EMC® Smarts® Dynamic Model

7.0

Dynamic Modeling Tutorial

300-003-967
REV A01

EMC Corporation
Corporate Headquarters:
Hopkinton, MA 01748-9103
1-508-435-1000
www.EMC.com

Copyright © 2004, 2006 EMC Corporation. All rights reserved.

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.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on EMC.com

All other trademarks used herein are the property of their respective owners.

Contents

Preface ix

Chapter 1 Introduction

Availability, features, and licensing 1-2
- Availability of dynamic modeling 1-2
- Features of dynamic modeling 1-2
- Licensing 1-2

Differences between MODEL and Dynamic Model 1-3
- refine interface keyword 1-3
- refine keyword 1-3

Restrictions of dynamic modeling 1-4
- Preprocessor macros 1-4
- Pragmas 1-4
- User-defined source code 1-4
- ICIM_ManagedDomain 1-4

Developing dynamic models 1-5
- Guidelines 1-5
- Development cycle 1-6
- Model enhancement examples 1-7

Location and loading of dynamic model files 1-8
- Loading dynamic models into a running Manager 1-8
- Loading dynamic models during Manager startup 1-9

Chapter 2 Exercise 1 - Adding a new printer class

Goals 2-2

Exercise 2-3
- Creating the dynamic model 2-3
- Compiling the dynamic model 2-3
- Loading the dynamic model 2-4
- Verifying that the new class is present (optional) 2-4
- Setting up the printer OID for discovery 2-5
- Reloading the dynamic model 2-6
- Discovering an instance of the SM_Printer class 2-7
- Adding the new class to the Global Manager 2-8

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

Goals 3-2

Exercise 3-3
Creating the dynamic model 3-3
Compiling the dynamic model 3-3
Loading the dynamic model 3-3
Verifying that the new attribute is present in topology 3-4
Discover an instance of Router 3-5
Populating the value of SM_SerialNumber 3-6
Forwarding the new attribute to Global Manager 3-6

Chapter 4 Exercise 3 - Polling for Values of Additional Attributes

Goals 4-2
Exercise 4-3
Creating the dynamic model 4-3
Compiling the dynamic model 4-5
Automatically subscribing to the event in a Domain Manager 4-5
Adding the polled event to the Global Manager 4-5
Loading the dynamic model 4-6
Discovering an instance of the Router class 4-7
Verify variable in Polling and Settings Threshold dialog 4-9
TS_HighOverrunRate event in Global Manager 4-10

Appendix A Guidelines and Syntax for Dynamic Model Files

Adding a class to the model A-2
Determining the parent of the new class A-2
Adding a new class using the interface keyword A-2
Updating an Existing Class in the Model A-3
refine interface keyword A-3
Refining inherited class properties A-4
Adding an attribute A-4
Adding a relationship A-4
Adding an event A-5
Adding an aggregate A-5
Adding a problem A-6
Adding a symptom A-6
Adding an operation A-7

Appendix B dynmodel Command Syntax and Options

Syntax and options B-2
Pragmas B-3

Appendix C Methods of Creating Classes and Populating Attributes

Using dmctl C-2
Command-line mode C-2
Interactively C-2
Batch mode C-3
Synchronizing topology changes with the Global Manager C-4
Populating attributes using an EMC Smarts API C-5
PERL API C-5
Using ASL C-5
Using the EMC Smarts discovery post-processing mechanism C-6
Creating the driver C-6
Editing custom-end-system.asl C-8

Creating the script to parse the value of SM_SerialNumber C-9
Restarting the Manager C-10

Appendix D Updating the Global Manager

Loading the dynamic model enhancements D-2
Transferring topology and events D-3
Transferring topology D-3
Transferring events D-4

Appendix E Troubleshooting

Compilation errors E-2
License server does not support compiling dynamic model files E-2
Loading errors E-3
Dynamic models are not processed when using --dynamic E-3
Duplicate OID entries E-4
Manager will not start without the --dynamic option E-4
Unable to open <ldmfilename> E-5
DYN-E-CLASS_NAME_ALREADY_EXIST E-5
DYN-E-STRUCT_NAME_ALREADY_EXISTS E-6
DYN-E-ENUMERATION_NAME_ALREADY_EXISTS E-6
DYN-E-ATTRIBUTE_NAME_ALREADY_EXISTS E-6
DYN-E-EVENT_NAME_ALREADY_EXISTS E-6
DYN-E-NO_AGGREG_OP E-6
Discovery errors E-7
Class property changes do not appear in Service Assurance E-7
Running the ASL scripts E-7

Appendix F MODEL and Dynamic Models

The *EMC Smarts ICIM Reference* F-2
Representing systems using MODEL F-3
Differences between MODEL and Dynamic Model F-4

Appendix G Understanding the sm_edit Utility

Index i-1

Figures

	Title	Page
1-1	Steps involved in developing dynamic models	1-6
2-1	SM_Printer class in topology of TEST_SERVER	2-5
2-2	Printer discovered as an instance of SM_Printer	2-7
2-3	SM_Printer class in the TEST_SA Global Manager	2-9
2-4	Map view of the SM_Printer instance	2-10
3-1	Serial number attribute added to the Router class	3-5
3-2	Router discovered with the SM_SerialNumber attribute	3-6
3-3	SM_SerialNumber in the TEST_SA Global Manager	3-9
4-1	Interface instance with the new TS_HighOverrunRate event	4-8
4-2	Interface instance with the new attributes before polling	4-8
4-3	Interface instance with the new attributes after polling	4-9
4-4	TS_OverrunThreshold attribute in the Polling and Threshold dialog	4-10
4-5	HiOverrunRate event in Global Manager	4-11
C-1	Resulting topology from dmctl batch file “test_dmctl_batch.txt”	C-4
C-2	Using the discovery post-processing mechanism	C-6
C-3	Modifying custom-end-system.asl	C-8
F-1	MODEL and Dynamic Model methodology comparison	F-5

Preface

This guide presents a tutorial introduction to Dynamic Model. Examples with detailed explanations about developing dynamic models are provided.

Audience for this guide

This guide is intended to introduce EMC Smarts users to the concepts and methods of dynamic modeling. Specifically, it is for those users who want to extend the EMC Smarts Common Information Model™ (ICIM), by either augmenting existing ICIM classes or defining new classes, and for whom a C^{++} compiler is not available for model development. 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 through *SMARTS Educational Services*.

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

Since this guide focuses on dynamic modeling, which is an extension of MODEL, the *EMC Smarts MODEL Reference Guide* should be used together with this document to understand basic EMC Smarts 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 UNIX, this location is: /opt/InCharge<*n*>/<*productsuite*>
- For Windows, this location is: C:\InCharge<*n*>\<*productsuite*>.

The <n> represents the software platform version number. The <productsuite> represents the EMC Smarts product suite to which the product belongs. This location is referred to as BASEDIR/smarts.

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

- UNIX: /opt/InCharge7
- Windows: C:\InCharge7

However, you cannot change the <productsuite> location under the root directory.

For more information about the directory structure of EMC Smarts software, refer to the *EMC Smarts System Administration Guide*.

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 guide may find the following related documentation helpful. It can be found in the BASEDIR/smarts/doc/pdf directory.

Note: These documents are updated periodically. Electronic versions of the updated manuals are available on the Powerlink website: `http://Powerlink.EMC.com`.

- *EMC Smarts Documentation Catalog*
- *EMC Smarts System Administration Guide*
- *EMC Smarts ICIM Reference*
- *EMC Smarts Common Information Model Infrastructure Models Chart*
- *EMC Smarts Common Information Model Application/Business Models Chart*
- *EMC Smarts ASL Reference Guide*
- *EMC Smarts Perl Reference Guide*

Suggestions for searching PDF files

You may search across multiple PDF files using the Adobe Acrobat Reader software.

1. If the documentation is not accessible to all users of the EMC Smarts product suite, copy the contents of the *BASEDIR/smarts/doc/pdf d*irectory to a central location, such as a shared drive on your LAN, that operators and others may view.
2. To search throughout the documentation library, open the Acrobat Reader software.
   1. Choose **Edit** > **Search**, and enter a word or phrase.
   2. In the **Where would you like to search** option, select **All PDF Documents in** and type the pathname of the location where the PDF documents reside.

If you have more than one EMC Smarts product suite installed, you can set up cross-product document searches by copying files from the BASEDIR/smarts/doc/pdf directory for each product suite into this common documentation directory path.

Conventions used in this guide

EMC uses the following conventions for notes and cautions.

Note: A note presents information that is important, but not hazard-related.

IMPORTANT

An important notice contains information essential to the operation of the software.

CAUTION

A caution contains information essential to avoid data loss or damage to the system or equipment. The caution may apply to hardware or software.

Typographical conventions

EMC uses the following conventions in this guide:

Normal font	In running text: • Interface elements (for example, button names, dialog box names) outside of procedures • Items that the user selects outside of procedures • Java classes and interface names • Names of resources, attributes, pools, Boolean expressions, buttons, DQL statements, keywords, clauses, environment variables, filenames, functions, menu names, utilities • Pathnames, URLs, filenames, directory names, computer names, links, groups, service keys, file systems, environment variables (for example, command line and text), notifications
Bold	In procedures: • Names of dialog boxes, buttons, icons, menus, fields, and maps • Selections from the user interface, including menu items, and field entries • Key names • Window names • Code examples • Commands • User interface tabs
Italic	Used for: • Full publications titles referenced in text In procedures: • Directory paths • Filenames • Scripts • Important/unique words • URLs • Utilities • Variables
`Courier`	Used for: • System output • User entry
< >	Angle brackets used to enclose parameter or variable values supplied by the user.
[ ]	Square brackets used to indicate optional values.
|	Vertical bar used to indicate alternate selections. The bar means “or”.

{ }	Braces used to indicate content that you must specify (that is, x or y or z).
...	Ellipses used to indicate nonessential information omitted from the example.
%	Used to indicate a C shell prompt.
#	Used to indicate a C shell superuser prompt.
▼▲	Used to indicate a command is wrapped over one or more lines. The command must be typed as one line.

Pathname conventions

Directory pathnames are shown with forward slashes (/). Users of the Windows operating systems should substitute back slashes (\) for forward slashes.

Graphical conventions

If there are figures illustrating consoles in this document, they represent the consoles as they appear in Windows. Under UNIX, the consoles appear with slight differences. For example, in views that display items in a tree hierarchy such as the Topology Browser, a plus sign displays for Windows and an open circle displays for UNIX.

Smarts Manager

Unless otherwise specified, the term Smarts Manager is used to refer to EMC Smarts programs such as Domain Managers, Global Managers, and adapters.

Where to get help

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

Product information — For documentation, release notes, software updates, or for information about EMC products, licensing, and service, go to the EMC Powerlink™ website (registration required) at:

http://Powerlink.EMC.com

Technical support — For technical support, go to EMC Customer Service on Powerlink. To open a service request through Powerlink, you must have a valid support agreement. Please contact your EMC sales representative for details about obtaining a valid support agreement or to answer any questions about your account.

Sales and customer service contacts — For the list of EMC sales locations, please access the EMC home page at:

http://EMC.com/contact

1 Introduction

Dynamic Model is an extension of the 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 a Domain Manager 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, at runtime when the dynamic models are loaded into a Domain Manager.

This chapter contains the following information:

- Availability, features, and licensing 1-2
- Differences between MODEL and Dynamic Model 1-3
- Restrictions of dynamic modeling 1-4
- Developing dynamic models 1-5
- Location and loading of dynamic model files 1-8

Availability, features, and licensing

This section describes the different products that currently support Dynamic Model 7.0. It also gives an overview of the features that are available, as well as the licenses that are required.

Availability of dynamic modeling

Dynamic Model 7.0 is supported on EMC Smarts IP Management Suite 7.0, and EMC Smarts 7.0 Service Assurance Management Suite. It is not currently available for other Managers.

Features of dynamic modeling

The following features are provided by Dynamic Model:

- Classes may be made known to the system at runtime, when they are loaded into a Manager as a dynamic model.
- A dynamic model compiler. After the compiler is invoked, it produces a file with a .ldm extension that can be loaded by a Manager.
- You can load a dynamic model when the Domain Manager is started or after the Domain Manager is running.
- When you load a dynamic model, the existing model class dependencies and the new dynamic model class dependencies are resolved.
- A **new class** may be defined that derives from any existing ICIM class that is part of the ICIM_ManagedElement and ICIM_MetaObject trees. Refer to the *EMC Smarts Common Information Model Application/Business Models Chart* and *EMC Smarts Common Information Model Infrastructure Models Chart* to determine 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 in an exisiting ICIM class.
- You can refine problems and symptoms in an existing ICIM class with a separate license.

Licensing

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 Domain Manager). This license also allows you to load the .ldm files into a Domain 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.

Refer to the *EMC Smarts System Administration Guide* for 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 1-1 on page 1-3 presents a summary of the differences between using MODEL to create a custom solution and using Dynamic Model to enhance an existing solution.

Table 1-1 Differences Between MODEL and Dynamic Model

Feature	Model	Dynamic Model
Define new classes	✓	✓
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

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.

refine keyword

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

Restrictions of dynamic modeling

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

Preprocessor macros

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

Pragmas

Only those pragmas listed in Table B-2 on page B-3, 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.

Developing dynamic models

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

Guidelines

The following guidelines should be followed 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.
- Edit the dynamic model files, using **sm_edit**.
- Dynamic model files can be loaded when a Manager is started or after the Domain Manager is already running.
- Perform a backup on the repository of the Manager before using dynamic modeling in a production environment.
- Precede any new attributes, classes, aggregates or events that are defined with a unique tag, such as “USR_”. You should not use "SM_" as this is already used by EMC Smarts.
- Install the Dynamic Modeling feature on each Manager whose model you want to extend using dynamic modeling. For example, if you want dynamic modeling to create new attributes in a class in an EMC Smarts IP Availability Manager, and you want those attributes reflected in the topology in the Global Manager, you must install dynamic modeling on both the EMC Smarts IP Availability Manager and the Global Manager.

Note: Performance and load changes on the Domain Manager are dependent on the changes and additions that have been made using dynamic modeling.

Development cycle

Several steps are involved in the development cycle of extending a model of a Domain Manager, using dynamic modeling.

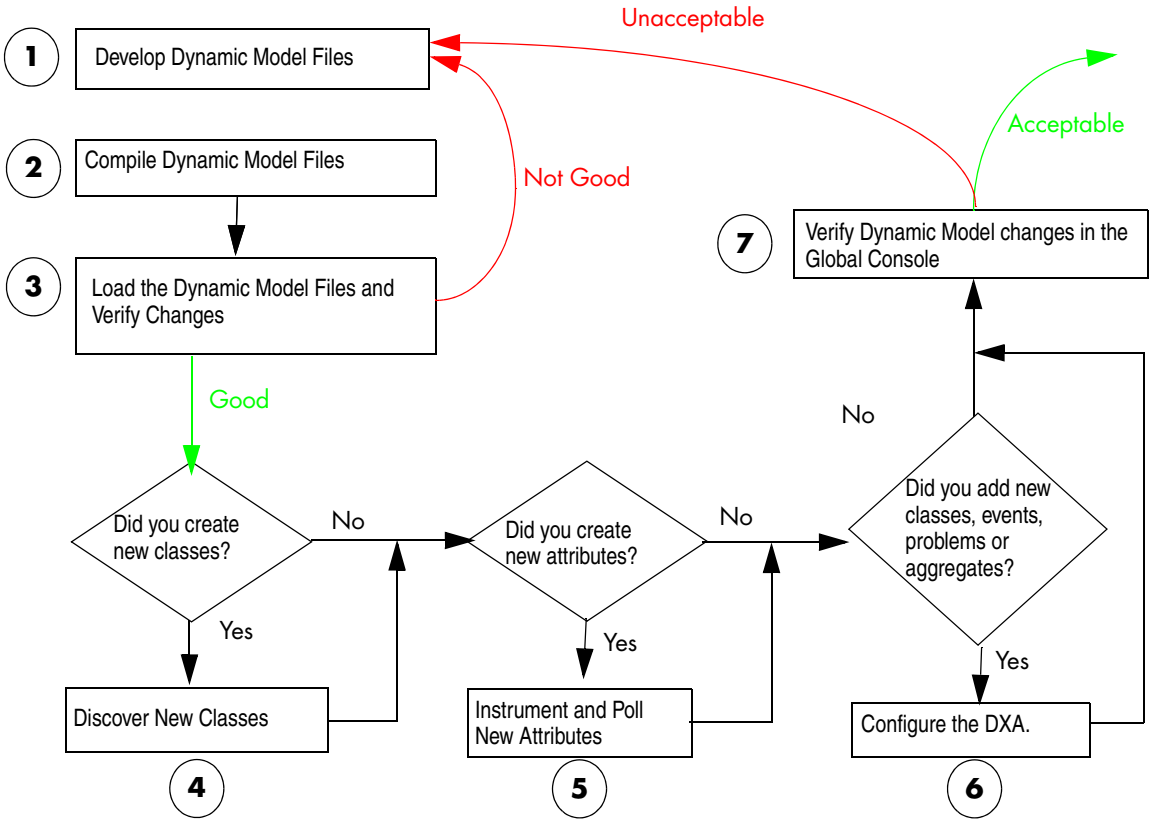


Figure 1-1 Steps involved in developing dynamic models

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

1. Add the new model elements to either a new dynamic model text file or an existing one. Refer to "Guidelines and Syntax for Dynamic Model Files" on page A-1.
2. Compile the dynamic model text file into a file that can be loaded into a Manager. Refer to "dynmodel Command Syntax and Options" on page B-1.
3. Load the compiled dynamic model file into a Domain Manager. Using the Global Console, verify that the new model elements have been added to the list of classes and/or the list of properties.
4. If you have defined new classes, then you need to create or discover instances of those classes. Refer to "Methods of Creating Classes and Populating Attributes" on page C-1.
5. If you have defined new attributes, then you need to populate the values of those attributes, either manually or using the EMC Smarts discovery mechanism. Refer to "Methods of Creating Classes and Populating Attributes" on page C-1.

6. If you want to view the new classes, attributes, problems, aggregates or events in the Global Manager, then configure the appropriate DXA. Refer to "Updating the Global Manager" on page D-1.
7. Verify the topology in the Global Console.

Model enhancement examples

This tutorial will guide you step-by-step through the process of enhancing the model of an EMC Smarts IP Availability Manager. Three different exercises are presented:

- "Exercise 1 - Adding a new printer class" on page 2-1, adds a new printer class to the model, and modifies the appropriate files so that printer instances are created during discovery.
- "Exercise 2 - Adding an Attribute to the Router Class" on page 3-1, adds a new attribute to an existing ICIM class.
- "Exercise 3 - Polling for Values of Additional Attributes" on page 4-1, 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 BASEDIR/smarts/examples/dynamic-model directory.

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.

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 start up. If your Manager is started using **sm_service**, edit the command string used by **sm_service**, to include the appropriate option to load your dynamic models. Refer to the *EMC Smarts System Administration Guide*.

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. Using **dmctl**, enter the following command:

```
▼# BASEDIR/smarts/bin/dmctl -s <domain_manager> -b <broker> invoke
   SM_System::SM-System loadModel <modelfilename> SM_DYNAMIC_LIBRARY▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

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

```
▼# BASEDIR/smarts/bin/dmctl -s TESTDM -b localhost:426 invoke
   SM_System::SM-System loadModel Example3 SM_DYNAMIC_LIBRARY▲
```

Note: You do not need to include the .ldm file extension.

You should verify that the model file was loaded. If you started your Domain Manager with the - - output option, you can check the log file. Your output should be similar to the following:

```
[20-Oct-2006 16:56:07+626ms EDT] t@75 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading
   'C:/InCharge7/IP/smarts/local/model/Example3.ldm'

Property TS_OverrunThreshold initial value = '10'
Setting initial property of TS_OverrunThreshold to 10

Actual initial value for TS_OverrunThreshold = 10

[20-Oct-2006 16:56:07+674ms EDT] t@75 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface
   NetworkAdapter_Performance_Setting
    refined from Example3.mdl:1

Property TS_OverrunThreshold initial value = '10'
Setting initial property of TS_OverrunThreshold to 10

Actual initial value for TS_OverrunThreshold = 10

[20-Oct-2006 16:56:07+678ms EDT] t@75 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface
   Interface_Performance_CiscoRouter
    refined from Example3.mdl:9
```

Accessing the **dmctl** utility is described in "Using dmctl" on page C-2.

Loading dynamic models during Manager startup

If you start a Manager without the **`--dynamic`** option, then only the .ldm files in *BASEDIR*/smarts/local/model /<*config*> will be loaded. In this example, a Global Manager is started without the **`--dynamic`** option.

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

The command above loads all of the .ldm files in *BASEDIR*/smarts/local/model /ics, since ics is the configuration directory specified in the command line.

Including the **`--dynamic`** option on the command line when you start up a Manager, loads all of the .ldm files in BASEDIR/smarts/local/model, in addition to all of the .ldm files in BASEDIR/smarts/local/model /<config>.

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

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

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

2

Exercise 1 - Adding a new printer class

In this exercise you use Dynamic Model to enhance an EMC Smarts IP Availability Manager and Global Manager. A new printer class, with no attributes or other properties, is added to an existing EMC Smarts IP Availability Manager and 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 automatically be discovered, based on the device's OID, when the Manager is started.

Note: This exercise should be performed on test versions of an EMC Smarts IP Availability Manager and Global Manager. Before starting this exercise, backup the repository on the two test Managers.

This chapter contains the following information:

- Goals 2-2
- Exercise 2-3

Goals

In this example you will accomplish several tasks:

- Add a new printer class to the models in an EMC Smarts IP Availability Manager and Global Manager
- Discover instances of the new class using the EMC Smarts IP Availability Manager
- View the new class in the Global Manager topology, using the Global Console

Exercise

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

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

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

A printer is a physical element, on the same level as a router or a host. Therefore, in this example, its parent class is the UnitaryComputerSystem, which is also the router's and host's parent class. A printer, router, and host all inherit the same attributes, events and relationships from the UnitaryComputerSystem class.

In the following steps, the syntax is the same as that used to declare any MODEL class. Refer to "Adding a new class using the interface keyword" on page A-2.

Note: EMC Smarts recommends that a unique prefix be used 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.

Unless otherwise noted, BASEDIR refers to the directory for the EMC Smarts IP Availability Manager (for example, /opt/InCharge7/IP).

Creating the dynamic model

To create the dynamic model, perform these steps:

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

   ```
   # BASEDIR/smarts/bin/sm_edit --create model/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 *Example1.mdl* file. The file is saved in the *BASEDIR/smarts/local/model* directory.

Compiling the dynamic model

From the *BASEDIR/smarts/local/model* directory, issue the command to compile the *Example1.mdl* model file:

```
# cd BASEDIR/smarts/local/model
# BASEDIR/smarts/bin/dynmodel Example1.mdl
```

The dynmodel command does not produce any output to the command-line console, if it is successful. In addition, the Example1.ldm file is created in the BASEDIR/smarts/local/model directory.

If you encounter errors, refer to "Compilation errors" on page E-2.

Loading the dynamic model

In this procedure, TEST_SERVER is the name of the EMC Smarts IP Availability Manager. Replace it with the name of your test Manager.

To load the dynamic model, perform these steps:

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

1. Load the compiled dynamic model file into the test EMC Smarts IP Availability Manager.

   Note: If the test EMC Smarts IP Availability Manager is already running, stop it first.

   ```
   ▼# BASEDIR/smarts/bin/sm_server -n TEST_SERVER -c icf
   --broker=localhost:426 --bootstrap=bootstrap-am.conf
   --dynamic▲
   ```

   Note: ▼▲ Indicates that this text is entered all on one line.

   This example uses *bootstrap-am.conf* as its bootstrap file. It also registers TEST_SERVER with the broker running on the host, "localhost," using port 426. Substitute the bootstrap file, host and port of your broker, if they are different.

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

   ```
   [11-Nov-2004 09:56:37+131ms EST] t@5 ldm file processing
   #1
   DYN-W-FILE_OPEN_OK-Reading
    'BASEDIR/smarts/local/model/Example1.ldm'

   [11-Nov-2004 09:56:39+220ms EST] t@5 ldm file processing
   #1
   DYN-W-CLASS_DEFINITION_CREATE-Interface SM_Printer
   created from
    Example1.mdl:3
   ```

   Note: Since the `--output=<log_file_name>` option is not specified with the **sm_server** command, all output goes to the command console (stdout). If you specify the `--output` option, output will go to a log file.

Verifying that the new class is present (optional)

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

1. Open a Global Console, attaching to the Domain Manager. For this example, attach to TEST_SERVER.
2. Right-click within the Topology Browser Console and select **Select.** Select **Show classes with no instances in current domain**, and select **OK**.

3. In the Topology Browser Console, verify that the SM_Printer class has been added, as shown in Figure 2-1 on page 2-5.

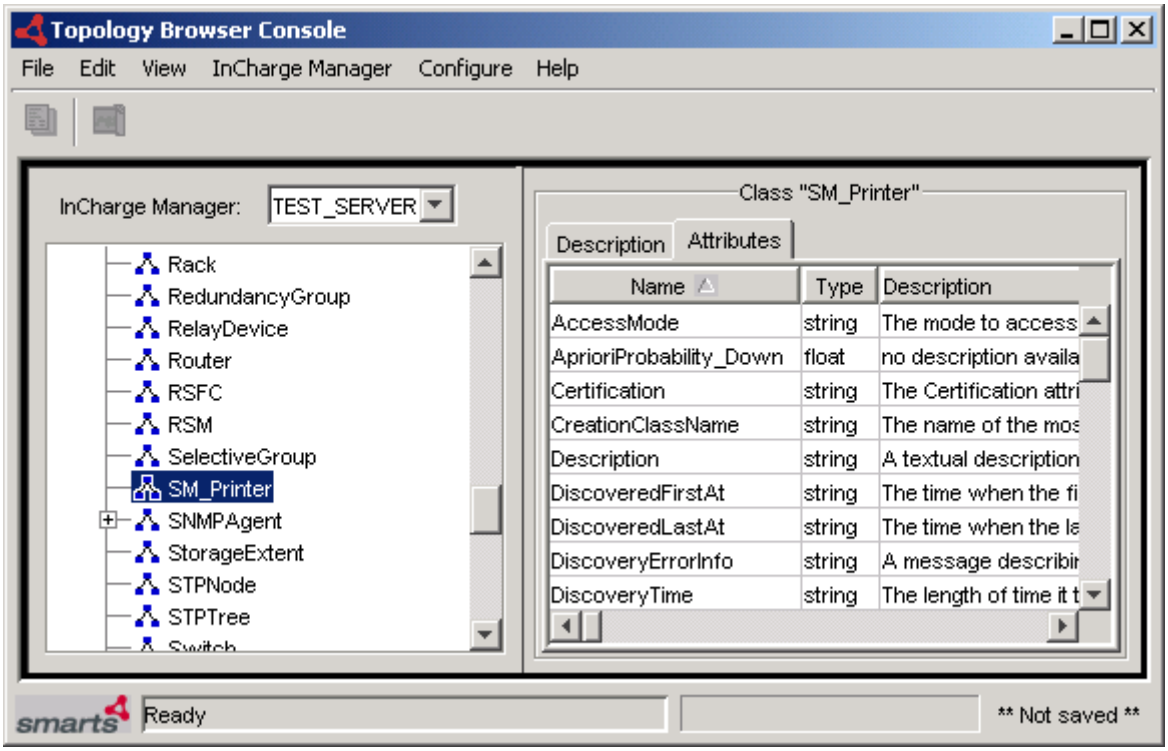


Figure 2-1 SM_Printer class in topology of TEST_SERVER

Setting up the printer OID for discovery

Printers are network devices, but they are classified generically as a Host by EMC Smarts's discovery process. You can, however, add such devices to a field certification file so that EMC Smarts properly recognizes these devices, and classifies them as instances of the new SM_Printer class.

First we need to see if the printer is already defined generically (TYPE = Host), in one of the existing EMC Smarts certification files. EMC Smarts certification files are named oid2type_*.conf and are found in BASEDIR/smarts/conf/discovery/.

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

For example, the printer OID, `.1.3.6.1.4.1.11.2.3.9.1`, is also defined in BASEDIR/smarts/conf/discovery/oid2type_Misc.conf.

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

```
▼# BASEDIR/smarts/bin/sm_edit
conf/discovery/oid2type_Misc.conf▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

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

```
#.1.3.6.1.4.1.11.2.3.9.1 {
#   TYPE = Host
#   VENDOR = HP
#   CERTIFICATION = VALIDATED
#
#INSTRUMENTATION:
#    Interface-Fault= MIB2
#   Interface-Performance= MIB2
#}
```

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

Next, the printer OID must be added to the field certification file.

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

```
# cd BASEDIR/smarts/bin
# sm_edit conf/discovery/oid2type_Field.conf
```

2. Add the following lines to the bottom of the *oid2type_Field.conf* file. These lines will create an instance of the SM_Printer class each time this model printer, designated by the OID (.1.3.6.1.4.1.11.2.3.9.1), is discovered in the network. The TYPE attribute designates this OID as a member of the SM_Printer class.

```
.1.3.6.1.4.1.11.2.3.9.1 {
   TYPE = SM_Printer
   VENDOR = HP
   MODEL = HP 8150DN Network Printer
   CERTIFICATION = TEMPLATE

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

Note: The OID used in this example is for a particular model HP 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 above. Refer to the *EMC Smarts IP Management Suite Discovery Guide* for further information concerning the *oid2type_Field.conf* file.

3. Save and close the *oid2type_Field.conf* file. The modified version of the file is saved to the *BASEDIR/smarts/local/conf/discovery* directory.

Reloading the dynamic model

Since the oid2type_Field.conf file is processed when the Domain Manager is first started, TEST_SERVER must be stopped and restarted for any changes to take effect. If TEST_SERVER is already running:

1. Stop TEST_SERVER.

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

```
▼# BASEDIR/smarts/bin/sm_server -n TEST_SERVER -c icf
--broker=localhost:426 --bootstrap=bootstrap-am.conf
--dynamic▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

This example uses bootstrap-am.conf as its bootstrap file. It also registers TEST_SERVER with the broker running on the host, "localhost," using port 426. Substitute the bootstrap file, host and port of your broker, if they are different.

Discovering an instance of the SM_Printer class

1. Open a Global Console, attaching to the Domain Manager. For this example, attach to TEST_SERVER.
2. Select **Configure> Domain Manager Administrative Console,** and select **TEST_SERVER**.
3. Initiate discovery. One method is to select **Topology > Add Agent,** and enter either the name or IP address of the printer. In addition, the *EMC Smarts IP Management Suite Discovery Guide* describes various methods of discovery.
4. Verify that the printer has been added to the topology, as an instance of the new SM_Printer class, as shown in Figure 2-2 on page 2-7.

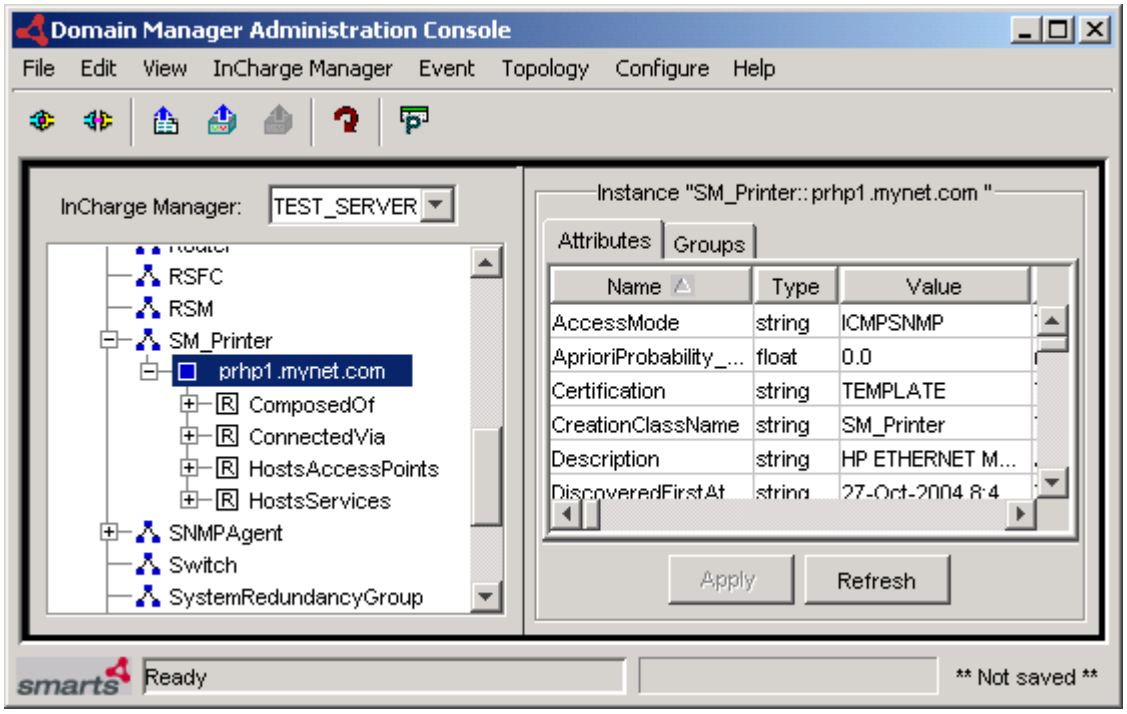


Figure 2-2 Printer discovered as an instance of SM_Printer

Adding the new class to the Global Manager

So far, the new SM_Printer class has only been loaded into the test EMC Smarts IP Availability Manager. The test Global Manager does not contain the new class in its model, therefore, the new class cannot be viewed in the Global Manager. This section explains the steps required to load the dynamic model into the Global Manager, and to transfer the topology information from the TEST_SERVER EMC Smarts IP Availability Manager to the Global Manager.

Note: In this section, BASEDIR is the directory for the Global Manager, (for example, /opt/InCharge7/SAM).

Transferring topology to the Global Manager

The BASEDIR/smarts/conf/ics/ics.conf file is the configuration file for the Global Manager. This file designates which underlying Managers transfer events and topology to the Global Manager.

If your test EMC Smarts IP Availability Manager has not been added to the *ics.conf* file, perform the following steps.

1. Edit the *ics.conf* file, using **sm_edit**. Issue:

```
# BASEDIR/smarts/bin/sm_edit conf/ics/ics.conf
```

2. Since, TEST_SERVER is a test EMC Smarts IP Availability Manager, add it to the DomainType section.

```
DomainSection
{
 DomainType
 { ConfFile          = "dxa-conn.conf";
   MinimumCertainty  = 0.24;
   SmoothingInterval = 65;
#     HookScript        = "ics/dxa-sample-hook.asl";
   Name          = "INCHARGE-AM";
   Name          = "TEST_SERVER";
}
```

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

Loading the dynamic model into the Global Manager

The dynamic model enhancements must be loaded into the Global Manager. Otherwise, the Global Manager has no knowledge of the new SM_Printer class. Perform the following steps:

1. Copy *Example1.ldm* (the version of the dynamic model file that can be loaded into a Domain Manager) from the test EMC Smarts IP Availability Manager's *BASEDIR/smarts/local/model* directory to the test Global Manager's *BASEDIR/smarts/local/model* directory.

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

```
▼# BASEDIR/smarts/bin/sm_server -n TEST_SA -c ics
--broker=localhost:426 --dynamic▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

This example is registering TEST_SA with the broker running on the host "localhost", using port 426. Substitute the host and port of your broker, if they are different.

Verify presence of SM_Printer class in the Global Manager

To verify the presence of the SM_Printer class, perform the following steps:

1. In the Global Console, attach to the TEST_SA Global Manager.
2. In the Topology Browser Console, verify that the SM_Printer class has been added, as shown in Figure 2-3 on page 2-9.

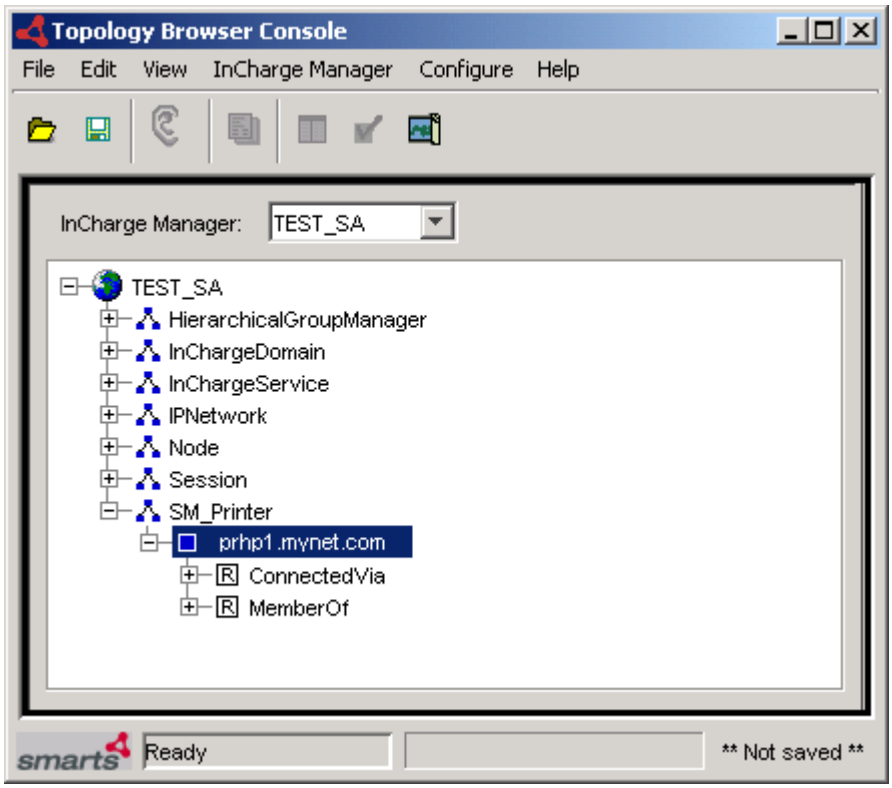


Figure 2-3 **SM_Printer class in the TEST_SA Global Manager**

3. Right-click on the prhp1.mynet.com instance and select **Show Map.** The prhp1.mynet.com instance is displayed in a Map Console, as shown in Figure 2-4 on page 2-10.

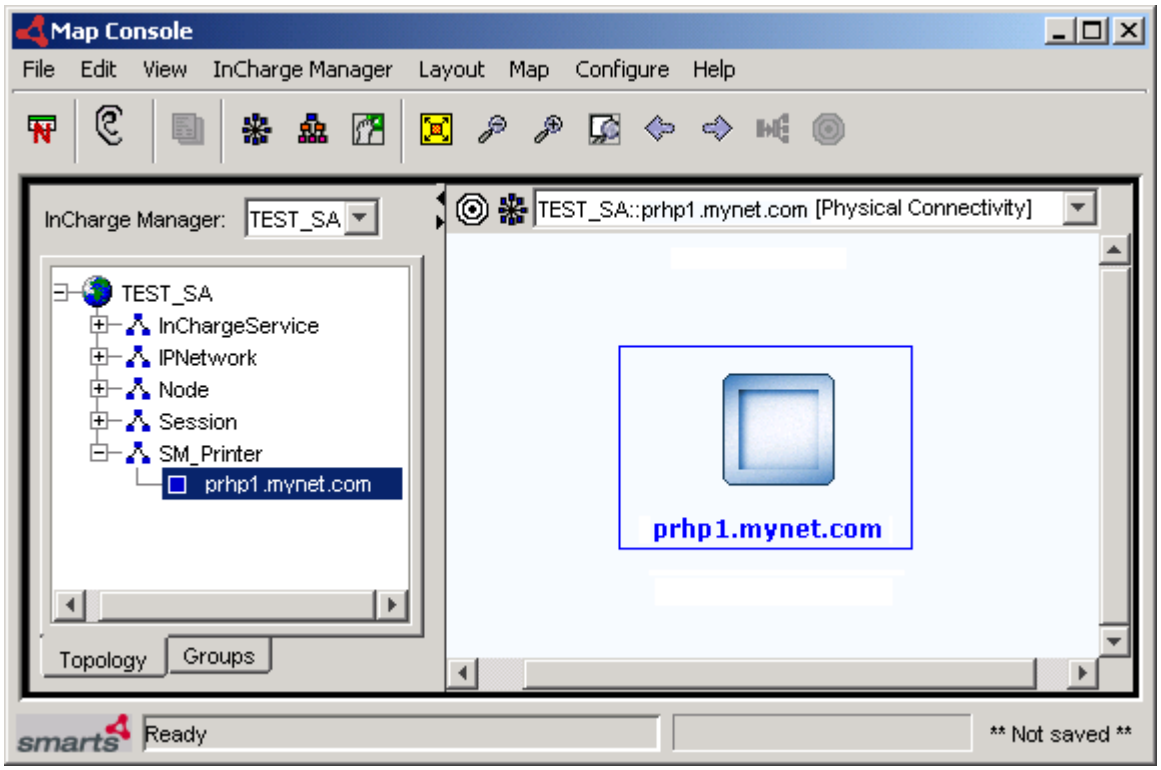


Figure 2-4 **Map view of the SM_Printer instance**

4. If you have a custom icon to represent a printer, you can modify the icon used to display an instance of SM_Printer. Refer to the section, *Custom Icons for Map Nodes* in the *EMC Smarts Service Assurance Manager Configuration Guide*.

3

Exercise 2 - Adding an Attribute to the Router Class

In this exercise, you add a new attribute to an existing ICIM class in an EMC Smarts IP Availability Manager. This exercise also demonstrates how the value of the new attribute can be populated using existing EMC Smarts capabilities.

Note: This exercise should be performed on test versions of an EMC Smarts IP Availability Manager and Global Manager. Before starting this exercise, backup the repository on the two test Managers.

This chapter contains the following information:

- Goals 3-2
- Exercise 3-3

Goals

In this example you will accomplish several tasks:

- Add an attribute to the existing Router class in the EMC Smarts IP Availability Manager model.
- View the new attribute in the Router class, using the Global Console.
- Populate the value of the new attribute in the EMC Smarts IP Availability Manager.
- Load the new Router attribute into the Global Manager model.
- Transfer the value of the new Router attribute from the EMC Smarts IP Availability Manager to the Global Manager.

Exercise

This exercise extends the EMC Smarts IP Availability Manager model by adding an attribute for the serial number to the existing Router class.

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

Note: We recommend that a unique prefix be used 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.

Unless otherwise noted, BASEDIR refers to the directory for the EMC Smarts IP Availability Manager, (for example, /opt/InCharge7/IP).

Creating the dynamic model

To create the dynamic model, perform these steps:

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

```
# BASEDIR/smarts/bin/sm_edit --create model/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 *Example2.mdl* file. The file is saved in the *BASEDIR/smarts/local/model* directory.

Compiling the dynamic model

From the *BASEDIR/smarts/local/model* directory, issue the command to compile the *Example2.mdl* model file:

```
# cd BASEDIR/smarts/local/model
# BASEDIR/smarts/bin/dynmodel Example2.mdl
```

The dynmodel command does not produce any output to the command-line console, if successful. In addition, the Example2.ldm file is created in the BASEDIR/smarts/local/model directory.

If you encounter errors, refer to "Compilation errors" on page E-2.

Loading the dynamic model

In this procedure, TEST_SERVER is the name of the EMC Smarts IP Availability Manager. Replace it with the name of your test Manager.

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

1. Load the compiled dynamic model file into the test EMC Smarts IP Availability Manager.

   Note: If the test EMC Smarts IP Availability Manager is already running, stop it first.

   ```
   ▼ # BASEDIR/smarts/bin/sm_server -n TEST_SERVER -c icf
   --broker=localhost:426 --bootstrap=bootstrap-am.conf
   --dynamic▲
   ```

   Note: ▼▲ Indicates that this text is entered all on one line.

   This example uses *bootstrap-am.conf* as its bootstrap file. It also registers TEST_SERVER with the broker running on the host, "localhost", using port 426. Substitute the bootstrap file, host and port of your broker, if they are different.

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

   ```
   [16-Nov-2004 12:56:49 PM+152ms Eastern Standard Time]
   t@2384 ldm file processing #1
   DYN-W-FILE_OPEN_OK-Reading
   'BASEDIR\smarts\local\model\Example2.ldm'

   [16-Nov-2004 12:56:56 PM+894ms Eastern Standard Time]
   t@2384 ldm file processing #1
   DYN-W-CLASS_REFINE_CREATE-Interface Router refined from
   Example2.mdl:1
   ```

   Note: Since the `--output=<log_file_name>` option is not specified with the sm_server command, all output goes to the command console (stdout). If you specify the `--output` option, output will go to a log file.

Verifying that the new attribute is present in topology

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 topology of the TEST_SERVER EMC Smarts IP Availability Manager.

1. Open a Global Console, attaching to the Domain Manager. For this example, attach to TEST_SERVER.
2. Right-click in the Topology Browser Console and select **Select.** Select **Show classes with no instances in current domain,** and select **OK**.
3. Select the Router class in the topology tree to display the attributes in the **Attributes** tab. The SM_SerialNumber attribute appears, as shown in Figure 3-1 on page 3-5.

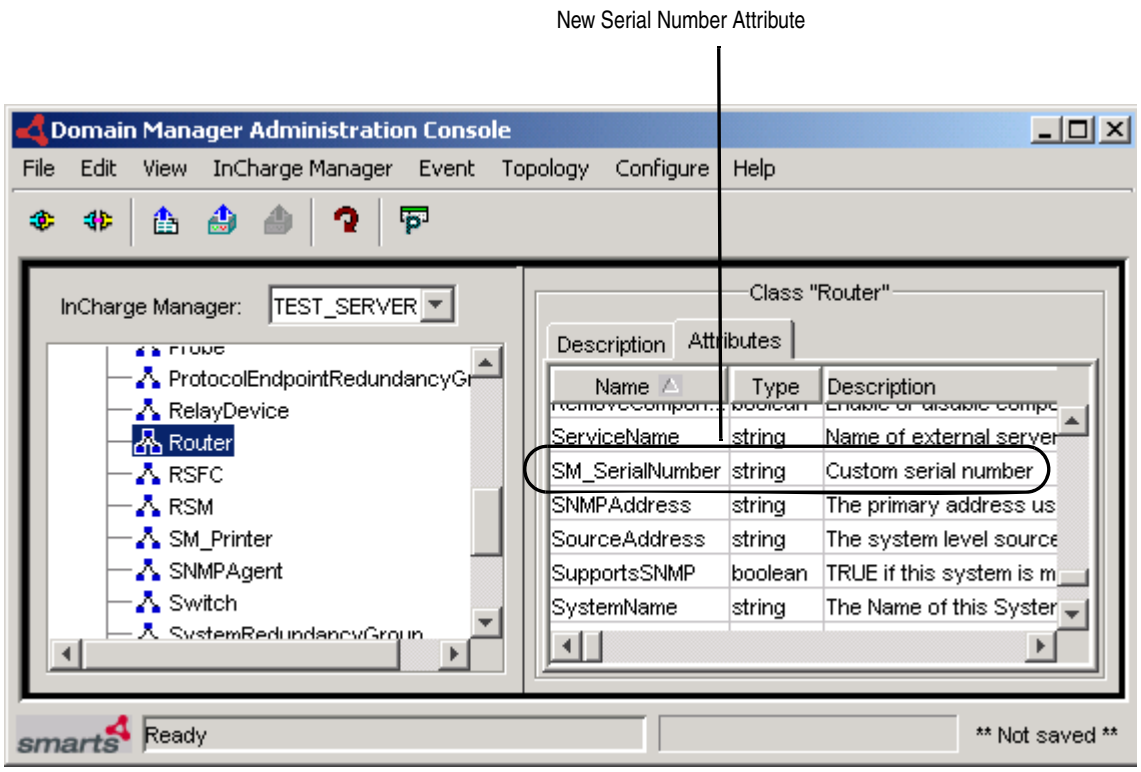


Figure 3-1 **Serial number attribute added to the Router class**

Discover an instance of Router

1. Select **Configure> Domain Manager Administrative Console**, and select **TEST_SERVER**.
2. Initiate discovery. One method is to select **Topology > Add Agent,** and enter either the name or IP address of the router.

   The *EMC Smarts IP Management Suite Discovery Guide* describes 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 3-2 on page 3-6.

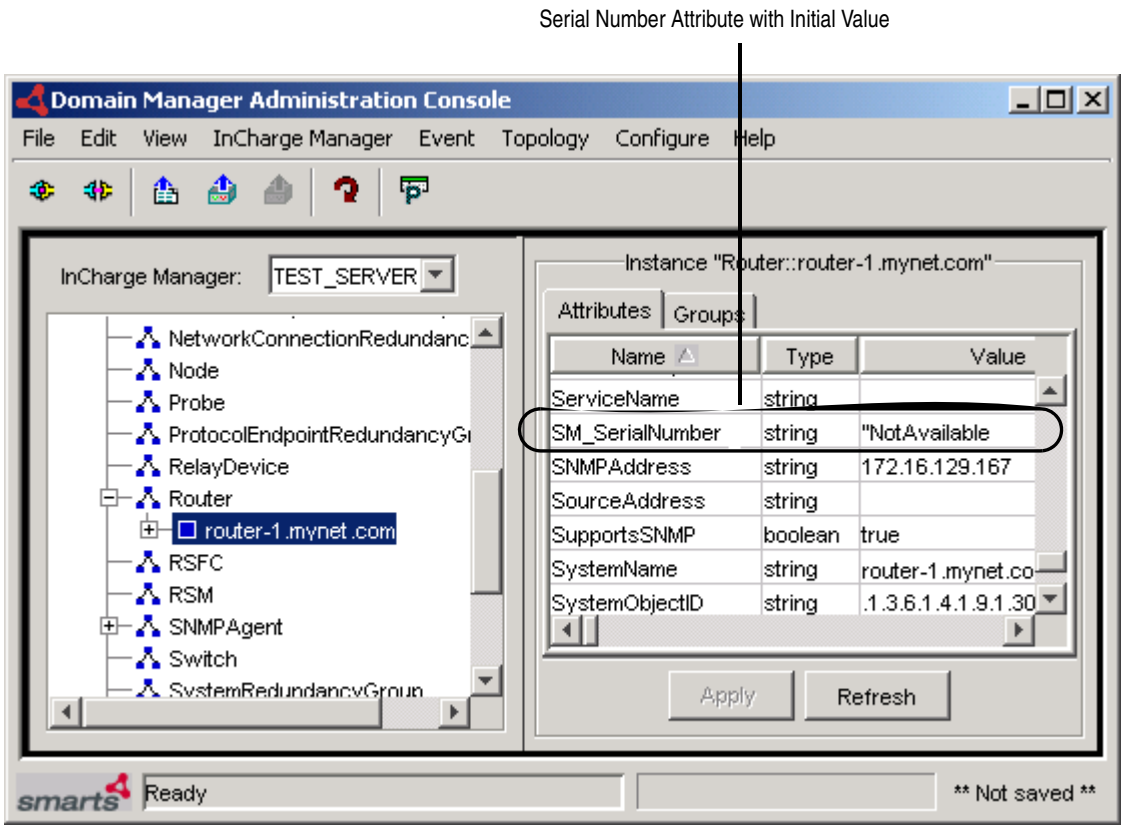


Figure 3-2 **Router discovered with the SM_SerialNumber attribute**

Populating the value of SM_SerialNumber

If you want to populate the values of the new Router attribute, use any of the methods described in "Methods of Creating Classes and Populating Attributes" on page C-1.

Forwarding the new attribute to Global Manager

Note: In this section, BASEDIR refers to the base directory for the Global Manager, (for example, /opt/InCharge7/SAM).

The BASEDIR/smarts/conf/ics/ics.conf file is the configuration file for the Global Manager. This file designates which dxa file is used to transfer events and topology from an individual Manager into the Global Manager.

In this example, dxa-conn.conf is the dxa file that defines which topology and events to transfer from the TEST_SERVER EMC Smarts IP Availability Manager to the Global Manager.

```
DomainSection
{
    DomainType
    {ConfFile           = "dxa-conn.conf";
      MinimumCertainty  = 0.24;
      SmoothingInterval = 65;
#        HookScript        = "ics/dxa-sample-hook.asl";
      Name          = "INCHARGE-AM";
      Name          = "TEST_SERVER";
}
```

1. Edit the *dxa-conn.conf* file, using **sm_edit**:

   ```
   # BASEDIR/smarts/bin/sm_edit conf/ics/dxa-conn.conf
   ```

2. The first section of the *dxa-conn.conf* file provides a listing of what classes, and attributes of those classes, will be transferred from the EMC Smarts IP Availability Manager to the Global Manager.

   Insert the following lines for the Router class (shown in **bold**), before the entry for the ICIM_UnitaryComputerSystem, as illustrated:

   ```
   # Remote Name of Class/Property       Mapped Name of Class/Property

            class   Router
            attr     SM_SerialNumber

            class   ICIM_UnitaryComputerSystem           Node
            attr    Certification
            attr    DisplayName
            attr    Description
            attr    Location
            attr    Model
            attr    PrimaryOwnerContact
            attr    Type
            attr    Vendor
            rel     ConnectedVia
   ```

   Adding these two lines configures the Global Manager to forward attribute changes for the new SM_SerialNumber attribute of the Router class.

   This is the only change that needs to be made to the *dxa-conn.conf* file.

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

Loading the dynamic model into Global Manager

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

In order for the Global Manager to display the new SM_SerialNumber attribute, the dynamic model enhancement must be loaded into the Global Manager.

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

In addition, the Global Manager model does not contain any events or problems. Therefore, if your dynamic model contains events or problems, you need to copy the original .mdl file to the Global Manager's BASEDIR/smarts/local/model directory, remove the events, and recompile it (using dynmodel) to create the .ldm file.

In this example, since only an attribute is added to the Router class, copy and load Example2.ldm into the Global Manager. You do not have to remove the events or problems, or recompile the file.

1. Copy *Example2.ldm* (the version of the dynamic model file that can be loaded into a Domain Manager) from the test EMC Smarts IP Availability Manager's *BASEDIR/smarts/local/model* directory to the test Global Manager's *BASEDIR/smarts/local/model* directory.
2. Load the compiled dynamic model file into the test Global Manager.

Note: *BASEDIR* is the directory for the Global Manager, (for example, */opt/InCharge7/SAM*).

```
▼# BASEDIR/smarts/bin/sm_server -n TEST_SA -c ics
--broker=localhost:426 --dynamic▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

In this procedure, TEST_SA is the name of the Global Manager. Replace it with the name of your test Manager.

Verify presence of SM_SerialNumber in Global Manager

1. In the Global Console, attach to the TEST_SA Global Manager.
2. In the Topology Browser Console, verify that the SM_SerialNumber attribute has been added to the Router class, as shown in Figure 3-3 on page 3-9.

Note: Figure 3-3 on page 3-9 illustrates the SM_SerialNumber attribute with a populated value. If you did not populate the value of the attribute (refer to "Populating the value of SM_SerialNumber" on page 3-6), then the value of SM_SerialNumber will still be "NotAvailable".

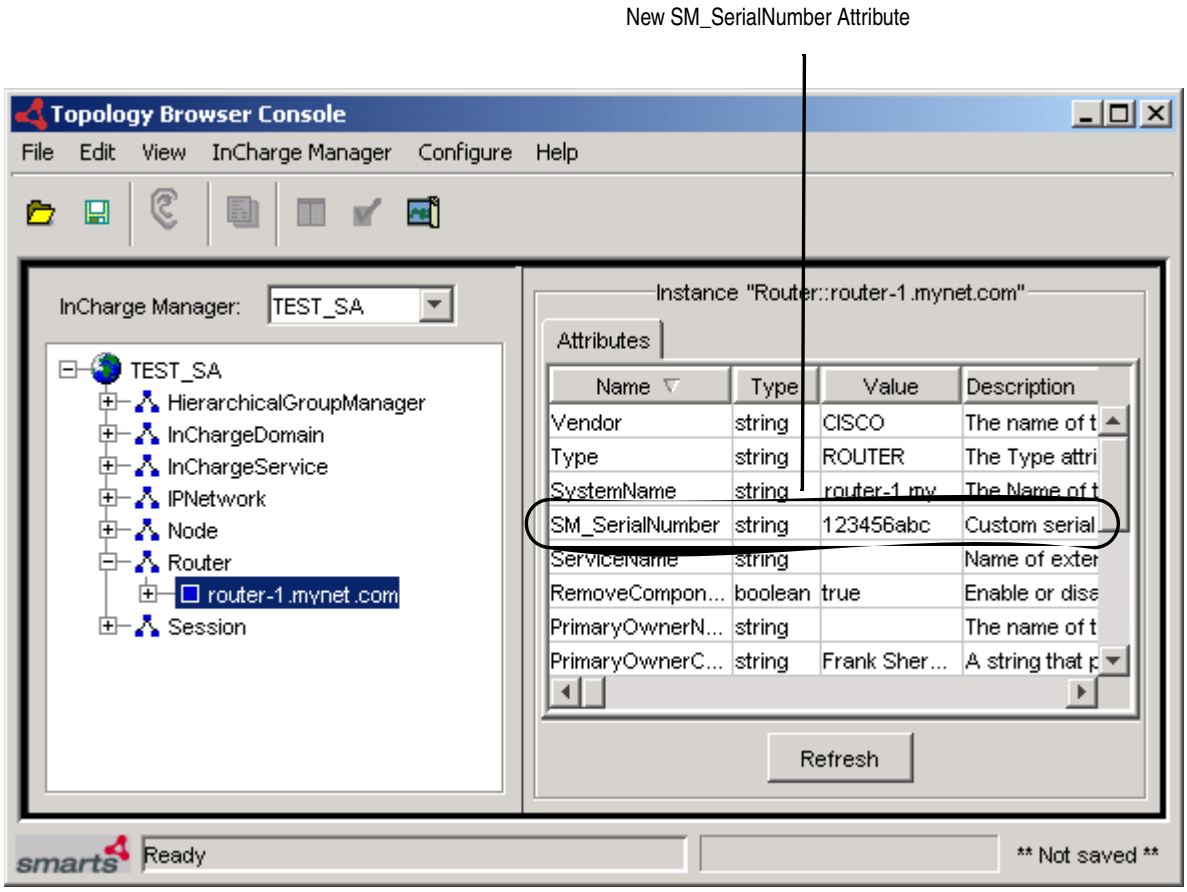


Figure 3-3 SM_SerialNumber in the TEST_SA Global Manager

4

Exercise 3 - Polling for Values of Additional Attributes

This exercise is a more complex example of using Dynamic Model to enhance an existing EMC Smarts IP Availability Manager and Global Manager for the purpose of polling for a new attribute and generating an event based on the value of the attribute. Another goal of this exercise is to forward the new event to a Global Manager so that it can be displayed as a notification.

Note: This exercise should be performed on test versions of an EMC Smarts IP Availability Manager and Global Manager. Before starting this exercise, backup the repository on the two test Managers.

This chapter contains the following information:

- Goals 4-2
- Exercise 4-3

Goals

In this example you will accomplish several tasks:

- Define an additional MIB variable that can be polled by the EMC Smarts IP Availability Manager and EMC Smarts IP Performance Manager.
- Define a setting class for the new attribute.
- Refine an instrumentation class whose function will be to define the MIB OID to poll.
- Poll the values of the new attributes (in real-time).
- Generate an event based on the values of this polled attribute.
- Have the Global Manager subscribe to and display notifications associated with events based on the value of this polled attribute.

Exercise

In order for an underlying Domain Manager to poll for a new attribute and generate an event, the following steps need to be taken:

1. Refine an instrumentation class whose function will be to define the MIB OID to poll.
2. Add an attribute to the existing instrumentation and setting classes. The value of the attribute is returned by the poll performed with the instrumentation class.
3. Modify the dxa configuration file for subscription, and for the event to be forwarded to the Global Manager as a notification.
4. Discover the Cisco Router device(s).
5. Enable polling for the device(s).

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

Note: EMC Smarts recommends that a unique prefix be used with new class names and other properties, since the name must differ from existing ICIM classes and properties. “TS_” is used as a prefix in this example.

Unless otherwise noted, BASEDIR refers to the base directory for the EMC Smarts IP Availability Manager. For example, /opt/InCharge7/IP.

Creating the dynamic model

To create the dynamic model, perform these steps:

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

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

2. Refine the setting class that sets the default value for the alarm threshold. 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 used for monitoring performance characteristics of a network adapter. These include basic parameters common to all network adapters, such as utilization, errors, broadcast, and packet drops.

3. Refine the existing Interface_Performance_CiscoRouter instrumentation class that will be used to poll for the attribute. Add the following lines to the *Example3.mdl* file:

```
refine interface Interface_Performance_CiscoRouter
{
    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;

    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 TrafficRate >= 0 && TS_OverrunErrorPct >
   TS_OverrunThreshold;

   export TS_HighOverrunRate;
}
```

In EMC Smarts, the instrumentation for an object, such as an interface, is generally defined in a separate class 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 "." in the specification of the OID.

The value returned by polling the OID is stored in an attribute defined as an instrumented attribute. Refer to the *EMC Smarts MODEL Reference Guide* for the syntax of an instrumented attribute.

EMC Smarts uses an event clause to define a new event. Events are defined in terms of expressions. In this example, a new EMC Smarts event is defined, called TS_HighOverrunRate. This event is triggered when TrafficRate >= 0 and 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. Refer to the *EMC Smarts MODEL Reference Guide* for details on defining and exporting events.

4. Save the *Example3.mdl* file. The file is saved in the *BASEDIR/smarts/local/model* directory.

Compiling the dynamic model

Compile the *Example3.mdl* model file, by typing the following commands:

```
# cd BASEDIR/smarts/local/model
# BASEDIR/smarts/bin/dynmodel Example3.mdl
```

The dynmodel command does not produce any output to the command-line console, if successful. In addition, the Example3.ldm file is created in the BASEDIR/smarts/local/model directory.

If you encounter errors, refer to "Compilation errors" on page E-2.

Automatically subscribing to the event in a Domain Manager

One of the strengths of EMC Smarts technology is that expressions defined in an event are not evaluated and polled unless there is a subscription to that event. In order for the Domain Manager to automatically begin polling and monitoring for the new event, add a subscription to an import file.

1. Create and edit an import file called *DynamicModelEvents.import*.

```
▼# BASEDIR/smarts/bin/sm_edit --create
conf/icf/DynamicModelEvents.import▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

2. Add the following lines to the *DynamicModelEvents.import*. file:

```
GA_SubscriberProfile::PROFILE-default
{
  subscriptions +=

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

The format is as follows:

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

where the possible values for subscription-level are:

```
#   77879 = problems, aggregates, events
#   77890= aggregates
#   77825= aggregates, events
#   77840= problems, aggregates
```

3. Save and close the import file. The file will be saved in the *BASEDIR/smarts/local/conf/icf* directory.

Adding the polled event to the Global Manager

Note: In this section, BASEDIR refers to the directory for the Global Manager (for example, /opt/InCharge7/SAM).

The BASEDIR/smarts/conf/ics/ics.conf file is the configuration file for the Global Manager. This file designates which dxa file is used to transfer events and topology from an individual Manager into the Global Manager.

In order for the event that is generated by the EMC Smarts IP Availability Manager to appear in the Global Manager, you need to edit the dxa configuration file to subscribe to the new event. Perform the following steps:

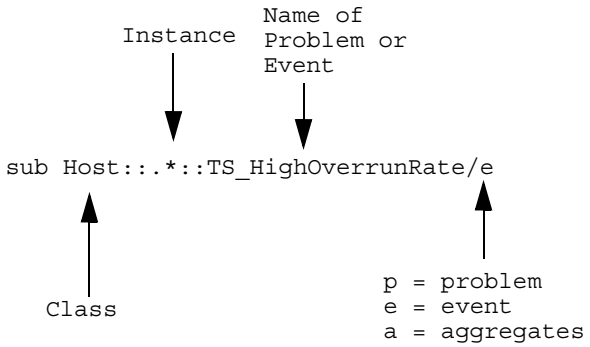
1. Start editing the dxa configuration file used by your test Global Manager. This example uses *dxa-conn.conf*. Type the following at a command prompt:

   ```
   # BASEDIR/smarts/bin/sm_edit conf/ics/dxa-conn.conf
   ```

2. Add the following line to the subscription section of the file.

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

   The format is as follows:

3. You can also set the severity and category of the notification that is created for this event. Refer to "Setting the notification characteristics of the event" on page D-5.

4. Save and close the *dxa-conn.conf* file. The modified file is saved in the *BASEDIR/smarts/local/conf/ics* directory.

Loading the dynamic model

In the following steps, TEST_SERVER is the name of the EMC Smarts IP Availability Manager.

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

1. Load the compiled dynamic model file into the test Domain Manager.

   ```
   ▼# BASEDIR/smarts/bin/sm_server -n TEST_SERVER -c icf
   --broker=localhost:426 --bootstrap=bootstrap-am-pm.conf
   --dynamic --output▲
   ```

Note: ▼▲ Indicates that this text is entered all on one line.

This example uses *bootstrap-am-pm.conf* as its bootstrap file because you need the performance-related monitoring, in addition to the availability-related monitoring. It also registers the TEST_SERVER Domain Manager with the broker running on the host, "localhost," using port 426. Substitute the bootstrap file, host and port of your broker, if different.

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

```
[07-Sep-2006 16:34:53+228ms EDT] t@8 LDM file processing #1
DYN-W-FILE_OPEN_OK- Reading
    'BASEDIR/smarts/local/model/Example3.ldm'


Property TS_OverrunThreshold initial value = '10'
Setting initial property of TS_OverrunThreshold to 10

Actual initial value for TS_OverrunThreshold = 10

[05-Oct-2006 12:53:29+300ms EDT] t@8 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface
   NetworkAdapter_Performance_Setting
    refined from Example3.mdl:1

Property TS_OverrunThreshold initial value = '10'
Setting initial property of TS_OverrunThreshold to 10

Actual initial value for TS_OverrunThreshold = 10

[05-Oct-2006 12:53:30+793ms EDT] t@8 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface
   Interface_Performance_CiscoRouter
    refined from Example3.mdl:10

Property TS_OverrunThreshold initial value = '10'
Setting initial property of TS_OverrunThreshold to 10

Actual initial value for TS_OverrunThreshold = 10

[07-Sep-2006 16:35:11+589ms EDT] t@8 LDM file processing #1
DYN-W-CLASS_REFINE_CREATED- Interface
   NetworkAdapter_Performance_Setting
    refined from Example3.mdl:12
```

Discovering an instance of the Router class

1. Open a Global Console, attaching to the Manager. For this example, attach to TEST_SERVER.
2. In the Global Console, select **Configure> Domain Manager Administrative Console**, and select the **TEST_SERVER** Domain Manager.
3. Initiate discovery. One method is to select **Topology > Add Agent,** and enter either the name or IP address of the router. In addition, the *EMC Smarts IP Management Suite Discovery Guide* describes various methods of discovery.
4. Verify that the router and interface have been added to the topology, with the new TS_HighOverrunRate event in the Interface instance, as shown in Figure 4-1 on page 4-8.

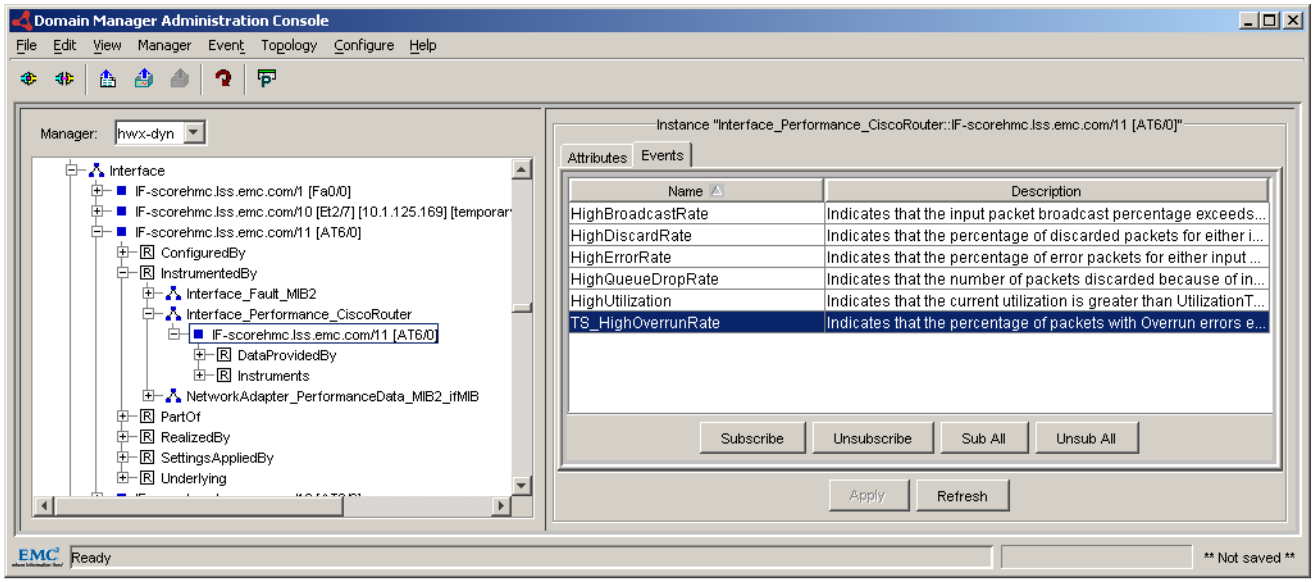


Figure 4-1 **Interface instance with the new TS_HighOverrunRate event**

5. Verify that the TS_locIfInOverrun, TS_locIfInOverrunRate, TS_OverrunErrorRate, and TS_OverrunErrorPct attributes have been added to the Interface instance, using the **Attributes** tab, as shown in Figure 4-2 on page 4-8.

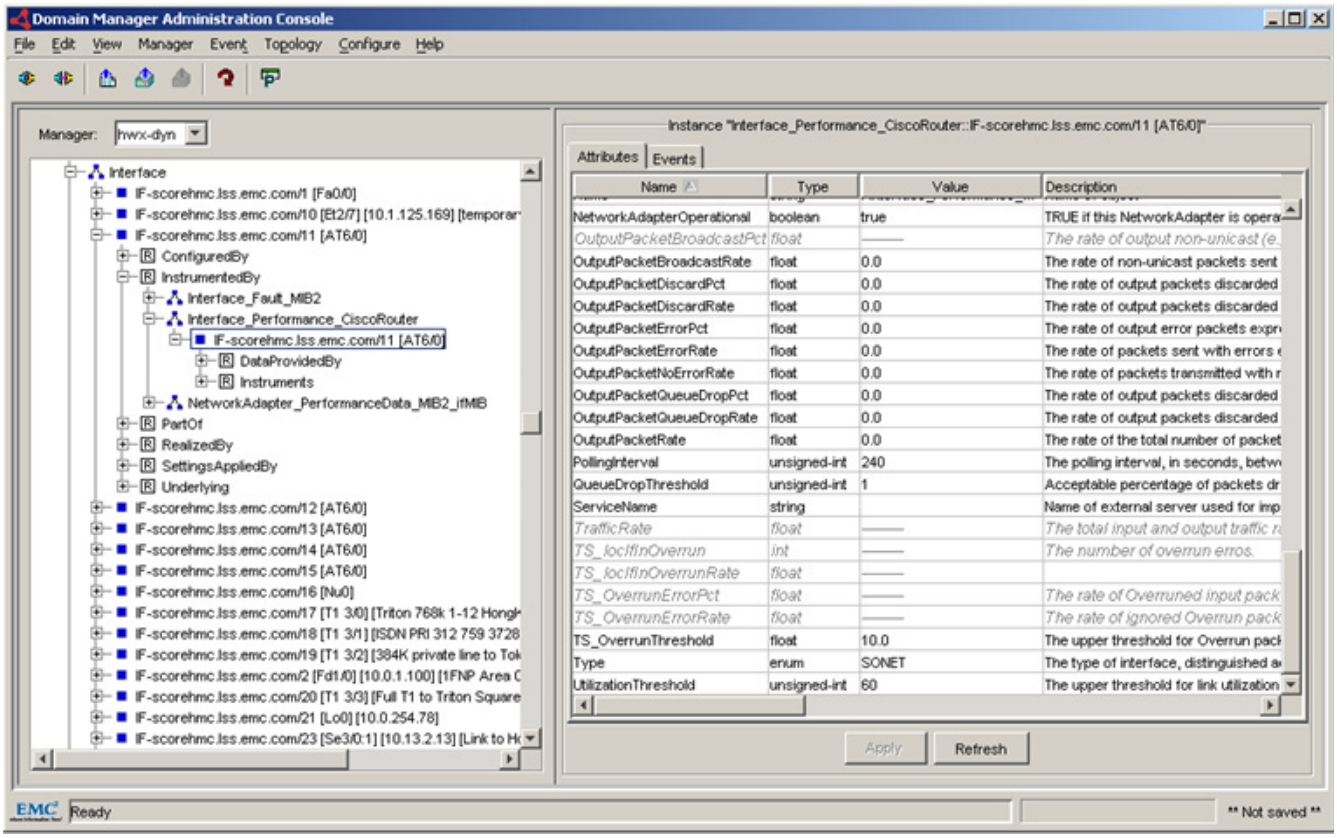


Figure 4-2 **Interface instance with the new attributes before polling**

6. Once polling has started, you will see the values of TS_locIfInOverrun, TS_locIfInOverrunRate, TS_OverrunErrorRate, and TS_OverrunErrorPct update, as shown in Figure 4-3 on page 4-9.

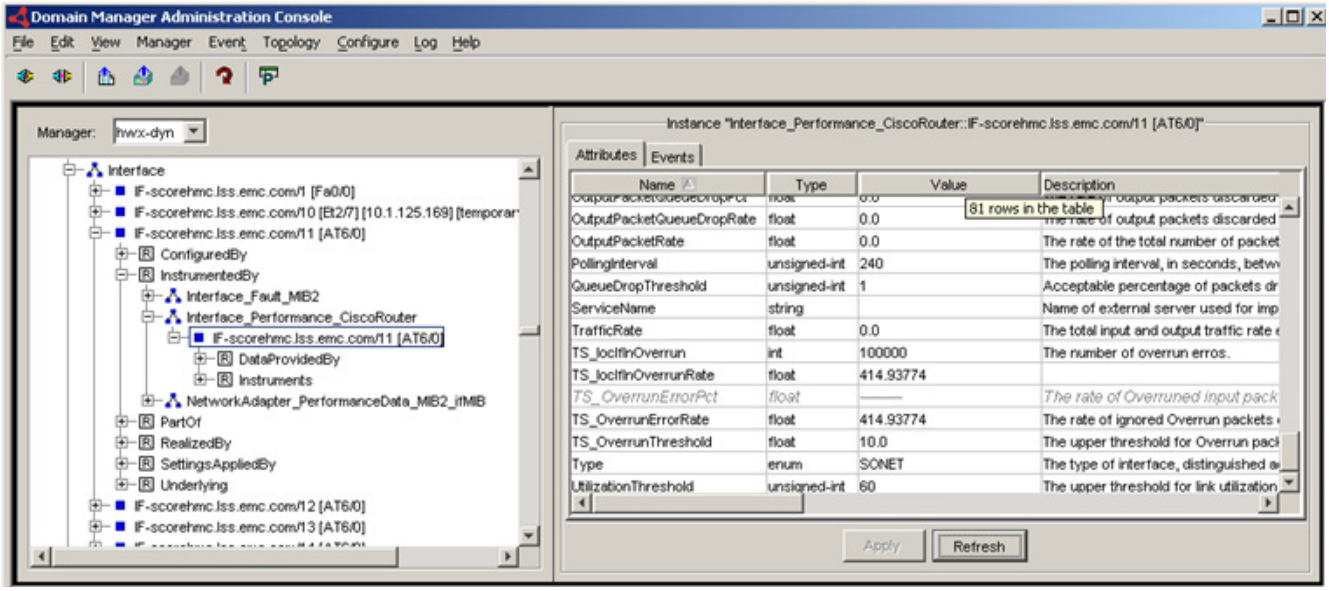


Figure 4-3 Interface instance with the new attributes after polling

Verify variable in Polling and Settings Threshold dialog

In this example, the TS_OverrunThreshold attribute was added by refining the NetworkAdapter_Performance_Setting class. Verify that the TS_OverrunThreshold attribute can be set, using the Polling and Threshold dialog.

The NetworkAdapter_Performance_Setting class defines parameters that are used for monitoring performance characteristics of a network adapter. These include basic parameters common to all network adapters, such as utilization, errors, broadcast, and packet drops.

1. In the Domain Manager Administration Console, select **Configure > Polling and Thresholds**. The **Polling and Thresholds** console opens.
2. In the left panel, select the **Tresholds** tab.
3. In the left panel, select **Interface Groups > Other Interfaces > Settings > Generic InterfacePort Performance**.

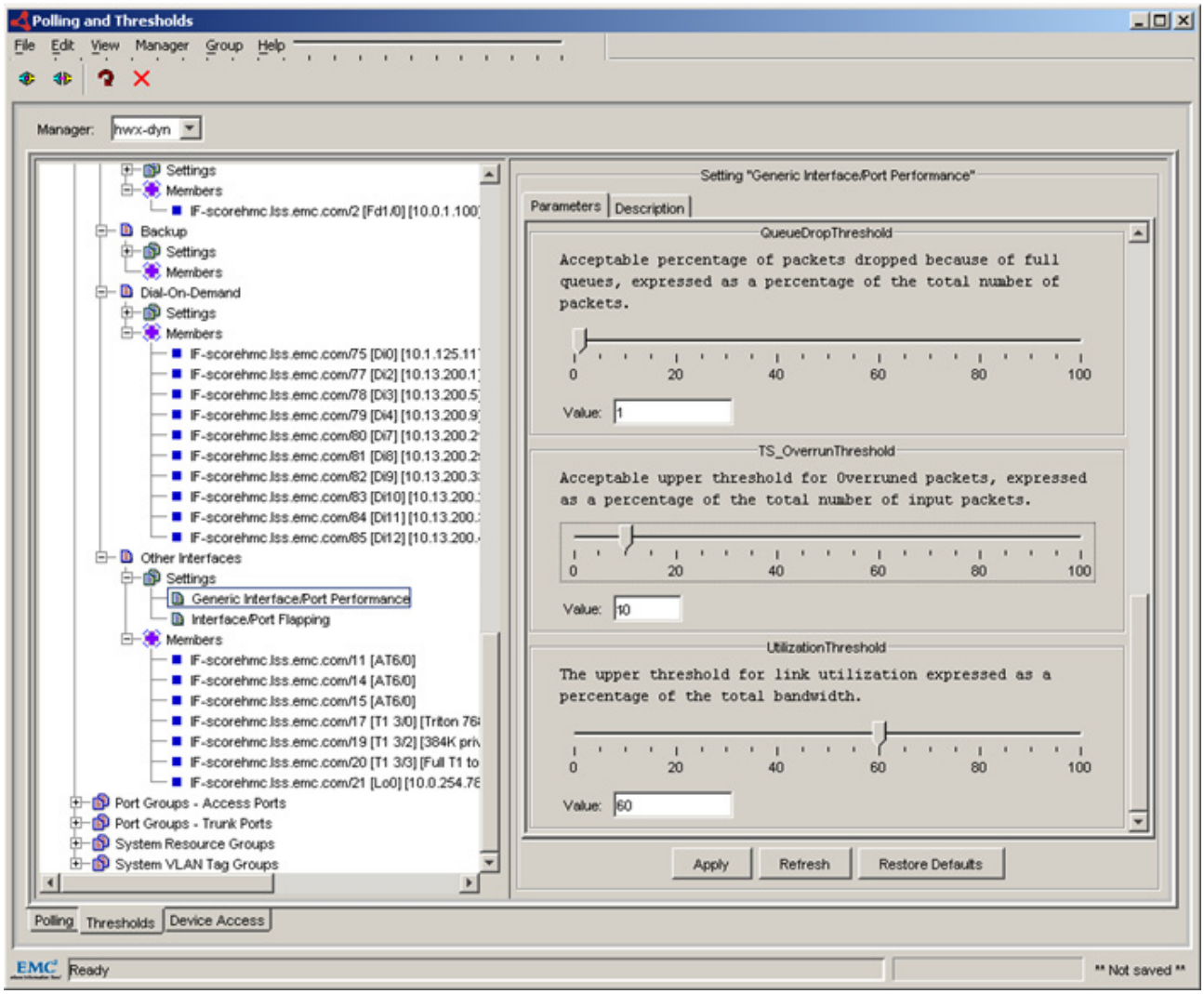


Figure 4-4 **TS_OverrunThreshold attribute in the Polling and Threshold dialog**

TS_HighOverrunRate event in Global Manager

To start the Global Manager, perform the following steps:

1. Start the Global Manager.

   ▼# *BASEDIR*`/smarts/bin/sm_server -n TEST_SA -c ics --broker=localhost:426`▲

   Note: ▼▲ Indicates that this text is entered all on one line.

   The `--dynamic` option is not required for this example, because the Global Manager only subscribes to the TS_HighOverrunRate event. The Global Manager data model does not have to be enhanced to include the event itself.

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

3. If the TS_HighOverrunRate event is triggered by the TS_OverrunErrorPct value, the TS_HighOverrunRate event is transferred to the Global Manager and displayed in the Global Console, as shown in Figure 4-5 on page 4-11.

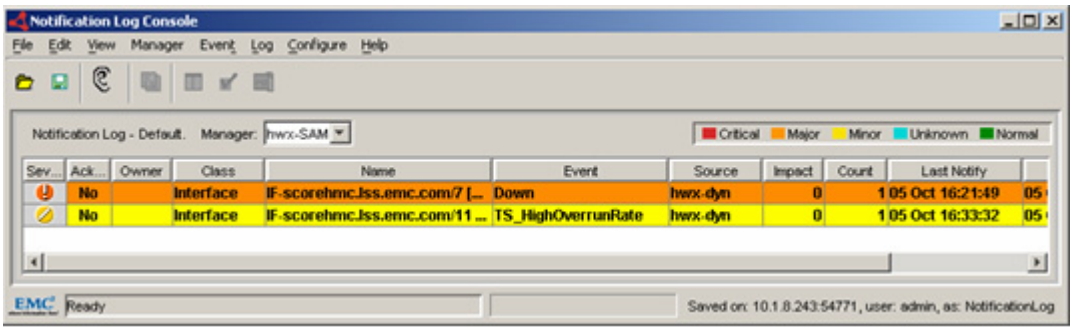


Figure 4-5 HiOverrunRate event in Global Manager

A

Guidelines and Syntax for Dynamic Model Files

A dynamic model file describes the model elements that extend the basic EMC Smarts data model. The following apply to dynamic model files:

- Typically, those classes that are related to each other are contained in the same dynamic modeling file.
- The dynamic model file must have a .mdl filename extension.
- The sm_edit command is used to create and edit the dynamic model files. (Refer to any of the exercises in this guide).
- Dynamic model files must follow the syntax rules described in this section.
- Dynamic model files must be located in either BASEDIR/smarts/local/model or BASEDIR/smarts/local/model/<config>.
- The Dynamic model files must be loaded into each Manager that requires knowledge of the new topology and events.
- When the option `--dynamic` is not added to the sm_server command, it indicates to the Manager that **all** dynamic model files that can be loaded into a Manager (those with extension .ldm) located in BASEDIR/smarts/local/model/<config> are to be loaded and incorporated in the run-time data model.

  Refer to "dynmodel Command Syntax and Options" on page B-1, for information on creating dynamic model files that can be loaded into a Manager.
- When the `--dynamic` option is specified, the Manager will load **all** dynamic model files that can be loaded into a Manager (those with extension .ldm) located in BASEDIR/smarts/local/model and BASEDIR/smarts/local/model/<config>.

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

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

This appendix contains the following information:

- Adding a class to the model A-2
- Updating an Existing Class in the Model A-3

Adding a class to the model

First, the class to be added needs to be declared (see "Adding a new class using the interface keyword" on page A-2). Next, it is compiled into a form that can be loaded into a Manager. Once the model containing 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 representing 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.

Adding a new class using the interface keyword

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>
}
```

Note: EMC Smarts recommends that a unique prefix be used 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.

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 Domain Manager. An existing class can be *extended* by adding any of the following:

- An attribute
- A relationship
- An event
- An aggregate
- A problem
- A symptom
- An operation

However, when a Manager, such as EMC Smarts IP Availability 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.

For example, if the library file contained the following class definition:

```
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.

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;
}
```

Note: 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*.

B

dynmodel Command 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 containing 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.

This appendix contains the following information:

- Syntax and options.....B-2
- Pragmas B-3

Syntax and options

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 B-1 on page B-2. For <mdl-files>, specify one or more files, separated by a space.

Table B-1 Command line options for the dynmodel command

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

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

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

The dynmodel command produces a file with the extension .ldm. The .ldm file has the same base name as the source file. For example, the following command creates two files in the current directory: Asset.ldm and AssetUCS.ldm. Asset.ldm is the version of the dynamic model file Asset.mdl 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.

```
# cd BASEDIR/smarts/local/model
# BASEDIR/smarts/bin/dynmodel Asset.mdl AssetUCS.mdl
```

Note: 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.

Pragmas

Pragmas can be used in dynamic model files to convey SNMP MIB qualities to the dynamic model compiler. Table B-2 on page B-3 lists the available pragmas.

Table B-2 Pragmas used in dynamic modeling

Pragma	Description
#pragma HexNotation	Specifies that an SNMP instrumented attribute is to be converted from hex notation when retrieved from an SNMP agent.
#pragma DotNotation	Specifies that an SNMP instrumented attribute is to be converted from dotted notation when retrieved from an SNMP agent.
#pragma WrapCounter	Specifies that an SNMP instrumented attribute is to be treated as a wrapping counter, which affects comparisons of pervious and current values.
#pragma SNMP table "table_file_name"	Specifies an OID table file, produced by mib2model, which informs the dynamic model compiler about the association between instrumented attribute names and MIB OIDs. Any number of such pragmas can be specified in a dynamic model file. If the same attribute name is given in more than one such file, the file specified last is used.

C

Methods of Creating Classes and Populating Attributes

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

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

- dmctl
- Text file accessed by one of the EMC Smarts APIs (PERL, C, Java)
- ASL scripts
- Working with scripts that are invoked as part of the post-processing phase of the EMC Smarts discovery mechanism

This appendix contains the following information:

- Using dmctl C-2
- Populating attributes using an EMC Smarts API C-5
- Using the EMC Smarts discovery post-processing mechanism C-6

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 user types.

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. Refer to "Synchronizing topology changes with the Global Manager" on page C-4.

Command-line mode

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

```
▼# BASEDIR/smarts/bin/dmctl -s <manager_name><command_name> <command
   parameters>▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

where <server_name> is the name of the Manager.

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

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

Note: ▼▲ Indicates that this text is entered all on one line.

Interactively

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

```
▼# BASEDIR/smarts/bin/dmctl -s <server_name>
   --broker=<host>:<port>▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

where <server_name> is the name of the Manager. The following command, starts dmctl interacting with the TEST_SERVER Manager registered with the broker running on the local host and port 426. At the prompts, enter the username and password.

```
# dmctl -s TEST_SERVER --broker=localhost:426
```

The output will appear similar to the following:

```
>dmctl -s TEST_SERVER --broker=localhost:426
Server TEST_SERVER User: admin
admin's Password: XXXXXXX
Domain Manager Control Program (v6.3) -- Type 'help' for a
   list of commands.
Attached to 'TEST_SERVER'
dmctl>
```

Once attached to the Manager, the create command is used to create an instance of a class:

```
create <class>::<instance>
```

For example, enter the following 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 using the put command:

```
put <class>::<instance>::<property> <value>
```

For example, to enter a value into 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
```

Note: 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 using a text editor and consists of a series of dmctl commands; 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::router-1.mynet.com::SM_SerialNumber 123456abc
```

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

```
▼# BASEDIR/smarts/bin/dmctl --server=TEST_SERVER
   --broker=localhost:426 --file=test_dmctl_batch.txt▲
```

Note: ▼▲ Indicates that this text is entered all on one line.

Figure C-1 on page C-4 shows the two new printers and the new value of the Router serial number in topology.

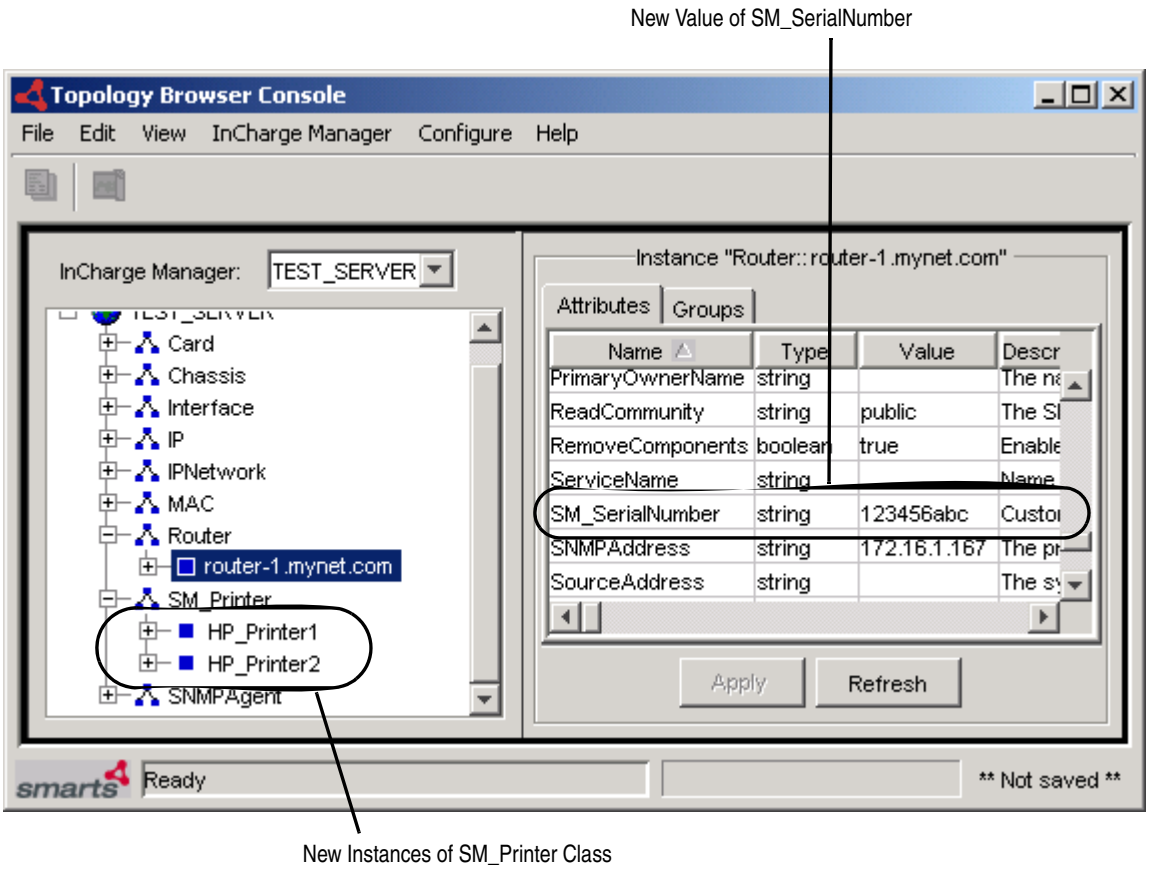


Figure C-1 **Resulting topology from dmctl batch file "test_dmctl_batch.txt"**

Synchronizing topology changes with the Global Manager

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

```
▼# BASEDIR/smarts/bin/dmctl -s <GLOBAL_MANAGER> invoke
   GA_Driver::<Domain Manager>_Topo-Driver start▲
```

Note: ▼▲ Indicates this text is entered all on one line.

where <*GLOBAL_MANAGER*> is the name of the Global Manager, and <*Domain Manager*> is the name of the underlying Domain Manager being synchronized.

For example:

```
▼# BASEDIR/smarts/bin/dmctl -s TEST_SA invoke
   GA_Driver::TEST_SERVER_Topo-Driver start▲
```

Note: ▼▲ Indicates this text is entered all on one line.

Populating attributes using an EMC Smarts API

This section describes how to populate attributes with 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. Refer to the *EMC Smarts Perl Reference Guide* for further information.

```
use InCharge::session;


#Establish a connection to the EMC Smarts Server
$session=InCharge::session->new(server=>"TEST_SERVER_EX2",
                                        broker =>"localhost:426",
                                        user=>"admin",
                                        password=>"changeme");



# 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}";
```

Using 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. Refer to the *EMC Smarts ASL Reference Guide* for further 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();
 }
```

Using the EMC Smarts discovery post-processing mechanism

This example populates the value of SM_SerialNumber (an attribute in the Router class), by querying a MIB. The EMC Smarts post-processing 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 EMC Smarts IP Availability Manager when it is started with the sm_server command.

During a discovery, EMC Smarts automatically runs the custom-end-system.asl script, after Phase III (discovery probing), for each device. 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, retrieving 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 C-2 on page C-6.

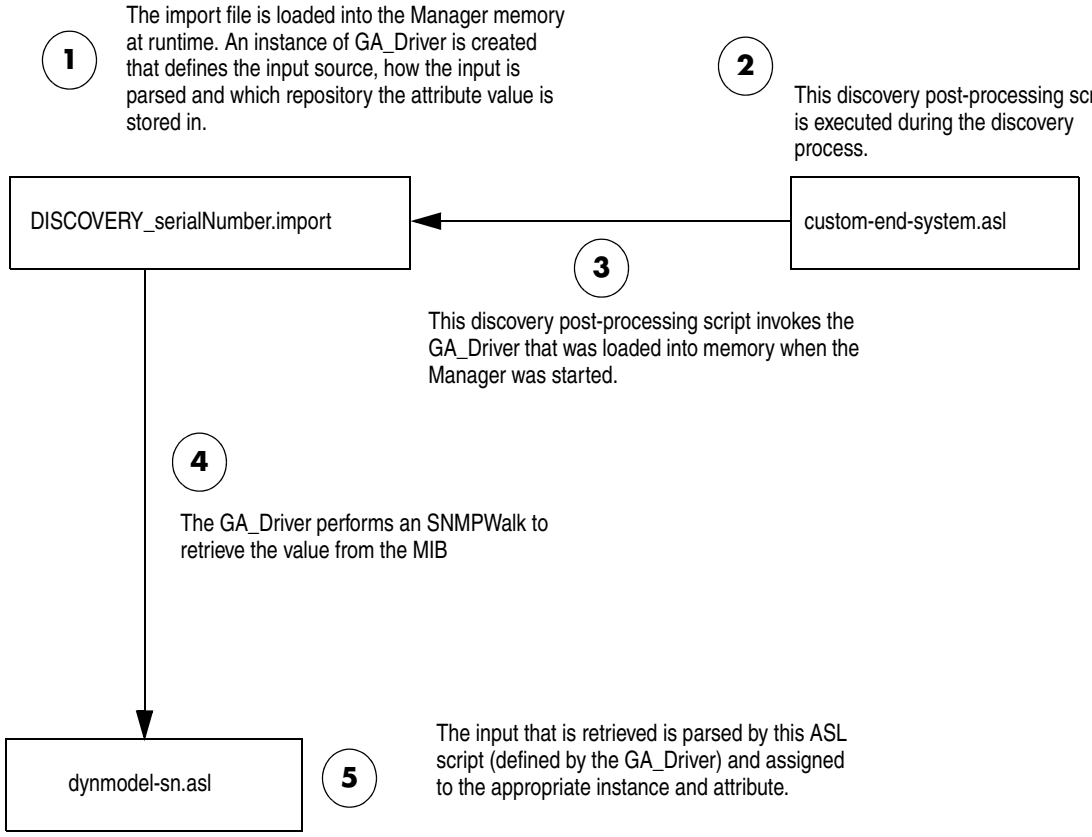


Figure C-2 Using the discovery post-processing mechanism

Creating the driver

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 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 that is loading the import file.

1. Start editing the import file.

```
▼# BASEDIR/smarts/bin/sm_edit --create
conf/discovery/DISCOVERY_serialNumber.import▲
```

Note: ▼▲ This text is entered all on one line.

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
}
```

Note: The value for the root OID in this example is for a Cisco router, Cisco chassis-id MIB (3)

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

   In this import file, an instance of the GA_Driver class is created, named DynModel-SerialNumber-Driver. Since an OID is queried for its value, a SNMP_WalkFE front end is used. The SNMP_WalkFE instance, DynModel-SN-FrontEnd, retrieves all of the MIB objects under a set of root OIDs, converts the values to ASCII and markers, and then 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, EMC Smarts automatically runs the custom-end-system.asl script, after Phase III (discovery probing), for each device. 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 C-3 on page C-8.

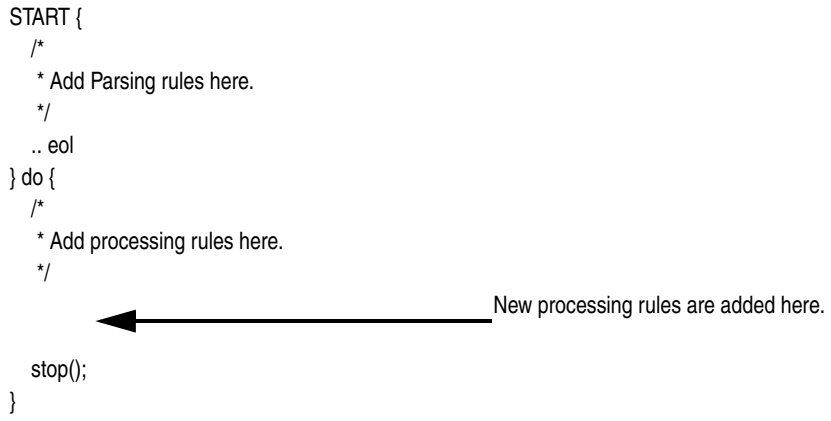


Figure C-3 Modifying custom-end-system.asl

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

```
# cd BASEDIR/smarts/bin
# sm_edit rules/discovery/custom/custom-end-system.asl
```

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

```
/* Check first if the object is a router since we 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 we kick off the GA _Driver that we have defined
   * in the import file whose purpose is to poll the device
   * for some MIB values and return it.   We 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);
}
```

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.

1. Start editing the ASL file.

```
# cd BASEDIR/smarts/bin
# 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 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, we 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();
}
```

Restarting the Manager

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

1. Stop the test Manager.
2. Reload the compiled dynamic model file into the test Manager.

```
# cd BASEDIR/smarts/local/model
▼# BASEDIR/smarts/bin/sm_server -n TEST_SERVER -c icf
--broker=localhost:426 --dynamic▲
```

Note: ▼▲ This text is entered all on one line.

This example registers TEST_SERVER with the broker running on the host, “localhost”, using port 426. Substitute the bootstrap file, host and port of your broker, if they are different.

D

Updating the Global Manager

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

Note: BASEDIR is the directory for the Global Manager, (for example, */opt/InCharge7/SAM*).

This appendix contains the following information:

- Loading the dynamic model enhancements D-2
- Transferring topology and events D-3

Loading the dynamic model enhancements

The dynamic model enhancements are loaded into the Global Manager by performing the following steps:

1. Copy the appropriate dynamic modeling files (*.ldm* files) to the Global Manager's *BASEDIR/smarts/local/model* directory.
2. Load the compiled dynamic model file into the Global Manager. For example:

```
cd BASEDIR/smarts/local/model
▼BASEDIR/smarts/bin/sm_server -n TEST_SA -c ics
--broker=localhost:426 --dynamic▲
```

Note: ▼▲ This text is entered all on one line.

Transferring topology and events

The BASEDIR/smarts/conf/ics/ics.conf file is the configuration file for the Global Manager. This file designates which dxa file is used to transfer events and topology from an individual Manager into the Global Manager.

In this example, BASEDIR/smarts/conf/ics/dxa-conn.conf is the dxa file that defines which topology and events to transfer from the TEST_SERVER EMC Smarts IP Availability Manager to the Global Manager.

```
DomainSection
{
    DomainType
    {ConfFile           = "dxa-conn.conf";
      MinimumCertainty  = 0.24;
      SmoothingInterval = 65;
#        HookScript        = "ics/dxa-sample-hook.asl";
      Name          = "INCHARGE-AM";
      Name          = "TEST_SERVER";
}
```

You must use sm_edit to modify the ics.conf file.

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 existing ICIM class, and you want that attribute to appear in the Global Manager, then the class and attribute must be added to this listing.

Note: Use sm_edit to edit the dxa configuration file.

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

```
# Remote Name of Class/Property  Mapped Name of Class/Property

class   Router
attr     SM_SerialNumber

class   ICIM_UnitaryComputerSystem    Node
attr    Certification
attr    DisplayName
attr    Description
attr    Location
attr    Model
attr    PrimaryOwnerContact
attr    Type
attr    Vendor
rel     ConnectedVia
```

Adding 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 the EMC Smarts IP Availability Manager to appear in the Global Manager, you need to edit the dxa configuration file to subscribe to the new event.

In EMC Smarts, 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 utilizing this value.

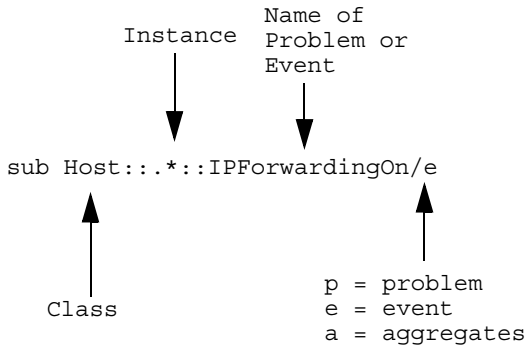
Note: Use sm_edit to edit the dxa configuration file.

Subscribing to the event

The next section of the dxa-conn.conf file provides a listing that specifies the events that are subscribed to.

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

The format is as follows:



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

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

Setting the notification characteristics of the event

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

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

The syntax is:

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

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

- 1 = red, critical
- 2 = orange, major
- 3 - yellow, minor
- 4 = cyan, unknown
- 5 = green, normal

Refer to the *EMC Smarts IP Performance Manager User's Guide* and *EMC Smarts IP Availability Manager User's Guide* for further 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
```

E
Troubleshooting

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.

This appendix contains the following information:

- Compilation errors E-2
- Loading errors E-3
- Discovery errors E-7

Compilation errors

This section describes common problems that you may encounter while compiling dynamic models.

License server does not support compiling dynamic model files

The following error may be encountered when you are trying to compile dynamic modeling files, using dynmodel. This error indicates that the license for DK_DYN_MODEL_COMPILE has not been installed for this product.

```
[inst-sol9: /opt/ramakrv_dynmod/IP/smarts/bin]dynmodel
LIC-E-ECHECKOUT-While attempting to check out license for
   DK_DYN_MODEL_COMPILE  LIC-EGENERIC-License server does not support
   this feature
  . Feature:       DK_DYN_MODEL_COMPILE
  . License path:  17955@fancy:17955@qa-fancy
  . FLEXlm error:  -18,147
  . For further information, refer to the FLEXlm End User
  . Manual, available at "www.macrovision.com".
```

Solution

Install the DK_DYN_MODEL_COMPILE license. Refer to the *EMC Smarts System Administration Guide* for information concerning license administration.

Loading errors

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

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

Dynamic models are not processed when using --dynamic

If the Dynamic Model feature has been installed on a Manager, the output from starting the Manager with the `--dynamic` option will include the following message for each of the dynamic models that are being loaded.

```
DYN-W-FILE_OPEN_OK-Reading
```

You will also see other messages that indicate that the new classes or other refinements in your .ldm files have been processed. For example:

```
[11-Nov-2004 09:56:37+131ms EST] t@5 ldm file processing #1
DYN-W-FILE_OPEN_OK-Reading
    '/opt/InCharge7/SAM/smarts/local/model/Example1.ldm'

[11-Nov-2004 09:56:39+220ms EST] t@5 ldm file processing #1
DYN-W-CLASS_DEFINITION_CREATE-Interface SM_Printer created from
    Example1.mdl:3
```

However, if the Dynamic Model feature has not been installed on a Manager, the output from starting the Manager with the `--dynamic` option does not include the DYN-W-FILE_OPEN_OK-Reading message. You will only receive a message indicating that the dynamic model files are being processed, but they are never opened.

```
processing dynamic model file:
   '/opt/ramakrv_dynmod/SAM/smarts/local/model/dm_model1.ldm'
ICF-N-LICENSEDFEATURE-Enabling the following licensed feature(s):
      InCharge For Service Assurance Manager
      Business Impact Manager
      Application Service Map
      Summary Views
      Status Views
      SAM Viewlets
      SAM Web Console
      Automatic Actions
      Configuration Manager For Service Assurance Manager
```

Solution

Install the Dynamic Modeling feature on each Manager that you want to start with the `--dynamic` option.

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/local/conf/discovery/oid2type_Field.conf so that this printer would be added to topology as an instance of SM_Printer. However, this OID is also defined in another oid2type file.

```
DSCV-W-While loading oid2type files, the following entries are
   duplicate
    .1.3.6.1.4.1.11.2.3.9.1 in file:
   /opt/InCharge7/smarts/local/conf/discovery/oid2type_Field.conf
```

Solution

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

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

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

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

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

```
#.1.3.6.1.4.1.11.2.3.9.1 {
#   TYPE = Host
#   VENDOR = HP
#   CERTIFICATION = VALIDATED
#
#INSTRUMENTATION:
#    Interface-Fault= MIB2
#   Interface-Performance= MIB2
#}
```

3. Save and close the file. The edited copy is saved in the *BASEDIR/smarts/local/conf/discovery/* directory.
4. Restart the Manager, using the `--dynamic` option.

Manager will not start without the --dynamic option

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

```
class: 'SM_Printer' is no longer defined in repos

ICF-N-RESTOREFINISH-PersistenceManager: restore finished at
   13-Dec-2004
    1:44:11 PM Eastern Standard Time
[13-Dec-2004 1:44:10 PM+838ms Eastern Standard Time] t@2548 InCharge
   Framework
ICF-E-EINIT-While initializing the InCharge application
 ICF-ERESTOREREPOS-While attempting to restore topology from file
    C:\InCharge7\IP\smarts\local\repos\icf\TEST_SERVER.rps
 CI-EGENERIC-Can't create SM_Printer::ratbert: SM_Printer
 MR-CLASS_NOT_FOUND-Class of given name not found
[13-Dec-2004 1:44:12 PM+881ms Eastern Standard Time]
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.

In the example above, 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, if you invoke the sm_server command without the `--dynamic` option, 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. However, the instance of SM_Printer still exists in the repository and the system is trying to restore the instance of SM_Printer that was previously discovered, but 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 include the `--dynamic` option when you invoke sm_server.

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

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

Note: ▼▲ Indicates that this text is entered all on one line.

Unable to open <ldmfilename>

The .ldm file, <ldmfilename>, cannot be opened by the Manager. BASEDIR/smarts/local/model directory.

Solution

Verify that <ldmfilename> has the correct read permissions. This error should be pretty rare, since most Managers are started by root (they have administrative privileges).

DYN-E-CLASS_NAME_ALREADY_EXIST

The name of a class declared in the .mdl source file, at the given line number, conflicts with an existing class name in the configuration already loaded.

Solution

You must choose a different class name. Edit the .mdl file, run dynmodel, and restart the Manager with the `--dynamic` option.

DYN-E-STRUCT_NAME_ALREADY_EXISTS

The name of a structure declared in the .mdl source file, at the given line number, conflicts with an existing structure name in the configuration already loaded.

Solution

You must choose a different structure name. Edit the .mdl file, run dynmodel, and restart the Manager with the `--dynamic` option.

DYN-E-ENUMERATION_NAME_ALREADY_EXISTS

The name of an enumeration declared in the .mdl source file, at the given line number, conflicts with an existing enumeration name in the configuration already loaded.

Solution

You must choose a different enumeration name. Edit the .mdl file, run dynmodel, and restart the Manager with the `--dynamic` option.

DYN-E-ATTRIBUTE_NAME_ALREADY_EXISTS

The name of an attribute declared in the .mdl source file, at the given line number, conflicts with an existing attribute name in the configuration already loaded.

Solution

You must choose a different attribute name. Edit the .mdl file, run dynmodel, and restart the Manager with the `--dynamic` option.

DYN-E-EVENT_NAME_ALREADY_EXISTS

The name of an event declared in the .mdl source file, at the given line number, conflicts with an existing name in the configuration already loaded.

Solution

You must choose a different event name. Edit the .mdl file, run dynmodel, and 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 run time, so it does not require an aggregation operator. This message is followed by the class name, propagated property name, and relationshipset name.

```
<classname>:<property>:<relationshipset>
```

Solution

Find the .mdl source file which declares the propagated property and add the aggregation operator to the propagated property declaration.

Discovery errors

This section explains some of the errors that you may encounter while discovering classes and populating attributes.

Class property changes do not appear in Service Assurance

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

Solution

You can synchronize the topology between an underlying Domain Manager and Service Assurance by:

- Initiating a discovery.
- Using dmctl. Refer to "Synchronizing topology changes with the Global Manager" on page C-4.

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
```

F

MODEL and Dynamic Models

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 Domain 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.

This appendix provides:

- An overview of data model concepts and the EMC Smarts implementation of data modeling, using ICIM and MODEL.
- A rationale for dynamic modeling

This appendix contains the following information:

- The EMC Smarts ICIM Reference F-2
- Representing systems using MODEL F-3
- Differences between MODEL and Dynamic Model F-4

The *EMC Smarts ICIM Reference*

The ICIM represents logical and physical components of managed systems across networked technology domains. It is a conceptual view of the problem domain, using an object-oriented representation, that provides Domain 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 especially useful for root-cause analysis are *relationship* and *event*. Relationship expresses the correlation dependencies between classes. Events are those actions that cause transitions within the model.
- The object-oriented *inheritance* construct allows new classes to be derived from existing classes. This is a mechanism for specialization within the model, while preserving model semantics for the new class. New classes *inherit* not only the properties of the base class but its semantics, as well.
- The object-oriented *instance* construct provides a mechanism for separating the model representation of a managed element from its actual implementation. This supports desirable modeling goals, where the model is conceptual and generalized, and independent of any particular implementation within the problem domain. With respect to ICIM, this means that the correlation model, represented by *classes*, is separated from the domain topology, represented by *instances*. ICIM is a true model, then, in the sense that it can accurately conceptualize, or model, particular implementations without prior knowledge.

Representing systems using MODEL

An ICIM class is actually defined by describing it as an interface. This is consistent with the desirable programming practice of information hiding, where program elements are known and fully described by their interface and not by their implementation. To accomplish this, a language is needed to establish syntactic rules for the interface itself and for programs that use the interface. For 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. This also allows tools, such as compilers and analysis programs, to be applied to the model. The MODEL also defines semantic constructs that support the correlation model.

The example shows a simple MODEL interface declaration.

```
interface MyExample : ICIM_ManagedSystemElement
"An interface to generate an event when value > threshold."
{
   attribute int MyValue = 0;

   attribute int MyThreshold = 50;

   event MyValueOverThreshold = MyValue > MyThreshold;
   export MyValueOverThreshold;
};
```

The interface has a name identifier, `MyExample`, and inherits from a base class, `ICIM_ManagedSystemElement`. (The interface may also include a comment describing the interface).

Constructs that support the correlation model include typed attributes, `attribute int`, and events that are computed by and exported from the interface, `event` and `export`.

The MODEL compiler generates the C++ source code needed to access an interface's attributes and to monitor the event conditions and generate an event notification. The generated source code is linked as an executable model library, which is loaded by a Manager whose analysis programs operate on the loaded model.

Add your model source file to the makefile used to build the system. Executing the appropriate 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 an Manager, using sm_server, and specify the model executable to load.

Differences between MODEL and Dynamic Model

A characteristic of the MODEL language and the methodology for developing model using MODEL is that ICIM models are bound to programs that operate on the models at compile time.

This imposes a limitation on applications that are not aware of all of an interface's properties at compile time. This typically occurs because EMC Smarts models tend to be general enough to support a large customer base. A particular customer, for whom more interface properties would be useful, must, however, limit their application programs to only what is made available by the ICIM model.

Dynamic modeling is a step toward solving this problem. It achieves this by allowing the time when binding occurs between the model and the programs that operate on the model to be delayed. Instead of binding at compile time, as with the traditional MODEL, interfaces may now effectively be made known to the system at runtime, when they are loaded into a Manager as a Dynamic Model.

Figure F-1 on page F-5 shows the differences between using MODEL and Dynamic Model. The obvious differences between the process of creating a traditional model executable and creating a Dynamic Model executable is that intermediate C++ source code is not generated and the dynamic model is not linked into the system during the 'build' phase. Instead, the dynamic model compiler is invoked to directly produce a file with a .ldm extension that can be loaded by a Manager. When the Manager is started, and the dynamic model is loaded, the existing model class dependencies and the new dynamic model class dependencies are resolved.

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

Note: 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.

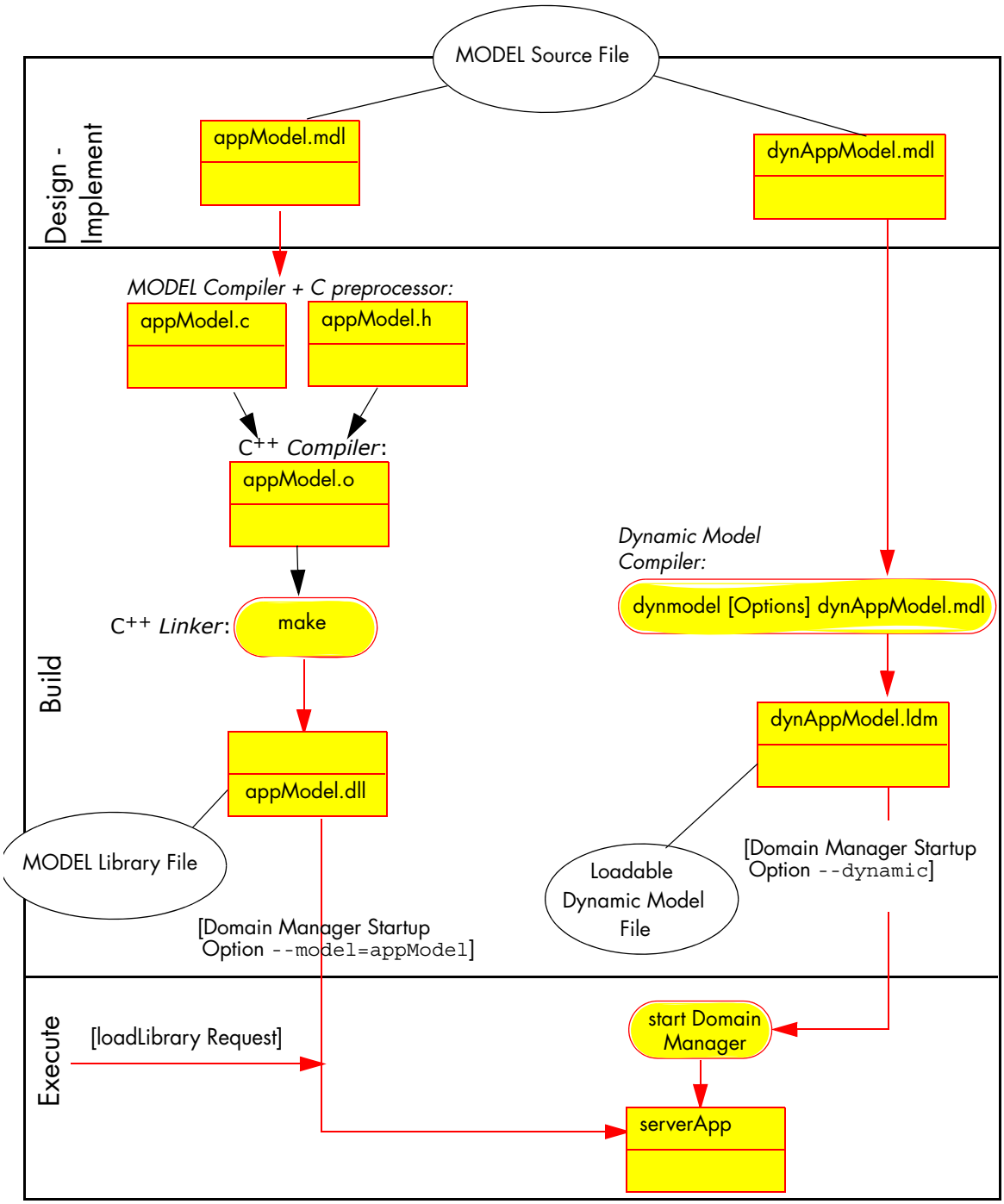


Figure F-1 MODEL and Dynamic Model methodology comparison

G

Understanding the sm_edit Utility

As part of the EMC Smarts deployment and configuration process, you need to modify certain files. User modifiable files include configuration files, rule set files, templates, and files (such as seed files, and security configuration files) that contain encrypted passwords. Original versions of these files are installed into appropriate subdirectories under the BASEDIR/smarts/ directory. For example, on UNIX operating systems the original versions of Global Manager configuration files are installed to /opt/InCharge7/SAM/smarts/conf/ics.

Original versions of files should not be altered. If a file requires modification, it must be stored as a local copy of the file in BASEDIR/smarts/local or one of its subdirectories. For example, a modified ics.conf file should be saved to /opt/InCharge7/SAM/smarts/local/conf/ics. Smarts software is designed to first search for user modifiable files in BASEDIR/smarts/local or one of its subdirectories. If a modified version of a file is not found in the local area, Smarts software then searches appropriate nonlocal directories.

To facilitate proper file editing, EMC Corporation provides the sm_edit utility with every Smarts product suite. When used to modify an original version of a file, this utility automatically creates a local copy of the file and places it in the appropriate location under BASEDIR/smarts/local. This ensures that the original version of the file remains unchanged. You can invoke sm_edit from the command line. Optionally, you can configure Windows so that sm_edit is automatically invoked when user-modifiable files are double-clicked in Windows Explorer.

To invoke the sm_edit utility from the command line, specify the path and the name of the file to edit under BASEDIR/smarts. If multiple Smarts products are running on the same host, invoke sm_edit from the bin directory of the product suite with the files to edit. For example, to edit the configuration file for the Global Manager, invoke the sm_edit utility as follows:

```
# /opt/InCharge7/SAM/smarts/bin>sm_edit conf/ics/ics.conf
```

In this example, the sm_edit utility automatically creates a local copy of the ics.conf file in the /opt/InCharge7/SAM/smarts/local/conf/ics directory and opens the file in a text editor. If a local version of the file already exists, the sm_edit utility opens the local version in a text editor. In addition, sm_edit creates any necessary subdirectories.

The *EMC Smarts System Administration Guide* provides additional information about the sm_edit utility.

Index

- -dynamic option A-1

A

Add device to field certification 2-6
Adding a new attribute to a class 3-3
Adding a new class 2-1
Adding an attribute
    Syntax A-4
Aggregate
    Syntax for adding A-5
ASL scripts C-5
ASL-ERROR_CREATE E-7
ASL-ERROR_PUT E-7
Attribute
    Forwarding to Global Manager 3-6
    Parse results of SNMPWalk C-9
    Populate value with ASL script C-5
    Populating value using dmctl C-2
    Populating with EMC Smarts API C-5

C

Compiling dynamic models
    Example 2-3
    Location of compiler B-1
    Producing a .ldm file B-2
    Troubleshooting Errors E-2
custom-end-system.asl C-8
custom-end-system.asl script C-8

D

Development cycle 1-6
Device
    Add to field certification list 2-5
Differences between MODEL and Dynamic Model 1-3
Discovery import file C-7
DISCOVERY_serialNumber.import C-7
dmctl command C-2
    Batch mode C-3
    Interactive mode C-2
Domain Manager
    Automatically subscribing to events 4-5
dxa-conn.conf 3-7, D-3
    Add new attribute
        Example 3-7
    Setting notification characteristics D-5
    Subscribing to events D-4
    Transferring topology D-3
Dynamic Model
    Features 1-2
    How it differs from MODEL 1-3
    Installing feature 1-5
    Location of compiler B-1
    Location of files A-1
Dynamic modeling and MODEL
    Differences in methodology F-4
Dynamic models
    Add a new attribute 3-3
    Adding a new class 2-1
    Compiling
        Example 2-3
    Creating
        Example 2-3
    Editing 1-5
    Instrumenting attributes 4-3
    Loading 1-8
    Loading into Domain Manager 2-4
    Loading into Global Manager 2-8
    Parent of New Class A-2
    Pragmas B-3
    Prefix for new classes and properties 1-5
DynamicModelEvents.import 4-5
DYN-E-ATTRIBUTE_NAME_ALREADY_EXISTS E-6
DYN-E-CLASS_NAME_ALREADY_EXIST E-5
DYN-E-ENUMERATION_NAME_ALREADY_EXISTS
        E-6
DYN-E-EVENT_NAME_ALREADY_EXISTS E-6
DYN-E-NO_AGGREG_OP E-6
DYN-E-STRUCT_NAME_ALREADY_EXISTS E-6
dynmodel
    Command line options B-2
    Location of compiler B-1
    Producing a .ldm file B-2

E

Editing dynamic models 1-5

EMC Smarts Common Information Model
Description F-2
EMC Smarts discovery post-processing scripts C-6
custom-end-system.asl C-8
Errors
Compiling dynamic models E-2
Discovering classes and populating attributes E-7
Loading dynamic models E-3
Events
Automatic subscription by Domain Manager 4-5
Subscribing 4-6
Syntax for adding A-5

F

Field certification file 2-6

G

GA_Driver class C-7
Example C-7
getClasses command C-3
getProperties command C-3
Global Manager C-4
Adding a new class 2-8
Displaying new attribute 3-7
Fowarding a new attribute 3-6
Loading dynamic models 2-8, D-2
Subscribing to a new event 4-6
Subscribing to events D-4
Synchronizing topology with Domain Manager C-4
Transfer new topology 2-8
Transfer topology from Domain Manager D-3
Transferring topology D-3
Guidelines 1-5

I

ics.conf 3-6
Installing Dynamic Model feature 1-5
Instrumentation Class 4-4
interface keyword A-2

L

Loading dynamic models 1-8
Example 2-4
Troubleshooting Errors E-3

M

MODEL
How it differs from Dynamic Model 1-3
Representing systems F-3
MODEL and dynamic modeling
Differences in methodology F-4

O

oid2type_Field.conf 2-6
Operation
Syntax for adding A-7

P

PERL API C-5
Platforms supported 1-2
Polling
Add instrumentation class 4-4
Pragmas 1-4
Used in dynamic models B-3
Prefix for new classes and properties 1-5
Problems
Syntax for adding A-6
put command C-3

R

refine interface keywords 1-3, A-3
refine keyword
Example 3-3
Relationship
Syntax for adding A-4
Restrictions 1-4

S

sm_server command 1-8
- -dynamic option A-1
sm_service 1-8
sm_service command A-1
SNMP_WalkFE C-7
SNMPWalk
Parsing results to get value of attribute C-9
Supported platforms 1-2
Symptoms
Syntax for adding A-6
Synchronizing topology C-4

T

Topology
Synchronizing C-4

U

Using discovery probing C-8