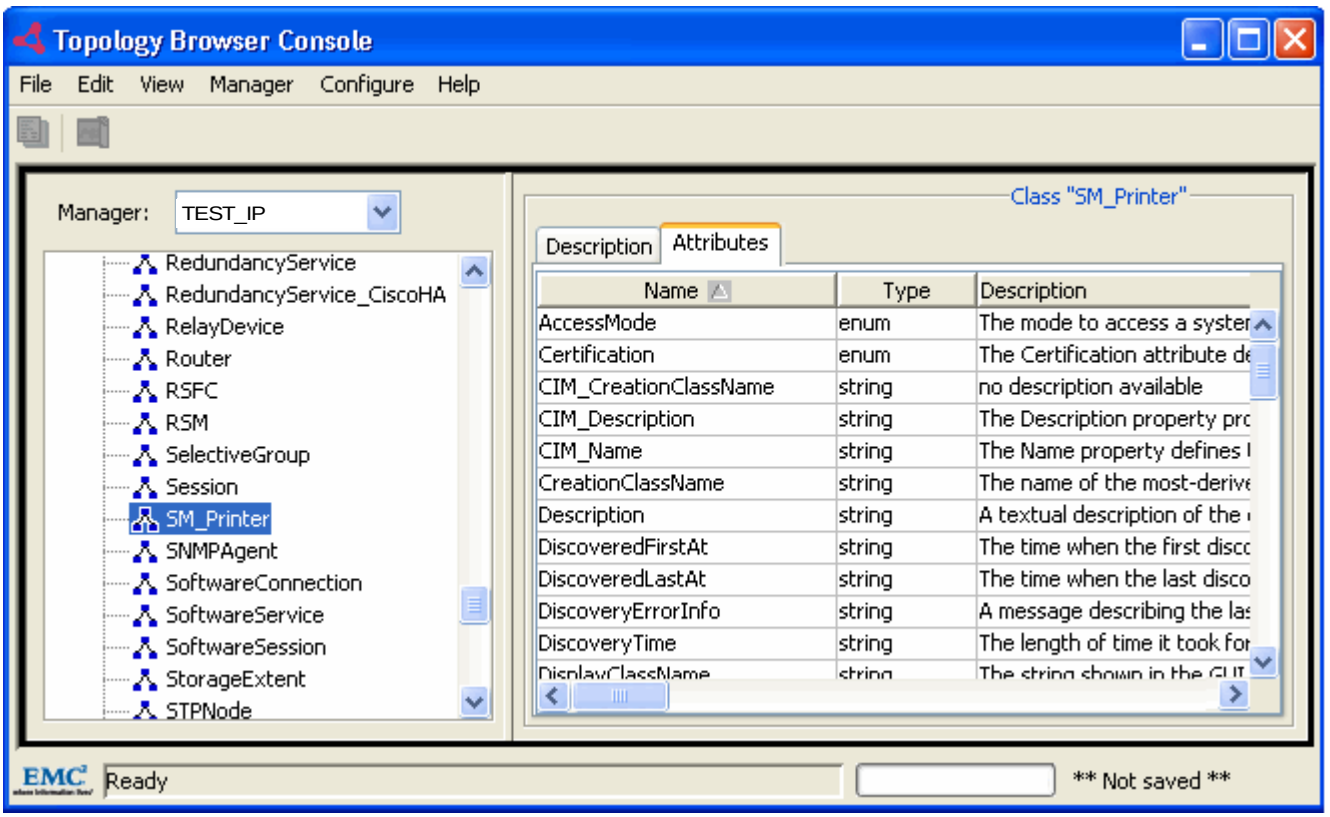

Figure 2 SM_Printer class in topology of TEST_IP

NOTICE

You can also use **dmctl** to verify whether the class is present in the topology. "Command-line mode" on page 92 provides further details.

Setting up the printer OID for discovery

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

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

Determine if the printer is already defined

You need to determine if the printer is already defined generically (TYPE = Host), in one of the existing EMC Smarts 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 the sm_edit utility 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. Go to the BASEDIR/smarts/bin directory and enter the following command to open the oid2type_Misc.conf file:

   ```
   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 modified version of the oid2type_Misc.conf file is saved to the BASEDIR/smarts/local/conf/discovery directory.

Add the printer OID to the field certification file

To add the printer OID to the field certification file:

1. Go to the BASEDIR/smarts/bin directory and enter the following command to open the oid2type_Field.conf file:

   sm_edit conf/discovery/oid2type_Field.conf

2. Add the following lines at the bottom of the file:

```
.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
}
```

   NOTICE

   The OID that is used in this example is for a particular model XEROX printer. Substitute this OID with an OID of a printer on your network. Modify the VENDOR and MODEL fields as required. The CERTIFICATION and INSTRUMENTATION fields *must* be entered as shown in this example. The IP Manager user documents provides additional information about the oid2type_Field.conf file.

   These lines create an instance of the SM_Printer class each time that 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.

3. Save and close the file.

   The modified version of the oid2type_Field.conf 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 Manager is first started, TEST_IP must be stopped and restarted for any changes to take effect. The new class is loaded into the IP Manager.

The IP Manager is started with the --dynamic command line option to pull the compiled dynamic model file from BASEDIR/smarts/local/model.

To reload the compiled dynamic model file into the test IP Manager:

1. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to stop the test IP Manager:

   ```
   dmquit --force --server=TEST_IP
   ```

2. Enter the following command to start the test IP Manager and reload the compiled dynamic model file:

   ```
   sm_server -n TEST_IP -c icf --broker=localhost:426
      --bootstrap=bootstrap-am.conf --output --dynamic
   ```

   The startup code uses bootstrap-am.conf as the bootstrap file for the test IP Manager. The startup code also registers the test IP Manager as TEST_IP with the Broker that is running on port 426 of the local host. Substitute the bootstrap file and the Broker location with your bootstrap file and Broker location.

Discovering an instance of the SM_Printer class

To discover a printer instance:

1. In the Topology Browser Console that is attached to the test IP Manager (TEST_IP), select **Configure › Domain Manager Administration Console** to launch a Domain Manager Administration Console.
2. In the Domain Manager Administration Console, initiate a discovery.

   One method is to select **Topology › Add Agent** to display the **Add Agent** dialog box. In the dialog box, specify either the hostname or IP address of the printer.

   The *EMC Smarts IP Manager Concepts Guide* and the *EMC Smarts IP Manager User Guide* describe 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 Figure 3 on page 37.

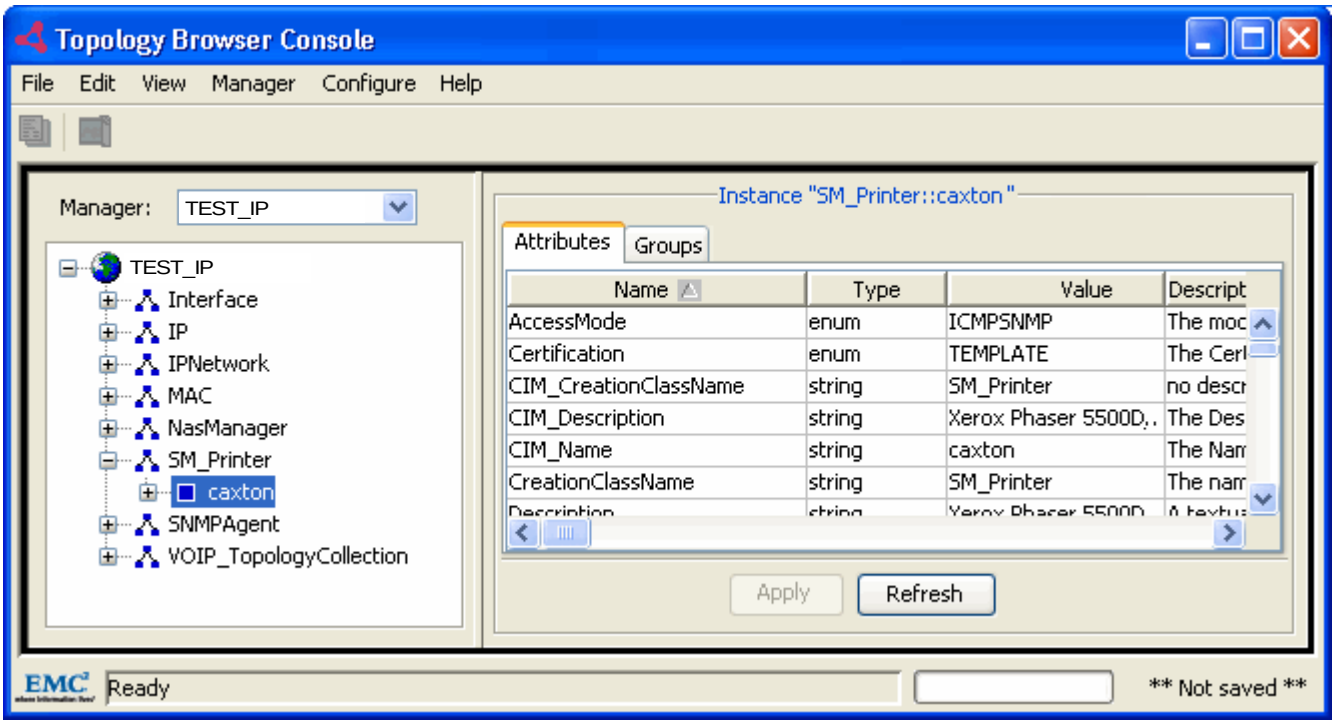

Figure 3 Printer discovered as an instance of SM_Printer

Adding the new class to the Global Manager

The new SM_Printer class has been loaded into the IP Manager but not into the Global Manager. You complete the following tasks to add the new class to the Global Manager and to enable the Global Manager to import the new class from the IP Manager:

- Compile and load the dynamic model into the Global Manager.
- Add the IP Manager as a source to the Global Manager.

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

Compile and load the dynamic model into the Global Manager

The dynamic model enhancements must be compiled and 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 file into the Global Manager:

1. Copy Example1.mdl from the IP Manager's BASEDIR/smarts/local/model directory to the Service Assurance Manager's BASEDIR/smarts/local/model directory.
2. Enter the following command to change to the BASEDIR/smarts/local/model directory in the Service Assurance Manager installation area:

   cd *BASEDIR*/**smarts/local/model**

3. From the BASEDIR/smarts/local/model directory, enter the following command to compile Example1.mdl:

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

   If the compilation 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/local/model directory:

   - Example1_mm

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

4. From the BASEDIR/smarts/local/model directory in the Service Assurance Manager installation area, copy the following files to the SM_SITEMOD/l10n directory:

   - Example1_mmbe.dat
   - Example1_mmle.dat

   By default, SM_SITEMOD is set to BASEDIR/smarts/local.

5. Go to the BASEDIR/smarts/bin directory in the Service Assurance Manager installation area and enter the following command to start the test Global Manager and load the Example1.ldm file:

   ```
   sm_server -n TEST_SA -c ics --broker=localhost:426 --output
      --dynamic
   ```

   The startup code registers the test Global Manager as TEST_SA with the Broker that is running on port 426 of the local host. Substitute the Broker location with your Broker location.

Add the IP Manager as a source to the Global Manager

If you started your test IP Manager (TEST_IP) as an IP Availability Manager (AM), perform the procedure in this section. If you started your test IP Manager as an IP Availability Manager and an IP Performance Manager running as a single process (AM-PM), go to "Add the event subscription to the IP Manager DXA file" on page 69 for more information.

To add the IP Manager as a source to the test Global Manager:

1. In the Domain Manager Administration Console that is attached to the test IP Manager (TEST_IP), select **File › New › Topology Browser Console** to launch a Topology Browser Console.
2. In the Topology Browser Console, select **Manager › Attach** to launch the **Attach Manager** dialog box.
3. In the dialog box, enter the test Global Manager name (**TEST_SA**) and the login information (for example, *admin* and *changeme*) and click **OK**.
4. In the Topology Browser Console, select **TEST_SA** from the **Manager** drop-down menu.
5. In the Topology Browser Console, select **Configure › Global Manager Administration Console** to launch a Global Manager Administration Console.
6. In the left panel of the Global Manager Administration Console, expand the configuration tree to display the **Domains**.
7. Right-click **Domains** and then click the pop-up menu to launch the **Domain Creation Wizard**.
8. Consult Figure 4 on page 40 and follow the directions to create an IP Manager domain named **TEST_IP**. (Type **TEST_IP** in the **Domain Name** field and select **INCHARGE-AM** in the **Copy Existing** drop-down list.)

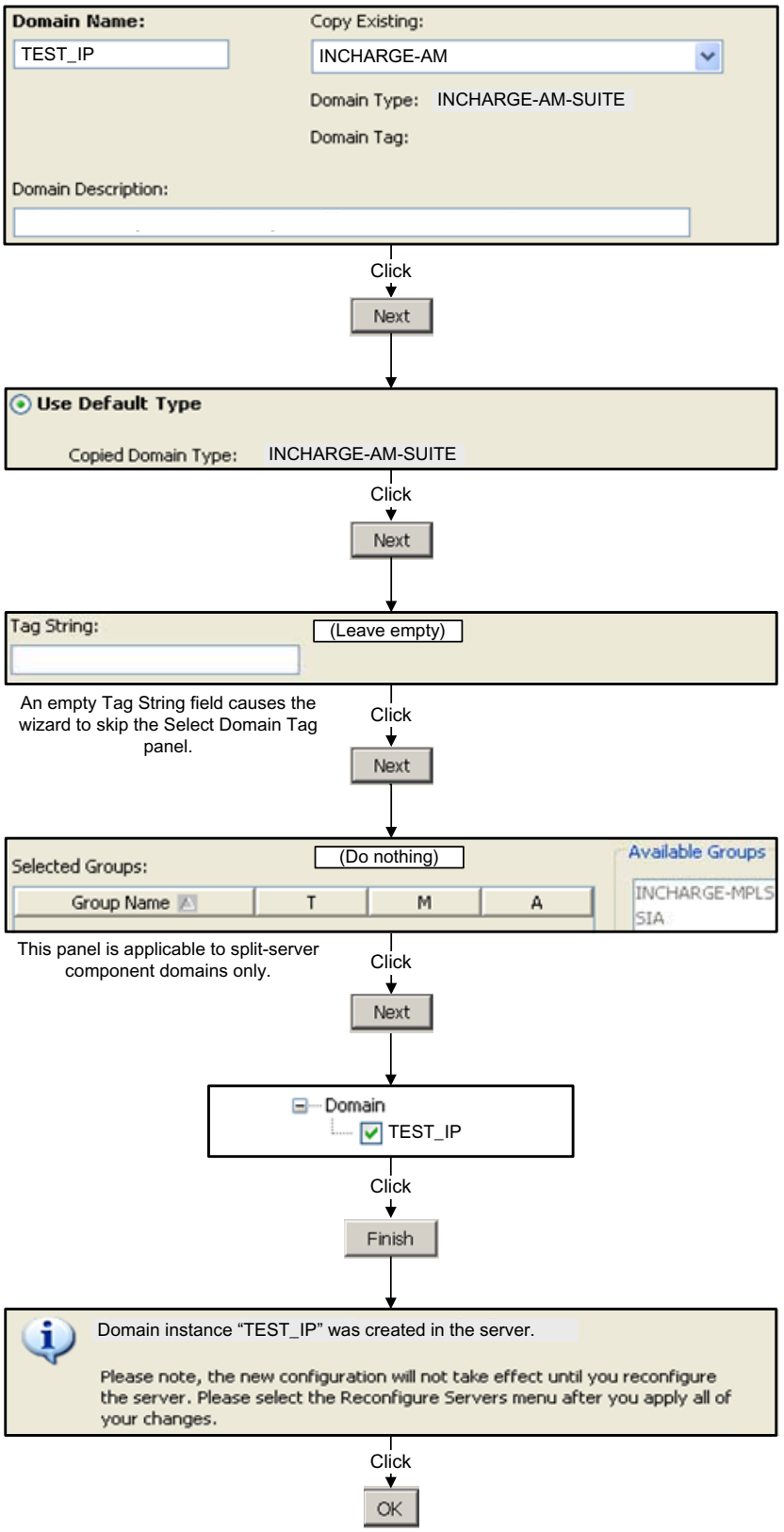

Figure 4 Using the Domain Creation Wizard to create the TEST_IP domain for Exercise 1

9. In the toolbar area of the Global Manager Administration Console, click the **Reconfigure** toolbar button ( ).
10. In the **Server Reconfiguration** confirmation dialog box that appears, click **Yes**.
11. In the **Server Reconfiguration** information dialog box that appears, click **Close**.

In response, the Global Manager:

- Creates an InChargeDomain object named **TEST_IP** and starts the appropriate topology and event drivers.
- Consults the Broker for the deployment and establishes a connection to the **TEST_IP** domain.
- Subscribes to and imports topology and events from the **TEST_IP** domain.

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

To verify the presence of the SM_Printer class in the test Global Manager:

1. In the Global Manager Administration Console that is attached to the test Global Manager (TEST_SA), select **File › New › Topology Browser Console** to launch a Topology Browser Console.
2. In the Topology Browser Console, verify that the SM_Printer class has been added, as shown in Figure 5 on page 41.

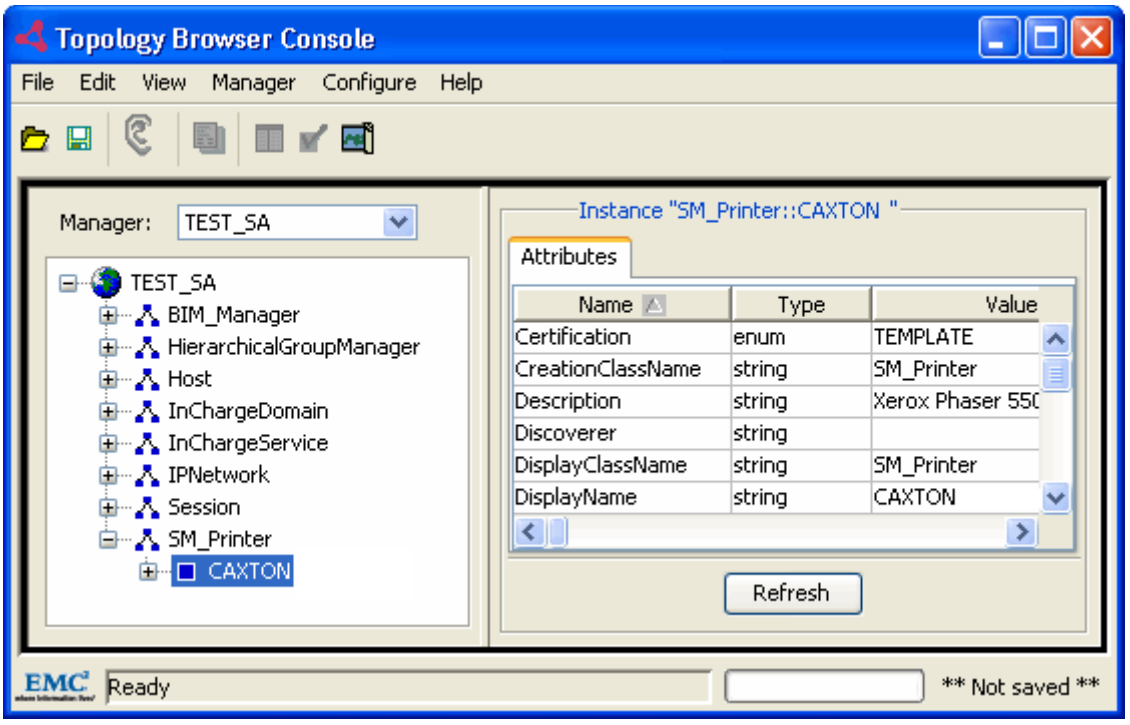

Figure 5 SM_Printer class in the TEST_SA Global Manager

3. Right-click the printer instance (**CAXTON** in this procedure) and select **Show Map** from the pop-up menu. The printer instance is displayed in a Map Console, as shown in Figure 6 on page 42.

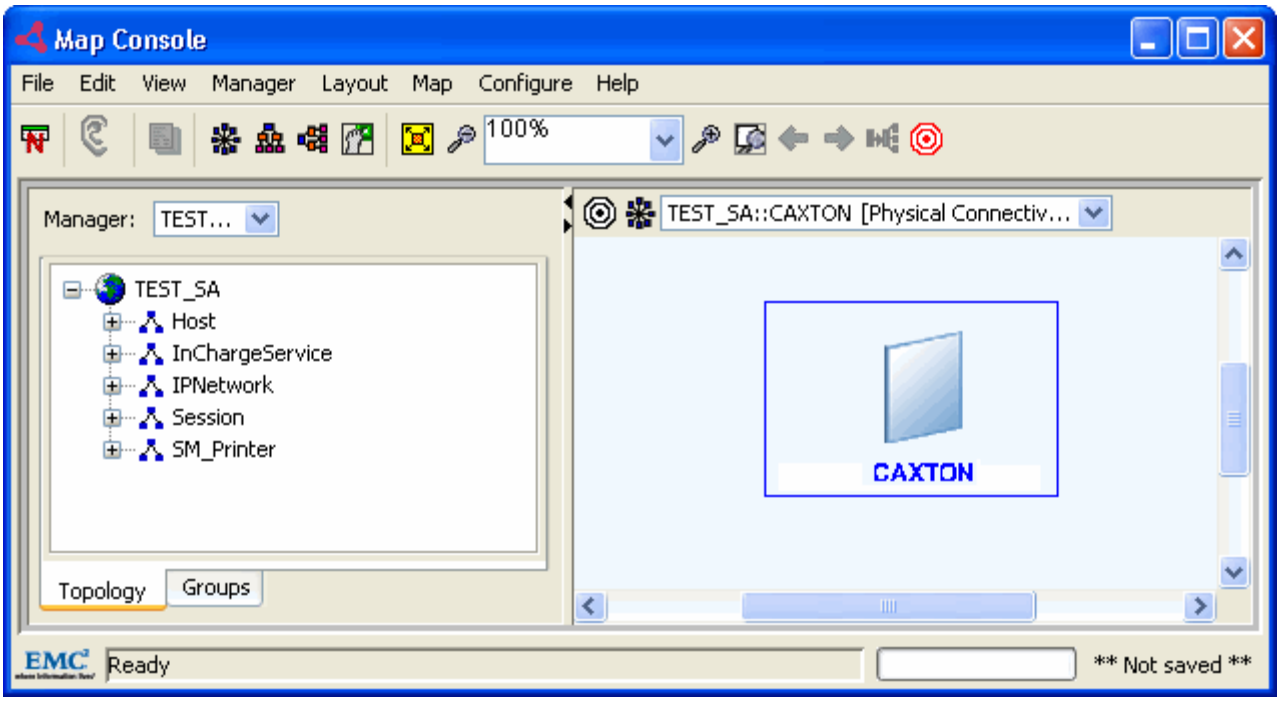

Figure 6 Map view of the SM_Printer instance

4. If you have a custom icon to represent a printer, you can modify the icon that is used to display an instance of SM_Printer. The *EMC Smarts Service Assurance Manager Configuration Guide* has more information on custom icons for map nodes.

CHAPTER 3

Exercise 2 - Adding an Attribute to the Router Class

This chapter consists of the following sections:

- Objectives 44
- Exercise 44
- Creating the dynamic model 45
- Compiling the dynamic model 46
- Copying files for localization 46
- Loading the new attribute into the IP Manager 47
- Verifying the new attribute in the IP Manager 48
- Discovering an instance of the Router class 49
- Populating the value of SM_SerialNumber 50
- Adding the new attribute to the Global Manager 50
- Verifying the presence of the SM_SerialNumber attribute in the Global Manager 54

Objectives

This exercise describes how to use Dynamic Model to add a new attribute to an existing ICIM class in the following Managers:

- IP Manager
- Global Manager

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

NOTICE

This exercise should be performed on test versions of a Broker, IP 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 IP Manager.
- Use the Global Console to view the new Router attribute.
- Use the IP Manager to discover an instance of the Router class.
- Add 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:

- IP Manager
- Global Manager

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

NOTICE

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 prefix is already used by EMC Smarts.

Creating the dynamic model

In this example, an instance of a compiled dynamic model file named Example2.ldm is created in the BASEDIR/smarts/local/model/‹*conf*› directory of the IP Manager installation area. The IP Manager 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. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to create a new model file named Example2.mdl:

   ```
   sm_edit --create model/Example2.mdl
   ```

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

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

3. Save and close the file.

   The newly created Example2.mdl file is saved to the BASEDIR/smarts/local/model directory.

Compiling the dynamic model

To compile the Example2.mdl model file:

1. Enter the following command to change to the BASEDIR/smarts/local/model directory in the IP Manager installation area:

   cd *BASEDIR*/**smarts/local/model**

2. From the BASEDIR/smarts/local/model directory, enter the following command to compile Example2.mdl:

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

If the compilation 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 in the IP Manager installation area, copy the following files to the SM_SITEMOD/l10n directory:

- Example2_mmbe.dat
- Example2_mmle.dat

By default, SM_SITEMOD is set to BASEDIR/smarts/local.

Loading the new attribute into the IP Manager

Once the dynamic model has been compiled, you must load it into the IP Manager. The Router class appears in the topology of the IP Manager.

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

If the test IP Manager is already running, stop it.

To load the compiled dynamic model file into the test IP Manager:

Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to start the test IP Manager and load the compiled dynamic model file:

```
sm_server -n TEST_IP -c icf --broker=localhost:426
   --bootstrap=bootstrap-am.conf --output --dynamic
```

The startup code uses bootstrap-am.conf as the bootstrap file for the test IP Manager. The startup code also registers the test IP Manager as TEST_IP with the Broker that is running on port 426 of the local host. Substitute the bootstrap file and the Broker location with your bootstrap file and Broker location. As an example, you could substitute bootstrap-am.conf with bootstrap-am-pm.conf to start the IP Manager as an IP Availability Manager and an IP Performance Manager running as a single process (AM-PM).

In addition, all messages will be logged to a log file, such as BASEDIR/smarts/local/logs/TEST_IP_en_US_UTF-8.log.

To check that the compiled dynamic model file was loaded:

Go to the BASEDIR/smarts/local/logs directory and use any text editor to open the server 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 the new attribute in the IP 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 the test IP Manager.

To view the presence of the new attribute:

1. Attach the Global Console to the test IP Manager (TEST_IP) and open a Topology Browser Console view of the Global Console.

   The *EMC Smarts Service Assurance Manager Operator Guide* provides information about starting the Global Console, logging in, and attaching to an EMC Smarts application. The default administrative account is username *admin* and password *changeme*.

2. In the topology tree panel of the Topology Browser Console, right-click an open space, and select **Select** from the pop-up menu to display the **Select tree content** dialog box.

3. In the dialog box, select **Show classes with no instances in current domain** and click **OK**.

4. In the topology tree panel of the Topology Browser Console, select the **Router** class to display the attributes in the **Attributes** tab.

   The SM_SerialNumber attribute appears, as shown in Figure 7 on page 48.

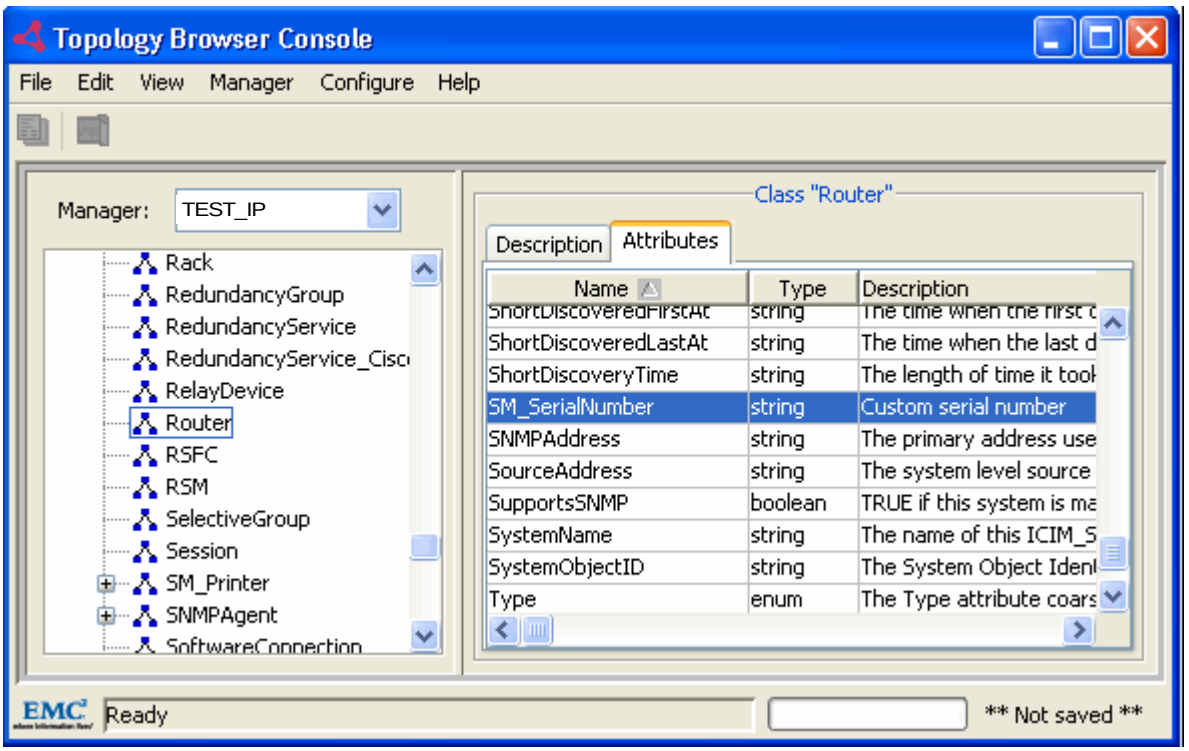

Figure 7 Serial number attribute added to the Router class in TEST_IP

Discovering an instance of the Router class

To discover a Router instance:

1. In the Topology Browser Console that is attached to the test IP Manager (TEST_IP), select **Configure › Domain Manager Administration Console** to launch a Domain Manager Administration Console.
2. In the Domain Manager Administration Console, initiate a discovery.

   One method is to select **Topology › Add Agent** to display the **Add Agent** dialog box. In the dialog box, specify either the hostname or IP address of the router.

   The *EMC Smarts IP Manager Concepts Guide* and the *EMC Smarts IP Manager User Guide* describe various methods of discovery.
3. 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 Figure 8 on page 49.

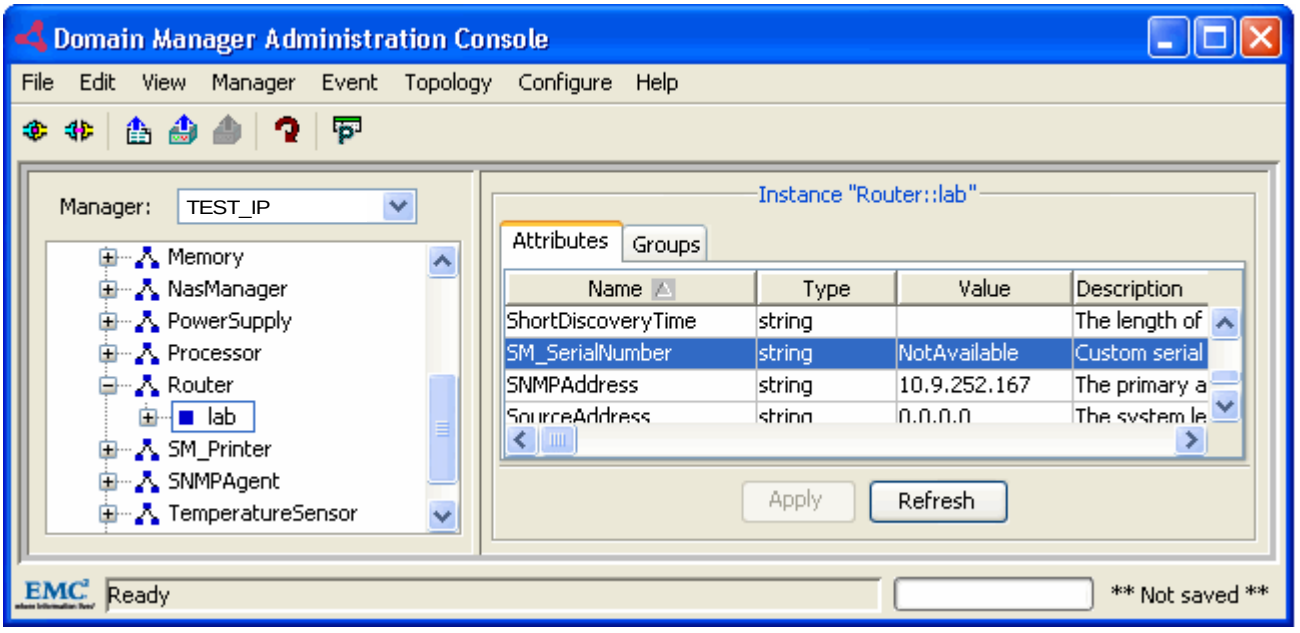

Figure 8 Router discovered with the SM_SerialNumber attribute in TEST_IP

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.

To rediscover the Router instance:

In the Domain Manager Administration Console that is attached to the test IP Manager (TEST_IP), right-click the Router instance and select **Rediscover** from the pop-up menu.

Adding the new attribute to the Global Manager

The new Router attribute has been loaded into the IP Manager but not into the Global Manager. Complete the following tasks to add the new attribute to the Global Manager and to enable the Global Manager to import the new attribute from the IP Manager:

- Compile and load the dynamic model into the Global Manager.
- Add the Router attribute to the IP Manager DXA configuration file in the Global Manager.
- Add the IP Manager as a source to the Global Manager.

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

Compile and load the dynamic model into the Global Manager

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

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 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/model directory. 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.mdl from the IP Manager's BASEDIR/smarts/local/model directory to the Service Assurance Manager's BASEDIR/smarts/local/model directory.
2. Enter the following command to change to the BASEDIR/smarts/local/model directory in the Service Assurance Manager installation area:

   cd *BASEDIR*/**smarts/local/model**

3. From the BASEDIR/smarts/local/model directory, enter the following command to compile Example2.mdl:

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

If the compilation 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:

- 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.

4. From the BASEDIR/smarts/local/model directory in the Service Assurance Manager installation area, copy the following files to the SM_SITEMOD/l10n directory:

   - Example2_mmbe.dat
   - Example2_mmle.dat

   By default, SM_SITEMOD is set to BASEDIR/smarts/local.

5. Go to the BASEDIR/smarts/bin directory in the Service Assurance Manager installation area and enter the following command to start the test Global Manager and load the Example2.ldm file:

   ```
   sm_server -n TEST_SA -c ics --broker=localhost:426 --output
      --dynamic
   ```

The startup code registers the test Global Manager as TEST_SA with the Broker that is running on port 426 of the local host. Substitute the Broker location with your Broker location.

Add the Router attribute to the IP Manager DXA file

In the Service Assurance Manager installation area, the BASEDIR/smarts/conf/ics/ics-default.xml file designates which DXA configuration file is to be used to transfer events and topology from the IP Manager to the Global Manager. The ics-default.xml file contains the following sections for the IP Manager and indicates the DXA file for the IP Manager:

```
<domaintype
    name="INCHARGE-AM-SUITE"
    dxaConfFile="dxa-conn.conf"
    MinimumCertainty="0.01"
    SmoothingInterval="65">
    <description>Type for INCHARGE-AM-SUITE</description>
    <hook enable="False">ics/dxa-sample-hook.asl</hook>
    <domainname>INCHARGE-AM</domainname>
</domaintype>

<domain name="INCHARGE-AM" enable="True"
    domaintypename="INCHARGE-AM-SUITE">
```

```
    <description>AM analysis domain</description>
</domain>

<domaintype
    name="INCHARGE-AM-PM-SUITE"
    dxaConfFile="dxa-conn-perf.conf"
    MinimumCertainty="0.01"
    SmoothingInterval="65">
    <description>Type for InCharge AM PM SUITE</description>
    <hook enable="False">ics/dxa-sample-hook.asl</hook>
</domaintype>
```

Because the test IP Manager (TEST_IP) was started as an IP Availability Manager (AM), its domain type is INCHARGE-AM-SUITE. If you started your test IP Manager as an IP Availability Manager and an IP Performance Manager running as a single process (AM-PM), you must load the INCHARGE-AM-PM-SUITE domain type into the Global Manager and then add the Router attribute to the dxa-conn-perf.conf file. Go to "Add the event subscription to the IP Manager DXA file" on page 69 for more information.

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

1. Go to the BASEDIR/smarts/bin directory in the Service Assurance Manager installation area and enter the following command to open the dxa-conn.conf file:

   `sm_edit conf/ics/dxa-conn.conf`

   The first section of the dxa-conn.conf file provides a listing of the classes, and attributes of those classes, that will be transferred from the IP Manager to the Global Manager.

2. Just before the entry for the ICIM_UnitaryComputerSystem, add following two lines for the Router class (shown in **bold**):

   ```
   # Remote Name of Class/Property         Mapped Name of Class/Property
   server  INCHARGE-AM

   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 receive attribute changes for the new SM_SerialNumber attribute of the Router class from the IP Manager.

   The addition of the two lines is the only change that needs to be made to the dxa-conn.conf file.

3. Save and close the file.

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

Add the IP Manager as a source to the Global Manager

To add the IP Manager as a source to the test Global Manager:

1. In the Domain Manager Administration Console that is attached to the test IP Manager (TEST_IP), select **File › New › Topology Browser Console** to launch a Topology Browser Console.
2. In the Topology Browser Console, select **Manager › Attach** to launch the **Attach Manager** dialog box.
3. In the dialog box, enter the test Global Manager name (**TEST_SA**) and the login information (for example, *admin* and *changeme*) and click **OK**.
4. In the Topology Browser Console, select **TEST_SA** from the **Manager** drop-down menu.
5. In the Topology Browser Console, select **Configure › Global Manager Administration Console** to launch a Global Manager Administration Console.
6. In the left panel of the Global Manager Administration Console, expand the configuration tree to display the **Domains**.
7. Right-click **Domains** and then click the pop-up menu to launch the **Domain Creation Wizard**.
8. Consult Figure 4 on page 40 and follow the directions to create an IP Manager domain named **TEST_IP**. (Type **TEST_IP** in the **Domain Name** field and select **INCHARGE-AM** in the **Copy Existing** drop-down list.)
9. In the toolbar area of the Global Manager Administration Console, click the **Reconfigure** toolbar button ( ).
10. In the **Server Reconfiguration** confirmation dialog box that appears, click **Yes**.
11. In the **Server Reconfiguration** information dialog box that appears, click **Close**.

In response, the Global Manager:

- Creates an InChargeDomain object named **TEST_IP** and starts the appropriate topology and event drivers.
- Consults the Broker for the deployment and establishes a connection to the **TEST_IP** domain.
- Subscribes to and imports topology and events from the **TEST_IP** domain.

Verifying the presence of the SM_SerialNumber attribute in the Global Manager

To verify the presence of the SM_SerialNumber attribute in the test Global Manager:

1. In the Global Manager Administration Console that is attached to the test Global Manager (TEST_SA), select **File › New › Topology Browser Console** to launch a Topology Browser Console.
2. In the Topology Browser Console, verify that the SM_SerialNumber attribute has been added to the Router class, as shown in Figure 9 on page 54.

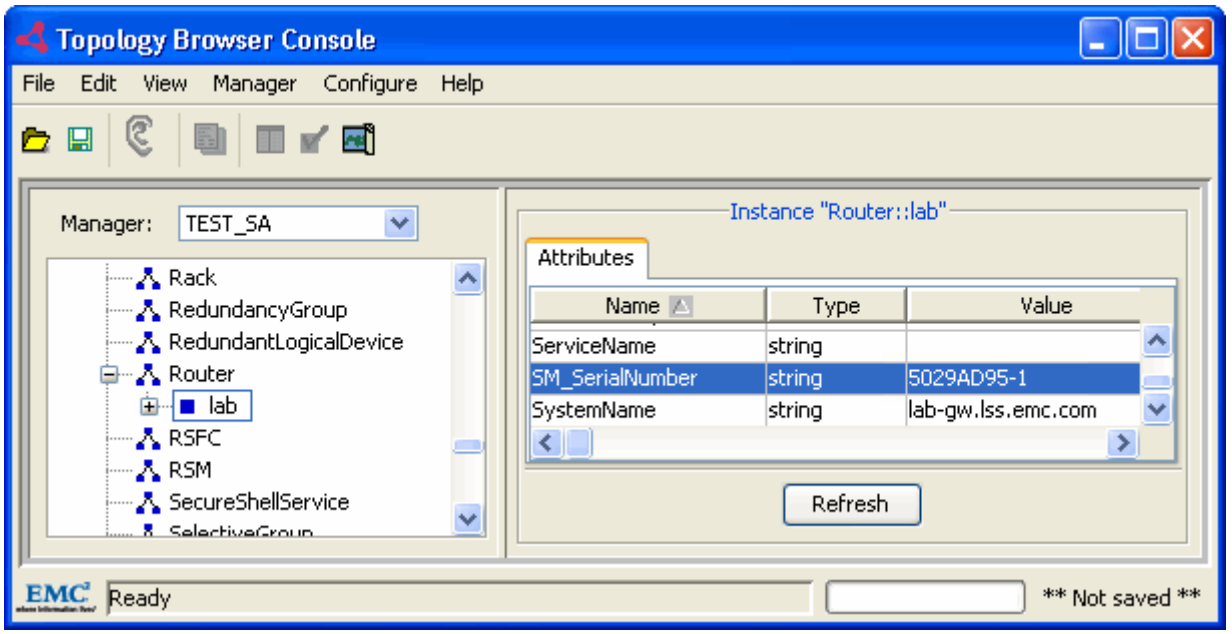

Figure 9 SM_SerialNumber in the TEST_SA Global Manager

Note that if you did not populate the value of the attribute (see "Populating the value of SM_SerialNumber" on page 50), the value of SM_SerialNumber will still be "NotAvailable."

CHAPTER 4
Exercise 3 - Polling for Values of Additional Attributes

This chapter consists of the following sections:

- Objectives 56
- Exercise 57
- Determining the instrumentation class 58
- Creating the dynamic model 59
- Compiling the dynamic model 61
- Copying files for localization 61
- Subscribing to the new event in the IP Manager 62
- Adding the new threshold to the adapter-setting.asl file 63
- Loading the dynamic model into the IP Manager 63
- Discovering an instance of the Router class 65
- Verifying the new attributes in the IP Manager 65
- Verifying the new threshold in the IP Manager 67
- Adding a new event subscription to the Global Manager 69
- Polling the new attributes 73

Objectives

This exercise is a complex example of using Dynamic Model to enhance the following Managers:

- IP Manager
- Global Manager

NOTICE

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

The purpose of the exercise is to:

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

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 IP 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 IP Manager.
- Add a new threshold setting to the adapter-setting.asl file in the IP Manager.
- Load the dynamic model into the IP Manager.
- Discover an instance of the Router class.
- Verify the new attributes of the instrumentation class in the IP Manager.
- Use the Polling and Thresholds Console to verify the new threshold setting.
- Add the polled event to the Global Manager.
- Poll the values of the new attributes of the instrumentation class in real-time.
- Generate an event based on the values of the polled attribute.
- Have the Global Manager subscribe to and display notifications that are associated with the event.

Exercise

In the EMC Smarts environment, an event is a stateful condition that is applied to a topology class in a Domain Manager. An event for a discovered topology object, at any given time, is either notified (active) or clear (inactive). Events are generated by the Domain Manager.

The Global Manager becomes aware of a generated event by subscribing to the event. The Global Manager will receive a NOTIFY message when the event becomes active, and a CLEAR message when the event clears.

Events are associated with topology objects, but are not objects themselves. Notifications, on the other hand, are objects that represent events. The Global Manager maps the data in an imported event to the attributes of a notification object.

In this exercise, the IP 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, exceeds 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.

NOTICE

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 prefix is already used by EMC Smarts.

Determining the instrumentation class

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

The IP 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 that represents the device, which is where the event logic is defined. The event logic might 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, which are located in the BASEDIR/smarts/conf/discovery directory. In this example, the OID for a Cisco 2621XM 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** in the device certification:

```
.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 that is 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.

In the device certification for the Cisco 2621XM router, for example, Interface_Performance_CiscoRouter_Ethernet is the instrumentation class that is used to instrument the performance of Ethernet interface (IF) connections on discovered 2621XM routers.

Creating the dynamic model

In this example, an instance of a compiled dynamic model file named Example3.ldm is created in the BASEDIR/smarts/local/model/‹*conf*› directory of the IP Manager installation area. The IP Manager 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. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to create a new model file named Example3.mdl:

   sm_edit --create model/Example3.mdl

2. To refine the setting class that sets the default value for the alarm threshold, add the following lines to 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;
   }
   ```

   The NetworkAdapter_Performance_Setting class defines parameters that are common to the performance of all network adapters, such as utilization, errors, broadcast, and packet drops.

3. To refine the Interface_Performance_CiscoRouter_Ethernet instrumentation class that is used to poll for the attribute, add the following lines to 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;
}
```

These lines add five attributes to the Interface_Performance_CiscoRouter_Ethernet class:

- TS_OverrunThreshold (threshold attribute)
- TS_locIfInOverrun (instrumented attribute)
- TS_locIfInOverrunRate (computed attribute)
- TS_OverrunErrorRate (computed attribute)
- TS_OverrunErrorPct (computed attribute)

The instrumentation for an object, such as an interface, is generally defined in a separate class that is called an instrumentation class. The definition to poll for the value of the TS_locIfInOverrun MIB attribute is added in this instrumentation class. Classes of this type inherit from the parent class called ICIM_Instrumentation.

The instrument SNMP{} block defines the OID value that needs to be polled. Note that there is no leading period (“.”) in the specification of the OID. The value that is returned by polling the OID is stored in an attribute that is defined as an instrumented attribute. The *EMC Smarts MODEL Reference Guide* provides the syntax of an instrumented attribute.

An event clause is used to define a new event. Events are defined in terms of expressions. In this example, a new event is defined that is called TS_HighOverrunRate. This event is triggered when the value of the following expression evaluates to true:

```
TS_OverrunErrorPct > TS_OverrunThreshold
if NetworkAdapterOperational and (TS_OverrunThreshold < 100) and
(MaxSpeed > 0)
```

To ensure that the event is available to a client, an export declaration is required. The *EMC Smarts MODEL Reference Guide* includes details on defining and exporting events.

4. Save and close the file.

   The newly created Example3.mdl file is saved to the BASEDIR/smarts/local/model directory.

Compiling the dynamic model

To compile the Example3.mdl model file:

1. Enter the following command to change to the BASEDIR/smarts/local/model directory in the IP Manager installation area:

   cd *BASEDIR*/**smarts/local/model**

2. From the BASEDIR/smarts/local/model directory, enter the following command to compile Example3.mdl:

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

If the compilation 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:

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

The following subdirectory 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 in the IP Manager installation area, copy the following files to the SM_SITEMOD/l10n directory:

- Example3_mmbe.dat
- Example3_mmle.dat

By default, SM_SITEMOD is set to BASEDIR/smarts/local.

Subscribing to the new event in the IP Manager

One of the strengths of EMC Smarts technology is that expressions that are defined in an event are not evaluated and polled unless there is a subscription to that event. In order for the IP Manager to automatically begin polling and monitoring for the new event, you need to add a subscription to the IP Manager's bootstrap-am-pm.conf file.

To add a subscription to the IP Manager's bootstrap-am-pm.conf file:

1. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to open the bootstrap-am-pm.conf file:

   `sm_edit conf/icf/bootstrap-am-pm.conf`

2. In the **GA_SubscriberProfile::PROFILE-default** section, add the following line:

```
{"Interface_Performace_CiscoRouter_Ethernet",".*","TS_HighOverrunRate",77879 }
```

   The format is as follows:

   `{ <class>, <instance>, <event>, <subscription-level> },`

   where:

   - *‹class›* is the name of the class in double quotes.
   - *‹instance›* is the name of the instance in double quotes.
   - *‹event›* is the name of the event in the class in double quotes.
   - *‹subscription-level›* can have the following values:
     - 77879 = problems, aggregates, events
     - 77890 = aggregates
     - 77825 = aggregates, events
     - 77840 = problems, aggregates

   NOTICE

   If your entry is the last line in the GA_Subscriber section, ensure that you type a comma (,) at the end of the line before your entry.

3. Save and close the file.

   The modified version of the bootstrap-am-pm.conf file is saved to the BASEDIR/smarts/local/conf/icf directory.

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

The new threshold attribute, TS_HighOverrunRate, must be added to the BASEDIR/smarts/rules/health/adapter-setting.asl file in the IP Manager installation area.

To add the threshold to the adapter-setting.asl file:

1. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to open the adapter-setting.asl file:

   ```
   sm_edit rules/health/adapter-setting.asl
   ```

2. In the **APPLY_SETTING()** section, add the following line (shown in **bold**):

   ```
   if (!setObj->isNull()) {
      REMOVE_PERF_INSTRUMENTATION(obj);
      continue;
   }

   threshold_map += "TS_OverrunThreshold";


   if (obj->CreationClassName == "Port") {
   ```

   NOTICE

   The name of the threshold in the instrumentation object and in the setting object *must* be exactly the same name.

3. Save and close the file.

   The modified version of the adapter-setting.asl file is saved to the BASEDIR/smarts/local/rules/health directory.

Loading the dynamic model into the IP Manager

Once the dynamic model has been compiled, you must load it into the IP Manager.

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

If the test IP Manager is already running, stop it.

To load the compiled dynamic model file into the test IP Manager:

Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to start the test IP Manager and load the compiled dynamic model file:

```
sm_server -n TEST_IP -c icf --broker=localhost:426
   --bootstrap=bootstrap-am-pm.conf --output --dynamic
```

The startup code uses bootstrap-am-pm.conf as the bootstrap file for the test IP Manager. The startup code also registers the test IP Manager as TEST_IP with the Broker that is running on port 426 of the local host. Substitute the Broker location with your Broker location.

In addition, all messages will be logged to a log file, such as BASEDIR/smarts/local/logs/TEST_IP_en_US_UTF-8.log.

To check that the compiled dynamic model file was loaded:

Go to the BASEDIR/smarts/local/logs directory and use any text editor to open the server log file.

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

```
[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
```

Discovering an instance of the Router class

To discover a Router instance:

1. In the Topology Browser Console that is attached to the test IP Manager (TEST_IP), select **Configure › Domain Manager Administration Console** to launch a Domain Manager Administration Console.
2. In the Domain Manager Administration Console, initiate a discovery.

   One method is to select **Topology › Add Agent** to display the **Add Agent** dialog box. In the dialog box, specify either the hostname or IP address of the router.

   The *EMC Smarts IP Manager Concepts Guide* and the *EMC Smarts IP Manager User Guide* describe various methods of discovery.

Verifying the new attributes in the IP Manager

Verify that the router and interface have been added to the topology by verifying that the new attributes for the Interface_Performance_CiscoRouter_Ethernet instrumentation class have been added to the data model of the test IP Manager.

To verify the presence of the new attributes:

1. In the Domain Manager Administration Console that is attached to the test IP Manager (TEST_IP), click an open space in the topology tree panel and press the **Ctrl-Alt-M** keys simultaneously to make all IP Manager classes available to the Global Console, including the meta classes.
2. In the topology tree panel of the Global Console, right-click an open space and select **Refresh tree** from the pop-up menu to refresh the content of the topology tree.
3. In the topology tree panel, click the **+** signs to expand the **Interface_Performance_CiscoRouter_Ethernet** class, then click an Interface_Performance_CiscoRouter_Ethernet instance to display the **Attributes** tab in the right panel.
4. In the **Attributes** tab, shown in Figure 10 on page 66, verify that the following attributes have been added to the Interface_Performance_CiscoRouter_Ethernet instance:

- TS_locIfInOverrun
- TS_locIfInOverrunRate
- TS_OverrunErrorPct
- TS_OverrunErrorRate
- TS_OverrunThreshold

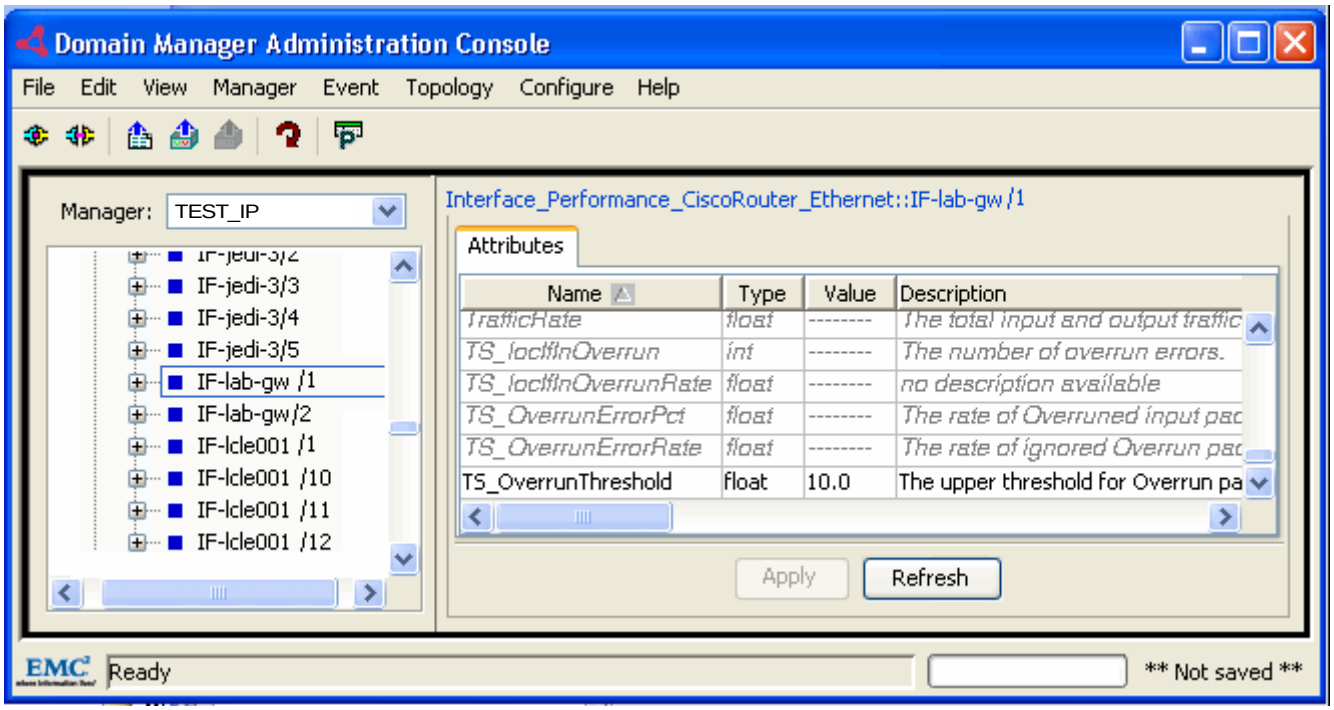

Figure 10 Attributes tab with the new attributes before polling

5. In the topology tree panel, right-click an open space and select **Refresh tree** from the pop-up menu to refresh the content of the topology tree.
6. Press the **Ctrl-Alt-M** keys simultaneously to make only the default IP Manager classes available.
7. In the topology tree panel, click the **+** sign next to the IP Manager's name to expand the topology tree. Only the default classes appear in the topology tree.

Verifying the new threshold in the IP Manager

In this example, the TS_OverrunThreshold attribute was added to the Ethernet Interface/Port Performance setting 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 that are common to all network adapters, such as utilization, errors, broadcast, and packet drops.

To verify that the TS_OverrunThreshold attribute was added to the Ethernet Interface/Port Performance setting:

1. In the Domain Manager Administration Console that is attached to the test IP Manager (TEST_IP), select **Configure › Polling and Thresholds** to launch the Polling and Thresholds Console.
2. In the Polling and Thresholds Console, click 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**.

   NOTICE

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

6. Verify that the value of the TS_OverrunThreshold attribute can be set as shown in Figure 11 on page 67.

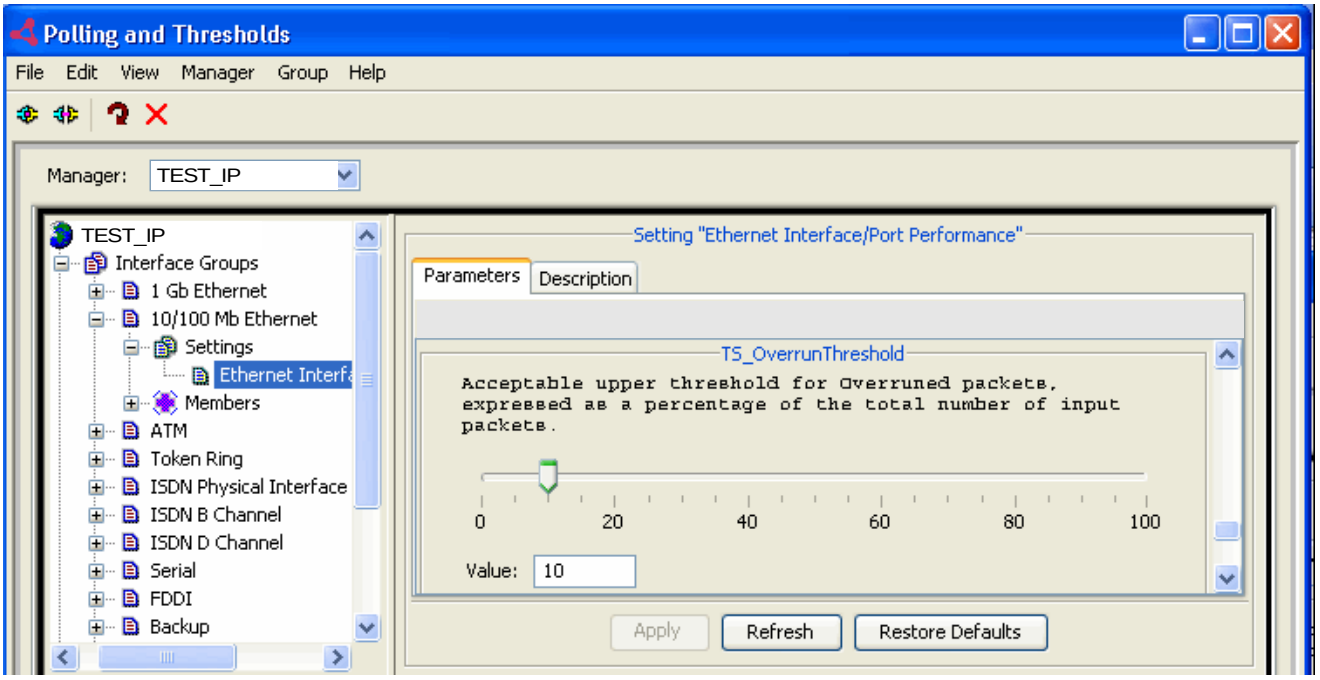

Figure 11 TS_OverrunThreshold attribute in the Thresholds tab

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 Figure 12 on page 68.

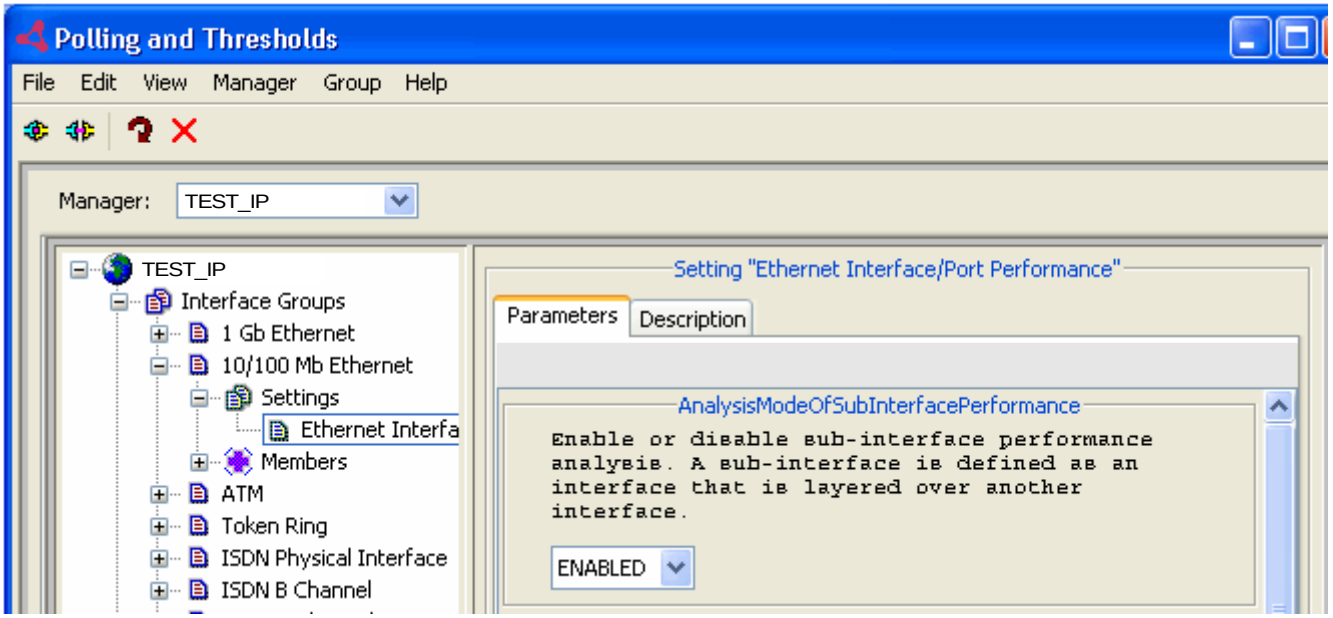

Figure 12 Enabled subinterface performance analysis

8. Click the **Apply** button to save your configuration changes.
9. Click the **Reconfigure** toolbar button ( ) to make your configuration changes take effect.

Adding a new event subscription to the Global Manager

You do not load the new event into the Global Manager because the Global Manager data model does not have to be enhanced to include the event itself.

You do, however, need to complete the following tasks to enable the Global Manager to import the new event from the IP Manager:

- Add the event subscription to the IP Manager DXA configuration file in the Global Manager.
- Add the IP Manager as a source to the Global Manager.

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

Add the event subscription to the IP Manager DXA file

In the Service Assurance Manager installation area, the BASEDIR/smarts/conf/ics/ics-default.xml file designates which DXA configuration file is to be used to transfer events and topology from the IP Manager to the Global Manager. The ics-default.xml file contains the following sections for the IP Manager and indicates the DXA file for the IP Manager:

```
<domaintype
    name="INCHARGE-AM-SUITE"
    dxaConfFile="dxa-conn.conf"
    MinimumCertainty="0.01"
    SmoothingInterval="65">
    <description>Type for INCHARGE-AM-SUITE</description>
    <hook enable="False">ics/dxa-sample-hook.asl</hook>
    <domainname>INCHARGE-AM</domainname>
</domaintype>

<domain name="INCHARGE-AM" enable="True"
    domaintypename="INCHARGE-AM-SUITE">
    <description>AM analysis domain</description>
</domain>

<domaintype
    name="INCHARGE-AM-PM-SUITE"
    dxaConfFile="dxa-conn-perf.conf"
    MinimumCertainty="0.01"
    SmoothingInterval="65">
    <description>Type for InCharge AM PM SUITE</description>
    <hook enable="False">ics/dxa-sample-hook.asl</hook>
</domaintype>
```

Because the test IP Manager (TEST_IP) was started as an IP Availability Manager and an IP Performance Manager running as a single process (AM-PM), its domain type is INCHARGE-AM-PM-SUITE. And because that domain type is not loaded into the Global Manager by default, you must load it into the Global Manager and then add the event subscription to the dm-conn-perf.conf file. The *EMC Smarts Service Assurance Manager Configuration Guide* provides the procedure for using the Import XML Domain Configuration server tool to load INCHARGE-AM-PM-SUITE.

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

The syntax is as follows:

sub ‹*classPattern*›::‹*instancePattern*›::‹*eventPattern*›|‹*modifiers*›

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 subscription to the dxa-conn-perf.conf file:

1. Go to the BASEDIR/smarts/bin directory in the Service Assurance Manager installation area and enter the following command to open the dxa-conn-perf.conf file:

   `sm_edit conf/ics/dxa-conn-perf.conf`

2. In the subscription section of the file, add the following lines:

```
# 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" on page 107 provides more information.

4. Save and close the file.

   The modified version of the dxa-conn-perf.conf file is saved to the BASEDIR/smarts/local/conf/ics directory.

Add the IP Manager as a source to the Global Manager

To add the IP Manager as a source to the test Global Manager:

1. In the Domain Manager Administration Console that is attached to the test IP Manager (TEST_IP), select **File › New › Topology Browser Console** to launch a Topology Browser Console.
2. In the Topology Browser Console, select **Manager › Attach** to launch the **Attach Manager** dialog box.
3. In the dialog box, enter the test Global Manager name (**TEST_SA**) and the login information (for example, *admin* and *changeme*) and click **OK**.
4. In the Topology Browser Console, select **TEST_SA** from the **Manager** drop-down menu.
5. In the Topology Browser Console, select **Configure › Global Manager Administration Console** to launch a Global Manager Administration Console.
6. In the left panel of the Global Manager Administration Console, expand the configuration tree to display the **Domains**.
7. Right-click **Domains** and then click the pop-up menu to launch the **Domain Creation Wizard**.
8. Consult Figure 13 on page 72 and follow the directions to create an IP Manager domain named **TEST_IP**. (Type **TEST_IP** in the **Domain Name** field and select **INCHARGE-AM-PM** in the **Copy Existing** drop-down list.)
9. In the toolbar area of the Global Manager Administration Console, click the **Reconfigure** toolbar button ( ).
10. In the **Server Reconfiguration** confirmation dialog box that appears, click **Yes**.
11. In the **Server Reconfiguration** information dialog box that appears, click **Close**.

In response, the Global Manager:

- Creates an InChargeDomain object named **TEST_IP** and starts the appropriate topology and event drivers.
- Consults the Broker for the deployment and establishes a connection to the **TEST_IP** domain.
- Subscribes to and imports topology and events from the **TEST_IP** domain.

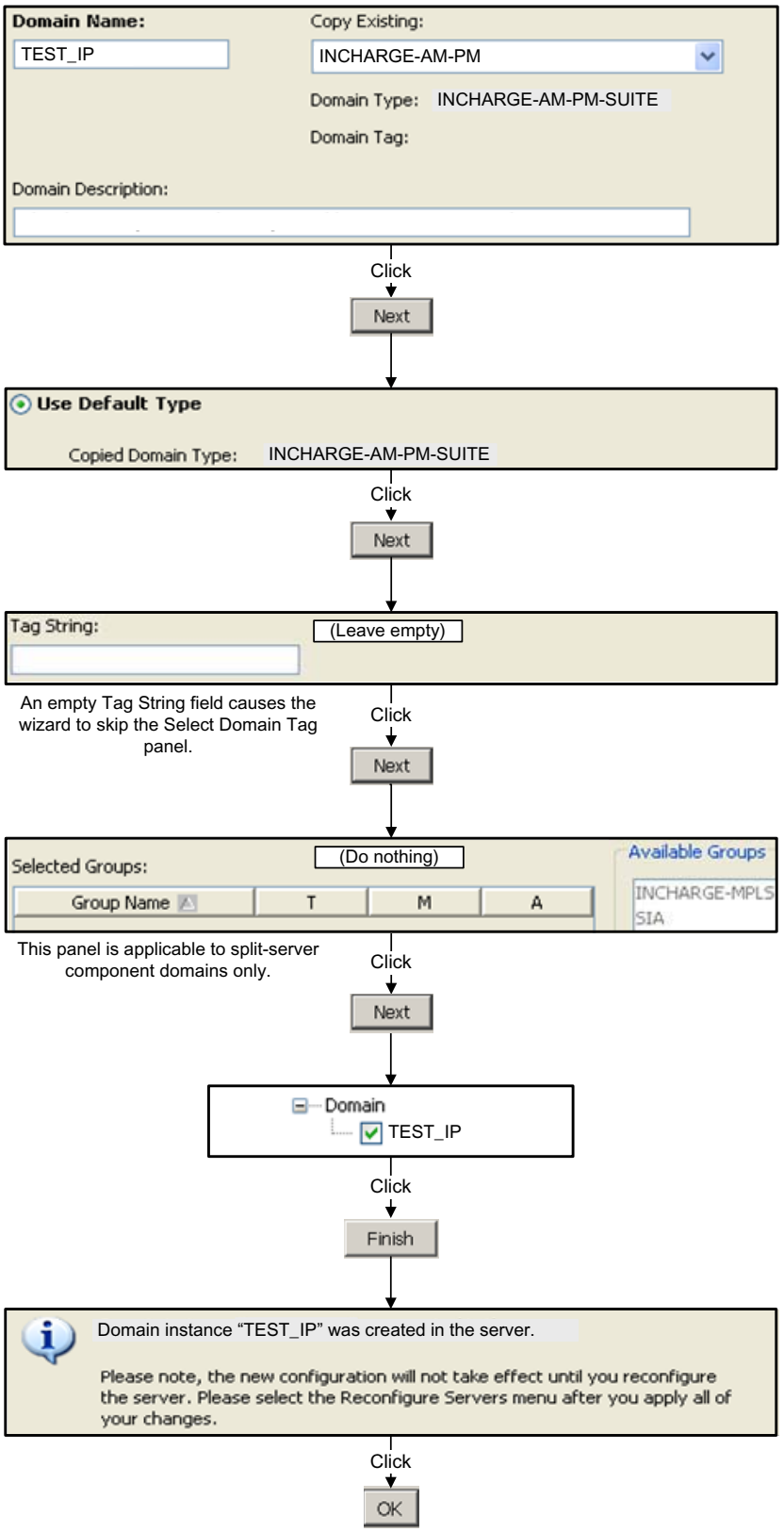

Figure 13 Using the Domain Creation Wizard to create the TEST_IP domain for Exercise 3

Polling the new attributes

The Global Manager must be restarted for the subscription to the TS_HighOverrunRate event to take effect, and for the polling of the new attributes to commence.

If the test Global Manager is already running, stop it.

To start the polling of the new attributes:

Go to the BASEDIR/smarts/bin directory in the Service Assurance Manager installation area and enter the following command to start the test Global Manager:

```
sm_server -n TEST_SA -c ics --broker=localhost:426 --output
```

The --dynamic option is not included in the startup command because the Global Manager data model does not have to be enhanced to include the event itself. Substitute the Broker location with your Broker location.

Once polling has started, the values of TS_locIfInOverrun, TS_locIfInOverrunRate, and TS_OverrunErrorPct, and TS_OverrunErrorRate will appear in the IP Manager's topology, similar to what is shown in Figure 14 on page 73.

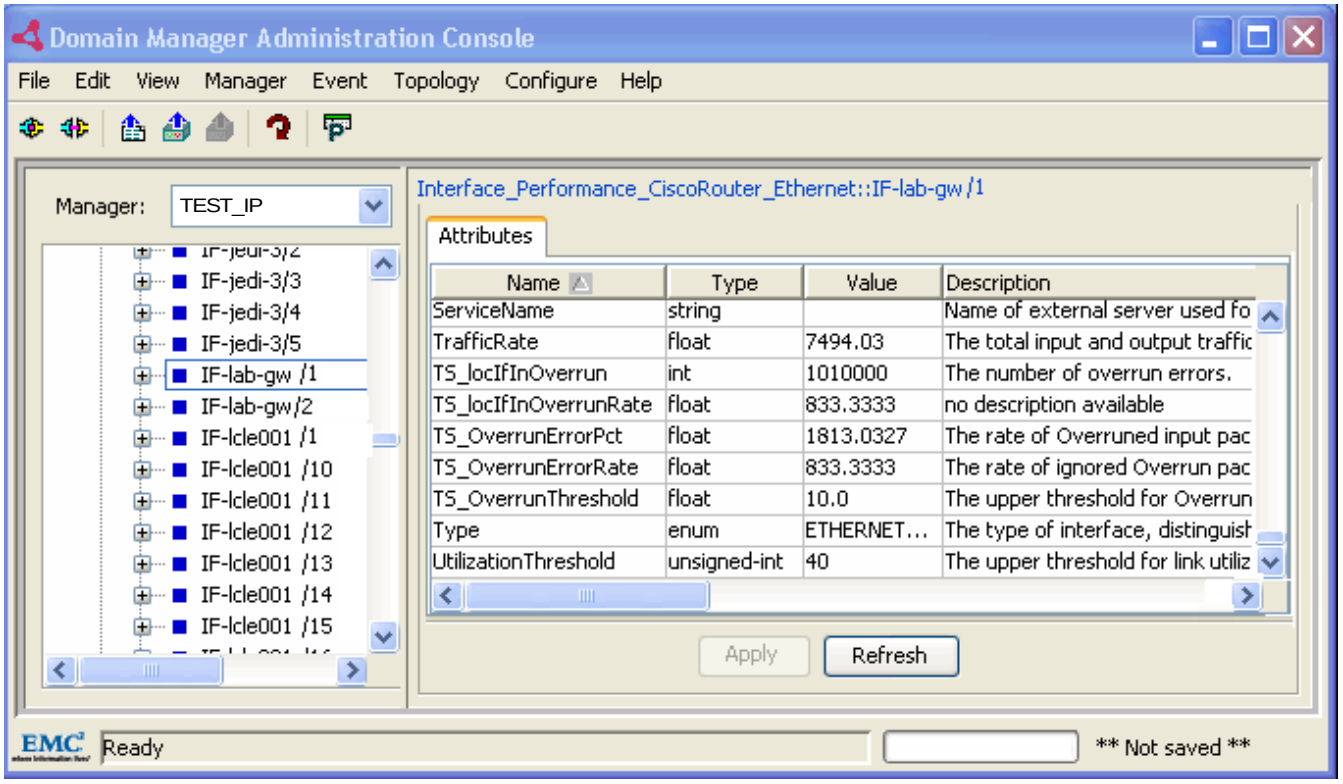

Figure 14 Attributes tab with the new attributes after polling

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 Notification Log Console.

APPENDIX A
Guidelines and Syntax for Dynamic Model Files

This appendix consists of the following sections:

- Guidelines and requirements 76
- Adding a class to the model 77
- Updating an existing class in the model 79

Guidelines and requirements

A dynamic model file describes the model elements that extend the basic EMC Smarts 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.
- Use the sm_edit utility to create and edit the dynamic model files (those with extension .mdl).
- Dynamic model files must follow the syntax rules that are described in "Adding a class to the model" on page 77 and "Updating an existing class in the model" on page 79.
- 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 EMC Smarts.
- Do not use the **external** keyword in any dynamic model files of any Manager.
- Dynamic model files must be located in either BASEDIR/smarts/local/model or BASEDIR/smarts/local/model/‹*config*›, where ‹*config*› refers to the configuration directory that is specified on the command line when the 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) that are located in:

  BASEDIR/smarts/local/model/‹*config*›

  where ‹*config*› refers to the configuration directory that is specified on the **sm_server** command line when the Manager is started.

  Appendix B, "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) that are located in:
  - BASEDIR/smarts/local/model
  - BASEDIR/smarts/local/model/‹*config*›

    The ‹*config*› refers to the configuration directory that is specified on the **sm_server** command line when the Manager is started.

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" on page 78. 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 EMC Smarts 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.

NOTICE

The IP Manager counts the instances of the ICIM_UnitaryComputerSystem toward the system volume license. 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 configuration file 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>
}
```

NOTICE

Use a unique prefix with new class names and other properties, since the name must differ from existing ICIM classes and properties. “SM_” is used as a prefix in this example. Do not use “SM_” as this is already used by EMC Smarts.

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 *EMC Smarts 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 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 an IP 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 *EMC Smarts 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 *EMC Smarts 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;
}
```

NOTICE

The inverse relationship, OSRunninOn, must also be defined in the SM_OperatingSystem class.

For more information about relationship declarations, see the *EMC Smarts 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 *EMC Smarts 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 *EMC Smarts 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 *EMC Smarts 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 *EMC Smarts 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 *EMC Smarts MODEL Reference Guide*.

APPENDIX B
The dynmodel Command Syntax and Options

This appendix consists of the following sections:

- Syntax and options 86
- Pragmas..... 89

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 Table 2 on page 86.
- ‹*mdl-files*› specify one or more files, separated by a space.

Table 2 Command line options for the dynmodel command

option	Description
-w	Suppress generation of warning messages that are 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.
--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 conjunction with the --locales option.
--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 conjunction with the --locales option.

In the following example, the command line suppresses the generation of warning messages that are produced when compiling the mymodel.mdl file:

```
dynmodel -w mymodel.mdl
```

The **dynmodel** command creates the following files in the BASEDIR/smarts/local/model directory, where ‹*modelname*› is the name of a dynamic model. Table 3 on page 87 describes the set of files that is created for each of the dynamic model files that are included on the command line.

Table 3 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 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 that is created for holding temporary files that are 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.mdl that can be loaded by a Manager.
```

NOTICE

The dynamic model source files can be created and compiled in any directory. However, all compiled dynamic model files (those files with the .ldm extension) *must* reside in either the BASEDIR/smarts/local/model directory or the BASEDIR/smarts/local/model/*‹config›* directory, in order to be loaded into a Manager. The ‹*config*› refers to the configuration directory that is specified on the **sm_server** command line when the Manager is started.

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, 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 the following example, **dynmodel** compiles two models, model1 and model2:

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

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.

If you do not include the `--l10nsrc` and `--locales` options, **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. Table 4 on page 89 lists the available pragmas.

Table 4 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.

APPENDIX C
Methods of Creating Classes and Populating Attributes

This appendix consists of the following sections:

- Overview 92
- Using dmctl 92
- Populating attributes by using an EMC Smarts API 95
- Discovery customization 96

Overview

For new classes that are 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 EMC Smarts APIs (PERL, C, Java)
- Adapter Scripting Language (ASL) scripts
- Working with scripts that are invoked as part of the postprocessing phase of the 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" on page 95 provides additional information.

Command-line mode

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

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

where:

- *‹managername›* 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.
- *‹commandname›* is the name of the command.
- *‹commandparams›* is the list of the parameters for the command.

NOTICE

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

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

```
BASEDIR/smarts/bin/dmctl -s TEST 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_IP getInstances <classname>
```

where ‹*classname*› is the name of the class that 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 <managername> [--broker=<host>:<port>]
```

where:

- ‹*managername*› 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.

NOTICE

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

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

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

The output appears similar to the following:

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

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

```
create <classname>::<instancename>
```

where:

- ‹*classname*› is the name of the class.
- ‹*instancename*› is the name of the instance of the class.

For example, enter 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 <classname>::<instancename>::<propertyname> <value>
```

where:

- *‹classname›* is the name of the class.
- *‹instancename›* is the name of the instance of the class.
- *‹propertyname›* is the name of the property in the class instance.
- *‹value›* is the value of the property.

For example, enter 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
```

NOTICE

The **getProperties** command lists all of the properties for a class. The syntax is **getProperties** *‹class›*.

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:

```
BASEDIR/smarts/bin/dmctl --server=TEST_IP --broker=localhost:426
   --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, enter the following command:

```
BASEDIR/smarts/bin/dmctl -s <GlobalManagername> invoke
   GA_Driver::<managername>_Topo-Driver start▲
```

where:

- ‹*GlobalManagername*› is the name of the Global Manager.
- ‹*managername*› is the name of the underlying Manager that is being synchronized.

For example:

```
BASEDIR/smarts/bin/dmctl -s TEST_SA invoke
   GA_Driver::TEST_IP_Topo-Driver start
```

Populating attributes by using an EMC Smarts 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 *EMC Smarts Perl Reference Guide* provides additional information.

```
use InCharge::session;


#Establish a connection to the EMC Smarts Server
$session=InCharge::session->new(server=>"TEST_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 *EMC Smarts 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 IP 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 97.

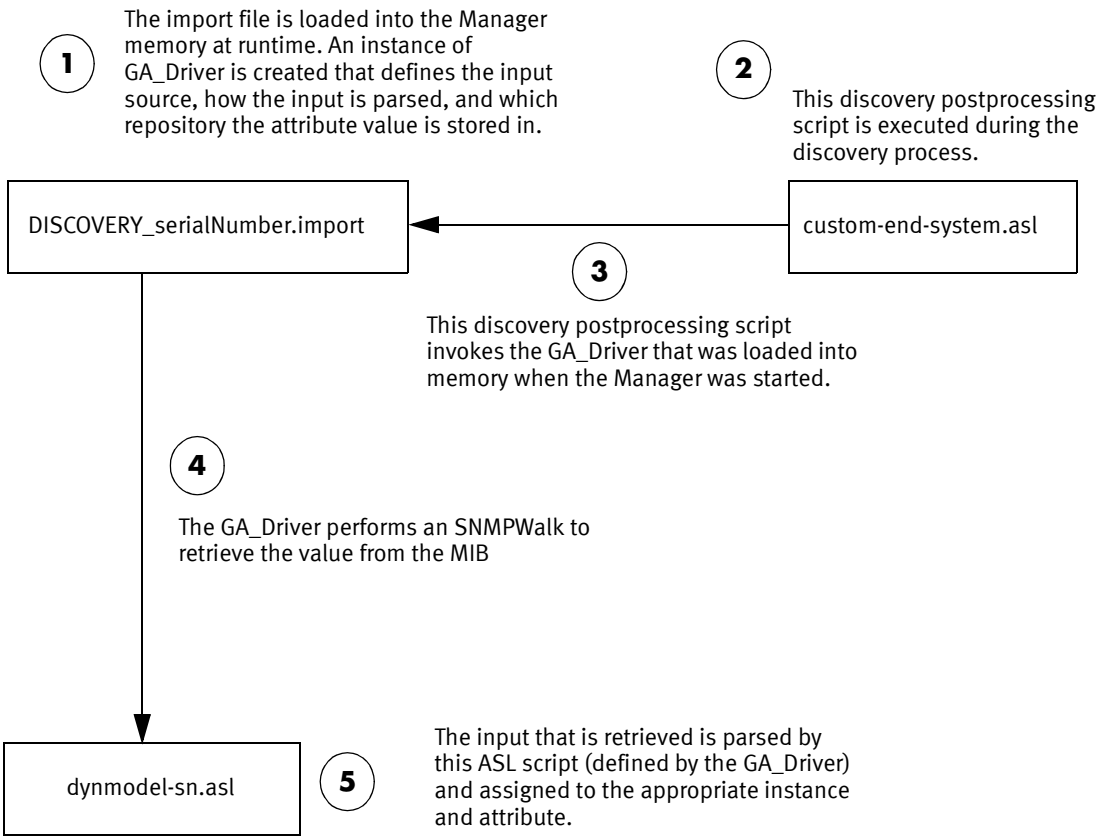

Figure 15 Using the discovery postprocessing mechanism

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:

1. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to create a new import file named DISCOVERY_serialNumber.import:

   sm_edit --create conf/discovery/DISCOVERY_serialNumber.import

2. Add the following lines to the DISCOVERY_serialNumber.import file:

```
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
}
```

3. Save and close the file.

   The newly created DISCOVERY_serialNumber.import file is saved to the BASEDIR/smarts/local/conf/discovery directory.

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

- 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 Figure 16 on page 99.

```
START {
   /*
    * Add Parsing rules here.
    */
   .. eol
} do {
   /*
    * Add processing rules here.
    */
                                        New processing rules are added
                                        here.
   stop();
}
```

Figure 16 Modifying custom-end-system.asl

To add custom processing:

1. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to open the custom-end-system.asl file:

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

2. Just after **Add processing rules here** in the START rule, add the following lines:

```
/* 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 file.

   The modified version of the custom-end-system.asl file is saved to 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. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to create a new ASL file named dynmodel-sn.asl:

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

2. Add the following lines to 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 file.

   The newly created dynmodel-sn.asl file is saved to the BASEDIR/smarts/local/rules/discovery directory.

Restarting the IP Manager

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

To restart the IP Manager and reload the compiled dynamic model file into the IP Manager:

1. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to stop the test IP Manager:

   ```
   dmquit --force --server=TEST_IP
   ```

2. Enter the following command to start the test IP Manager and reload the compiled dynamic model file:

   ```
   sm_server -n TEST_IP -c icf --broker=localhost:426
      --bootstrap=bootstrap-am.conf --output --dynamic
   ```

   The startup code uses bootstrap-am.conf as the bootstrap file for the test IP Manager. The startup code also registers the test IP Manager as TEST_IP with the Broker that is running on port 426 of the local host. Substitute the bootstrap file and the Broker location with your bootstrap file and Broker location.

APPENDIX D
Updating the Global Manager

This appendix contains the following information:

- Loading the dynamic model enhancements..... 104
- Transferring topology and events 105

Loading the dynamic model enhancements

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

To compile and 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/model directory.
2. Remove any problems or events from the dynamic model files.
3. Enter the following command to change to the BASEDIR/smarts/local/model directory in the Service Assurance Manager installation area:

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

4. Compile the dynamic model files. For example, from the BASEDIR/smarts/local/model directory, enter the following command to compile Example2.mdl:

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

   If the compilation 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:

   - 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

5. Copy the dynamic model string files to the SM_SITEMOD/l10n directory. Using the compilation of Example2.mdl as an example, from the BASEDIR/smarts/local/model directory, copy the following files to the SM_SITEMOD/l10n directory:

   - Example2_mmbe.dat
   - Example2_mmle.dat

6. Go to the BASEDIR/smarts/bin directory in the Service Assurance Manager installation area and enter the following command to load the compiled dynamic model files into the Global Manager:

   ```
   sm_server -n TEST_SA -c ics --broker=localhost:426 --output
      --dynamic
   ```

Transferring topology and events

In the Service Assurance Manager installation area, the ics-default.xml file in the BASEDIR/smarts/conf/ics directory is the configuration file for the Global Manager. This file designates which DXA file is used to transfer events and topology from a particular Manager to the Global Manager. The DXA files are also located in the BASEDIR/smarts/conf/ics directory.

The ics-default.xml file contains the following sections for the IP Manager and indicates the DXA file for the IP Manager:

```
<domaintype
    name="INCHARGE-AM-SUITE"
    dxaConfFile="dxa-conn.conf"
    MinimumCertainty="0.01"
    SmoothingInterval="65">
    <description>Type for INCHARGE-AM-SUITE</description>
    <hook enable="False">ics/dxa-sample-hook.asl</hook>
    <domainname>INCHARGE-AM</domainname>
</domaintype>

<domain name="INCHARGE-AM" enable="True"
    domaintypename="INCHARGE-AM-SUITE">
    <description>AM analysis domain</description>
</domain>

<domaintype
    name="INCHARGE-AM-PM-SUITE"
    dxaConfFile="dxa-conn-perf.conf"
    MinimumCertainty="0.01"
    SmoothingInterval="65">
    <description>Type for InCharge AM PM SUITE</description>
    <hook enable="False">ics/dxa-sample-hook.asl</hook>
</domaintype>
```

For the example in Exercise 2, because the test IP Manager was started as an IP Availability Manager, the ics-default.xml file designates **dxa-conn.conf** for the IP Manager.

Transferring topology

The first section of the DXA configuration file 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.

You use the sm_edit utility to edit the DXA configuration file.

For the example in Exercise 2, you use sm_edit to open the **dxa-conn.conf** file and add following two lines for the Router class (shown in **bold**) just before the entry for the ICIM_UnitaryComputerSystem:

```
# Remote Name of Class/Property          Mapped Name of Class/Property
server  INCHARGE-AM

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 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.

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 Manager.

The syntax is as follows:

sub ‹*classPattern*›::‹*instancePattern*›::‹*eventPattern*›/‹*modifiers*›

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

```
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 ICIM_ComputerSystem         Unresponsive    Operational   4
sev ICIM_UnitaryComputerSystem  DiscoveryError  Discovery     3
.
```

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 EMC Smarts IP Availability Manager documentation provides 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
```

APPENDIX E
Troubleshooting

This appendix consists of the following sections:

- Overview 110
- Compilation errors 111
- Loading errors 112
- Discovery errors 117

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.

Compilation errors

License server does not support compiling dynamic model files

The following error may be encountered when using the **dynmodel** compiler to compile dynamic model files. This error indicates that the license for the DK_DYN_MODEL_COMPILE feature has not been installed for this product.

```
BASEDIR\smarts\local\model>..\..\bin\dynmodel.exe Example2.mdl
[October 7, 2009 4:35:21 PM GMT+05:30 +950ms] t@3584 <Primary Thread>
LIC-E-ECHECKOUT-While attempting to check out license for
   DK_DYN_MODEL_COMPILE
LIC-EFLEXLM-FlexLM returned the following error message: "No such
   feature exists.

  . Feature:       DK_DYN_MODEL_COMPILE
  . License path: BASEDIR/smarts/local/conf/SAM80-IP81-all-temp90.lic
  . FLEXnet Licensing error:-5,357
  . For further information, refer to the FLEXnet Licensing End User
   Guide,
  . available at "www.macrovision.com"."
```

Solution

Install the SMA-SDM-C license. The *EMC Smarts System Administration Guide* provides additional information concerning license administration.

License server does not support compiling dynamic model files with problems and symptoms

The following error may be encountered when you are trying to compile dynamic model files that contain new problems or symptoms, or when you are trying to refine existing problems or symptoms. This error indicates that the SMA-SDM-C license has not been installed for this product.

```
BASEDIR\smarts\local\model>..\..\bin\dynmodel.exe Example3.mdl
[October 7, 2009 4:36:55 PM GMT+05:30 +254ms] t@5016 <Primary Thread>
LIC-E-ECHECKOUT-While attempting to check out license for
   DK_DYN_MODEL_PROBLEM
LIC-EFLEXLM-FlexLM returned the following error message: "No such
   feature exists.

  . Feature: DK_DYN_MODEL_PROBLEM
  . License path: BASEDIR/smarts/local/conf/SAM80-IP81-all-temp90.lic
  . FLEXnet Licensing error:-5,357
  . For further information, refer to the FLEXnet Licensing End User
   Guide,
  . available at "www.macrovision.com"."
```

Solution

Install the SMA-SDM-C license. The *EMC Smarts System Administration Guide* provides additional information concerning license administration.

Loading errors

If you include the --output option with the **sm_server** command that is used to load dynamic models into a Manager, the following error messages are written to the server log file. Otherwise, the 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 using use the **dynmodel** compiler to compile the dynamic model files.

```
[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
```

NOTICE

ICU, for International Components for Unicode, is an Open Source set of C/C++ and Java libraries for Unicode and software internationalization support.

Solution

From the BASEDIR/smarts/local/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

You must copy the ‹*modelname*›_mmbe.dat and ‹*modelname*›_mmle.dat files to the SM_SITEMOD/l10n directory every time that you use **dynmodel** to compile the ‹*modelname*›.mdl file.

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 the 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 the sm_edit utility 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. Go to the BASEDIR/smarts/bin directory in the IP Manager installation area and enter the following command to open the oid2type_Misc.conf file:

   ```
   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 modified version of the oid2type_Misc.conf file is saved to 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_IP.rps
 MR-CANTCREATE_INSTANCE-'SM_Printer::CAXTON' could not be created.  ; in file
   "/work/blackcurrent/DMT-9.0.0.X/1201/smarts/local/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/local/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, 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/local/model
- BASEDIR/smarts/local/model/*‹config›*

where *‹config›* refers to the configuration directory that is specified on the command line when the Manager is started.

If you start a Manager without the --dynamic option, only the .ldm files in the following directory will be loaded:

- BASEDIR/smarts/local/model/*‹config›*

where *‹config›* refers to the configuration directory that is specified on the command line when the Manager is started.

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

```
BASEDIR/smarts/bin/sm_server -n TEST -c icf --broker=localhost:426
   --bootstrap=bootstrap-am.conf    --ignore-restore-errors
```

Unable to open ‹ldmfilename›

The .ldm file, *‹ldmfilename›*, cannot be opened by the Manager from the BASEDIR/smarts/local/model directory.

Solution

Verify that *‹ldmfilename›* has the correct read permissions. This error should be fairly rare because most Managers are started by root. They have administrative privileges.

DYN-E-CLASS_NAME_ALREADY_EXIST

The name of a class that is declared in the .mdl source file, at the given line number, conflicts with an existing class name in the configuration that has already been 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 that is declared in the .mdl source file, at the given line number, conflicts with an existing structure name in the configuration that has already been 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 that is declared in the .mdl source file, at the given line number, conflicts with an existing enumeration name in the configuration that has already been 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 that is declared in the .mdl source file, at the given line number, conflicts with an existing attribute name in the configuration that has already been 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 that is declared in the .mdl source file, at the given line number, conflicts with an existing name in the configuration that has already been 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 that declares the propagated property and add the aggregation operator to the propagated property declaration.

Discovery errors

Class property changes do not appear in the Global Manager

You modified class properties, such as attribute values, in the underlying Manager, but you do not see those changes reflected in the Global Manager. The problem is that the topology in the underlying Manager and the Global Manager are no longer synchronized.

Solution

You can synchronize the topology between an underlying Manager and the Global Manager by:

- Initiating a discovery.
- Using the **dmctl** command. "Synchronizing topology changes with the Global Manager" on page 95 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
```

APPENDIX F
MODEL and Dynamic Models

This appendix consists of the following sections:

- The ICIM Reference 120
- Representing systems with MODEL 121
- Differences between MODEL and Dynamic Model 122

The ICIM Reference

Managed Object Definition Language (MODEL) is integral to the EMC Smarts application development process. Its purpose is to render the schema and semantics of the *EMC Smarts ICIM Reference* specification. MODEL is compiled, producing data model libraries that describe the model. These libraries are loaded by a Manager and are operated on by EMC Smarts applications. For an EMC Smarts 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 Managers with semantics that support root-cause analysis.

EMC Smarts models are object-oriented.

- The notion of *class* provides a mechanism for encapsulating a managed element's state or properties. Two semantic properties that are 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. The construct 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 construct 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 the class as an interface. This behavior 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 end, a language is needed to establish syntactic rules for the interface itself and for programs that use the interface. For EMC Smarts, 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. The MODEL also enables tools, such as compilers and analysis programs, to be applied to the model, and defines semantic constructs that support the correlation model.

The following 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 that is 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 that is used to build the system. Executing the appropriate makefile invokes 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 EMC Smarts 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

EMC Smarts ICIM models are typically compiled into shared libraries, with their interfaces exported as C++ header files. Applications, such as the IP Manager and the Global Manager, that use the MODEL libraries link to the libraries at compile time.

The binding of applications to models at compile time imposes a limitation on applications because the applications are not aware of all properties of an interface at compile time. This limitation occurs because ICIM models are intended to be general enough to support a large customer base.

A particular customer, for whom more interface properties would be useful, must, however, limit an application's programs to only what is made available by the ICIM models.

Dynamic modeling is a step toward solving this problem. It allows delays in binding between the model and the applications that operate on the model. Instead of binding at compile time, as with the traditional MODEL, interfaces may now effectively be made known at runtime, when the interfaces are loaded into a Manager as a dynamic model.

Figure 17 on page 123 shows the differences between using MODEL and Dynamic Model.

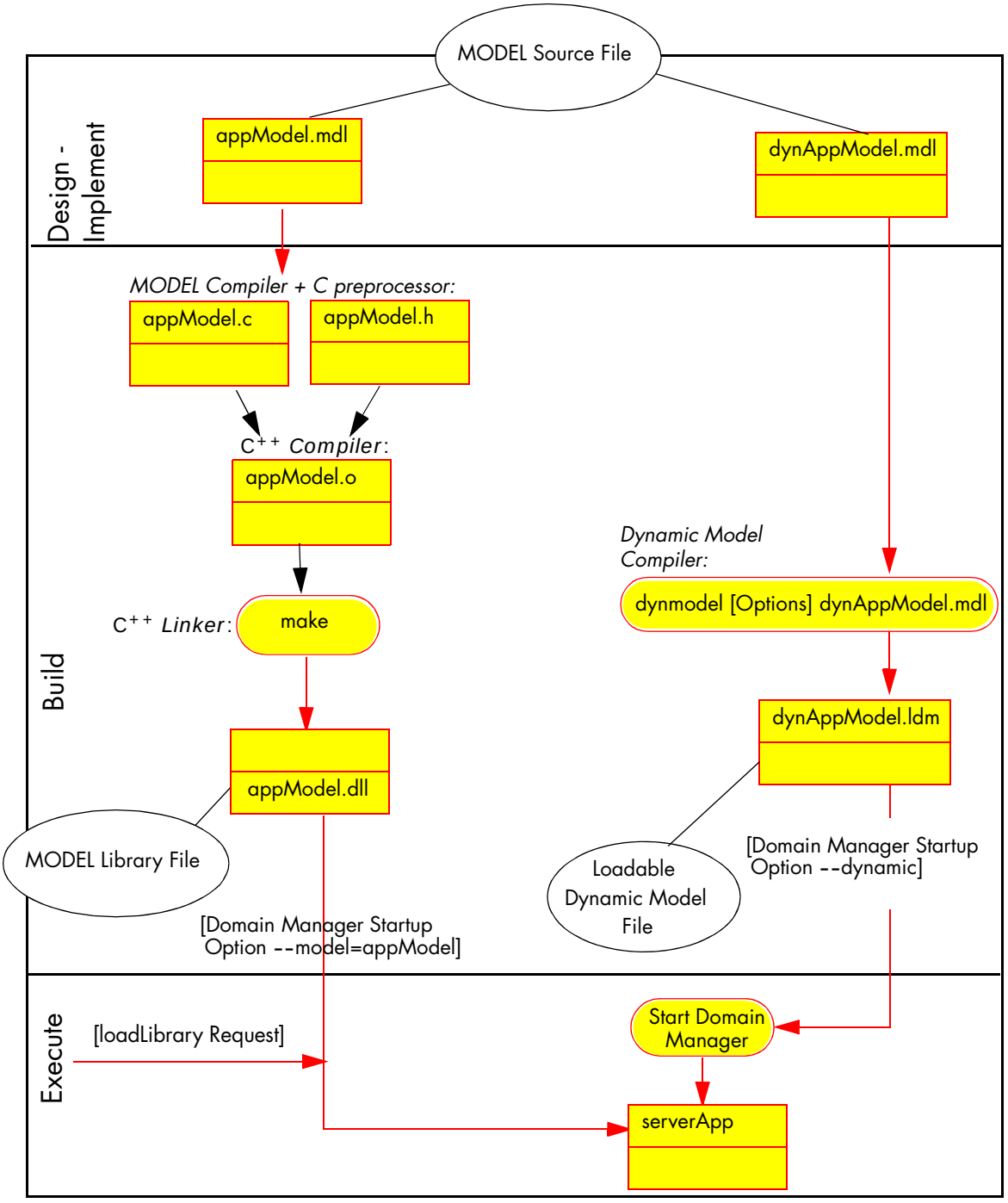

Figure 17 Simplified MODEL and Dynamic Model methodology comparison

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.

Dynamic model files can be loaded when a Manager is started or after the Manager is already running.

Note the following:

- There are other files that are produced by the MODEL compiler and by the Dynamic Model compiler. Figure 17 on page 123 is a simplified view to highlight the major differences. A more detailed explanation of the MODEL compiler is available in the *EMC Smarts MODEL Reference Guide*.
- 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 *EMC Smarts MODEL Reference Guide* is useful when working with Dynamic Model.

INDEX

A

adapter-setting.asl 63
Adding
  Attribute
    Example 45
    Syntax 80
  Device to field certification 35
  New attribute 44
  New attributes to instrumentation class 56
    Example 59
  New class 28
  New event 56
  New threshold setting
    To adapter-setting.asl 63
  Topology Source
    IP Manager to Global Manager 39, 53, 71
Aggregate
  Syntax for adding 82
ASL scripts 96
ASL-ERROR_CREATE 118
ASL-ERROR_PUT 117
Attribute
  Adding to dynamic model file
    Example 45
    Syntax 80
  Adding to instrumentation class
    Example 59
  Forwarding to Global Manager 51, 105
  Parse results of SNMPWalk 100
  Populate value with ASL script 96
  Populating value using dmctl 92
  Populating with API 95
  Starting polling 73

C

Class
  Adding a new one
    Example 30
    Syntax 78
    Verifying in Global Console 33
  Extending
    Syntax 79
  Show classes with no instances 33
Compiling dynamic models
  Command line options for dynmodel command 86
  Example 31, 38, 46, 50, 61
  Location of compiler 86
  Service Assurance Manager 104
  Troubleshooting Errors 111
custom-end-system.asl 99

D

Development cycle 22
Device
  Add to field certification list 34
Differences between MODEL and Dynamic Model 18
Discovery import file 98
discovery postprocessing scripts 96
DK_DYN_MODEL_COMPILE 111
DK_DYN_MODEL_PROBLEM 111
dmctl command 92
  Batch mode 94
  Interactive mode 92, 93
DXA file
  Setting notification characteristics 107
  Setting notification subscriptions 106
  Transferring topology 105
dxa-conn.conf
  Add new attribute 52
  Transferring topology 105
dxa-conn-perf.conf
  Add new event 70
Dynamic Model
  Features 16
  How it differs from MODEL 18
  Location of compiler 86
  Location of files 21, 76
Dynamic modeling and MODEL
  Differences in methodology 122
Dynamic models
  Add a new attribute 45
  Add new attributes to instrumentation class 59
  Adding a new attribute 44
  Adding a new class 28
  Adding a new event 56
  Adding new attributes to instrumentation class 56
  Compiling
    Command line options 86
    Example 31, 38, 46, 50, 61
    In Service Assurance Manager 104
    Locales 88
  Creating
    Example 30
  Editing 20
  Guidelines 20
  Loading into a running Manager 25
  Loading into Global Manager 38, 50
  Loading into IP Manager 32, 47, 64
  Loading into Manager
    With the --dynamic option 26
    Without the --dynamic option 26
  Parent of New Class 77
  Pragmas 89
  Prefix for new classes and properties 20, 76
  Requirements 21
  Restrictions 19
dynamic option, --dynamic option 76
DYN-E-ATTRIBUTE_NAME_ALREADY_EXISTS 116

DYN-E-CLASS_NAME_ALREADY_EXIST 115
DYN-E-ENUMERATION_NAME_ALREADY_EXISTS 116
DYN-E-EVENT_NAME_ALREADY_EXISTS 116
DYN-E-NO_AGGREG_OP 117
DYN-E-STRUCT_NAME_ALREADY_EXISTS 115
dynmodel
    Command line options 86
    Generated files 87
    Location of compiler 86
DYN-W-CANT_LOAD_DAT 112

E

Editing dynamic models 20
Errors
    Compiling dynamic models 111
    Discovering classes and populating attributes 117
    Loading dynamic models 112
Event
    Forwarding to Global Manager 69
Events
    Subscribing to in Global Manager 106
    Subscription by IP Manager 62
    Syntax for adding 81
Exercises
    Location of source files 24

F

Field certification file 35

G

GA_Driver class 97
    Example 98
getClasses command 94
getProperties command 94
Global Console
    Show classes with no instances 33
Global Manager 95
    Adding a new class 38, 50
    Adding a new event subscription 69
    Displaying new attribute 52
    Displaying new event 70
    Forwarding a new attribute 51, 105
    Forwarding a new event 69
    Loading dynamic models 38, 50, 104
    Subscribing to events 106
    Synchronizing topology with IP Manager 95
    Transferring topology 105
Guidelines for developing dynamic models 20

I

ICF-ERESTOREREPOS 114
ics-default.xml 51, 69, 105
Instrumentation class 59
interface keyword 78
IP Manager
    Subscribing to events 62

L

--l10nsrc command line option 88
ldm files
    location for loading 26
LIC-E-ECHECKOUT 111
Loading dynamic models 25
    Example 32, 47, 64
    Into a running Manager 25
    Troubleshooting errors 112
    With the --dynamic option 26
    Without the --dynamic option 26
Localization
    Compiling locales 88
Location of .ldm files 26

M

MODEL
    How it differs from Dynamic Model 18
    Representing systems 121
MODEL and dynamic modeling
    Differences in methodology 122

N

Notifications
    Setting characteristics 107
    Setting subscriptions 106

O

oid2type_Field.conf 35
Operation
    Syntax for adding 83

P

PERL API 95
Polling
    Add attributes to instrumentation class 59
    Starting 73
Polling and Thresholds Console 67
Pragmas 19
    Used in dynamic models 89
Prefix for new classes and properties 20, 76
Problems
    Syntax for adding 82
put command 94

R

refine interface keywords 18, 79
refine keyword
    Example 45
Relationship
    Syntax for adding 81
Requirements for dynamic models 21
Restrictions of dynamic modeling 19

S

Setting
    Ethernet Interface/Port Performance 67

Show classes with no instances 33
sm_server command 25
--dynamic option 76
sm_service 25
sm_service command 77
SM_SITEMOD 21, 31, 38, 46, 51, 61, 104, 112, 115, 116
SNMP_WalkFE 98
SNMPWalk
Parsing results to get value of attribute 100
Source files for the exercises 24
Symptoms
Syntax for adding 83
Synchronizing topology 95

T

Threshold setting
Adding to adapter-setting.asl 63
Verifying 67
Topology
Add Source
IP Manager to Global Manager 39, 53, 71
Problems restoring 114
Synchronizing 95
Transferring
To Global Manager 105

U

Using discovery probing 99