International Technical Support Organization

SG24-4508-00

Software Distribution for AIX: A Solution for Installation and Configuration of Pristine AIX Environments

February 1996

IBM

International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-4508-00

Software Distribution for AIX: A Solution for Installation and Configuration of Pristine AIX Environments

February 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xv.

First Edition (February 1996)

This edition applies to Releases 1.2, 1.2.1 of NetView DM/6000 and Version 3.1 of Software Distribution for AIX, Program Number 5765-196 for use with the AIX Operating System Version 3.2.5 or higher.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Abstract

This document shows a practical example of how to design and implement a solution to install and configure pristine AIX machines in an automated way. It focuses on the installation and customization of Software Distribution for AIX and necessary communication subsystems, such as SNA Server. Two alternatives for storing the configuration data - the AIX internal database ODM and DB2/6000 - are also explained.

The book is written for customers and IBM personnel who will install a large number of AIX machines and want to take a structured approach to maintain control over a growing environment.

Some knowledge of AIX, including Korn Shell, Software Distribution for AIX and the basic concepts of change and configuration management, is needed.

(464 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help customer staff and IBM personnel to plan and implement large networks of AIX workstations. The information in this publication is not intended as the specification of any programming interfaces that are provided by NetView DM/6000 and Software Distribution for AIX. See the PUBLICATIONS section of the IBM Programming Announcement for NetView DM/6000 and Software Distribution for AIX for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIX/6000
AIXwindows	APPN
DATABASE 2	DB2
DRDA	IBM
InfoExplorer	OS/2
OS/400	RISC System/6000
VTAM	400

The following terms are trademarks of other companies:

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

HP	Hewlett-Packard Company
Motif	Open Software Foundation Incorporated
ODM	Optical Disk Mastering, Incorporated
NFS, Solaris, Sun	Sun Microsystems, Incorporated

Other trademarks are trademarks of their respective companies.

Preface

In this redbook we will show a solution for installing and configuring pristine AIX machines in a highly automated way. The redbook focuses on the installation and customization of the AIX operating system, Software Distribution for AIX and necessary communication subsystems such as SNA server. We also explain two alternatives for storing the configuration data - the AIX internal database ODM and DB2/6000.

The book guides you in getting started in the disciplines of change and configuration management in a new environment. The original design and implementation that is the basis for this book is used in a customer environment and has already proven its usability in covering daily business needs during a roll-out of several hundred machines.

The book is written for technical staff who will install a large number of AIX machines and want to take a structured approach to maintain control over the configuration data in a growing distributed environment. It can also be used by project managers to get an impression of what can be done even if they are not interested in implementation details.

As this book deals intensively with AIX, including Korn Shell, Software Distribution for AIX as well as change management and configuration management aspects, some basic knowledge is assumed in these areas.

Yes or No ?

No: It is a *competitive* solution that covers major aspects of configuration management - for example inventory discovery.

No: The customer is locked in with this solution and cannot take advantage of future configuration management features.

Yes: The data access modules are implemented in a way so that they can be adapted easily to, for example, a configuration database under the SystemView family.

Yes: Once it is implemented and adapted to an environment it saves a lot of time and avoids typical errors that occur through manual configuration.

How This Redbook Is Organized

The redbook is basically divided into two main parts:

- Chapters 2 through 10 set up the framework and explain how to implement this kind of solution with NetView DM/6000 and the ODM database; how to extend it and give you some ideas on how to transfer the test environment to a production environment.
- From Chapter 11 to 16 we will enhance the configuration procedure, use Software Distribution for AIX Version 3.1, use DB2/6000 to store the configuration data and guide you through the process of developing a simple graphical interface to maintain the database.

The chapters are organized as follows:

- Chapter 1, “The Overall Picture” gives you an overview of what will be covered in the following chapters and will also explain the background of why certain parts are designed and implemented the way they are.
- Chapter 2, “Base of Automated Configuration” provides an introduction to the automated configuration, sets the objective for the following chapters and defines which node types will be supported.
- In Chapter 3, “Designing a Data Model for Configuration Data” we design a data model that represents the node types that we want to configure. As for the first start, we use the AIX internal database ODM.
- Chapter 4, “Designing and Implementing the Configuration Procedure” will guide you through the different basic configuration procedure to install and customize NetView DM/6000 and SNA server. The objective is to have an installation that allows you to use means of change management from then on.
- Chapter 5, “Testing the Automatic Configuration Script” shows you the result of the configuration with an example of how a network can be set up and tested to ensure the correctness and usability of the setup.
- Chapter 6, “Using the ODM Editor to Change the Configuration” gives you some hints on how you can use and exploit the ODM database and some of its tools, for example the ODM editor.
- Chapter 7, “Customizing and Extending the Configuration Procedure” advises you if you wish to enhance the configuration procedure and include additional products that need specific customization steps.
- Chapter 8, “Enhancing the Configuration Procedure” guides you through a situation where the data model and certain parts of the procedure must be changed. We use the configuration of an intermediate node scenario as an example.
- In Chapter 9, “Configuring a Production Environment” we explain what needs to be done in order to use this approach in a production environment and show an example of how to develop a roll-out strategy.
- In Chapter 10, “Pristine Installation” we exploit some new features of NetView DM/6000 that allow us to back a system up and install this image as a base system on pristine machines.
- Chapter 11, “Migrating the Procedure to Software Distribution for AIX V3.1” shows you a way to adapt the scenario to the new version of Software Distribution for AIX in order to take advantage of the new features and functions.
- In Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” we transfer the ODM data model to DB2/6000 and use SQL to access the database. You will also find a way to isolate the configuration procedure from the data model in order to keep it as independent as possible.
- Chapter 13, “Testing the Automatic Configuration Procedure with Software Distribution for AIX V3.1 with DB2/6000” gives you an example of the behavior of the system. It shows the definitions to be made in the configuration for a complete scenario and output that you get when you run the configuration procedures on the different node types.

- In Chapter 14, “Converting the Data Model between ODM and DB2/6000” we give you an example of how to start with the implementation on ODM and move it to DB2/6000. It also shows you how you can take advantage of the strengths of both systems and automate the whole process even more.
- In Chapter 15, “Modifying Configuration Data Using a Graphical User Interface” we guide you through an example where we develop a simple graphical interface that allows you to maintain the configuration data base. It is written in such a way that you can easily adapt it to the needs in your environment.
- Chapter 16, “Cloning Systems Using Software Distribution for AIX 3.1” deals with the new functions that are available in Software Distribution for AIX to install an AIX operating system using the change management product.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *NetView DM/6000 R1.2 Concepts and Overview*, GH19-5001
- *NetView DM/6000 R1.2 Installation and Customization Guide*, SH19-5002
- *NetView DM/6000 R1.2 Installation and Configuration Guide*, SH19-5005
- *NetView DM/6000 R1.2 User's Guide*, SH19-5003
- *NetView DMA/6000 V1R1 User's Guide*, SH19-4071
- *NetView DM/6000 R1.2 Message and Error Recovery Guide*, SH19-5004
- *Software Distribution 3.1 for AIX Concepts*, GH19-4161
- *Software Distribution 3.1 for AIX Getting Started*, SH19-4162
- *Software Distribution 3.1 for AIX User's Guide*, SH19-4163
- *Software Distribution 3.1 for AIX Installation and Customization Guide*, SH19-4164
- *DB2/6000 Programming Reference*, SC09-1573
- *DATABASE 2 SQL Reference*, SC09-1574
- *DB2 Call Level Interface Reference and Guide*, SC09-1626
- *AIXwindows Programming Guide*, SC23-2632
- *AIX User Interface Programming Concepts, Volume 1*, SC23-2404

International Technical Support Organization Publications

- *The NetView Distribution Manager/6000 Cookbook, GG24-4246*
- *NetView Distribution Manager/6000 Release 1.2 Agents and Advanced Scenarios, GG24-4490*
- *Software Distribution for AIX: Migration Aspects, GG24-4621* (will be available second quarter 1996)
- *Distributed Relational Database Cross Platform Connectivity and Application, SG24-4311*

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOCAT TXT. This package is updated monthly.

How to Order ITSO Technical Publications

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Most major credit cards are accepted. Outside the USA, customers should contact their local IBM office. For guidance on ordering, send a PROFS note to BOOKSHOP at DKIBMVM1 or E-mail to bookshop@dk.ibm.com.

Customers may order hardcopy ITSO books individually or in customized sets, called BOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain books on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser to the following URL:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. The internal Redbooks home page may be found at the following URL:

```
http://w3.itsc.pok.ibm.com/redbooks/redbooks.html
```

Subscribing to Internet Listserver

IBM redbook titles/abstracts are now available through Internet E-mail via the IBM Announcement Listserver. With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. All it takes is a few minutes to set up a profile, and you can get news (in ASCII format) from selected categories.

To initiate the service, send an E-mail note to:

`announce@webster.ibmlink.ibm.com`

with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

To obtain more details about this service, employees may type the following:

```
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
```

Note: INEWS users can select RelInfo from the action bar to execute this command automatically.

Acknowledgments

This project was designed and managed by:

Wolfgang Geiger
International Technical Support Organization, Raleigh Center

The authors of this document are:

Plamen Kiradjiev
IBM Germany

Stefan Uelpenich
IBM Germany

This publication is the result of a residency conducted at the International Technical Support Organization, Raleigh Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

David Boone
Kathryn Casamento
Robert Macgregor
Linda Robinson
Gail Wojton
International Technical Support Organization, Raleigh Center

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Matteo Bonavita
IBM Network Systems Laboratory, Rome, Italy

Monica Galiano
IBM Network Systems Laboratory, Rome, Italy

Linda Harrell
IBM Network Systems Laboratory, Rome, Italy

Chapter 1. The Overall Picture

This book provides guidance on the installation and configuration of large AIX-based production environments. It focuses on the configuration of NetView Distribution Manager/6000 and the following version of this product called Software Distribution for AIX which is part of the SystemView for AIX product family. It also gives some advice on how to configure large networks of AIX workstations in general.

The book is intended mainly for system designers and administrators who have to plan and implement large networks of workstations. It may also be useful for project managers to get an overview about what effort is required to establish the base infrastructure for a large production environment.

This redbook is based on real customer requirements that have been implemented, tested and used in the "real world" to set up a production environment of several hundred AIX machines. The experiences that were noted during the test phase and usage of the package lead to the different enhancements that are also reflected in the book.

This approach is *not intended* to be:

- A "competitive" solution for any configuration management product that may become part of the SystemView family. This approach shows you a way to get started with configuration and change management and because the access routines are isolated, this solution can be adapted easily to a new, more complex configuration data base.
- A one-size-fits-all solution - meaning that for very large projects you will probably have to extend the package, for example, to better control administrative work from a security point of view.

  Also, for very small projects it may be that it is more efficient to to do certain parts manually to save the (relatively small) overhead that is related to this implementation.

This approach tries to:

- Encourage you to take a planned and systematic approach when you have to install and configure a large network of AIX workstations even if we do not have a full blown configuration database yet
- Show the major advantages when you use a change management product rather than some kind of an installation tool
- Be a guide that allows you to follow step by step to get to a solution that meets production requirements

The effort needed to establish the network infrastructure for a large network is often underestimated. In large, distributed networks it is especially important that the machines to be used at remote locations are configured automatically and can be upgraded while unattended.

The configuration might include several network components, such as network adapters, communication protocols and communication products.

Throughout this book we will design and implement a configuration procedure that can be used to configure the NetView DM/6000/Software Distribution for AIX nodes in our software distribution network from a central configuration server. This procedure will completely configure any node type in our example scenario nearly without any interaction required at the workstations to be configured.

Preparing a software distribution network includes more tasks than just the configuration of the change management product. The reason for this is that NetView DM/6000/Software Distribution for AIX needs some prerequisites, for example, a properly configured base operating system and communications subsystem.

Therefore we define the following general tasks which have to be completed to achieve a ready-to-use software distribution network:

- Installation and configuration of the base operating system
- Installation and configuration of network communication products
- Installation and configuration of the change management product, NetView DM/6000 or Software Distribution for AIX

  It is fully intended to support both versions of NetView DM/6000/Software Distribution for AIX because there are cases, for example, when you have many OS/2 and Windows agents, where you would still take NetView DM/6000 instead of Software Distribution for AIX. For more information on this topic refer to the redbook *Software Distribution for AIX: Migration Aspects, GG24-4621*.

Getting the Code

You can get a copy of all the files needed for this project via anonymous FTP:

- For users outside the IBM network:
  1. Connect to ftp.almaden.ibm.com using FTP user ID anonymous.
  2. Download file /redbooks/GG244508/README.first in ASCII.
  3. Download file /redbooks/GG244508/4508CODE.zip in binary.
- For users inside the IBM network:
  1. Connect to rsserver.itso.ral.ibm.com using FTP user ID anonymous.
  2. Download file /pub/GG244508/README.first in ASCII.
  3. Download file /pub/GG244508/4508CODE.zip in binary.

IBM employees can also request the code by typing:

```
TOOLS SENDTO ROMEPPC LABFORUM NVDMREPO GET 4508CODE PACKAGE
```

The detailed steps needed to configure a specific environment will differ from scenario to scenario. In order to develop our sample configuration we have chosen an example scenario that represents the production environment we want to configure.

In our scenario we will have:

- Two NetView DM/6000/Software Distribution for AIX servers

- A NetView DM/6000/Software Distribution for AIX agent
- A NetView DM/MVS focal point system

The production environment that we will eventually configure includes the following components:

- A central NetView DM/MVS system located at the headquarters and acting as the focal point for all software distributions
- Approximately 320 RS/6000 systems, located at head offices and acting as NetView DM/6000/Software Distribution for AIX servers
- Approximately 2200 RS/6000 systems, located at branch offices and acting as NetView DM/6000/Software Distribution for AIX agents.

The systems are interconnected using different types of networks, including a WAN (Wide Area Network) as well as several LAN (Local Area Network) networks.

In our example configuration procedure we will only configure LAN communications, namely a token-ring network. However, in most production environments there will also be WAN communications, for example, an X.25 network.

This book is divided into several parts:

In the first chapters we will design and implement a configuration procedure based on a sample scenario. This will include the design of a data model which can be used to hold configuration data.

As soon as the configuration procedure has been implemented, we will show how to apply it to our example environment.

In chapter Chapter 6, "Using the ODM Editor to Change the Configuration" on page 99 we show how the configuration database can be easily modified using the AIX ODM editor (`odme`).

The configuration procedure that we describe in this book was developed for this specific project and is therefore most likely to need customization for your own environment. Therefore we have included some guidance on how to adapt and extend the procedure in Chapter 7, "Customizing and Extending the Configuration Procedure" on page 109.

Also we have included Chapter 8, "Enhancing the Configuration Procedure" on page 131 in which we introduce new features to the configuration procedure that were not included in the original procedure.

In chapter Chapter 10, "Pristine Installation" on page 167 we will combine the configuration procedure with the pristine installation procedure supplied with NetView DM/6000 Version 1.2.1 in order to show the completely automatic configuration of a NetView DM/6000 server.

This chapter can also be used separately if you just need information about this new pristine installation feature of NetView DM/6000.

In order to fully benefit from the contents of this book you should have a good understanding of the NetView DM/6000/Software Distribution for AIX product; for example, you should have some experience in using the NetView DM/6000/Software Distribution for AIX command line interface.

Also, you should have at least some knowledge about shell programming, because you will have to adapt the scripts presented in this book to your own needs.

Chapter 2. Base of Automated Configuration

In this chapter we will describe the base of the automatic configuration for nodes in a NetView DM network.

Throughout the book we will use this base to implement an automatic configuration procedure for NetView DM/6000.

2.1 Objective and Overview

Our objective is to configure each node in a NetView DM/6000 network automatically, no matter whether it is a CC server, a CC client or a preparation system.

This process is considered to be part of a complete roll-out for a large number of RS/6000 systems constituting a target network for a production environment.

Consider the following situation: We have a production environment with a large number of RS/6000 systems, let's say about 1000, which need to be installed at several different remote locations. Since it is very time consuming to install all these systems manually, we need procedures to configure each system automatically so that it can be used right after it has been installed.

To do so, several installation and configuration steps need to be performed including the installation of the base operating system, configuration of network adapters, etc. As soon as NetView DM/6000 has been configured automatically, we can perform any additional configuration steps using NetView DM/6000.

This chapter will concentrate on the configuration of NetView DM/6000, but we will also show some methods for the general configuration of pristine RS/6000 systems.

Configuration of NetView DM/6000 nodes in your network might be quite easy if you have only a small number of CC servers. This is because then you can configure your CC servers manually. Furthermore the configuration of a CC client requires only a few steps, basically the customization of the `nvdm.cfg` file and the local target definition on the server, so this could be done by a simple script.

However, the task becomes more challenging as soon as you have a larger number of CC servers on your network. Because configuration of a NetView DM/6000 server is normally more demanding and requires a large number of configuration steps, therefore we did not want to do it manually but use an automatic configuration routine instead.

The configuration routine that we develop in this chapter includes all tasks required to configure any type of NetView DM/6000 node, including those configuration steps not directly related to NetView DM/6000. For example, a connection to NetView DM/MVS requires a working SNA connection, so we will develop a procedure to configure SNA Server automatically so that it will establish an LU 6.2 session to NetView DM/MVS.

It is not intended to cover all configuration possibilities of the product because in most cases you will only have a certain subset. Therefore we show how to extend the configuration script and database in Chapter 7, “Customizing and Extending the Configuration Procedure” on page 109.

The configuration procedure is developed and tested in a test scenario. In order to configure a large production environment we also show methods to configure a "real" production environment.

The transition from the test environment to the production environment is made just by replacing the configuration database the configuration script uses. Therefore we also develop a simple data model to represent the configuration data needed to configure a NetView DM/6000 node.

We include a complete description of every configuration step we perform so that you can use the procedures we create as building blocks for your own configuration scripts.

The following figure shows the environment in which we develop and test our configuration scripts:

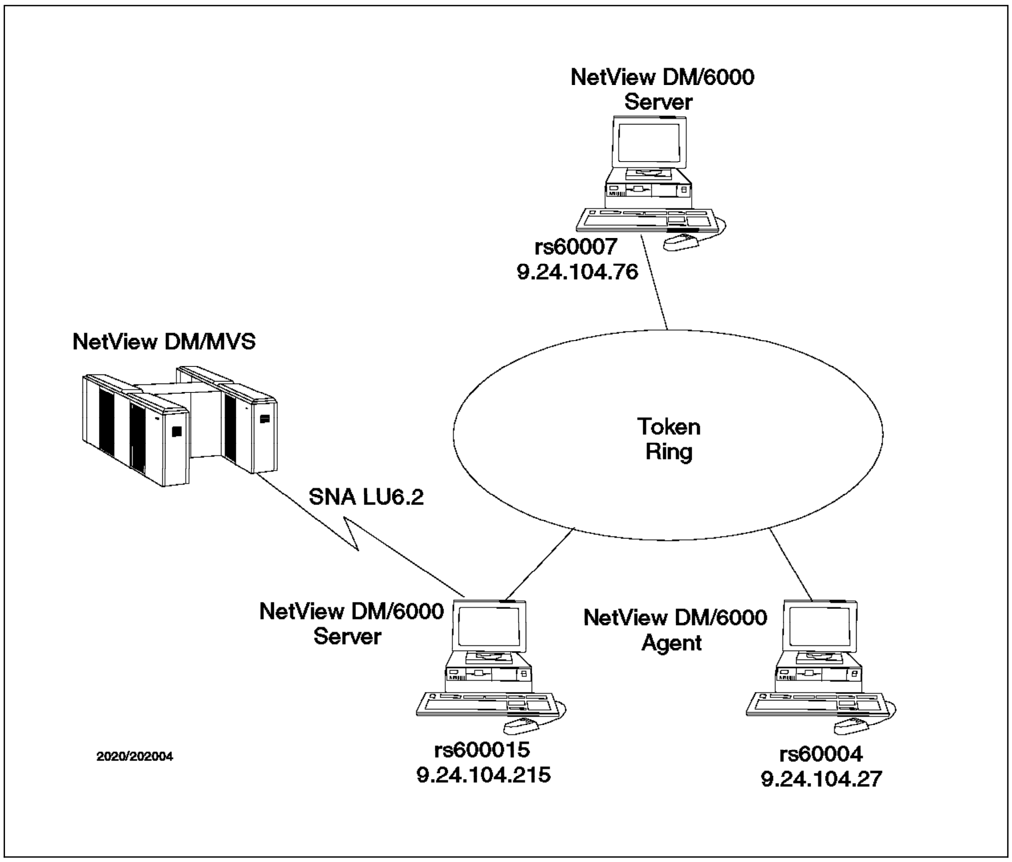


Figure 1. Scenario for Automatic Configuration of NetView DM/6000

Creating an automatic configuration script for NetView DM/6000 is divided into the following tasks:

- Defining which types of nodes we want to configure
- Defining which configuration activities are required for each type of node
- Defining interfaces to other configuration activities and prerequisites
- Designing a data model that represents the configuration

- Designing and implementing a shell procedure for each configuration step
- Creating test data for configuring the test environment
- Testing the automatic configuration script in the test environment
- Showing procedures to replace test data with real production data
- Showing procedures to configure a large NetView DM/6000 network

For this part you should have a good understanding of NetView DM/6000 and the AIX operating system. Depending on which components you need to configure in your specific environment, you should also have at least a basic understanding of these components. For example, if you have SNA connections you should have a basic understanding of how to configure SNA Server.

Also, you should have some knowledge about Korn shell programming as well as of some common UNIX tools, such as `sed`, `awk`, etc.

Note

As far as the configuration of SNA connections is concerned we perform all configuration activities using SNA Server commands. If you intend to use SNA Services instead, you will have to adapt the procedures. If you need guidance on how to configure either SNA Server or SNA Services for use with NetView DM/6000, you should consult the redbook *The NetView Distribution Manager/6000 Cookbook* GG24-4246.

2.2 Defining Which Types of Nodes We Want to Configure

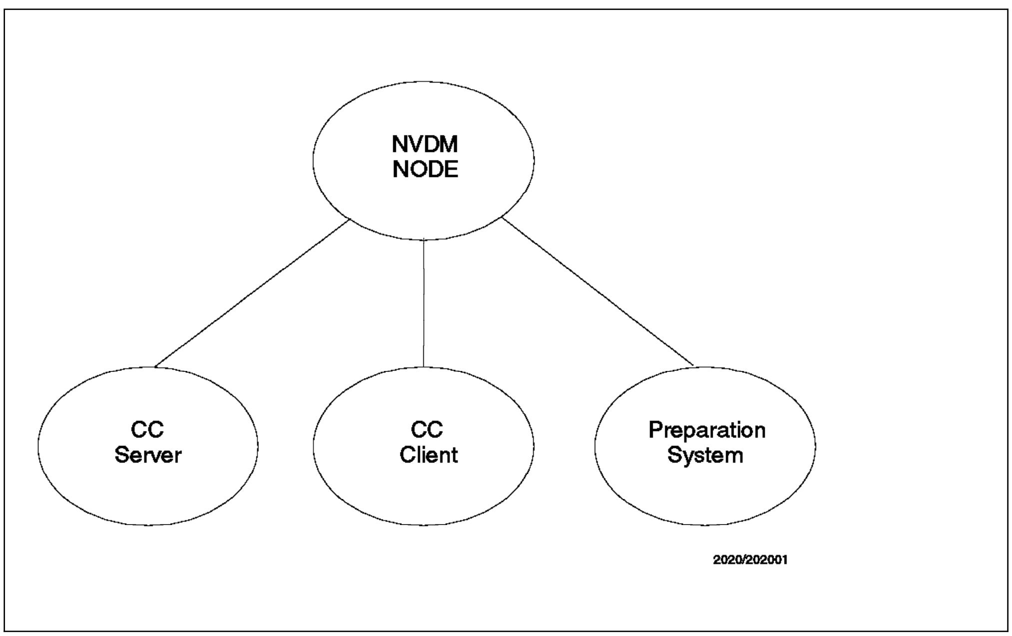


Figure 2. NetView DM/6000 Node Types

We want to be able to configure any type of NetView DM/6000 node, where we differentiate between the following:

- NetView DM/6000 server (CC server)

- NetView DMA/6000 (CC client)
- NetView DM/6000 preparation system (CC server)

The preparation system can normally be treated as a CC server; nevertheless we use a distinct type of node for preparation systems. The reason for this is that all configuration data is held in a configuration database, so you can tell at any time for example, how many servers, preparation systems and agents you have in your network by just examining this database.

As far as configuration is concerned a CC server and a preparation system are treated in the same way.

Note

We do not have a different configuration track for preparation systems in our specific configuration procedure but use the same track used for a NetView DM/6000 server.

In your specific environment, however, you might want to have a distinct track for preparation systems. For example, you may wish to create some file systems on preparation systems.

We use the same configuration script for all types of nodes, so the script is able to detect which type of node it is about to configure.

2.3 Defining Configuration Activities

We now define which configuration steps need to be performed for the different node types.

Note

We assume that a preparation system can be treated as a NetView DM/6000 server, so we have only two types of nodes, servers and agents.

For a NetView DM/6000 agent the following steps need to be performed:

- Configuring of the `Workstation Name` and `Server` fields in the `nvdm.cfg` file
- Configuring of product parameters like the size of log files etc. in `nvdm.cfg`
- Adding NetView DM/6000 users to AIX

For a NetView DM/6000 server the following steps need to be performed:

- Configuring SNA Server, including all link and session profiles
- Configuring SNA/DS connection profiles
- Configuring SNA/DS routing table
- Configuring all local targets
- Configuring all remote targets
- Configuring a focal point system
- Configuring all target groups

Note

Since a NetView DM/6000 server includes an agent, we imbed the configuration steps necessary for an agent into the server and thus perform all steps necessary for an agent also on a server. You should also note that the above list is just an example of what is needed to perform a complete customization of NetView DM/6000. This list needs to be extended for *your* specific environment. Also, some steps may not be necessary in your environment; for example, if you do not have a focal point in your network you can ignore the procedure to configure a focal point system.

2.4 Defining Interfaces and Prerequisites

As mentioned before we will not cover every single step needed to install every system on our network from scratch in this scenario. Instead, we concentrate on configuring NetView DM/6000.

We try however, to give some examples of what else could be necessary to achieve a completely automatic configuration of an RS/6000 system. Since we will not discuss these additional steps in full detail, we define some of these steps as prerequisites for our NetView DM/6000 configuration.

This means that for a completely automatic roll-out we assume that these steps have been performed before the NetView DM/6000 configuration is done.

In our example we assume that the following steps have been performed before we start configuring NetView DM/6000:

- The AIX base operating system has been installed on the node to be configured.
- The NetView DM/6000 product has been installed on the node to be configured (either server or agent).
- All other necessary products have been installed, for example SNA Server.
- A TCP/IP network connection to a central server has been configured.
- TCP/IP hostnames can be resolved on each node to be configured.

Note

It is essential that each system which needs to be configured has a working TCP/IP network connection to a central server. The configuration of each node on the network will be initiated from that central server. How this can be done is shown in 5.3, “Automating the Configuration Process” on page 89. For the TCP/IP connection to work for example, the network adapter in each system has to be configured.

We describe a pristine installation procedure in Chapter 10, “Pristine Installation” on page 167 that performs all the above prerequisites. It is combined with the configuration procedure and can therefore be used to completely configure our software distribution network.

Chapter 3. Designing a Data Model for Configuration Data

One of the most important steps in creating an automatic configuration procedure for NetView DM/6000 is to describe a data model that can hold the complete configuration data to configure the software distribution network.

The intention of this chapter is to show you how to get started. In Chapter 7, "Customizing and Extending the Configuration Procedure" on page 109 we explain how you can easily extend the data model and adapt it to your needs.

Before we start we should have a look at the requirements for the configuration data:

- The configuration database must contain all parameters necessary to completely configure any NetView DM/6000 node on the network.
- The configuration database must be easily accessible from a Korn shell script.
- Data changes in the configuration database must not require any changes in the configuration script.
- It must be possible to change the way of storing configuration data, for example by using a relational database system. This should be transparent to the configuration part of the script. The only change required for the script is replacing the database access procedures. See Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219 for the implementation of the NetView DM/6000 configuration data model in DB2/6000.

3.1 Defining a Method to Store Configuration Data

We could store all configuration data in flat ASCII files. However, we decided to use the AIX Object Data Manager (ODM) to store all configuration data in the first approach.

This has the following advantages:

- The way the ODM stores data forces you to structure your configuration data using object classes. This is a great advantage over flat files where you do not have any implicit structure.
- The ODM ensures integrity of your data because it will only let you add data which is in the form predefined by the ODM class definition.
- The creation of ODM classes and objects is quite easy, as is access to objects stored in the database.
- You can, for example, use the ODM editor `odme` to add new objects to your software distribution network. We will show an example for this in Chapter 6, "Using the ODM Editor to Change the Configuration" on page 99.
- It should be easily possible to move ODM data to another type of database, for example a relational database later.
- The ODM is available on every AIX system for free.

Note

We move the database to IBM Database 2/6000 in Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219. We also show how to transfer data stored in the ODM to DB2/6000 tables and vice versa.

3.2 Defining Object Classes

Note

This data model has been developed for a customer scenario and therefore only reflects the requirements for this project. You should make your own database design in any case.

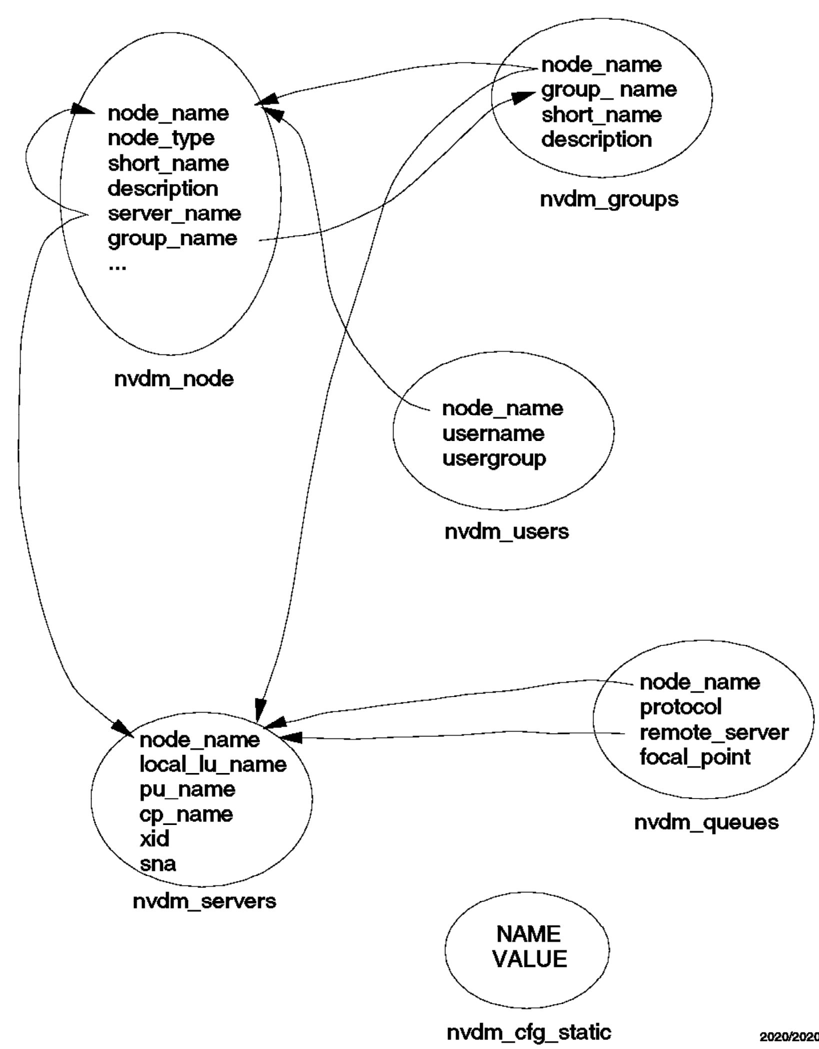


Figure 3. Configuration Object Classes

The main task in defining the ODM database is to find the necessary object classes and assign the necessary attributes.

To define classes we have to find objects in our software distribution network which have the same kind of attributes. All objects which share the same attributes can then be grouped together forming an ODM class.

These classes will then be filled with instances representing the configuration of our software distribution network.

The first thing we can define is that each RS/6000 system that we want to configure will be a NetView DM/6000 node, so we will need to have an object representing each node in our network. Since a node can either be a server, a preparation system or an agent, we need to find out what all these types of nodes have in common. Then this information can be stored in this general node class.

For example, each NetView DM/6000 node is a target, so this class must contain all parameters needed to configure a target, such as a target short name, description, etc. (Also, each node has a NetView DM/6000 server, even a server itself because it acts as its own server.)

Furthermore, we have configuration parameters that are only needed for a server. For example, only a server can have connections to other servers or a focal point system using SNA/DS.

We will now discuss every class we use in this configuration in detail.

3.2.1 The nvdm_node Object Class

As mentioned above this class contains an instance for every node in our software distribution network. This class holds all attributes which are needed for every node.

The first attribute is the `node_name`. This is a special attribute because it will be used by the ODM access methods as the search criteria when searching the ODM database for matching objects.

When we fill the ODM database we will set this attribute to the IP hostname of the node to be configured.

The `node_name` attribute will be included as a key in all ODM classes we define. The configuration script can then query any ODM class with the IP hostname of the node as the search argument. This enables us to pass only that one parameter, the IP hostname, to the configuration script. The script can then obtain all the data it needs from the ODM database by specifying just the `node_name` parameter.

The class `nvdm_node` will contain the following attributes:

- `node_name`: the IP hostname of the node to be configured
- `node_type`: the node type of the node to be configured
- `short_name`: the target short name of the node to be configured
- `description`: a description of the target
- `contact_name`, `owning_manager`, `telephone_number`, `customer_name`: descriptive information
- `server_name`: the NetView DM/6000 server for that target

- `group_name`: the name of the target group this node belongs to
- `target_os`: the target operating system

Explanation:
Since every node acts as a target, they need all parameters needed to configure a target. Parameters for a target that are the same for all nodes will not be stored with the object, but hard-coded in our configuration script.

We have, however, included the target operating system for a target in the `nvdm_node` class, although this attribute is always set to AIX in our example. The procedure could be extended to be able to configure agents or servers for other operating systems too.

> **Note**
>
> Examples where our procedure could be extended to handle other operating systems are the NetView DM/6000 agents for HP-UX, Sun OS and Sun Solaris. Since we have defined that configuring an agent only requires configuration of `nvdm.cfg` and addition of operating system users, these configuration steps can easily be transferred to these agents. The configuration of `nvdm.cfg` can remain unchanged, only the commands to add and change operating system users (`mkuser` and `chuser` on AIX) need to be adapted to the other operating systems.
>
> However, because the ODM only exists in AIX, the database access routines would need to be adapted. For example, all ODM queries could be made remotely to an AIX server by using `rsh`.
>
> Another example where the data model and procedure could be extended is the use of target specific tokens or an attribute that whether if a target is a push-mode or pull-mode target.

Some of the attributes in the object class are mandatory, others are not. Each node needs to have a `short_name`, a `node_type` and a `server_name`. A node, which is always a target, can belong to a target group. If so, the `group_name` field contains the name of this target group.

Each target has a corresponding NetView DM/6000 `server_name`. For a NetView DM/6000 server node, this field contains the server itself because each server has a target definition for itself.

> **Note**
>
> Which parameters are mandatory is not only dependent on the NetView DM/6000 product, but also on how we code the configuration script later. There are some parameters which are essentially needed to configure the product, such as server-agent relationships, so they must be supplied in the database.
>
> Others can be set to defaults within the configuration script if no match is found in the database. For example, if the configuration script does not find a match for the target OS in the database it could set it to AIX by default.

To demonstrate how this class can be used, imagine the following example:

We want to know which targets are defined for a NetView DM/6000 server. For that purpose we just need to start an ODM query to display all objects in the nvdm_node class where the server field matches the IP hostname of the server we are looking for. We will discuss the syntax and the results of such a query later.

3.2.2 The nvdm_groups Object Class

The nvdm_groups object class will be used to store objects representing NetView DM/6000 target groups.

It will also have an attribute node_name which identifies the node on which this target group has to be created. Since target groups will be defined on NetView DM/6000 servers only this class will only be accessed on nodes which are servers.

This class will contain one object for each group to be created on a NetView DM/6000 server.

The class nvdm_groups will have the following attributes:

- node_name: the IP hostname of the node to be configured
- group_name: the group name of the group to be created
- short_name: the short name of the group to be created
- description: a description of the group to be created

Explanation:
The group_name attribute is a link to the attribute with the same name in the nvdm_node object class. This enables the configuration script to find all nodes belonging to the target group when creating it.

Note

Since there is only one entry possible for the group_name field in the nvdm_node class, a target can only belong to one target group in this example.

If you want a target to be able to belong to more than one target group you will have to adapt the data model for that purpose.

3.2.3 The nvdm_users Object Class

The nvdm_users object class is used to store NetView DM/6000 users.

We will have one object for each user to be defined on each NetView DM/6000 server.

The class nvdm_users will contain the following attributes:

- node_name: the IP hostname of the node to be configured
- username: the user name of the user to be created
- usergroup: the user group of the user to be created

Explanation:

Here the `node_name` field identifies the target on which the user `username` has to be added, where `usergroup` is the AIX user group this user will be in (FNDUSER, FNDBLD, FNDADMN).

This class will be accessed on both the agent and server. The users for a target will be configured on its server. On the target the user needs to be added as an AIX user and assigned to the right AIX user group.

3.2.4 The nvdm_servers Object Class

By creating the `nvdm_node` class we have grouped all nodes in one class, whether they are server or agent, because the attributes contained in this class are needed for every node.

There are, however, attributes which are only needed for servers. These attributes will be stored in the classes `nvdm_servers` and `nvdm_queues`.

The class `nvdm_servers` contains the parameters needed to configure SNA server.

The attributes are in detail:

- `node_name`: the IP hostname of the node to be configured
- `local_lu_name`: the name of the LU 6.2 used for NVDM communications, for example, to a focal point system
- `pu_name`: the name of the SNA Physical Unit used for NVDM communications
- `cp_name`: the name of the SNA Control Point used for NVDM communications
- `xid`: the XID used for the server node
- `sna`: a flag indicating whether this node needs SNA configuration

Explanation:
The attributes stored in this class are the parameters that are unique for every system when configuring SNA server. They will be discussed in detail when creating the SNA Server configuration procedure.

The `sna` attribute can be set to yes or no indicating whether the server `node_name` needs SNA configuration or not.

3.2.5 The nvdm_queues Object Class

The `nvdm_queues` object class will be used to configure SNA/DS queues. An SNA/DS queue will be used for both types of communications, APPC and TCP/IP.

This class will contain the following attributes:

- `node_name`: the IP hostname of the node to be configured
- `protocol`: the communications protocol to be used
- `remote_server`: the short name of the remote server or focal point to which this queue should be connected
- `focal_point`: flag indicating whether the remote server is a focal point or not

Explanation:

The protocol parameter must be set to either APPC or TCP/IP. The `remote_server` is defined by its short name. The configuration script will, among other things, also create a remote target for that system.

If the `focal_point` flag is set to yes the remote target will be configured as a `report_to` focal point.

3.2.6 The nvdm_cfg_static Object Class

The classes defined before will contain objects which are unique for every node.

However, we also have parameters that are unique in the entire software distribution network and thus the same for all nodes to be configured. One example is the SNA network name which normally exists only once in the whole network.

Note

In this example we assume that we will only configure SNA connections to one central NetView DM/MVS system, so we will have only one SNA network name. However, if you have more SNA connections that you want to configure in your network, you will have to make some changes to the data model and the script. We will give you some hints on how to do that later.

Since we do not want to hard-code these parameters in our script we will also store these values in the ODM.

To do so we create a simple class, `nvdm_cfg_static`, which holds all data being the same for every node.

This class will contain only two attributes, `NAME` and `VALUE`, where `NAME` identifies the global parameter and `VALUE` stores its value. For the SNA Network name, `NAME` could be set to SNA_NET_NAME and `VALUE` could be set to USIBMRA assuming that USIBMRA is our SNA network name.

3.3 Creating Test Data for Configuring the Test Environment

We now define the configuration data for our test scenario and store it in ODM classes.

First of all, we need to create the ODM classes used to store the configuration data.

The following figure shows the ODM class definition file for the ODM database we want to create:

```
#
# Create ODM class files for NVDM configuration DB
#

#
# the nvdm_groups class defines the target groups to be defined
# on a server
#

class nvdm_groups {
  char group_name[25];
  char description[25];
  char short_name[9];
  char node_name[25];
}

#
# the nvdm_node class describes the name (IP Hostname) and
# type (Server, Agent, Prep Site) of the node, where
# 0 : NVDM Server
# 1 : NVDM Agent
# 2 : NVDM Prep Site
# also included are attributes required for every node, like
# the name of the NVDM/6000 Server, etc.
#
# group_name is a link to the nvdm_groups class specifying
# the group this target belongs to

class nvdm_node {
  char  node_name[25];
  short node_type;
  char  short_name[9];
  char  target_os[12];
  char  description[25];
  char  contact_name[25];
  char  owning_manager[25];
  char  telephone_number[20];
  char  customer_name[20];
  char  x_25_number[15];
  char  server_name[25];
  link  nvdm_groups nvdm_groups group_name group_name;
}

#
# nvdm_users is a class containing the users
# for a target. this class will be used on
# servers and targets to define users
#
```

Figure 4 (Part 1 of 2). Class Definition File

```
class nvdm_users {
  link  nvdm_node nvdm_node node_name node_name;
  char  username[9];
  char  usergroup[12];
}

#
# nvdm_cfg_static contains all parameters being
# unique for all targets
#

class nvdm_cfg_static {
  char  NAME[20];
  char  VALUE[128];
}

#
# the nvdm_servers class contains parameters only
# needed to configure NVDM/6000 Servers
#

class nvdm_servers {
  link nvdm_node nvdm_node node_name node_name;
  char local_lu_name[13];
  char pu_name[9];
  char cp_name[9];
  char xid[9];
  char sna[4];
}

#
# the nvdm_queues class contains connections to
# remote servers
# e.g. a Focal Point or remote administrator
#
# Protocol must be "APPC" or "TCP/IP"
# if Protocol is TCP/IP the remote_server
# field must be filled with the IP hostname
# of the remote server
#
# This class will also be used to define
# The remote server as a remote target automatically
#

class nvdm_queues {
  link nvdm_node nvdm_node node_name node_name;
  char protocol[8];
  char remote_server[25];
  char focal_point[4];
}
```

Figure 4 (Part 2 of 2). Class Definition File

The class definition file should have a file name ending with .cre. Assuming that our class definition is stored in the file `config_db.cre` the command for creating the ODM classes is:

```
odmcreate -c config_db
```

Note

You must have `root` user authority when invoking the above command as well as all other commands mentioned in this chapter.

The syntax of the class definitions is similar to a structure definition in the C programming language.

The attributes in a class are defined using data types such as `char` and `short`. The `link` type is a special data type used to link an attribute in a class to an attribute in another class with the same definition.

Note

If you need more information about the format of the class definition file you can refer to the manpage for `odmcreate` or to the appropriate InfoExplorer document.

This command will create a file for each class we have defined in the ODM directory which is `/etc/objrepos` by default. If you want to store them in another directory you can set the environment variable ODMDIR to that directory, for example, in a Korn shell:

```
export ODMDIR=/myodm
```

The `odmcreate` command will produce the following output:

```
nvdm_groups
nvdm_node
nvdm_users
nvdm_cfg_static
nvdm_servers
nvdm_queues
```

Figure 5. Output from odmcreate Command

Now that we have created the class files we can fill them with our configuration data.

Data can be added to the ODM using the `odmadd` command. The argument supplied with this command is the name of an ODM data file.

We will also refer to these data files as ODM definition files in this book.

This file has to be in a special format readable by the `odmadd` command. The following figure shows the file which contains the data definition for the objects to be added to the `nvdm_node` class:

```
nvdm_node:
   node_name = "rs60007"
   node_type = 0
   short_name = "RS60007"
   target_os = "AIX"
   description = "ITSO Raleigh development"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "4711"
   customer_name = "IBM"
   x_25_number = ""
   server_name = "rs60007"
   group_name = "Group1"

nvdm_node:
   node_name = "rs600015"
   node_type = 0
   short_name = "RS600015"
   target_os = "AIX"
   description = "ITSO Raleigh test server"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "4711"
   customer_name = "IBM"
   x_25_number = ""
   server_name = "rs600015"
   group_name = "Group2"

nvdm_node:
   node_name = "rs60004"
   node_type = 1
   short_name = "RS60004"
   target_os = "AIX"
   description = "ITSO Raleigh test client"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "4711"
   customer_name = "IBM"
   x_25_number = ""
   server_name = "rs60007"
   group_name = "Group1"
```

Figure 6. Data Definition File for nvdm_node Class

Explanation:
The data definition file contains the object definitions for the three nodes in our example network, where `rs60007` and `rs600015` are servers and `rs60004` is an agent, indicated by the `node_type` attribute set to either 1 or 0.

The server for agent `rs60004` is `rs60007`, indicated by the `server_name` field set to `rs60007` in the object defintion for `rs60004`.

Targets `rs60007` and `rs60004` belong to the same target group `Group1`, whereas target `rs600015` belongs to a target group `Group2`.

Each instance starts with the name of the class that we want to create an object for, for example, nvdm_node followed by a colon (:).

The attributes for an object are specified in the form:

```
attribute_name = value
```

For example, to set the node_name for an object to rs60007 we have to specify the line:

```
node_name = "rs60007"
```

There does not have to be a line for optional attributes. For example, if you do not want to have a description for a target, you can leave out the description attribute in the object definition. This field will then be set to an empty string automatically.

You can store the data definition for all classes in one file but we recommend that you use a single file for each class for ease of maintenance.

To add the definitions for the nvdm_node class we type:

```
odmadd nvdm_node.odmadd
```

assuming that the definitions are stored in the file nvdm_node.odmadd.

You should know that if you invoke this command several times, the ODM adds the entries several times because it does not check if an entry already exists.

To prevent this we write a simple script called build_db which contains the following lines:

```
odmcreate -c config_db
odmadd nvdm_cfg_static.odmadd
odmadd nvdm_groups.odmadd
odmadd nvdm_node.odmadd
odmadd nvdm_queues.odmadd
odmadd nvdm_users.odmadd
odmadd nvdm_servers.odmadd
```

Figure 7. build_db Shell Script

Whenever we change a definition in one of the *.odmadd files we will invoke this script to update the ODM database.

The odmcreate command at the beginning of the script will create the ODM classes and therefore clear the database every time it is invoked. This avoids double entries in the database.

The following figure shows the data definition file for the nvdm_groups class in our example network:

```
nvdm_groups:
   group_name = "Group1"
   description = "Raleigh Group1"
   short_name = "GROUP1"
   node_name = "rs60007"

nvdm_groups:
   group_name = "Group2"
   description = "Raleigh Group2"
   short_name = "GROUP2"
   node_name = "rs600015"
```

Figure 8. Data Definition File for nvdm_groups Class

Explanation:
The data definition file contains the description of the two target groups we have in our example network.

The `node_name` field indicates on which NetView DM/6000 server the target group shall be created.

The following figure shows the data definition file for the `nvdm_users` class in our example network:

```
nvdm_users:
   node_name = "rs60007"
   username = "root"
   usergroup = "FNDADMN"

nvdm_users:
   node_name = "rs60007"
   username = "suelpen"
   usergroup = "FNDADMN"

nvdm_users:
   node_name = "rs600015"
   username = "root"
   usergroup = "FNDADMN"

nvdm_users:
   node_name = "rs600015"
   username = "suelpen"
   usergroup = "FNDUSER"

nvdm_users:
   node_name = "rs60004"
   username = "root"
   usergroup = "FNDADMN"

nvdm_users:
   node_name = "rs60004"
   username = "mike"
   usergroup = "FNDBLD"
```

Figure 9. Data Definition File for nvdm_users Class

Explanation:
On target `rs60007` we defined two users, `root` and `suelpen`, both being NetView DM/6000 administrators.

On target `rs600015` we defined the same user names, but this time user `suelpen` being a NetView DM/6000 user.

On target `rs60004` we have defined user `root` to be an administrator and user `mike` to be a builder.

The following figure shows the data definition file for the `nvdm_servers` class in our example network:

```
nvdm_servers:
   node_name = "rs600015"
   local_lu_name = "RA60015B"
   pu_name = "RA60015"
   cp_name = "RA6015CP"
   xid = ""
   sna = "yes"

nvdm_servers:
   node_name = "rs60007"
   local_lu_name = "A"
   pu_name = "B"
   cp_name = "C"
   xid = ""
   sna = "no"
```

Figure 10. Class Definition File for nvdm_servers Class

Explanation:
The `nvdm_servers` class contains the SNA definitions needed to configure a NetView DM/6000 server for SNA communications.

Of the two servers on our network, only `rs600015` has an SNA connection to NetView DM/MVS, indicated by the `sna` attribute set to `yes`. The object definition for `rs600015` contains the SNA parameters being unique for that system, which are the local LU name, the PU name and the CP name.

The `xid` field is left blank, so the configuration script configures the corresponding SNA profiles to use the control point XID instead.

Since `rs60007` does not have SNA connections we filled in some dummy values for the LU name, PU name and CP name.

The following figure shows the data definition file for the `nvdm_queues` class in our example network:

```
nvdm_queues:
   node_name = "rs60007"
   protocol = "TCP/IP"
   remote_server = "rs600015"
   focal_point = "no"

nvdm_queues:
   node_name = "rs600015"
   protocol = "TCP/IP"
   remote_server = "rs60007"
   focal_point = "no"

nvdm_queues:
   node_name = "rs600015"
   protocol = "APPC"
   remote_server = "RA39TCF1"
   focal_point = "yes"
```

Figure 11. Class Definition File for nvdm_queues Class

Explanation:
We want to configure an SNA/DS connection between `rs60007` and `rs600015` using TCP/IP, so we have to define this connection on both nodes.

Further, `rs600015` will also have an SNA/DS connection to NetView DM/MVS using APPC, where NetView DM/MVS is a focal point system.

To add all the above definitions to the ODM database we typed:

```
./build_db
```

Now that we have added our test data to the ODM we can start implementing the configuration shell script.

3.4 Limitations

As mentioned before the data model that we have described in this chapter was designed for a specific customer environment and only covers the specific requirements for that project.

Hence, the data model presented here has the following limitations:

- There is no support for the automatic configuration of intermediate nodes (IN).
- There is no support for the automatic configuration of user interface (UI) only targets.
- The SNA configuration covers only a connection to a central NetView DM/MVS system.

The above points were not implemented because they were not needed in the scenario for which the procedure was originally developed (because we cannot cover configuration of NetView DM/6000 here).

However, we give you advice on how to customize the data model as well as the configuration procedure to meet the requirements of your specific environment.

The intent of this book is to provide you with the basic knowledge to allow you to create your own, specific configuration procedure.

Note

We show a way to extend the configuration procedure to be able to configure Intermediate Nodes (IN) in 8.1, "Configuring Intermediate Nodes" on page 131.

Chapter 4. Designing and Implementing the Configuration Procedure

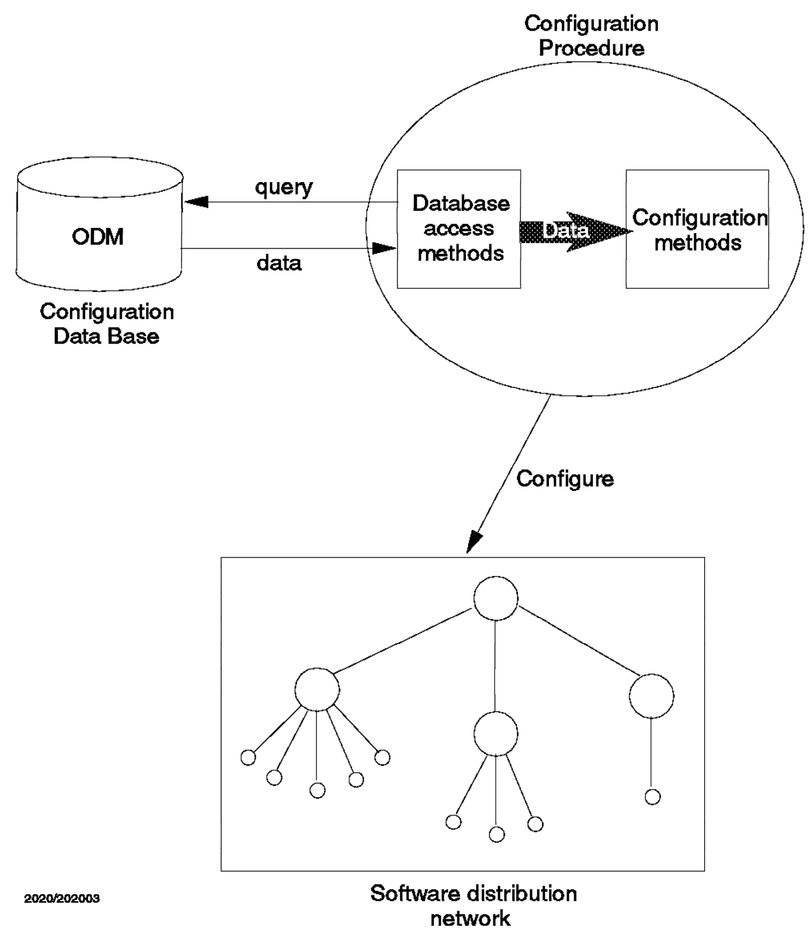


Figure 12. Structure of the Configuration Procedure

We now create the shell script which can be used to configure nodes in our software distribution network.

One design goal is to keep the database access independent from the configuration code. To achieve this we encapsulate the database access, in our case ODM queries, in separate shell procedures.

Note

Encapsulating the database access procedures enables us to replace the ODM by another database later. We show how to replace the ODM by DB2/6000 in Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219.

4.1 General Recommendations

In the project, the configuration procedure described in this book was originally developed for, we found that it is very useful to use a version control system to keep track of the changes made to the configuration procedure.

This is especially useful if you have to create a procedure for a complex environment or if there is more than one person working on the configuration procedure. Also, you might have procedures to configure other components of your AIX system, for example, other products. Then a version control system enables you to keep track of changes more easily and also allows you to roll back to a previous release of the script if necessary.

The easiest way is to use Source Code Control System (SCCS) which comes free with every AIX system. This is quite an easy to use tool to keep track of source code levels.

Note

To get a description of `sccs` type `man sccs` or refer to the corresponding InfoExplorer document.

When developing or testing your configuration procedure it is also a good idea to keep different versions of the configuration database. This can, for example, be useful when testing a procedure with certain configuration parameters.

You can manage to hold different versions of the configuration database in the ODM by setting the `ODMDIR` shell variable to the appropriate directory.

Note

You should be careful when setting the `ODMDIR` variable.

Let's assume that you hold a test version of the configuration database in the directory `/usr/mydata`. You could achieve this by setting the `ODMDIR` variable to `/usr/mydata` before using the `odmcreate` and `odmadd` commands to create and fill the necessary ODM class files: `ODMDIR=/usr/mydata and export ODMDIR`.

When you run the configuration procedure, afterwards you will have problems if there is a configuration part that uses the ODM itself.

For example, SNA Server stores its configuration in the ODM, which is by default located in the `/etc/objrepos` directory. If you redirect the ODM directory SNA will not be able to find and update its configuration information.

You can solve this by copying the necessary class files from `/etc/objrepos` to your own directory. However, you should not make permanent changes to the configuration while the ODM path is redirected. Use this only for testing purposes.

4.2 Database Access Procedures

The only way a configuration procedure can retrieve data is by calling the database access procedures, thus hiding the underlying database.

Later we can replace the ODM with another way of storing data. The only procedures which need to be changed are the database access procedures. The configuration procedures can remain unchanged.

Note

It is very important that the configuration scripts access data stored in the configuration database only by using the procedures described below. This allows us to easily replace the underlying database later (see Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219).

Since we have decided to store our data in the ODM, we first have to develop the access procedures to retrieve data from the ODM.

The first procedure, `get_attribute`, will retrieve an attribute value from an object matching a specified criteria and look as follows:

```
#
# get single parameters
# $1 = class name
# $2 = search field
# $3 = search field value
# $4 = attribute name
#

get_attribute ()
{
 VALUE=`odmget -q $2=$3 $1 | grep "$4 =" | cut -d '=' -f2 | sed "s/\"//g" |\
 cut -c2-79`
}
```

Figure 13. get_attribute Shell Procedure

The procedure will query the ODM with the search criteria specified in the command line parameters, cut the relevant information from the `odmget` output and store the result in the shell variable VALUE which is accessible in the entire shell script.

For example, you could issue the following command in a configuration procedure:

```
get_attribute nvdm_node node_name rs60012 server
```

This will search the class `nvdm_node` for an object where the `node_name` attribute is set to `rs600012`.

If an object is found the value for the `server` attribute for that node is stored in VALUE.

Note

The `get_attribute` procedure can only be used for queries where you expect no or exactly one match.

In our scenario we have queries where you could have more than one match. For example, when asking the database which nodes have a certain NetView DM/6000 server:

```
get_attribute nvdm_node server MASTER node_name
```

If there is more than one node in the network which has MASTER as its server the query will return a list of matches.

Therefore we will write another procedure, `get_attribute_list`, which can be used to make queries where you expect a list of answers:

```
#
# get list parameters from odm_class
# $1 = class name
# $2 = search field
# $3 = search field value
# $4 = attribute name
# The list of parameters is stored in the VALUE_LIST variable
# The number of parameters is stored in VALUE_NUM
#

get_attribute_list ()
{
 VALUE_LIST=`odmget -q $2=$3 $1 | grep "$4 =" | cut -d'=' -f2 |\
 sed "s/\"//g" | cut -c2-79`
 VALUE_NUM=`odmget -q $2=$3 $1 | grep "$1:" | wc -l`
}
```

Figure 14. get_attribute_list Shell Procedure

The `get_attribute_list` procedure will store the result in the shell variable VALUE_LIST. Also it will store the number of elements in VALUE_LIST in the shell variable VALUE_NUM.

Sometimes it will be necessary to have two search criteria connected with AND, therefore we, will have another procedure allowing this type of query:

```
#
# get single parameters (AND)
# $1 = class name
# $2 = search field1
# $3 = search field value1
# $4 = search field2
# $5 = search field value2
# $6 = attribute name
#

get_attribute_and ()
{
 VALUE=`odmget -q "$2=$3 AND $4=$5" $1
 sed "s/\"//g" | cut -c2-79`
}
```

Figure 15. get_attribute_and Shell Procedure

This procedure is necessary to query attributes of objects with more than one instance per node. For example, we can have more than one user defined for a target. If we want to query the usergroup attribute for one of these users we can connect the search arguments by typing:

```
get_attribute_and nvdm_users node_name rs600012 username mike usergroup
```

Note

You might also want to add other database access procedures that use, for example, other features of the `odmget` command. One example is the use of the `LIKE` statement of the `odmget` command, which can be used to do a fuzzy search.

However, the configuration scripts that we will develop in this chapter will only use the database access procedures shown above.

4.3 How the Configuration Script Works

We now develop a configuration routine for every configuration step needed to configure NetView DM/6000.

These routines will be implemented as Korn Shell procedures. Then the single procedures will be combined in one single Korn Shell script, `config_nvdm`, which will be used to configure every node in our software distribution network.

The only parameter passed to this script will be the IP hostname of the node to be configured.

The complete configuration script is listed in Appendix A, “The Configuration Script Listings” on page 331.

It is important to understand that in our scenario we have an ODM database holding the complete configuration of our software distribution network. Therefore, the ODM files need to be copied to every node that needs to be configured. How this can be achieved will be described later.

The configuration script are a combination of the database access routines. The configuration routines we describe below.

The way data is passed within the script is as follows:

- The ODM query routines which we have described previously will be used to retrieve data from the ODM and store it in the shell variables `VALUE` or `VALUE_LIST`.
- A configuration procedure will call the ODM access procedures and then copy the data retrieved to a global shell variable accessible within the entire configuration script.

  For example, one configuration routine will need the NetView DM/6000 server name, so the code to get that data looks like the following:

  ```
  get_attribute nvdm_node node_name rs600012 server_name
  SERVER=$VALUE
  ```

  The above code will determine the server for node rs600012 and then store the server name in the global shell variable `SERVER`.
- The main body of the script combines the different configuration steps together for each type of node.
- Since this defines the configuration steps to take place for the different node types, you will have to make changes here if you want to adapt the script to your specific environment.

  For example, if you want to leave out the configuration of the SNA Initial Node Setup you can just remove the line calling the `sna_initial` procedure from the `config_nvdm` script.

The following figure shows the steps being performed in our scenario:

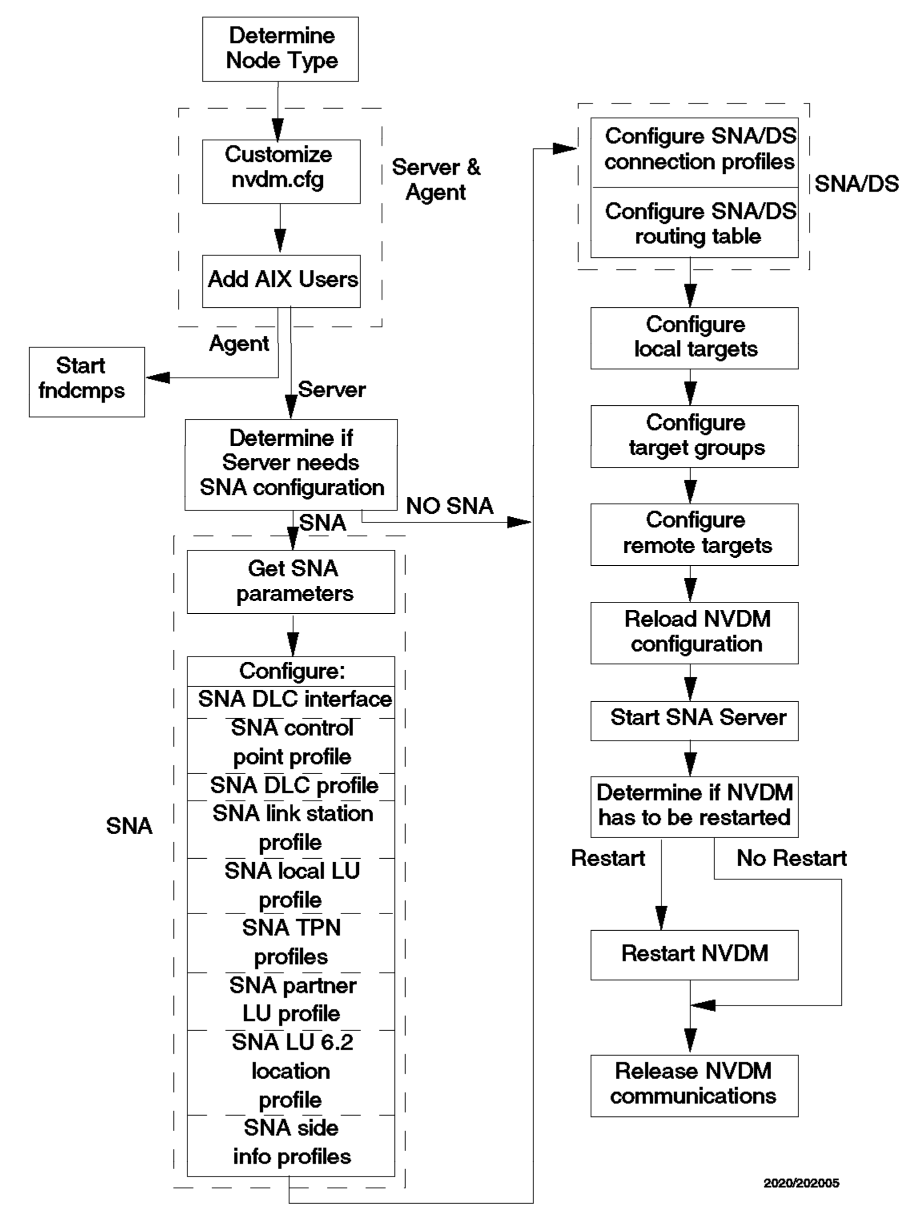


Figure 16. Configuration Steps

In the following we describe the configuration procedures that we use in our scenario. They cover all the configuration activities that we have defined before.

> **Note**
>
> All configuration procedures will be implemented as Korn-Shell procedures. If you are not familiar with the Korn-Shell syntax, you should refer to the Korn-Shell manpage by typing `man ksh` or to the documentation listed in the preface of this book.

Within the procedures we use several commands. These commands are AIX commands, as well as SNA Server and NetView DM/6000 commands.

We use the following AIX operating system commands:

- `mv` to move files
- `rm` to remove files
- `cd` to change the working directory
- `sed` to modify files
- `grep` to search in strings and files
- `cut` to cut sub-strings from strings
- `odmget` to query the ODM
- `lsuser` to list defined AIX users and their attributes
- `mkuser` to add AIX users
- `chuser` to change the attributes of existing AIX users
- `mkdev` to create devices (chuser)

We recommend

If you are not familiar with any of the above commands, you should at least consult the manpage for that command.

Moreover, it is always a good idea to consult the appropriate section of InfoExplorer to get a more comprehensive overview.

We use the following SNA Server commands:

- `mk_qcinit` to perform the Initial Node Setup
- `mksnaobj` to create SNA Server profiles
- `chsnaobj` to change existing SNA Server profiles
- `exportsna` to export the current SNA Server profiles

We recommend

If you need a description of the above SNA Server commands and especially about the flags these commands use, you should refer to the SNA Server documentation.

We use the following NetView DM/6000 commands:

- `nvdm lstg` to list existing targets
- `nvdm addtg` to add a target to a NetView DM/6000 server configuration
- `nvdm updtg` to update target information
- `nvdm deltg` to delete targets from a NetView DM/6000 server configuration
- `nvdm lsgp` to list existing target groups
- `nvdm addgp` to add a target group to a NetView DM/6000 server configuration
- `nvdm delgp` to delete groups from a NetView DM/6000 server configuration
- `nvdm inv` to update the NetView DM/6000 software inventory

- `nvdm lscm` to list existing change management records
- `nvdm delcm` to delete change management records
- `nvdm uncat` to remove a change file from the catalog
- `nvdm bld` to build a new change file
- `nvdm inst` to install a change file
- `nvdm start` to start the NetView DM/6000 server
- `nvdm stop` to stop the NetView DM/6000 server

We recommend

If you need a description of the above NetView DM/6000 commands and especially of the flags these commands use, you should refer to the manpages for NetView DM/6000, by typing `man nvdm_addtg`; this command will give you a description of the `addtg` sub-command including a description of all possible flags.

4.4 Customizing the NetView DM/6000 Configuration File

The first thing we discuss is the customization of the NetView DM/6000 configuration file `nvdm.cfg`.

Note

All configuration procedures described in this chapter were developed for NetView DM/6000. Generally these procedures can also be used for Software Distribution for AIX. However, some of the procedures need to be adapted to be usable with Software Distribution for AIX. The required modifications are described in Chapter 11, "Migrating the Procedure to Software Distribution for AIX V3.1" on page 197.

The `nvdm.cfg` file is the NetView DM/6000 base configuration file and is needed on every server and agent on the network.

We wrote a configuration procedure `configure_nvdm_cfg` that can be used to change any parameter in the `nvdm.cfg` file.

The following figure shows that procedure:

```
#
# Set Attributes in nvdm.cfg file
# $1 parameter name (e.g. WORKSTATION NAME, SERVER)
# $2 parameter value
#

configure_nvdm_cfg ()
{
 mv $CONFIG /tmp/config
 print "NVDM CONFIG : Setting nvdm.cfg ($1) to $2"
 #
 # the TCP/IP port parameter is special
 # because it contains a / in its name
 # and also needs modification of
 # /etc/services
 #
 if [ "$1" = "TCP/IP PORT" ]
 then
   sed "s/TCP\/IP PORT:.*/TCP\/IP PORT:          $2/" \
/tmp/config >$CONFIG
   mv /etc/services /tmp/services
   sed "s/NetViewDM6000.*\/tcp/NetViewDM6000   $2\/tcp/" \
/tmp/services >/etc/services
   return
 fi
 #
 # adjust to right column
 #
 len=`echo $1 | wc -c`
 SUBST=$2
 while [ $len -lt 22 ]
 do
  SUBST=" "$SUBST
  len=`expr $len + 1`
 done
 #
 # replace parameter
 #
 sed "s/$1:.*/$1:$SUBST/" /tmp/config >$CONFIG
}
```

Figure 17. configure_nvdm_cfg Shell Procedure

Explanation:
You have to pass two arguments when calling the procedure. The first argument is the keyword of the parameter you want to change, and the second is the value you want to set for this parameter.

For example, to change the `WORKSTATION NAME` field to `rs600011` the call would be:

```
configure_nvdm_cfg "WORKSTATION NAME" rs600011
```

In our configuration script we will use this procedure to change the `WORKSTATION NAME`, `SERVER` and `LOG FILE SIZE` fields.

You can, however, use this procedure to change any valid field in the nvdm.cfg file.

The values to be filled in the nvdm.cfg file will be retrieved from the ODM by using the ODM access procedures before calling the configuration procedure.

The following figure shows a code fragment which can be used to retrieve the NetView DM/6000 server from the database and then set the appropriate field in the configuration file:

```
get_attribute nvdm_node node_name $1 server_name
SERVER=$VALUE
configure_nvdm_config "SERVER" $SERVER
```

First the server name is retrieved from the ODM using the get_attribute procedure to query the nvdm_node class. The call shown above assumes that the variable $1 contains the IP hostname of the node to be configured.

The result is then stored in the shell variable VALUE, so we will assign the value of that variable to another shell variable SERVER.

Then the configuration procedure configure_nvdm_cfg is called passing the field name and the server name as parameters.

You should notice that the configuration of the TCP/IP port used for NetView DM/6000 is different from the other parameters, and therefore treated as a special case.

First, it contains a slash (/) character in the parameter name and can therefore not be directly processed by the sed command used to alter the corresponding line in nvdm.cfg.

Second, a change of the TCP/IP port in nvdm.cfg requires also a change of the corresponding entry in /etc/services.

Note

If you want to be completely flexible in using different TCP/IP ports for NetView DM/6000, you must add the code to change the other ports NetView DM/6000 uses.

The code supplied will only be able to change the port NetViewDM6000 and is considered to be an example.

4.5 Adding NetView DM/6000 Users to AIX

In order to enable an AIX user to use NetView DM/6000 the user ID has to be defined twice:

1. The user has to be defined as a user of the NetView DM/6000 target.
2. The user has to be defined to the AIX operating system itself.

Defining the user as a NetView DM/6000 target user is covered by the configuration procedure that does the target configuration and is therefore described there.

The routine we describe now will add the users to AIX.

The following figure shows the configuration procedure:

```
#
# add user at OS level (AIX)
# $1 = IP Hostname
# $2 = Type: either "server" or "target"
#            use "target", when you want to add a user to AIX
#            add a target workstation; the user will always be
#            assigned group FNDADMN
#            use "server", when you want to add a user to AIX
#            add a server workstation; the user will be assigned
#            the appropriate usergroup defined in the database
#

add_users_aix ()
{
print "NVDM CONFIG : --> Adding AIX users for NVDM..."
get_attribute_list nvdm_users node_name $1 username
if [ $VALUE_NUM != 0 ]
then
  for i in $VALUE_LIST
  do
    #
    # First, add NVDM user to operating system...
    # check if user exists
    #
    lsuser $i 2>/dev/null 1>&2
    #
    # if not (RC 2 from lsuser command)
    #
    if [ $? = 2 ]
    then
      print "NVDM CONFIG : Adding user $i to AIX OS."
      mkuser $i
    fi
    #
    # check if user has NVDM group
    #
    get_attribute_and nvdm_users node_name $1 username $i usergroup
    GRP=$VALUE
    #
    # if we configure a target, set group to FNDADMN
    #
    if [ "$2" = "target" ]
    then
      GRP=FNDADMN
    fi
    DEFGRP=`lsuser -a groups $i | cut -d'=' -f2`
    # if user is not in NVDM group, add him
    if [ "`echo $DEFGRP | grep $GRP`" = "" ]
```

Figure 18 (Part 1 of 2). add_users_aix Shell Procedure

```
    then
      chuser groups="$DEFGRP,$GRP" $i
    fi
  done
fi
}
```

Figure 18 (Part 2 of 2). add_users_aix Shell Procedure

Explanation:
This procedure has to be called on the server *and* on the target for which a user has to be defined.

On the server, all targets defined for this server have to be detected. Then all users for all targets have to be defined on the server.

On a target, all users defined for this target have to be defined to the AIX operating system.

First, the `add_users_aix` procedure queries the class `nvdm_users` to get all users for the target to be configured.

If a NetView DM/6000 user does not exist yet, it will be added to AIX, then the group to which this user shall belong will be determined from the `nvdm_users` class.

Since on an agent there is only the FNDADMN user group, the AIX user will always be assigned to this user group on an agent.

If the user does not belong to the defined group yet this group will be added to the AIX group set for that user. The user group must be one of the groups created by NetView DM/6000 (FNDUSER, FNDADMN, FNDBLD).

If a user already exists, then only the group set is extended, if necessary.

4.6 Configuring SNA Server

The automatic configuration of SNA Server for use with NetView DM/6000 is one of the most difficult tasks in creating an automatic roll-out procedure for NetView DM/6000 nodes.

The reason for this is that a lot of configuration parameters are needed to configure SNA, so it is very hard to find a scenario to represent a "typical" environment.

Therefore this part of the configuration is most likely to need customization for your specific environment. For that reason, we tried to make the scripts as flexible and modular as possible so that you can use them as building blocks for your own environment. See Chapter 7, “Customizing and Extending the Configuration Procedure” on page 109 for information on how to customize the procedure.

The steps performed to configure SNA Server for use with NetView DM/6000 were initially taken from *The NetView Distribution Manager/6000 Cookbook* GG24-4246. So, if you need help in customizing the scripts for use in your environment, it might be a good idea to consult this redbook before doing so.

Customizing SNA Server is divided into several steps, where most of the steps represent the configuration of an SNA Server profile.

The steps needed to configure SNA are as follows:

- Configuring the SNA DLC interface
- Configuring the SNA Initial Node Setup profile
- Configuring the SNA Control Point profile
- Configuring the SNA DLC profile
- Configuring the SNA Link Station profile
- Configuring the SNA Local LU profile
- Configuring the SNA Mode profile
- Configuring the SNA TPN profiles
- Configuring the SNA Partner LU profile
- Configuring the SNA LU 6.2 Location profile
- Configuring the SNA Side Information profiles

Moreover, we have another procedure to determine all the parameters that we need to configure SNA Server. We start with that procedure and then describe all the single configuration steps to be performed.

4.6.1 Determining SNA Configuration Parameters

Most of the parameters needed to configure SNA are unique across the network. This is because most of them describe the focal point system which we have only once in our entire network.

Note

The configuration procedure described in this book can only be used to configure SNA connections to a central NetView DM/MVS system. If you want to be able to configure SNA connections between any systems, you will have to modify the scripts as well as the data model.

Therefore most of the configuration parameters are stored in the ODM class `nvdm_cfg_static` because this is the place we decided to store data which exists only once in the entire network.

The following table shows all SNA parameters that we will store in `nvdm_cfg_static`:

Table 1 (Page 1 of 2). SNA Configuration Parameters

Parameter Name	Parameter Description	Parameter Value
VTAM_CP_NAME	Name of VTAM Control Point	RAK
SOLICIT_SSCP	Flag	yes
I_FIELD_SIZE	I-Field Size	2042
LOCAL_SAP	Local SAP address	04

Table 1 (Page 2 of 2). SNA Configuration Parameters		
Parameter Name	**Parameter Description**	**Parameter Value**
REMOTE_SAP	Remote SAP address	04
INITIATE_CALL	Flag	yes
ACTIVATE_START	Flag	yes
RESTART_NORMAL	Flag	yes
RESTART_ABNORMAL	Flag	yes
DATALINK_DEVICE	Network adapter to be used for SNA communications	tok0
REM_LINK_ADDR	Remote Link Address for SNA communications	400001240000
SNA_NET_NAME	SNA network name	USIBMRA
TPN_PROF_NAME_SND	Name of TPN profile for send	NVDMSND
TPN_PROF_NAME_RCV	Name of TPN profile for receive	NVDMRCV
MODE_PROF_NAME	Name of Mode profile	NVDMNORM
MODE_NAME	Name of Mode for NetView DM/MVS	NVDMNORM
PARTNER_LU_NAME	Name of focal point LU6.2	RA39TCF1
SIDE_INFO_PROF_SND	Side Information profile name (Send)	NVDMSIDS
SIDE_INFO_PROF_RCV	Side Information profile name (Receive)	NVDMSIDR

Configuration data that is different for each server node includes data describing the SNA characteristics of that specific node and not data being unique in the network.

This data will be stored in the `nvdm_servers` class because only NetView DM/6000 servers can have SNA connections.

The parameters stored in that class are:

- Local PU name
- Local LU name
- Local Control Point name
- XID

The procedure shown below will retrieve the necessary configuration parameters from the configuration database and store them in global shell variables so that they are accessible from every SNA configuration procedure.

```
#
# get all static SNA attributes (SNA Net Name, etc.)
# $1 = IP Hostname of node to be configured
#

get_sna_attributes ()
{

  #
  # get static SNA parameters
  #

  for i in SNA_NET_NAME DATALINK_DEVICE REM_LINK_ADDR MODE_PROF_NAME\
 MODE_NAME TPN_PROF_NAME_SND TPN_PROF_NAME_RCV PARTNER_LU_NAME\
 SIDE_INFO_PROF_SND SIDE_INFO_PROF_RCV SOLICIT_SSCP I_FIELD_SIZE\
 LOCAL_SAP REMOTE_SAP INITIATE_CALL ACTIVATE_START RESTART_NORMAL\
 RESTART_ABNORMAL VTAM_CP_NAME
  do
    get_attribute nvdm_cfg_static NAME $i VALUE
    case $i in
      SNA_NET_NAME)      text="SNA Network Name"
                         SNA_NET=$VALUE ;;
      DATALINK_DEVICE)    text="SNA Datalink Device"
                          DEVICE=$VALUE ;;
      REM_LINK_ADDR)      text="SNA Remote Link Address"
                          ADDR=$VALUE ;;
      MODE_PROF_NAME)     text="SNA NVDM Mode Profile Name"
                          MPROF=$VALUE ;;
      MODE_NAME)          text="SNA NVDM Mode Name"
                          MODE=$VALUE ;;
      TPN_PROF_NAME_SND)  text="SNA TPN Profile Name (Send)"
                          SND=$VALUE ;;
      TPN_PROF_NAME_RCV)  text="SNA TPN Profile Name (Receive)"
                          RCV=$VALUE ;;
      PARTNER_LU_NAME)    text="SNA Partner LU Name (MVS Host)"
                          PARTNER=$VALUE ;;
      SIDE_INFO_PROF_SND) text="SNA Side Info Profile Name (Send)"
                          SIDS=$VALUE ;;
      SIDE_INFO_PROF_RCV) text="SNA Side Info Profile Name (Receive)"
                          SIDR=$VALUE ;;
      SOLICIT_SSCP)       text="Solicit SSCP Field (yes|no)"
                          SOLICIT=$VALUE ;;
      I_FIELD_SIZE)       text="I-Field Size"
                          IFIELD=$VALUE ;;
      LOCAL_SAP)          text="SNA Local SAP No."
                          LSAP=$VALUE ;;
      REMOTE_SAP)         text="Remote SAP No."
                          RSAP=$VALUE ;;
```

Figure 19 (Part 1 of 3). get_sna_attributes Shell Procedure

```
      INITIATE_CALL)      text="SNA Initiate Call Field (yes|no)"
                     ICALL=$VALUE ;;
      ACTIVATE_START)     text="SNA Activate on start (yes|no)"
                          ACTSTART=$VALUE ;;
      RESTART_NORMAL) text="SNA Restart on normal termination (yes|no)"
                          RNORM=$VALUE ;;
      RESTART_ABNORMAL) text="SNA Restart on abnormal termination (yes|no)"
                          RABNORM=$VALUE ;;
      VTAM_CP_NAME) text="SNA VTAM CP Name (for LU6.2 Location Profile)"
                          VTAMCP=$VALUE ;;

    esac
    if [ "$VALUE" = "" ]
    then
      abort "Could not determine $text. Exiting..."
    else
      print "NVDM CONFIG : Setting $text to $VALUE"
    fi
  done

  get_attribute nvdm_servers node_name $1 pu_name
  PUNAME=$VALUE
  if [ "$PUNAME" = "" ]
  then
    abort "Could not determine PU NAME for $1 configuration
. Exiting..."
  fi

  print "NVDM CONFIG : Setting PU NAME for $1 to $PUNAME "

  get_attribute nvdm_servers node_name $1 local_lu_name
  LLUNAME=$VALUE
  if [ "$LLUNAME" = "" ]
  then
    abort "Could not determine Local LU Name for $1 configu
ration. Exiting..."
  fi

  print "NVDM CONFIG : Setting Local LU Name for $1 to $LLUNAME "

  get_attribute nvdm_servers node_name $1 cp_name
  CP_NAME=$VALUE
  if [ "$CP_NAME" = "" ]
  then
    abort "Could not determine Control Point Name for $1.\
 Exiting..."
  fi
  CP_TYPE=appn_end_node

  print "NVDM CONFIG : Setting Control Point Name for $1\
```

Figure 19 (Part 2 of 3). get_sna_attributes Shell Procedure

```
to $CP_NAME"

  get_attribute nvdm_servers node_name $1 xid
  XID=$VALUE
  if [ "$XID" = "" ]
  then
    print "NVDM CONFIG : Could not determine XID for $1 configu
ration."
    print "NVDM CONFIG : Setting USE_CP_XID to yes"
    USE_CP_XID="yes"
    # set XID to dummy value
    XID=07100000
  else
    print "NVDM CONFIG : Setting XID for $1 to $XID "
    print "NVDM CONFIG : Setting USE_CP_XID to no"
    USE_CP_XID="no"
  fi
}
```

Figure 19 (Part 3 of 3). get_sna_attributes Shell Procedure

4.6.2 Saving the Current SNA Configuration

Before making any changes to the existing SNA Server configuration, it is a good idea to save the current configuration.

The best way to do this is to use the SNA Server feature to export the SNA configuration profiles. In case of an error you can import the profiles again to restore the previous configuration.

The following shell procedure can be used to export the SNA configuration profiles:

```
#
# export existing SNA profiles
# in case they need to be restored if
# NVDM configuration fails
#
# $1 = name of export file
#
export_sna ()
{
 print "NVDM CONFIG : Exporting existing SNA profiles to $1 . & exportsna -A -f $1 -r -UT -C
}
```

Figure 20. export_sna Shell Procedure

4.6.3 Configuring the SNA DLC interface

For every communications adapter that we intend to use for SNA, we need a DLC (Data Link Control) interface for that adapter.

The following procedure can be used to configure token-ring, Ethernet and X.25 adapters for that purpose:

```
#
# configure SNA dlc
# for all SNA communications a DLC for the
# communications adapter is needed.
# if the DLC already exists, the mkdev command
# will print an error message - this will be
# redirected to /dev/null
#

configure_sna_dlc ()
{
 print "NVDM CONFIG : Adding DLC Device for $DEVICE"
 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) mkdev -c dlc -s dlc -t tokenring 1>/dev/null 2>&1 ;;
  "ent" ) mkdev -c dlc -s dlc -t ethernet 1>/dev/null 2>&1 ;;
  "x25" ) mkdev -c dlc -s dlc -t x25_qllc 1>/dev/null 2>&1 ;;
  "*"   ) print "NVDM CONFIG : Device type $CHECK unknown." ;;
 esac
}
```

Figure 21. configure_sna_dlc Shell Procedure

Explanation:
The script assumes that the script `get_sna_attributes` has been invoked before so that the the shell variable `DEVICE` contains the data link device to be used for SNA communications.

For example, this variable can hold the value `tok0` indicating that we want to use the first token-ring adapter in the machine for SNA.

By examining the first three characters in the device name, the script then determines which kind of adapter we have and then calls the appropriate configuration command.

4.6.4 Configuring the SNA Initial Node Setup Profile

The shell procedure `sna_initial` will be used to configure the SNA initial node setup:

```
#
# SNA initial node setup
#

sna_initial ()
{
 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="token_ring" ;;
  "ent" ) DEV_TYPE="ethernet" ;;
  "fdd" ) DEV_TYPE="fddi" ;;
  "x25" ) DEV_TYPE="x.25_call_SVC" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
    abort "No device type found for $DEVICE."
 fi

 print "NVDM CONFIG : Configuring SNA Initial Node Setup"
set -x
 mk_qcinit -y $DEV_TYPE -t $CP_TYPE -w $SNA_NET -d $CP_NAME
set +x

}
```

Figure 22. sna_initial Shell Procedure

For the initial node, set up the device type, control point type and SNA network name needed.

These are expected to be stored in certain global shell variables. For a complete list of all global shell variables used in the configuration procedures, you can refer to B.1, "Shell Variables" on page 411.

4.6.5 Configuring the SNA Control Point Profile

For the configuration of the control point we need the following information:

- The SNA network name
- The control point name of the node to be configured
- The control point type of the node to be configured

This information is assumed to be stored in the shell variables `SNA_NET`, `CP_NAME` and `CP_TYPE`.

The following figure shows the script:

```
#
# configure SNA Control Point Profile
#
# SNA_NET contains SNA Network Name
# CP_NAME contains SNA Control Point Name
# CP_TYPE contains SNA Control Point Type
#

configure_sna_cp ()
{
 print "NVDM CONFIG : Configuring SNA Control Point Profile"
 line
set -x
 chsnaobj -t 'control_pt' -e "$SNA_NET" -a "$CP_NAME" -A "$CP_NAME"\
 -N "$CP_TYPE" node_cp
set +x
 line
}
```

Figure 23. configure_sna_cp Shell Procedure

4.6.6 Configuring the SNA DLC Profile

The SNA DLC profile connects SNA Server to the communications adapter that shall be used for SNA communications.

To configure this profile we need the following configuration parameters:

- The device type of the communications adapter
- The device name of the communications adapter
- Flags

The device type will be determined by examining the first three characters of the device name. The script supports token-ring, Ethernet, FDDI and X.25 adapters.

The device name is assumed to be stored in the `DEVICE` shell variable. This variable is filled by the `get_sna_attributes` shell procedure which gets data from the configuration data base and stores it in shell variables. As a consequence, the `get_sna_attributes` procedure must be called before any other configuration procedure.

In our scenario we will use some flags that can change; others are hard-coded in the script. For example, the Local SAP address is stored in the `LSAP` shell variable.

Which flags you will store in variables and which you decide to hard-code depends on your specific environment.

The device name of the adapter will also be used as the name for the profile to be created.

The following figure shows the script that will configure the SNA DLC profile:

```
#
# configure SNA dlc profile
#

configure_sna_dlc_profile ()
{
 # determine type of DLC from datalink device name
 # get only first 3 characters from device name
 # e.g. if datalink device is x25s1, then x25 determines
 # the type to be X.25

 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="sna_dlc_token_ring" ;;
  "ent" ) DEV_TYPE="sna_dlc_ethernet" ;;
  "fdd" ) DEV_TYPE="sna_dlc_fddi" ;;
  "x25" ) DEV_TYPE="sna_dlc_x.25" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
    abort "No device type found for $DEVICE."
 fi

 #
 # create new DLC Profile
 # use Datalink Device Name as Profile Name
 #

 print "NVDM CONFIG : Configuring SNA DLC Profile"
 line
set -x
# change !!!
 if [ "$DEV_TYPE" = "sna_dlc_x.25" ]
 then
   mksnaobj -t "$DEV_TYPE" "$DEVICE"
   RC=$?
 else
   mksnaobj -t "$DEV_TYPE" -d "$DEVICE" -b $SOLICIT -w yes -m $IFIELD \
   -H $LSAP -c no -q 0 "$DEVICE"
   RC=$?
 fi
set +x
 line

 if [ $RC = 255 ]
 then
```

Figure 24 (Part 1 of 2). configure_sna_dlc_profile Shell Procedure

```
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
# change !!!
   if [ "$DEV_TYPE" = "sna_dlc_x.25" ]
   then
     chsnaobj -t "$DEV_TYPE" "$DEVICE"
   else
     chsnaobj -t "$DEV_TYPE" -d "$DEVICE" -b $SOLICIT -w yes -m $IFIELD \
 -H $LSAP -c no -q 0 "$DEVICE"
   fi
set +x
   line
 fi

}
```

Figure 24 (Part 2 of 2). configure_sna_dlc_profile Shell Procedure

4.6.7 Configuring the SNA Link Station Profile

For the configuration of the SNA Link Station profile we will need several parameters:

- The device type of the communications adapter to be used
- The device name of the communications adapter to be used
- The remote link address
- The XID
- The Local PU name
- Flags

The device name and type are the same that we used in the configuration of the SNA DLC profile before.

The remote link address is the MAC address of the SNA system to connect to, for example, the token-ring address of a host communications controller.

The local PU name will be used as the profile name for the SNA Link Station profile.

In our example we use some flags that we store in variables, for example, the remote SAP address and some restart flags.

This script, as well as all the following SNA Server configuration scripts, first tries to create a new profile. If there is a profile with that name already existing, it will change the existing one to reflect any changes made.

Therefore the entire configuration script can be used not only for an initial configuration but also for reconfiguration of a node.

The following figure shows the script:

```
#
# configure SNA Link Station Profile
#

configure_sna_link ()
{
 # determine type of DLC from datalink device name
 # get only first 3 characters from device name

 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="token_ring" ;;
  "ent" ) DEV_TYPE="ethernet" ;;
  "fdd" ) DEV_TYPE="fddi" ;;
  "x25" ) DEV_TYPE="x.25" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
   abort "No device type found for $DEVICE. Exiting"
 fi

 #
 # create new Link Station Profile
 # use Datalink Device Name as DLC Profile Name
 #

 print "NVDM CONFIG : Configuring SNA Link Station Profile"
 line
set -x
# change !!!
 if [ "$DEV_TYPE" = "x.25" ]
 then
   mksnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -q "$X25_TYPE"\
   -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
   -s "$ADDR" "$PUNAME"
   RC=$?
 else
   mksnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -d "$ADDR" -l $XID\
   -s $RSAP -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
   -c "$USE_CP_XID" "$PUNAME"
   RC=$?
 fi
set +x
 line

 if [ $RC = 255 ]
```

Figure 25 (Part 1 of 2). configure_sna_link Shell Procedure

```
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   if [ "$DEV_TYPE" = "x.25" ]
   then
     chsnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -q "$X25_TYPE" \
     -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
     -s "$ADDR" "$PUNAME"
   else
     chsnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -d "$ADDR" -l $XID\
     -s $RSAP -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
     -c "$USE_CP_XID" "$PUNAME"
   fi
set +x
   line
 fi
}
```

Figure 25 (Part 2 of 2). configure_sna_link Shell Procedure

Note

You should be aware that the configuration parameters in the SNA DLC profile and Link Station profile might be different for the different types of network adapters.

In the example procedures shown above, for example, we differentiate between an X.25 adapter and other types of adapters, such as token-ring or Ethernet.

This is necessary because the flags used when configuring the profiles are slightly different for X.25 adapters. For example, you have to specify the Network User Address (NUA) of the remote adapter with the `-s` flag when configuring an X.25 connection, while you have to specify the Medium Access Control (MAC) address with the `-d` flag when configuring a token-ring or Ethernet connection.

If you use an X.25 adapter you also need to provide the circuit type in the `X25_TYPE` variable which can be either switched or permanent.

In case you want to use a network adapter different from token-ring, Ethernet or X.25 you should check for the appropriate configuration flags before implementing the configuration script. How this can be done is described in 7.1.3, “Determining SNA Server Commands” on page 116.

4.6.8 Configuring the SNA Local LU Profile

NetView DM/6000 uses SNA LU 6.2 to communicate with NetView DM/MVS, therefore we need to configure the necessary LU 6.2 profiles.

The first LU 6.2 profile we configure is the local LU profile.

For that profile we need the following parameters:

- The local LU name
- The local LU alias
- The local LU profile name

Note

In our scenario we assume that the local LU is independent.

In our scenario we use the local LU name for the local LU alias and the profile name. This parameter is assumed to be stored in the `LLUNAME` shell variable.

The following figure shows the script:

```
#
# configure local LU profile for node
#

configure_sna_local_lu ()
{
 print "NVDM CONFIG : Configuring SNA Local LU Profile"

 #
 # create new Local LU Profile
 # use Local LU Name as Profile Name
 #

 line
set -x
 mksnaobj -t local_lu -u lu6.2 -l "$LLUNAME" -L "$LLUNAME"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_lu -u lu6.2 -l "$LLUNAME" -L "$LLUNAME"
set +x
   line
 fi

}
```

Figure 26. configure_sna_local_lu Shell Procedure

4.6.9 Configuring the SNA Mode Profile

The SNA Mode profile is used to describe the LU 6.2 mode used for NetView DM communications.

We use the mode name and the mode profile name as the configuration parameters, where the mode name is stored in the `MODE` variable and the mode profile name is stored in the `MPROF` variable.

We recommend that you use the default value, `NVDMNORM`, for both values.

The following figure shows the procedure used to configure the mode profile:

> **Note**
>
> In our example all mode parameters, like session winners (parameter `-w 0`) and losers (parameter `-l 0`) are hard-coded.
>
> If there is any need to change this, feel free to do so.

```
#
# configure SNA Mode Profile
#

configure_sna_mode ()
{
 #
 # create new Mode Profile
 #

 print "NVDM CONFIG : Configuring SNA Mode Profile"
 line
set -x
 mksnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N "#CONNECT" -m "$MODE" "$MPROF"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N "#CONNECT" -m "$MODE" "$MPROF"
set +x
   line
 fi

}
```

Figure 27. configure_sna_mode Shell Procedure

4.6.10 Configuring the SNA TPN Profiles

NetView DM/6000 uses two LU 6.2 transaction programs to transfer data using SNA. These transaction programs are delivered with the product and need to be defined to SNA Server. For that purpose, we need to create two SNA TPN profiles.

The parameters needed to configure these profiles are mostly predetermined by the product, for example, the transaction program name and the path where the program is located.

The only parameter that we have to define ourselves is the profile name for each of the two profiles.

We recommend that the TPN profile for send is named `NVDMSND` and that for receive it is named `NVDMRCV` to be consistent with the manuals.

The following figure shows the procedure to configure the TPN profile for send:

```
#
# configure TPN send profile
#

configure_sna_send ()
{
 #
 # create TPN Profile (Send)
 #

 print "NVDM CONFIG : Configuring SNA TPN Profile (SEND)"
 line
set -x
 mksnaobj -t local_tp -n 21F0F0F7 -h yes -c basic \
 -w /usr/lpp/netviewdm/bin/fndts -s none "$SND"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_tp -n 21F0F0F7 -h yes -c basic \
 -w /usr/lpp/netviewdm/bin/fndts -s none "$SND"
set +x
   line
 fi

}
```

Figure 28. configure_sna_send Shell Procedure

The following figure shows the procedure to configure the TPN profile for receive:

```
#
# configure TPN receive profile
#

configure_sna_receive ()
{
 #
 # create TPN Profile (Receive)
 #

 print "NVDM CONFIG : Configuring SNA TPN Profile (Receive)"
 line
set -x
 mksnaobj -t local_tp -n 21F0F0F8 -h yes -c basic \
 -w /usr/lpp/netviewdm/bin/fndtr -s none "$RCV"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_tp -n 21F0F0F8 -h yes -c basic \
 -w /usr/lpp/netviewdm/bin/fndtr -s none "$RCV"
set +x
   line
 fi

}
```

Figure 29. configure_sna_receive Shell Procedure

4.6.11 Configuring SNA Partner LU Profile

For communications between the system we want to configure and the NetView DM/MVS system we will use an LU 6.2 session.

We have shown a procedure to create an SNA Server profile describing the local LU used for that purpose in 4.6.8, “Configuring the SNA Local LU Profile” on page 53. Moreover, we also need to describe the partner LU we want to communicate with.

The parameters we need to configure this profile are the partner LU name, the partner LU alias, the network name, and the profile name.

For the partner LU alias and the profile name we will use the same name as for the partner LU. This value is assumed to be stored in the `PARTNER` shell variable. The SNA network name is assumed to be stored in the `SNA_NET` shell variable.

The following figure shows the procedure used to configure the partner LU profile:

```
#
# Configure partner LU profile (Focal Point)
#

configure_sna_partner ()
{
 #
 # create LU 6.2 Partner Profile
 #

 print "NVDM CONFIG : Configuring SNA LU6.2 Partner LU"
 line
set -x
 mksnaobj -t partner_lu6.2 -p no -P "$SNA_NET"."$PARTNER" \
 -A "$PARTNER" "$PARTNER"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t partner_lu6.2 -p no -P "$SNA_NET"."$PARTNER" \
-A "$PARTNER" "$PARTNER"
set +x
   line
 fi

}
```

Figure 30. configure_sna_partner Shell Procedure

4.6.12 Configuring the SNA LU 6.2 Location Profile

To configure the SNA LU 6.2 Location profile, we need the SNA network name, the partner LU name and the name of the VTAM control point as the configuration parameters. The name of the VTAM control point is assumed to be stored in the `VTAMCP` shell variable.

Note

This profile is only required if the remote control point is an APPN Low Entry Node (LEN). The remote node in this case is VTAM. VTAM does not support End Nodes (EN) or Network Nodes (NN) until Version 4.1.

The following figure shows the configuration procedure:

```
#
# configure LU6.2 location profile
#

configure_sna_location ()
{
 print "NVDM CONFIG : Configuring SNA LU 6.2 Location Profile"

 #
 # create new LU 6.2 Location Profile
 # use Local LU Name as Profile Name
 #

 line
set -x
 mksnaobj -t partner_lu6.2_location -P "$SNA_NET.$PARTNER" \
 -O "$SNA_NET.$VTAMCP" -m link_station -l $LLUNAME \
 -s $PUNAME $PARTNER
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
 chsnaobj -t partner_lu6.2_location -P "$SNA_NET.$PARTNER" \
 -O "$SNA_NET.$VTAMCP" -m link_station -l $LLUNAME \
 -s $PUNAME $PARTNER
set +x
   line
 fi

}
```

Figure 31. configure_sna_location Shell Procedure

4.6.13 Configuring the SNA Side Information Profiles

We need to configure a Side Information profile for the send and receive transaction programs used by NetView DM/6000.

In order to configure the Side Information profile for the Send program, we need to know the control point name, the SNA network name, the partner LU name, the mode name and the profile name for the Side Information profile.

The following figure shows the procedure used to configure the Side Information profile (Send):

```
#
# configure Side Info Profile (Send)
#

configure_side_snd ()
{
 #
 # create Side Info Profile (Send)
 #

 print "NVDM CONFIG : Configuring SNA Side Info Profile (Send)"
 line
set -x
 mksnaobj -t side_info -L "$CP_NAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F7 -h yes "$SIDS"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t side_info -L "$CP_NAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F7 -h yes "$SIDS"
set +x
   line
 fi

}
```

Figure 32. configure_side_snd Shell Procedure

Note

It is important to choose profile names that you will also use in the SNA/DS connection profiles, for example, in the `SEND TP SYMBOLIC DESTINATION` field. In our scenario this is guaranteed because the procedure used to configure the SNA/DS connection profiles uses the same shell variables to determine the profile names.

The configuration for the Side Information Profile for receive is very similar to that for Send, except that we use the local LU name instead of the control point alias.

We use the control point alias for the one profile and the local LU name for the other to avoid warnings when verifying the SNA profiles.

```
#
# configure Side Info Profile (Receive)
#

configure_side_rcv ()
{
 #
 # create Side Info Profile (Receive)
 #

 print "NVDM CONFIG : Configuring SNA Side Info Profile (Receive)"
 line
set -x
 mksnaobj -t side_info -L "$LLUNAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F8 -h yes "$SIDR"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t side_info -L "$LLUNAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F8 -h yes "$SIDR"
set +x
   line
 fi

}
```

Figure 33. configure_side_rcv Shell Procedure

4.7 Configuring SNA/DS Connection Profiles

NetView DM/6000 uses SNA/DS to communicate with other NetView DM/6000 servers or with NetView DM/MVS. For this to work, SNA/DS has to be configured properly.

Note

Unlike configuring SNA Server, which is a separate product, the configuration of SNA/DS is a part of configuring the NetView DM/6000 product.

For each connection to another server we need an SNA/DS connection profile, so we will now develop a procedure to configure these connection profiles automatically.

In our configuration database we define remote systems by adding objects to the nvdm_queues class. This makes sense because any connection to a remote server uses a queue.

Therefore, we first create a simple routine to query the nvdm_queues class. The information we need to configure an SNA/DS connection to a remote system contains basically the name of the remote system and the protocol to be used.

The following shell procedure will obtain this information:

```
#
# get queues defined for a server
# since this class can contain more
# than one entry for a server, we have
# to store the result in a list
#
# $1 = server name
#

get_queues ()
{
 #
 # first, determine number of entries for
 # that server
 #

 #
 # Fill in Fields
 #

 get_attribute_list nvdm_queues node_name $1 protocol
 NUM_QUEUE=$VALUE_NUM
 if [ $NUM_QUEUE = 0 ]
 then
   return
 fi

 PROTOCOL=$VALUE_LIST
 get_attribute_list nvdm_queues node_name $1 remote_server
 REMOTE_SERVER=$VALUE_LIST

}
```

Figure 34. get_queues Shell Procedure

The get_queues shell procedure will store the data in the global variables PROTOCOL and REMOTE_SERVER.

The configuration procedure shown below will use this data to configure the SNA/DS connection profiles:

```
#
# configure SNA/DS connection profiles
#
# $1 = IP Hostname of system to be configured
#

configure_sna_ds_conn ()
{

#
# perform SNA/DS configuration (connection profiles)
#

#
# remove demo profile CONNSNA,CONNTCP if existent
#
cd $SNA_DS_DIR
rm CONNSNA 2>/dev/null
rm CONNTCP 2>/dev/null

get_queues $1

if [ $NUM_QUEUE != 0 ]
then
  a=1
  for i in $PROTOCOL
  do
    print "NVDM CONFIG : Configuring $i connection"
    if [ "$i" != "APPC" -a  "$i" != "TCP/IP" ]
    then
      abort "Protocol is neither APPC nor TCP/IP. Exiting..
    fi
    if [ "$i" = "APPC" ]
    then
      configure_sna_ds_appc
    else
      REMSERV=`echo $REMOTE_SERVER | cut -d' ' -f "$a"`
      configure_sna_ds_tcpip $REMSERV
      a=`expr $a + 1`
    fi
  done
fi
}
```

Figure 35. configure_sna_ds_conn Shell Procedure

Explanation:
First the procedure removes the default connection files CONNSNA and CONNTCP, which are delivered with NetView DM/6000. This is just done to avoid these demo queues appearing in the NetView DM/6000 queues window and leading to confusion.

The shell variable SNA_DS_DIR holds the pathname of the directory where SNA/DS connection profiles are stored. Normally this is /usr/lpp/netviewdm/db/snads_conn.

Then the script processes the list of remote systems to be configured. Depending on whether the system to be configured is connected via APPC or TCP/IP it will call the configuration procedures configure_sna_ds_appc or configure_sna_ds_tcpip.

The next figure shows configure_sna_ds_appc:

```
#
# Configure SNA/DS connection configuration file (APPC)
#

configure_sna_ds_appc ()
{
 print "NVDM CONFIG : Configuring SNA/DS connection\
 configuration file $SNA_DS_DIR/$PARTNER"
 echo "PROTOCOL:              APPC
SEND TP SYMBOLIC DESTINATION:     $SIDS
RECEIVE TP SYMBOLIC DESTINATION:  $SIDR
NEXT DSU:                         $SNA_NET.$PARTNER
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" > $SNA_DS_DIR/$PARTNER
}
```

Figure 36. configure_sna_ds_appc Shell Procedure

Explanation:
The configuration procedure will create a file in the SNA/DS directory that represents the connection to be configured. The shell variable PARTNER holds the name of the partner system to be configured. The variables SIDS and SIDR hold the names of the side information profiles also used in the configuration of SNA Server; SNA_NET holds the SNA Network name.

Note

The procedure shown above is quite simple because in our scenario we only configure an APPC connection to NetView DM/MVS. Therefore, we use the same data that we used to configure the SNA Server connection to the MVS system.

However, the configuration becomes more complicated if you want to configure SNA connections to other RS/6000 systems. If you intend to do that, you will have to modify the above procedure.

In this example communications to other NetView DM/6000 servers will use TCP/IP as the communications protocol.

The following figure shows the procedure to configure SNA/DS connections using TCP/IP:

```
#
# Configure SNA/DS connection configuration file (TCP/IP)
# $1 = TCP/IP Hostname of remote system
#

configure_sna_ds_tcpip ()
{
 #
 # get short name of remote server
 #

 get_attribute nvdm_node node_name $1 short_name
 A=$VALUE
 print "NVDM CONFIG : Configuring SNA/DS connection configuration file.
 print "NVDM CONFIG : (TCP/IP) for remote Server $A."

 echo "PROTOCOL:                         TCP/IP
REMOTE SERVER NAME:               $1
TCP/IP TIME-OUT:                  300
NEXT DSU:                         $A.$A
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" >$SNA_DS_DIR/$A
}
```

Figure 37. configure_sna_ds_tcpip Shell Procedure

Explanation:
The procedure `configure_sna_ds_tcpip` is basically the same as `configure_sna_ds_appc`. It is, however, more flexible. The TCP/IP hostname of the remote system is passed to the procedure as a command parameter. The short name of this system is determined by using the `nvdm_node` class.

Therefore this procedure can be used to configure any TCP/IP connection using SNA/DS.

4.8 Configuring SNA/DS Routing Table

The SNA/DS routing table contains the routes to be used for SNA/DS and STS traffic. We will only use SNA/DS connections in our example.

There is one routing table for all connections.

The following figure shows the procedure to automatically configure the SNA/DS routing table:

```
#
# configure SNA/DS routing table
# $1 = IP Hostname
#

configure_routetab ()
{
 #
 # first, determine what network protocols we have
 #
 a=0
 b=0
 print "NVDM CONFIG : Configuring SNA/DS routing table."
 cd $SNA_DS_DIR
 HAVET=`grep PROTOCOL * | grep TCP/IP`
 if [ "$HAVET" != "" ]
 then
   print "NVDM CONFIG : System has TCP/IP connection to remote server.
   a=1
 fi

 HAVEA=`grep PROTOCOL * | grep APPC`
 if [ "$HAVEA" != "" ]
 then
   print "NVDM CONFIG : System has APPC connection to remote server."
   b=1
 fi

 if [ $a -eq 0 -a $b -eq 0 ]
 then
   print "NVDM CONFIG : There are no connections defined."
   return
 fi

 if [ $a -eq 1 -a $b -eq 1 ]
 then
   RPROT="BOTH"
 fi

 if [ $a -eq 1 -a $b -eq 0 ]
 then
   RPROT="TCP/IP"
 fi

 if [ $a -eq 0 -a $b -eq 1 ]
 then
   RPROT="APPC"
 fi
```

Figure 38 (Part 1 of 2). configure_routetab Shell Procedure

```
 print "NVDM CONFIG : Writing routing table to $SNA_DS_ROUTE"
 echo "NETWORK PROTOCOL: $RPROT

#
# SNA connections
#
" >$SNA_DS_ROUTE

#
# get all SNA Routes
#

 cd $SNA_DS_DIR
 SNA_R=`grep -p APPC * | grep "NEXT DSU" | cut -d':' -f2`
 if [ "$SNA_R" != "" ]
 then
   for i in $SNA_R
   do
     ONE=`echo $i | cut -d'.' -f1`
     TWO=`echo $i | cut -d'.' -f2`
     if [ "$TWO" = "*" ]
     then
       echo "$i    ANY  ANY  ANY  ANY  $ONE 5" >>$SNA_DS_ROUTE
     else
       echo "$i    ANY  ANY  ANY  ANY  $TWO 5" >>$SNA_DS_ROUTE
     fi
   done
 fi

 echo "
#
# TCP/IP connections
#
" >>$SNA_DS_ROUTE
 TCP_R=`grep -p TCP/IP * | grep "NEXT DSU" | cut -d':' -f2`
 if [ "$TCP_R" != "" ]
 then
   for i in $TCP_R
   do
     ONE=`echo $i | cut -d'.' -f1`
     echo "$ONE.*                      $ONE" >>$SNA_DS_ROUTE
   done
 fi
}
```

Figure 38 (Part 2 of 2). configure_routetab Shell Procedure

Explanation:
First, the procedure determines what protocols we use for SNA/DS connections, APPC, TCP/IP or both. This information is needed to fill in the `NETWORK PROTOCOL` field in the `routetab` file.

The shell variable `SNA_DS_ROUTE` holds the file name of the SNA/DS routing table which is normally `/usr/lpp/netviewdm/db/routetab`.

Information is then gathered from the SNA/DS connection files needed to construct the SNA/DS routes.

Note

In the above configuration procedure we just fill in the fields regarding the existing SNA/DS connection files. For the route configuration itself we use the defaults as recommended in the NetView DM/6000 manuals.

If you need to change these defaults you will have to modify the procedure, for example, by replacing the ANY fields in the APPC routes.

4.9 Configuring Local Targets

On a NetView DM/6000 server all targets defined for this server need to be configured automatically.

The procedure we develop to do this should be able to cover both types of configuration:

- Initial Configuration
- Reconfiguration

The entire configuration script cannot only be used to initially configure our software distribution network but also to make changes to this network.

As far as targets are concerned, this means that the server-agent relationships in the network may change in one of the following ways:

1. A target is added to a server.
2. A target is removed from a server.
3. The characteristics of a target are changed.
4. A target is moved from one server to another.

To meet all of the above requirements we have to write several scripts to perform the steps necessary to maintain the target configuration.

The first procedure will delete all targets from a server that are not contained in the current configuration database for that server. This will primarily happen when you want to delete a target from a server's configuration.

The following figure shows the procedure to do so:

```
#
# delete local targets from NVDM Server configuration
# $1 = Server IP Hostname
#

nvdm_delete_targets()
{
 #
 # get list of existing targets
 #

 TLIST=`nvdm lstg '*' | grep "Target:" | cut -d':' -f2`

 #
 # get list of all defined targets for this server
 #

 get_attribute_list nvdm_node server_name $1 node_name
 XLIST=$VALUE_LIST

 #
 # delete all targets which are not defined for this server
 #

 for i in $TLIST
 do
  match=0
  for x in $XLIST
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    nvdm_save_history $i
    print "NVDM CONFIG : Deleting Target $i from Server $1 configuration."
    nvdm deltg $i -f
  fi
 done
}
```

Figure 39. nvdm_delete_targets Shell Procedure

Explanation:
The procedure first determines which targets are currently defined for the server to be configured. Then the database is queried to return all targets currently in the configuration database for that server.

All targets that are currently defined, but not in the configuration database, will then be removed.

Possibly, a target not contained in the configuration database for a server anymore has been moved to another server. When this happens we not only want to remove the target from one server and add it to another but also move the target history with the target.

For saving the target history we write another shell procedure `nvdm_save_history` which will be called before removing a target from a server:

```
#
# Save NVDM target history by creating software inventory
# file and copying it to corresponding node
# requires /.rhosts file on target
# $1 = target name
#

nvdm_save_history ()
{
 print "NVDM CONFIG : Saving target history for $1"
 nvdm inv
 SLIST="`nvdm lscm -w $1 '*' | grep 'Global file name:' | cut -d':' -f2`"
 >/tmp/inv
 if [ "$SLIST" != "" ]
 then
   for o in $SLIST
   do
     print "NVDM CONFIG : Adding $o to software inventory file."
     print "PRODUCT: "$o >>/tmp/inv
     print "DESCRIPTION: Target has been moved!" >>/tmp/inv
   done
   print "NVDM CONFIG : Copying inventory file $SW_INV to $1."
   echo "GLOBAL NAME:                   HISTORY.REF.1
CHANGE FILE TYPE:              GEN
COMPRESSION TYPE:              LZW
REBOOT REQUIRED:               NO
PACK FILES:                    NO
SECURE PACKAGE:                NO
OBJECT:
SOURCE NAME:               /tmp/inv
TARGET NAME:               /usr/lpp/netviewdm/fndswinv
TYPE:                      FILE
ACTION:                    COPY
INCLUDE SUBDIRS:           NO" >/tmp/hist.pro
   nvdm delcm HISTORY.REF.1 -w '*'
   nvdm uncat HISTORY.REF.1 -d -f
   nvdm bld /tmp/hist.pro -f
   nvdm inst HISTORY.REF.1 -w $1 -f -i
   print "CONFIG NVDM : Sleeping for 5 secs."
   sleep 5
 fi
}
```

Figure 40. nvdm_save_history Shell Procedure

Explanation:
The script will save the target history using the software inventory file `fndswinv` on the target to be moved. First, the current change history of the node to be moved is determined using the `nvdm lscm` command. For this to work the target to be

moved still has to be active and connected to the server from which we initiate the move. Therefore, we have to do this before finally removing the target from the original server.

From the information gathered by `nvdm lscm`, a software inventory file is created and stored in `/tmp/inv` at the server.

This inventory file then has to be copied to the agent for which it has been created. To do so,we create a change file that contains the inventory file and then send this change file to the agent.

Note

We use a change file to transmit the inventory file to make use of the servers ability to copy files to the agent. If we used, for example, remote copy (`rcp`) to do so we would need to have a lot of `/.rhosts` files to cover the possible server-agent relationships.

The following figure contains a sample inventory file generated by `nvdm_save_history`:

```
PRODUCT: HUGO.REF.1
DESCRIPTION: Target has been moved!
PRODUCT: IBM.NDM6000.CLBOOKS.FIX.112.U436
DESCRIPTION: Target has been moved!
PRODUCT: IBM.NDM6000.CLBOOKS.REF.112
DESCRIPTION: Target has been moved!
PRODUCT: IBM.NDM6000.CLGI.FIX.112.U436929
DESCRIPTION: Target has been moved!
PRODUCT: IBM.NDM6000.CLGI.REF.112
DESCRIPTION: Target has been moved!
PRODUCT: IBM.NDM6000.CLIENT.FIX.112.U4369
DESCRIPTION: Target has been moved!
PRODUCT: IBM.NDM6000.CLIENT.REF.112
DESCRIPTION: Target has been moved!
PRODUCT: SUELPEN.DEMO.REF.1
DESCRIPTION: Target has been moved!
```

Figure 41. Sample Software Inventory File

After the target has been moved to another server the target history for that target can still be retrieved because it is stored in the software inventory file.

Warning

The procedure shown above is a quite simple way to save the target history. It assumes that the agent to be moved is connected to the server which tries to save the history. If you reconfigure a complete network you must ensure that you reconfigure the server *before* you reconfigure the agent that is to be moved. This is because if you reconfigured the agent first to belong to a new server, the server currently defined for that target could not create and transmit the history file.

We now show another procedure, `nvdm_configure_targets`, which we use to add or modify targets.

The following figure shows the procedure:

```
#
# configure Targets for an NVDM/6000 Server
# $1 = Server IP Hostname
#

nvdm_configure_targets ()
{
 #
 # First, determine all Nodes which have these Server
 # defined as their NVDM/6000 server
 #

 # access database

 get_attribute_list nvdm_node server_name $1 node_name
 TLIST=$VALUE_LIST

 for i in $TLIST
 do

   print "NVDM CONFIG : Defining Target $i on server $1"

   nvdm lstg $i 1>/dev/null 2>&1

   #
   # if return code = 0 then target exists already
   #

   if [ $? -ne 0 ]
   then
     COMMAND="nvdm addtg $i"
   else
     COMMAND="nvdm updtg $i"
     print "NVDM CONFIG : Target already exists. Updating.
   fi

   #
   # get required target attributes
   #

   for a in short_name target_os description contact_name\
 owning_manager telephone_number customer_name
   do
     get_attribute nvdm_node node_name $i $a
     v=$VALUE
     if [ "$v" != "" ]
     then
       case $a in
```

Figure 42 (Part 1 of 2). nvdm_configure_targets Shell Procedure

```
          short_name)     COMMAND=$COMMAND" -s '$v'" ;;
          target_os)      COMMAND=$COMMAND" -y '$v'" ;;
          description)    COMMAND=$COMMAND" -d '$v'" ;;
          contact_name)   COMMAND=$COMMAND" -q '$v'" ;;
          owning_manager) COMMAND=$COMMAND" -o '$v'" ;;
          telephone_number) COMMAND=$COMMAND" -t '$v'" ;;
          customer_name)    COMMAND=$COMMAND" -r '$v'" ;;
        esac
      fi
    done
    echo $COMMAND
    eval $COMMAND

#
# add users for target
#

get_attribute_list nvdm_users node_name $i username
if [ $VALUE_NUM != 0 ]
then
  print "NVDM CONFIG : Adding Target Users..."
  for x in $VALUE_LIST
  do
    print "NVDM CONFIG : Adding $x User"
    nvdm updtg $i -u $x
  done
fi

done

}
```

Figure 42 (Part 2 of 2). nvdm_configure_targets Shell Procedure

Explanation:
First the database is queried to return all targets currently in the configuration database for that server. Then the procedure determines if the target already exists or not. If the target does not exist yet, we will use the `nvdm addtg` command to add this target to the server; otherwise we will use `nvdm updtg` to just update the target characteristics.

In our scenario we retrieve several target characteristics from the configuration database, including a target description, etc. What you will include in your own configuration database depends on your specific environment. For example, you may not need to include a target description in your database.

However, some parameters are essential and necessary so they must be included in your configuration database. As far as the target definition is concerned, the target short name must be included in the database because it is needed to run the configuration command.

Note

In case of the short name you can, of course, also decide to automatically generate the short name in the configuration script. For example, you could take the IP hostname and convert it into uppercase to use it as the short name if you limit the length of the hostname to eight characters. Then you would not need to have the short name as an attribute in the `nvdm_node` class.

After the target base characteristics have been configured, the script will add the users of this target to the configuration. For that purpose the ODM class containing the target users will be consulted.

4.10 Configuring Target Groups

The task of configuring local target groups is similar to that of configuring local targets.

We also have a shell procedure to remove target groups that are not in the configuration database anymore from the configuration of a server.

However, since target groups do not have a change management history, we will not have to save history information for target groups.

The following figure shows the procedure to remove target groups:

```
#
# Delete all existing groups before adding groups from
# configuration database
# $1 = IP Hostname of server to be configured
#

nvdm_delete_groups ()
{
 #
 # determine existing groups
 #
 GP=`nvdm lsgp '*' | grep -E "Push|Pull" | cut -d' ' -f1`
 #
 # determine list of defined groups
 #
 get_attribute_list nvdm_groups node_name $1 group_name
 XGP=$VALUE_LIST

 for i in $GP
 do
  match=0
  for x in $XGP
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    print "NVDM CONFIG : Deleting group $i from $1 configuration."
    nvdm delgp $i -f
  fi
 done
}
```

Figure 43. nvdm_delete_groups Shell Procedure

The process of removing target groups from a server's configuration is nearly the same as that for removing targets.

Also we have a similar script to configure local target groups:

```
#
# configure groups defined for NVDM/6000 server
#

nvdm_configure_groups ()
{
 print "NVDM CONFIG : Configuring Target Groups for $1"
 get_attribute_list nvdm_groups node_name $1 group_name
 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No groups defined"
   return
 fi
 GROUP_LIST=$VALUE_LIST
 for i in $GROUP_LIST
 do
   print "NVDM CONFIG : Adding group $i"
   get_attribute nvdm_groups group_name $i short_name
   SHORT=$VALUE
   get_attribute nvdm_groups group_name $i description
   DESC=$VALUE
   #
   # get all targets being defined for this group
   #

   get_attribute_list nvdm_node group_name $i node_name

   for a in $VALUE_LIST
   do
     eval nvdm addgp $i $a -s "'$SHORT'" -d "'$DESC'"
   done
 done
}
```

Figure 44. nvdm_configure_groups Shell Procedure

4.11 Configuring Remote Targets

We can have two types of remote targets:

- Remote servers and clients
- Focal points

Remote servers are other NetView DM servers (NetView DM/6000, NetView DM/MVS, etc.).

Remote NetView DM/6000, NetView DM/MVS, NetView DM/2 and System Manager/400 servers can also be confirmed as focal points.

Connections to other NetView DM servers are made using SNA/DS queues. Therefore we can detect which remote targets we have to define by examining the database containing the queue definition for the server to be configured. In our scenario this is the `nvdm_queues` ODM class.

If the remote target is a focal point, we will use the same configuration parameters that we used to configure SNA Server.

If the remote target is another NetView DM/6000 server, it must also have a corresponding object in the `nvdm_node` class representing it, so we can consult that class to get the target short name.

The following figure shows the shell procedure that can be used to configure remote targets and focal points:

```
#
# configure Remote Targets
# $1 = IP Hostname
#

nvdm_remote_targets ()
{
 #
 # First, get all remote targets defined for this server
 # Remote Targets are determined by searching the nvdm_queues
 # class because any connection to a remote system requires a
 # queue

 get_attribute_list nvdm_queues node_name $1 remote_server

 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No remote targets defined"
   return
 fi

 for i in $VALUE_LIST
 do
   print "NVDM CONFIG : Defining remote target for $i"

   #
   # determine if system to be configured is a Remote Target or
   # a Focal Point
   #
   get_attribute_and nvdm_queues node_name $1 remote_server $i focal_point

   if [ "$VALUE" = "yes" ]
   then
     print "NVDM CONFIG : $i will be configured as focal point."
     # for the MVS focal point short name will be the same as node name
     # network id will be the SNA Network Name

set -x
     eval nvdm addtg $i -m report_to -s $i -n $SNA_NET -d "'NVDM_MVS'"
set +x
   else
     # get short name for remote server from class nvdm_node
     get_attribute nvdm_node node_name $i short_name
     if [ "$VALUE" = "" ]
     then
       abort "No Short Name defined for $i in class nvdm_node. Exiting..."
     fi
     RSHORT=$VALUE
```

Figure 45 (Part 1 of 2). nvdm_remote_targets Shell Procedure

```
      #
      # This remote server is assumed to be connected via TCP/IP
      # so, we set the network name to be the same as the short name
      #
      nvdm addtg $i -m remote -s $RSHORT -n $RSHORT
    fi
  done
}
```

Figure 45 (Part 2 of 2). nvdm_remote_targets Shell Procedure

Note

In the above script we assume that if we have to configure a focal point that this is also a report to focal point system, so we use the `-m report_to` parameter with the `nvdm addtg` command.

If you do not want the focal point system to be a report_to focal point you will have to change this. Of course you can also introduce a new attribute to the corresponding ODM class to make the configuration script more flexible.

4.12 Miscellaneous Matters

The configuration shell procedures described in the previous sections will be combined into one single shell script used to perform the configuration of a NetView DM/6000 node.

Besides the configuration procedures, this script also contains some additional routines, for example, how to print error messages.

The following procedure is used to start or restart NetView DM/6000 after a server has been configured:

```
restart_nvdm ()
{
 print "NVDM CONFIG : --> In order for the changes to become active"
 print "NVDM CONFIG :     NetView DM/6000 will be restarted on this node"

 #
 # determine if nvdm is running
 #

 nvdm stat 1>/dev/null 2>&1

 if [ $? = 121 ]
 then
   print "NVDM CONFIG : NVDM is not running. It will be started now."
   nvdm start
   nvdm start
 else
   print "NVDM CONFIG : Stopping NVDM."
   nvdm stop -x 1>/dev/null 2>&1
   s=1
   print "NVDM CONFIG : Restarting NVDM."
   while [ $s = 1 ]
   do
     print "NVDM CONFIG : Restarting NVDM."
     nvdm start
     nvdm stat
     if [ $? != 121 ]
     then
       s=0
     fi
   done
 fi
}
```

Figure 46. restart_nvdm Shell Procedure

Explanation:
The procedure first examines if NetView DM/6000 is already running by invoking the `nvdm stat` command. If NetView DM/6000 is not running yet it will be started; otherwise the server will be stopped and then started again.

Note

We recommend that the procedure to start NetView DM/6000 contains the statement `nvdm start` two times.

If NetView DM/6000 has just been stopped and `nvdm start` is called afterwards, the command will produce the following error message if there has not been enough time elapsed since the stopping of the server:

```
FNDCL232E: Unable to start the system as the D&CC Agent is shutting down
```

In that case the server will not be started. However, if you invoke `nvdm start` a second time, it will start the server in any case, because it then waits for the server to be stopped before trying to restart it.

4.13 Limitations

All the shell procedures described in this chapter previously are used to configure RS/6000 nodes in our software distribution network.

However, we might also have to configure other operating systems. For example, if we have a focal point system running NetView DM/MVS, we should also have procedures to configure NetView DM/MVS as well as other necessary MVS components.

The reason for this is simple. Assuming that we have a large number of RS/6000 nodes in our software distribution network, we now have the procedures to configure these nodes automatically. Nevertheless, we still have to do all the configuration work on MVS manually, for example defining all NetView DM/6000 nodes to NetView DM/MVS.

Although we will not provide the procedures needed to configure an MVS host, in this book we give some hints on how this task could be performed.

In order to automate the process of configuring NetView DM/MVS, we need to figure out which components have to be configured. For example, the following steps could be included:

- Adding node definitions for all NetView DM/6000 nodes to NetView DM/MVS
- Adding SNA LU 6.2 definitions for all RS/6000 nodes to VTAM

Note

The automatic configuration of VTAM can be quite complicated and will not be discussed here. If you want to create a procedure to automatically configure VTAM you will need a very good knowledge of SNA and MVS.

In order to automatically configure MVS components we normally use CLISTs which are similar to the Shell procedures for AIX developed in this book. NetView DM/MVS also supplies macros that can be used, for example, to add node definitions in batch mode.

The problem is that our configuration database is stored in the AIX ODM database which is only available on AIX. In order to configure MVS components we need to access the same configuration data that is used to configure RS/6000 nodes. Therefore, we have to transfer the configuration data from AIX to MVS.

One way to do this is as follows:

We create a shell script on AIX that queries the ODM database for the necessary data and then generates the CLIST or macro procedures needed to configure, for example, node definitions in NetView DM/MVS. These procedures can then be transferred to MVS and be executed there to perform the necessary configuration tasks.

We will show a simple example for such a procedure in 8.2, "Configuring NetView DM/MVS" on page 142.

Another way to exchange configuration data between AIX and MVS

is the use of a DB2 database (see Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219).

In this case the MVS host holds the configuration database while the AIX configuration server is configured as a Distributed Database Connection Services (DDCS) gateway, enabling the targets, configured as DB2/6000 clients, to access the host database transparently.

We are neither going to show the setup of the MVS host as a database server, nor the configuration of the DDCS feature of DB2/6000 in this book. For details about the latter you can refer to *Distributed Relational Database Cross Platform Connectivity and Application, SG24-4311*.

Note that the SQL scripts for the creation of the configuration database presented in Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219 apply to DB2 on the MVS host, too.

Chapter 5. Testing the Automatic Configuration Script

In Chapter 4, “Designing and Implementing the Configuration Procedure” on page 29 we created the shell procedures to perform the configuration activities needed to configure NetView DM/6000.

We now show how to apply the configuration script in combination with the configuration data to our test environment.

The different shell procedures will be combined into one single shell script, `config_nvdm`, which is listed in the appendix.

To initiate configuration of a node in a NetView DM/6000 software distribution network,we have to invoke this script and pass the IP hostname of the node as the command line parameter, for example, `config_nvdm rs60007`.

The script will then use the IP hostname as the search criteria to obtain configuration data for that node. Since we have filled the ODM database previously with the configuration data describing our specific scenario we can now start to configure our NetView DM/6000 nodes.

5.1 Prerequisites for Node Configuration

In order for the automatic node configuration to work we performed some prerequisite steps in our test environment.

These are the following:

- The AIX 3.2.5 operating system has been installed on all systems in our network.
- NetView DM/6000 Server Version 1.2 has been installed on `rs60007` and `rs600015`.
- NetView DM/6000 Client Version 1.2 has been installed on `rs60004`.
- SNA Server Version 2.1 has been installed on `rs600015`.
- TCP/IP has been configured to run on the token-ring adapter in `rs60004`,`rs60007` and `rs600015`.
- TCP/IP name resolution is provided on all hosts in our network.

5.2 Starting the Node Configuration

To configure the node `rs60007` and redirect the script output in a log file we type:

```
config_nvdm rs60007 2>&1 | tee rs60007.log
```

The following figure shows the log file produced for the configuration of `rs60007`:

```
NVDM CONFIG : --> Trying to configure node rs60007
NVDM CONFIG : Node type is 0 (0 = Server, 1 = Agent, 2 = Prep)
NVDM CONFIG : --> NVDM Base Node Configuration
NVDM CONFIG : Setting nvdm.cfg (WORKSTATION NAME) to rs60007
NVDM CONFIG : Setting nvdm.cfg (SERVER) to rs60007
NVDM CONFIG : Setting nvdm.cfg (LOG FILE SIZE) to 250000
NVDM CONFIG : Setting nvdm.cfg (TCP/IP PORT) to 729
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Adding user suelpen to AIX OS.
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Adding user mike to AIX OS.
NVDM CONFIG : Setting SNA Network Name to USIBMRA
NVDM CONFIG : Setting SNA Datalink Device to tok0
NVDM CONFIG : Setting SNA Remote Link Address to 400001240000
NVDM CONFIG : Setting SNA NVDM Mode Profile Name to NVDMNORM
NVDM CONFIG : Setting SNA NVDM Mode Name to NVDMNORM
NVDM CONFIG : Setting SNA TPN Profile Name (Send)
to NVDMSND
NVDM CONFIG : Setting SNA TPN Profile Name (Receive)
to NVDMRCV
NVDM CONFIG : Setting SNA Partner LU Name (MVS Host)
to RA39TCF1
NVDM CONFIG : Setting SNA Side Info Profile Name (Send)
to NVDMSIDS
NVDM CONFIG : Setting SNA Side Info Profile Name (Receive)
to NVDMSIDR
NVDM CONFIG : Setting Solicit SSCP Field (yes|no) to yes
NVDM CONFIG : Setting I-Field Size to 2042
NVDM CONFIG : Setting SNA Local SAP No. to 04
NVDM CONFIG : Setting Remote SAP No. to 04
NVDM CONFIG : Setting SNA Initiate Call Field (yes|no) to yes
NVDM CONFIG : Setting SNA Activate on start (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on normal termination
(yes|no) to yes
NVDM CONFIG : Setting SNA Restart on abnormal termination
(yes|no) to yes
NVDM CONFIG : Setting SNA VTAM CP Name
(for LU6.2 Location Profile) to RAK
NVDM CONFIG : Setting PU NAME for rs60007 to B
NVDM CONFIG : Setting Local LU Name for rs60007 to A
NVDM CONFIG : Setting Control Point Name for rs60007to C
NVDM CONFIG : Could not determine XID for rs60007 configu
ration.
NVDM CONFIG : Setting USE_CP_XID to yes
NVDM CONFIG : Configuring TCP/IP connection
NVDM CONFIG : Configuring SNA/DS connection configuration file.
NVDM CONFIG : (TCP/IP) for remote Server RS600015.
NVDM CONFIG : Configuring SNA/DS routing table.
NVDM CONFIG : System has TCP/IP connection to remote server.
```

Figure 47 (Part 1 of 2). Configuration Log File rs60007.log (Part 1)

```
NVDM CONFIG : Writing routing table
to /usr/lpp/netviewdm/db/routetab
NVDM CONFIG : Defining Target rs60007 on server rs60007
NVDM CONFIG : Target already exists. Updating...
nvdm updtg rs60007 -s 'RS60007' -y 'AIX'
-d 'ITSO Raleigh development' -q 'Stefan Uelpenich'
-o 'Wolfgang Geiger' -t '4711' -r 'IBM'
WARNING: The Network ID of this domain has been changed
to RS60007.
NVDM CONFIG : Adding Target Users...
NVDM CONFIG : Adding root User
NVDM CONFIG : Adding suelpen User
NVDM CONFIG : Defining Target rs60004 on server rs60007
nvdm addtg rs60004 -s 'RS60004' -y 'AIX'
-d 'ITSO Raleigh test client' -q 'Stefan Uelpenich'
-o 'Wolfgang Geiger' -t '4711' -r 'IBM'
NVDM CONFIG : Adding Target Users...
NVDM CONFIG : Adding root User
NVDM CONFIG : Adding mike User
NVDM CONFIG : Configuring Target Groups for rs60007
NVDM CONFIG : Adding group Group1
NVDM CONFIG : Defining remote target for rs600015
0513-029 The sna Subsystem is already active.
Multiple instances are not supported.
NVDM CONFIG : --> In order for the changes to become active
NVDM CONFIG :     NetView DM/6000 will be restarted on this node
NVDM CONFIG : NVDM is not running. It will be started now.
Trying to connect to default server (rs60007).
Connected to server rs60007.
NVDM CONFIG : Releasing NVDM SNA communications.
NVDM CONFIG : !!! Configuration of Server completed successfully !!!
```

Figure 47 (Part 2 of 2). Configuration Log File rs60007.log (Part 1)

Note

In order for the script to work, the ODM database needs to be filled with the configuration data. We filled the database in our scenario before by invoking the `build_db` command, which is listed in Figure 7 on page 22.

If the script cannot find the necessary configuration data in the database it will print an error message and quit.

We will now have a look at the NetView DM/6000 configuration for `rs60007` to see what the configuration script has configured.

The first thing done by the configuration script is modifying the NetView DM/6000 main configuration file `nvdm.cfg`:

```
WORKSTATION NAME:      rs60007
MESSAGE LOG LEVEL:     N
LAN AUTHORIZATION:     0
CONFIGURATION:         REMOTE_ADMIN_SERVER
MACHINE TYPE:          AIX
LOG FILE SIZE:         250000
TRACE FILE SIZE:       1000000
API TRACE FILE SIZE:   500000
TCP/IP PORT:           729
MAX TARGETS:           600
MAX CONNECTIONS:       50
MAX USER INTERFACES:   20
SERVER:                rs60007
REPOSITORY:            /usr/lpp/netviewdm/repos
SERVICE AREA:          /usr/lpp/netviewdm/service
BACKUP AREA:           /usr/lpp/netviewdm/backup
WORK AREA:             /usr/lpp/netviewdm/work
```

Figure 48. /usr/lpp/netviewdm/db/nvdm.cfg File on rs60007

The configuration script has changed the `WORKSTATION NAME`, `LOG FILE SIZE` and `SERVER` fields.

To examine the users created or changed by the script, we type:

```
lsuser -a groups root,suelpen,mike
```

This should produce the following output:

```
root groups=system,bin,sys,security,cron,audit,FNDADMN
suelpen groups=staff,FNDADMN
mike groups=staff,FNDBLD
```

Figure 49. Output from lsuser Command

Users `root` and `suelpen` have been defined to be NetView DM/6000 administrators, whereas user `mike` has been defined to be a NetView DM/6000 builder.

For the SNA/DS connection to `rs600015` the script has created a connection file `/usr/lpp/netviewdm/db/snads_conn/RS600015`:

```
PROTOCOL:                          TCP/IP
REMOTE SERVER NAME:               rs600015
TCP/IP TIME-OUT:                  300
NEXT DSU:                         RS600015.RS600015
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120
```

Figure 50. /usr/lpp/netviewdm/db/snads_conn/RS600015 File

The SNA/DS routing table looks like the following:

```
NETWORK PROTOCOL: TCP/IP

#
# SNA connections
#



#
# TCP/IP connections
#

RS600015.*                       RS600015
```

Figure 51. /usr/lpp/netviewdm/db/routetab File on rs60007

To see the targets created by the configuration script we type:

```
nvdm lstg -l '*'
```

This should produce the following output:

```
Target:                 rs600015
Description:
Customer name:
Contact name:
Telephone number:
Manager:
Mailing address:
Mode:                   Remote
Short name:             RS600015
Network ID:             RS600015

Target:                 rs60004
Description:            ITSO Raleigh test client
Customer name:          IBM
Contact name:           Stefan Uelpenich
Telephone number:       4711
Manager:                Wolfgang Geiger
Mailing address:
Mode:                   Push
Operating system:       AIX
Short name:             RS60004
Network ID:             RS60007
LAN address:
CM window:              00:00:00 - 23:59:00
Distribution window:    00:00:00 - 23:59:00
Logging level:          Normal
Tracing state:          Off
Installation parms:     None.
Hardware parms:         None.
Discovered inventory:   None.
Users:                  mike
                        root

Target:                 rs60007
Description:            ITSO Raleigh development
Customer name:          IBM
Contact name:           Stefan Uelpenich
Telephone number:       4711
Manager:                Wolfgang Geiger
Mailing address:
Mode:                   Push
Operating system:       AIX
Short name:             RS60007
Network ID:             RS60007
LAN address:
CM window:              00:00:00 - 23:59:00
Distribution window:    00:00:00 - 23:59:00
Logging level:          Normal
```

Figure 52 (Part 1 of 2). Output from lstg Command

```
Tracing state:         Off
Installation parms:    None.
Hardware parms:        None.
Discovered inventory:  None.
Users:                 root
                       suelpen
```

Figure 52 (Part 2 of 2). Output from lstg Command

To see the groups created by the configuration script we type:

```
nvdm lsgp '*'
```

This should produce the following output:

```
Group                          Mode      Description

Group1                         Push      Raleigh Group1
```

Figure 53. Ouput from lsgp Command

5.3 Automating the Configuration Process

In the previous example, we started the configuration for node `rs60007` manually.

If we have a large number of nodes to be configured we do not want to copy the configuration files and initiate the configuration process on each node manually, therefore we create a simple script to perform the configuration of nodes on the network from a central configuration server.

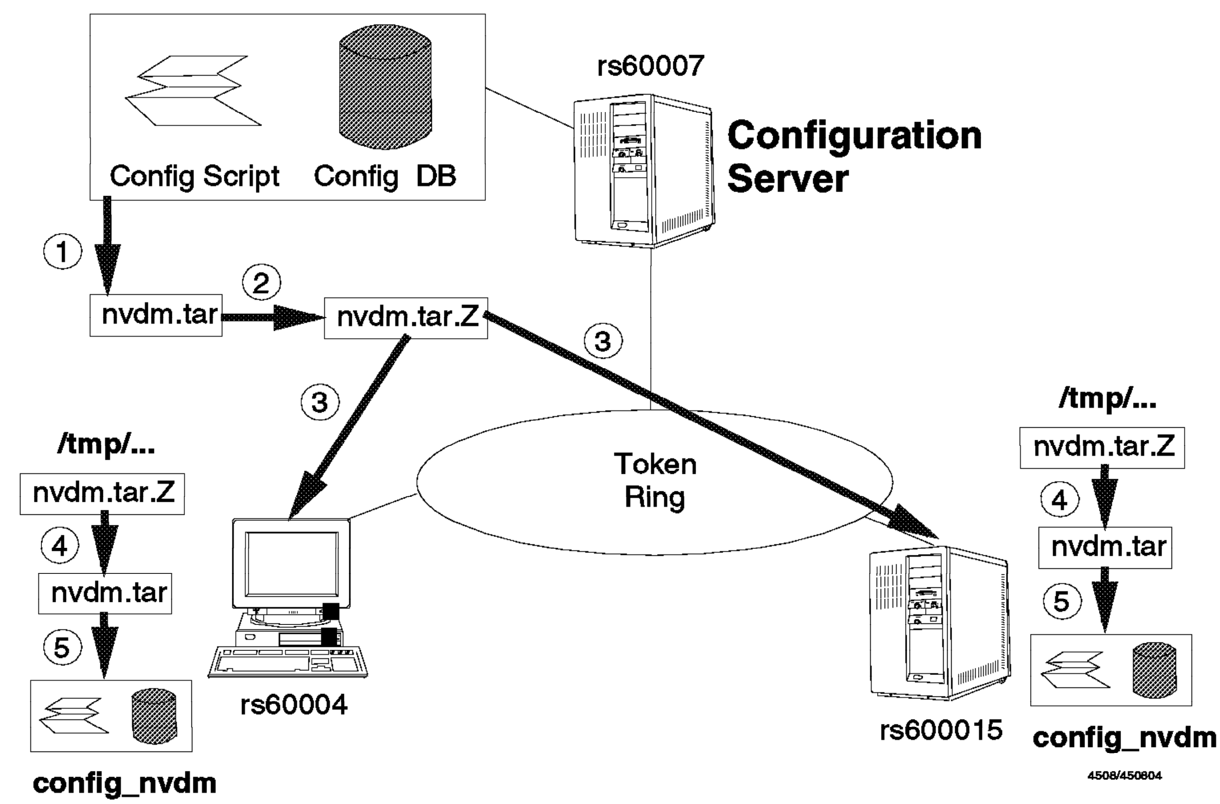


Figure 54. Automating the Configuration Process

The script will perform the following tasks:

1. Create a `tar` archive containing all files needed to configure a node, including the ODM database files.
2. Compress the `tar` file using `compress`.
3. Copy the compressed `tar` file to the node to be configured.
4. Decompress the file on the node to be configured using `uncompress`.
5. Extract the `tar` archive on the node to be configured.
6. Invoke the configuration script `config_nvdm` on the node to be configured thus initiating the configuration process.

Note

In our scenario it is normally not necessary to compress the `tar` file before transmitting because we use a relatively fast network. However, if you want to configure nodes, for example, in a WAN network such as X.25, it might be a good idea to compress the files thus saving transmission time.

In order to perform the configuration of the remote nodes the configuration server uses the commands `rsh` and `rcp`. Therefore all nodes to be configured have to have a file `/.rhosts` containing the name of the configuration server, in our case `rs60007`.

Hence, in our example the `/.rhosts` file on nodes `rs60004` and `rs600015` contains the line:

```
rs60007.itso.ral.ibm.com
```

Note

Remember to use the fully qualified host name when making the entry in the `/.rhosts` file and to refresh the inetd subsystem by typing `refresh -s inetd`.

The following figure shows the script that performs remote configuration of NetView DM/6000 nodes:

```
#!/bin/ksh
#
# Copy Configuration to all Nodes and execute configuration script
# For this to work each system to be configured has to have
# an entry for the central installation system in it's /.rhosts file
# Author : Stefan Uelpenich / IBM Germany
#

print "**** CONFIGURING NETVIEW DISTRIBUTION MANAGER/6000 ****"

#
# to reduce network traffic we will compress the
# installation files before transmitting them
#

print "** Creating tar archive"
tar -cvf/tmp/nvdm.tar . >/dev/null
SIZE=`ls -l /tmp/nvdm.tar | awk '{ print $5 }'`
print "Size before compressing : $SIZE"
print "** Crunching tar archive"
rm /tmp/nvdm.tar.Z 2>/dev/null
compress /tmp/nvdm.tar
SIZE=`ls -l /tmp/nvdm.tar.Z | awk '{ print $5 }'`
print "Size after compressing : $SIZE"

LIST=`cat node_list`
if [ "$LIST" != "" ]
then
for i in $LIST
do
   print "*** Processing node : $i"
   print "** Copy compressed archive"
   rcp /tmp/nvdm.tar.Z $i:/tmp
   print "** Uncrunching compressed archive"
   rsh $i rm /tmp/nvdm.tar
   rsh $i uncompress /tmp/nvdm.tar
   print "** Extracting files from tar archive"
   rsh $i cd /tmp
   rsh $i "cd /tmp ; tar -xvf/tmp/nvdm.tar 1>/dev/null 2>&1"
   print "Creating ODM DB ..."
   rsh $i /tmp/build_net_db
   print "Invoking configuration script..."
   rsh $i /tmp/config_nvdm $i
done
fi
```

Figure 55. configure_network Shell Script

The nodes to be configured remotely are listed in the file `node_list`, so in our scenario this file has the following entries:

```
rs60004
rs600015
```

The script is started on rs60007 by typing ./configure_network. To redirect the output to a log file we type:

```
./configure_network 2>&1 | tee network.log
```

Note

The configure_network shell script requires that all files needed to configure a node are stored in the same directory where the script itself resides.

This is because the script creates a tar archive of all files needed to configure a node. For simplicity this tar archive will contain all files that are located in the same directory as configure_network.

You have to supply the following files in the directory:

- The configuration script config_nvdm
- The ODM creation file, for example, config_db2.cre
- The ODM definition files, for example, nvdm_node.odmadd
- The program to modify the root.cli file (uicfg)

Since the tar archive is decompressed at the /tmp directory of the target system, you also need to supply a slightly modified version of the build_db shell script, which has to be named build_net_db.

The following figure shows this script:

```
odmcreate -c /tmp/config_db2
odmadd /tmp/nvdm_cfg_static.odmadd
odmadd /tmp/nvdm_groups.odmadd
odmadd /tmp/nvdm_node.odmadd
odmadd /tmp/nvdm_queues.odmadd
odmadd /tmp/nvdm_users.odmadd
odmadd /tmp/nvdm_servers.odmadd
```

Figure 56. build_net_db Shell Script

The following figure shows the configuration protocol:

```
**** CONFIGURING NETVIEW DISTRIBUTION MANAGER/6000 ****
** Creating tar archive
Size before compressing : 1136640
** Crunching tar archive
Size after compressing : 313686
*** Processing node : rs600015
** Copy compressed archive
** Uncrunching compressed archive
** Extracting files from tar archive
Creating ODM DB ...
nvdm_groups
nvdm_node
nvdm_users
nvdm_cfg_static
nvdm_servers
nvdm_queues
Invoking configuration script...
NVDM CONFIG : --> Trying to configure node rs600015
NVDM CONFIG : Node type is 0 (0 = Server, 1 = Agent, 2 = Prep)
NVDM CONFIG : --> NVDM Base Node Configuration
NVDM CONFIG : Setting nvdm.cfg (WORKSTATION NAME) to rs600015
NVDM CONFIG : Setting nvdm.cfg (SERVER) to rs600015
NVDM CONFIG : Setting nvdm.cfg (LOG FILE SIZE) to 250000
NVDM CONFIG : Setting nvdm.cfg (TCP/IP PORT) to 729
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Setting SNA Network Name to USIBMRA
NVDM CONFIG : Setting SNA Datalink Device to tok0
NVDM CONFIG : Setting SNA Remote Link Address to 400001240000
NVDM CONFIG : Setting SNA NVDM Mode Profile Name to NVDMNORM
NVDM CONFIG : Setting SNA NVDM Mode Name to NVDMNORM
NVDM CONFIG : Setting SNA TPN Profile Name (Send) to NVDMSND
NVDM CONFIG : Setting SNA TPN Profile Name (Receive) to NVDMRCV
NVDM CONFIG : Setting SNA Partner LU Name (MVS Host) to RA39TCF1
NVDM CONFIG : Setting SNA Side Info Profile Name (Send) to NVDMSIDS
NVDM CONFIG : Setting SNA Side Info Profile Name (Receive) to NVDMSIDR
NVDM CONFIG : Setting Solicit SSCP Field (yes|no) to yes
NVDM CONFIG : Setting I-Field Size to 2042
NVDM CONFIG : Setting SNA Local SAP No. to 04
NVDM CONFIG : Setting Remote SAP No. to 04
NVDM CONFIG : Setting SNA Initiate Call Field (yes|no) to yes
NVDM CONFIG : Setting SNA Activate on start (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on normal termination (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on abnormal termination (yes|no) to yes
NVDM CONFIG : Setting SNA VTAM CP Name (for LU6.2 Location Profile) to RAK
NVDM CONFIG : Setting PU NAME for rs600015 to RA60015
NVDM CONFIG : Setting Local LU Name for rs600015 to RA60015B
NVDM CONFIG : Setting Control Point Name for rs600015to RA6015CP
NVDM CONFIG : Could not determine XID for rs600015 configu
ration.
```

Figure 57 (Part 1 of 4). Configuration Log File network.log

```
NVDM CONFIG : Setting USE_CP_XID to yes
NVDM CONFIG : --> Configuring SNA
NVDM CONFIG : Adding DLC Device for tok0
NVDM CONFIG : Configuring SNA Initial Node Setup
+ mk_qcinit -y token_ring -t appn_end_node -w USIBMRA -d RA6015CP
The SNA DLC Profile 'tok0.00001' has been created successfully.
+ chsnaobj -t control_pt -e USIBMRA -a RA6015CP -A RA6015CP
 -N appn_end_node node_cp
NVDM CONFIG : Configuring SNA Control Point Profile
======================================================================
+ mksnaobj -t sna_dlc_token_ring -d tok0 -b yes -w yes -m 2042
 -H 04 -c no -q 0 tok0
Profile type 'control_pt' name 'node_cp' CHANGED.
======================================================================
NVDM CONFIG : Configuring SNA DLC Profile
======================================================================
+ RC=0
Profile type 'sna_dlc_token_ring' name 'tok0' ADDED.
======================================================================
+ mksnaobj -t link_station -w token_ring -y tok0 -d 400001240000
 -l 07100000 -s 04 -a yes -O yes -F yes -h yes -z yes -c yes RA60015
+ RC=0
NVDM CONFIG : Configuring SNA Link Station Profile
======================================================================
Profile type 'link_station_token_ring' name 'RA60015' ADDED.
+ mksnaobj -t local_lu -u lu6.2 -l RA60015B -L RA60015B RA60015B
+ RC=0
======================================================================
NVDM CONFIG : Configuring SNA Local LU Profile
======================================================================
Profile type 'local_lu_lu6.2' name 'RA60015B' ADDED.
+ mksnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N #CONNECT -m NVDMNORM NVDMNORM
+ RC=0
+ mksnaobj -t local_tp -n 21F0F0F7 -h yes -c basic -d 0 -P yes
 -w /usr/lpp/netviewdm/bin/fndts -s none NVDMSND
======================================================================
NVDM CONFIG : Configuring SNA Mode Profile
======================================================================
Profile type 'mode' name 'NVDMNORM' ADDED.
======================================================================
NVDM CONFIG : Configuring SNA TPN Profile (SEND)
======================================================================
+ RC=0
+ mksnaobj -t local_tp -n 21F0F0F8 -h yes -c basic -d 0 -P yes
 -w /usr/lpp/netviewdm/bin/fndtr -s none NVDMRCV
Profile type 'local_tp' name 'NVDMSND' ADDED.
======================================================================
NVDM CONFIG : Configuring SNA TPN Profile (Receive)
======================================================================
+ RC=0
```

Figure 57 (Part 2 of 4). Configuration Log File network.log

```
+ mksnaobj -t partner_lu6.2 -p no -P USIBMRA.RA39TCF1 -O none
-A RA39TCF1 RA39TCF1
Profile type 'local_tp' name 'NVDMRCV' ADDED.
=====================================================================
NVDM CONFIG : Configuring SNA LU6.2 Partner LU
=====================================================================
Profile type 'partner_lu6.2' name 'RA39TCF1' ADDED.
+ RC=0
+ mksnaobj -t partner_lu6.2_location -P USIBMRA.RA39TCF1
-O USIBMRA.RAK -m link_station -l RA60015B -s RA60015 RA39TCF1
+ RC=0
=====================================================================
NVDM CONFIG : Configuring SNA LU 6.2 Location Profile
=====================================================================
Profile type 'partner_lu6.2_location' name 'RA39TCF1' ADDED.
+ mksnaobj -t side_info -L RA6015CP -P USIBMRA.RA39TCF1
-m NVDMNORM -d 21F0F0F7 -h yes NVDMSIDS
+ RC=0
+ mksnaobj -t side_info -L RA60015B -P USIBMRA.RA39TCF1
-m NVDMNORM -d 21F0F0F8 -h yes NVDMSIDR
=====================================================================
NVDM CONFIG : Configuring SNA Side Info Profile (Send)
=====================================================================
Profile type 'side_info' name 'NVDMSIDS' ADDED.
=====================================================================
NVDM CONFIG : Configuring SNA Side Info Profile (Receive)
=====================================================================
+ RC=0
Profile type 'side_info' name 'NVDMSIDR' ADDED.
=====================================================================
NVDM CONFIG : Updating SNA Server...
NOTE: The following profiles can ONLY be refreshed if there
      are currently no active resources using them.

Profile type 'mode' name 'NVDMNORM' CHANGED.

NOTE: The following profile refreshes will take effect when
      all active resources using these profiles deactivate.

Profile type 'side_info' name 'NVDMSIDS' CHANGED.

verifysna command OK.
The profiles listed above have been dynamically updated successfully.
NVDM CONFIG : Configuring TCP/IP connection
NVDM CONFIG : Configuring SNA/DS connection configuration file.
NVDM CONFIG : (TCP/IP) for remote Server RS60007.
NVDM CONFIG : Configuring APPC connection
NVDM CONFIG : Configuring SNA/DS connection configuration file
/usr/lpp/netviewdm/db/snads_conn/RA39TCF1
NVDM CONFIG : Configuring SNA/DS routing table.
```

Figure 57 (Part 3 of 4). Configuration Log File network.log

```
NVDM CONFIG : System has TCP/IP connection to remote server.
NVDM CONFIG : System has APPC connection to remote server.
NVDM CONFIG : Writing routing table to /usr/lpp/netviewdm/db/routetab
NVDM CONFIG : Defining Target rs600015 on server rs600015
NVDM CONFIG : Target already exists. Updating...
nvdm updtg rs600015 -s 'RS600015' -y 'AIX'
-d 'ITSO Raleigh test server' -q 'Stefan Uelpenich'
-o 'Wolfgang Geiger' -t '4711' -r 'IBM'
WARNING: The Network ID of this domain has been changed to RS600015.
NVDM CONFIG : Adding Target Users...
NVDM CONFIG : Adding root User
NVDM CONFIG : Adding suelpen User
NVDM CONFIG : Configuring Target Groups for rs600015
NVDM CONFIG : Adding group Group2
NVDM CONFIG : Defining remote target for rs60007
NVDM CONFIG : Defining remote target for RA39TCF1
NVDM CONFIG : RA39TCF1 will be configured as focal point.
+ eval nvdm addtg RA39TCF1 -m report_to -s RA39TCF1
-n USIBMRA -d 'NVDM_MVS'
+ nvdm addtg RA39TCF1 -m report_to -s RA39TCF1 -n USIBMRA -d NVDM_MVS
0513-029 The sna Subsystem is already active.
Multiple instances are not supported.
NVDM CONFIG : --> In order for the changes to become active
NVDM CONFIG :     NetView DM/6000 will be restarted on this node
NVDM CONFIG : Stopping NVDM.
NVDM CONFIG : Restarting NVDM.
NVDM CONFIG : Releasing NVDM SNA communications.
NVDM CONFIG : !!! Configuration of Server completed successfully !!!
*** Processing node : rs60004
** Copy compressed archive
** Uncrunching compressed archive
** Extracting files from tar archive
Creating ODM DB ...
nvdm_groups
nvdm_node
nvdm_users
nvdm_cfg_static
nvdm_servers
nvdm_queues
Invoking configuration script...
NVDM CONFIG : --> Trying to configure node rs60004
NVDM CONFIG : Node type is 1 (0 = Server, 1 = Agent, 2 = Prep)
NVDM CONFIG : --> NVDM Base Node Configuration
NVDM CONFIG : Setting nvdm.cfg (WORKSTATION NAME) to rs60004
NVDM CONFIG : Setting nvdm.cfg (SERVER) to rs60007
NVDM CONFIG : Setting nvdm.cfg (LOG FILE SIZE) to 250000
NVDM CONFIG : Setting nvdm.cfg (TCP/IP PORT) to 729
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Adding user mike to AIX OS.
NVDM CONFIG : Starting NVDM Agent (fndcmps)....
```

Figure 57 (Part 4 of 4). Configuration Log File network.log

Chapter 6. Using the ODM Editor to Change the Configuration

We now show a simple example where we use the ODM editor to change the configuration of our software distribution network.

The ODM editor is a simple tool which assists in changing or adding objects (instances) in ODM classes. Basically it is a user interface to the `odmadd`, `odmget` and `odmdelete` commands.

Note

The ODM editor is only available in AIX 3.2, whereas in AIX 4.1 this tool has been removed.

Therefore in AIX 4.1 you have to use the above commands. What you can also do use the `odme` executable file `/bin/odme` from an AIX 3.2 system at the AIX 4.1 system.

The example shows how easily the configuration of our software distribution network can be changed. A change of the configuration requires the following steps:

- Changing the configuration database by changing the ODM object classes (using the ODM editor)
- Distributing the updated database to all nodes that are affected by the reconfiguration
- Processing the configuration script on all nodes that are affected by the reconfiguration

Warning

You can use the ODM editor `odme` to edit any object class on your AIX system. This includes the classes containing the operating system configuration, for example, `CuDv`. We do not recommend that you do this unless you are very experienced in AIX. Otherwise, this may cause unpredictable results on your system.

6.1 Editing the Configuration

The ODM editor is invoked by typing `odme`. This will pop up the following screen:

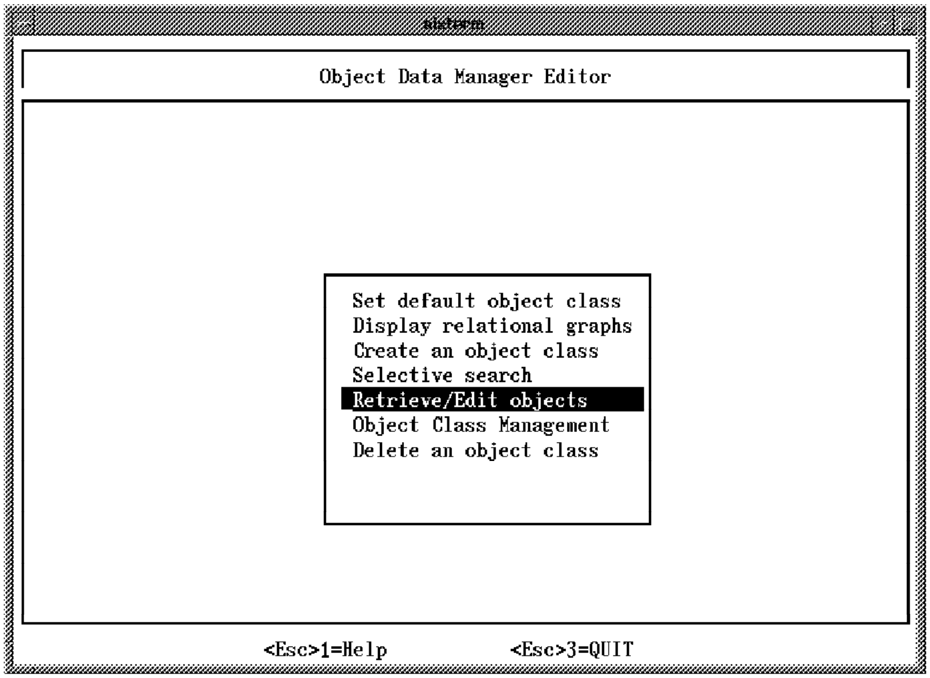


Figure 58. odme Startup Window

Note

We have developed a simple script `build_db` in Figure 7 on page 22 which we used to update the ODM database. You must not use this script after you have changed the ODM manually by using `odme`. This is because `build_db` clears all classes and refills them with the data from the *.odmadd files which would remove the changes made using `odme`.

To prevent this from happening we introduce a second script `rebuild_db` which will keep the ODM data files up to date:

```
#!/bin/ksh
#
# rebuild ODM definition files (odmadd)
# from existing ODM class files
#
# S.Uelpenich
#

# determine all classes beginning with "nvdm"

LIST=`ls /etc/objrepos/nvdm*`

for i in $LIST
do
    CLASS=`basename $i`
    print "$CLASS"
    odmget $CLASS >$CLASS".odmadd"
done
```

Figure 59. rebuild_db Shell Script

The script will look for all ODM files located in the default ODM directory `/etc/objrepos` and their names starting with nvdm.

Note

If you decide to have your ODM class names not starting with nvdm you will have to modify the `rebuild_db` script to reflect that change.

Then an `odmget` command is applied to all of these class files, thus getting the ODM data file, which is the source file from which this ODM class file has been created.

The output from the `odmget` command will be redirected to a file `classname`.odmadd.

Anytime you have used `odme` to change data in the ODM you have to invoke `rebuild_db` to update the corresponding data definition files.

When you invoke `build_db` then this will not remove the changes made using `odme`.

To ensure that you use `rebuild_db` after using `odme` to change data we recommend that you use another shell script `edit_db`:

```
#!/bin/ksh
# edit configuration database
# first call odme, then rebuild the
# odmadd files
#

if [ $# -ne 1 ]
then
  print "Syntax : $0 [ODM class name]"
  exit 1
fi

odme $1
./rebuild_db
```

Figure 60. edit_db Shell Script

To edit the configuration database you can type `edit_db` **`classname`**. The script will call `odme` and `rebuild_db` afterwards automatically.

We will change the configuration of our software distribution network now by moving target `rs60004` from server `rs60007` to `rs600015`.

To achieve this we have to perform the following steps:

1. Edit the `nvdm_node` object class
2. Run the configuration script on all affected nodes

To edit the `nvdm_node` object class we type:

```
./edit_db nvdm_node
```

The `odme` startup screen will appear as shown in Figure 58 on page 100.

We select **Retrieve/Edit objects**. This will pop up the following screen:

```
                                   aixterm
┌─────────────────────────────────Object Display─────────────────────────────────┐
│ Object Class : nvdm_node         Object: 3      Descriptor: 1   of  12           │
├──────────────────────────────────────────────────────────────────────────────────┤
│ node_name                  node_type  short_name  target_os    description      │
│ ODM_CHAR                   ODM_SHORT  ODM_CHAR    ODM_CHAR     ODM_CHAR         │
├──────────────────────────────────────────────────────────────────────────────────┤
│ rs60007                    0          RS60007     AIX          ITSO Raleigh d   │
│ rs600015                   0          RS600015    AIX          ITSO Raleigh t   │
│ rs60004                    1          RS60004     AIX          ITSO Raleigh t   │
│                                                                                  │
└──────────────────────────────────────────────────────────────────────────────────┘
 <Esc>1=Help   <Esc>2=Search   <Esc>3=EXIT     <Esc>4=Add    <Esc>5=Delete
 <Esc>6=Copy   <Esc>7=PgUp     <Esc>8=PgDown   <Esc>9=Left   <Esc>0=Right
```

Figure 61. odme Retrieve/Edit Objects Window

We do the following:

1. Move the cursor down to `rs60004` using the arrow down key.
2. Move the cursor to the `server_name` column using the Tab key.

You should see the following screen:

```
                                   aixterm
┌─────────────────────────────────Object Display─────────────────────────────────┐
│ Object Class : nvdm_node         Object: 3      Descriptor: 11  of  12           │
├──────────────────────────────────────────────────────────────────────────────────┤
│ telephone_number     customer_name        x_25_number     server_name           │
│ ODM_CHAR             ODM_CHAR             ODM_CHAR        ODM_CHAR              │
├──────────────────────────────────────────────────────────────────────────────────┤
│ 4711                 IBM                                  rs60007               │
│ 4711                 IBM                                  rs600015              │
│ 4711                 IBM                                  rs60007               │
│                                                                                  │
└──────────────────────────────────────────────────────────────────────────────────┘
 <Esc>1=Help   <Esc>2=Search   <Esc>3=EXIT     <Esc>4=Add    <Esc>5=Delete
 <Esc>6=Copy   <Esc>7=PgUp     <Esc>8=PgDown   <Esc>9=Left   <Esc>0=Right
```

Figure 62. odme Retrieve/Edit Objects Window

To move target rs60004 from server rs60007 to rs600015, we simply change the entry for server_name from rs60007 to rs600015.

To leave odme we press F3. A window will pop up asking if we want to commit changes. To do so we hit the Y key. This will get us back to the odme main window.

We press F3 again to leave odme. The following lines appear, indicating that edit_db updates the *.odmadd files:

```
nvdm_cfg_static
nvdm_groups
nvdm_node
nvdm_queues
nvdm_servers
nvdm_users
```

Before we run the configuration script again and thus move target rs60004 to server rs600015 we install a change file on rs60004 from server rs60007 to show how the change management history for the target is moved.

To do so we start the NetView DM/6000 graphical user interface on rs60007:

```
nvdmgi &
```

The following panel will pop up:

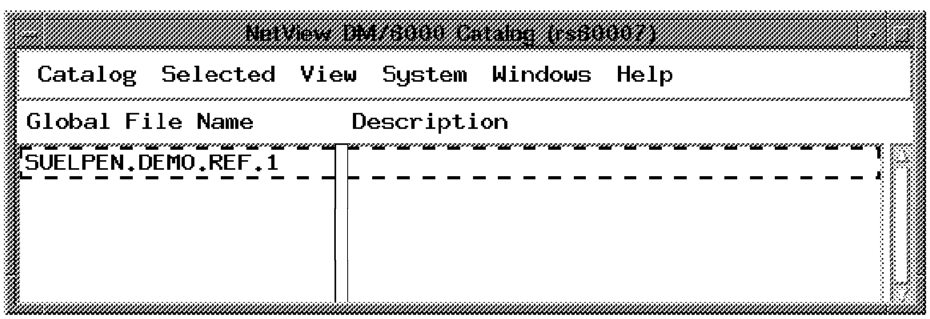


Figure 63. NetView DM/6000 Catalog Window (rs60007)

We install the dummy change file SUELPEN.DEMO.REF.1, which we have created before, on target rs60004. This change file is installed just to show the change management history, so its content is not important.

To install the change file we select **Selected** from the menu bar and then **Install...** from the pull-down menu.

The following panel will appear:

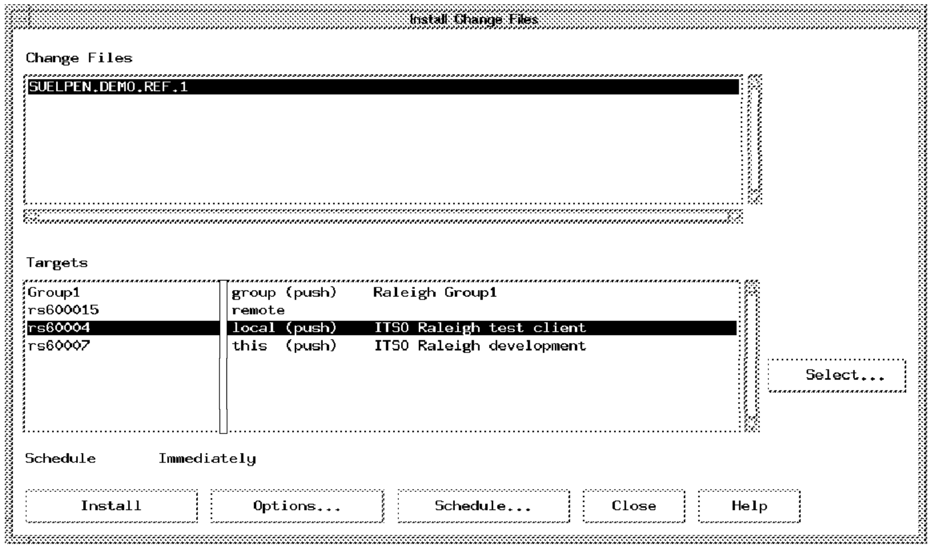


Figure 64. Install Change Files Window

We do the following:

1. Select target `rs60004` and then the **Install** push button. This will install the change file on `rs60004`.
2. Select the **Close** push button to close the Install Change Files window.
3. In the Catalog window select **Windows** from the menu bar and then **Targets** from the pull-down menu. This will open the Targets window.
4. In the Targets window select `rs60004`.
5. Select **Selected** from the menu-bar.
6. From the pull-down menu select **Open** and then **History...** from the cascaded menu.

The Target History window will appear:

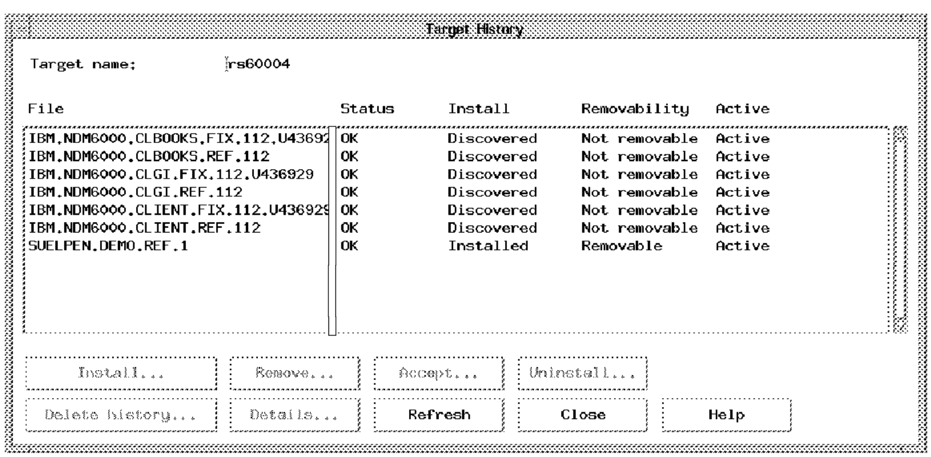


Figure 65. Target History Window

6.2 Reconfiguring the Network

We now reconfigure our network. First, we reconfigure `rs60007` by typing the following command on `rs60007`:

```
config_nvdm rs60007
```

The script will recognize from the database that target `rs60004` is no longer configured for server `rs60007` and therefore saves the target history.

Since `rs60004` and `rs600015` are also affected by the change in the database, they will also need reconfiguration. To perform the reconfiguration we will use the script `configure_network` on `rs60007`.

The `node_list` file must contain the following entries:

```
rs600015
rs60004
```

After the `configure_network` script has completed we start `nvdmgi &` on `rs600015` to see if the target as well as the target history has been moved to `rs600015`.

The following panel will appear:

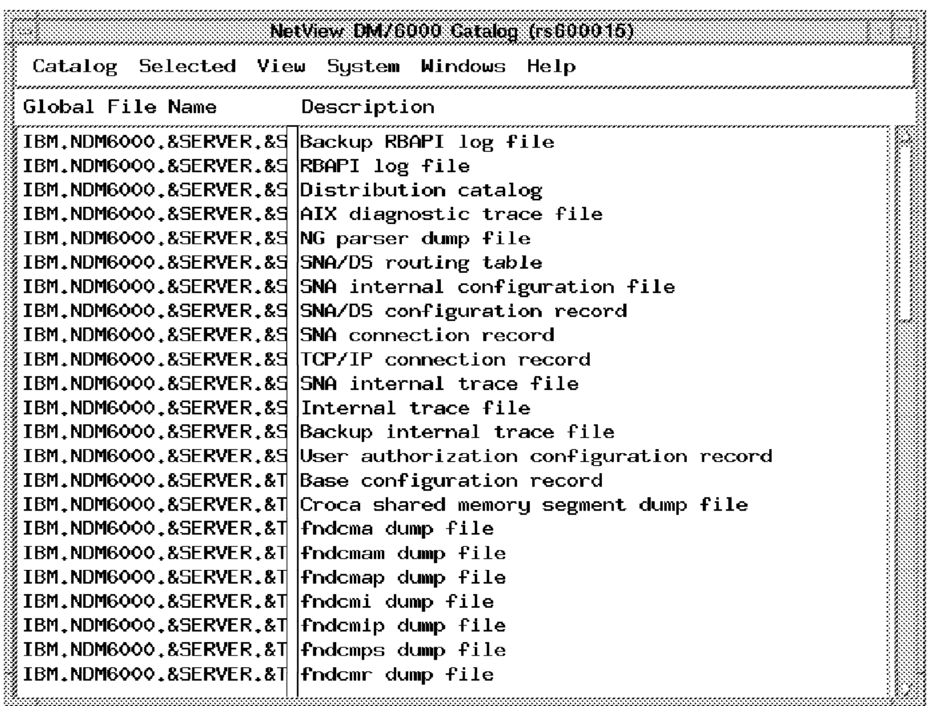


Figure 66. NetView DM/6000 Catalog (rs600015) Window

We select **Windows** from the menu-bar and then **Targets...** from the pull-down menu.

This will pop up the Targets window:

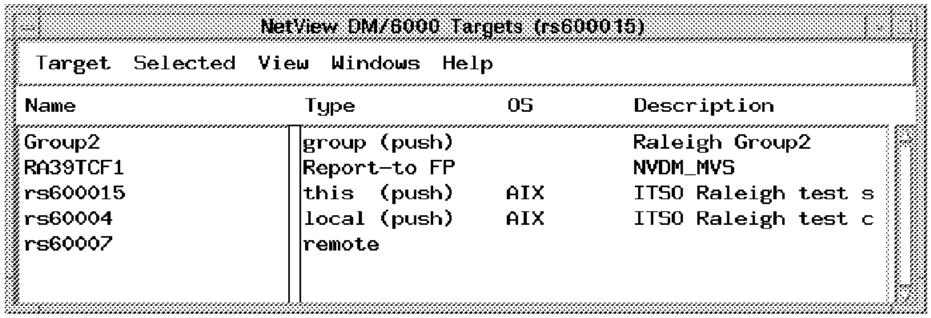


Figure 67. NetView DM/6000 Targets (rs600015) Window

We do the following:

1. Select target `rs60004` from the target list.
2. Select **Selected** from the menu bar.
3. Select **Open** from the pull-down menu.
4. Select **History...** from the cascaded menu.

The Target History window will appear:

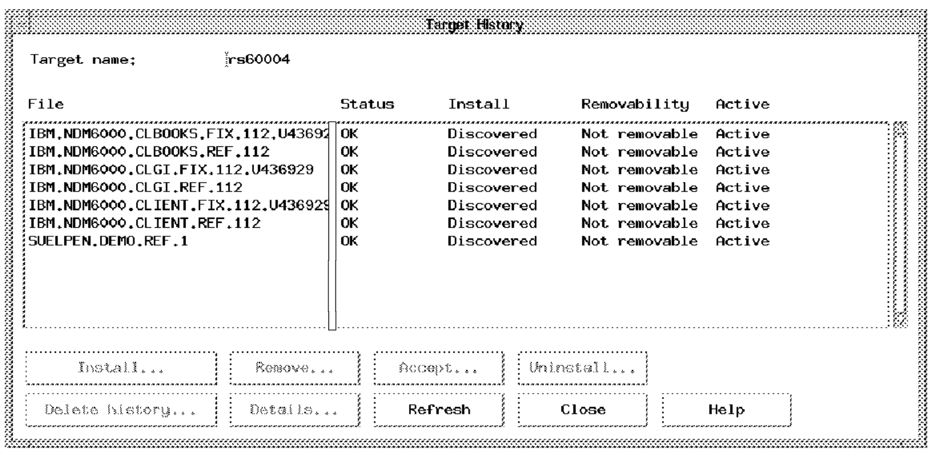


Figure 68. Target History Window After configure_network Completed

As you can see, the target history has been moved to `rs600015` together with the target itself.

Note

You should notice that the change file `SUELPEN.DEMO.REF.1` now shows up as Discovered, Not removable whereas on `rs60007` it was Installed, removable. This is because now the change management history for the target is gathered from the software inventory file on `rs60004`. All entries in the software inventory file will show up as Discovered, Not removable in the target history.

6.3 Other Ways to Store Configuration Data

In the above scenario we used the ODM editor because we stored our configuration database in ODM classes.

Consider the following alternative:

- We could have stored configuration data in a relational database system such as DB2/6000. Of course we would have to change the database access procedures in our configuration script then. This would enable us to change the configuration of the software distribution network by simply using SQL commands to modify the configuration database (see Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219).
- Let's say we have a large organization which has a network containing 100 NetView DM/6000 servers all connected to a central NetView DM/MVS system using SNA LU 6.2. At some point the organization decides to change its naming scheme for LU 6.2 names. This will require the SNA Server configuration to be changed on all NetView DM/6000 servers.
- If we stored the configuration in an SQL database we could just issue a simple `update` command to change all relevant table entries. Then we could run the configuration script again on all servers thus updating the SNA Server configuration on all nodes. (We realize that it is not that simple.)

Note

In fact we show how to use DB2/6000 to store configuration data in Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219. Further we show how to edit data stored in DB2/6000 in 15.1, "Using the Graphical Interface for Changing Configuration Data" on page 295.

Chapter 7. Customizing and Extending the Configuration Procedure

In this chapter we give you guidance on how to customize the configuration procedure for your own environment.

We do this by giving you some general hints about the following tasks:

- Determining configuration commands
- Changing configuration files
- Adjusting the data model

For this chapter, it is required that you be familiar with some general programming concepts, including shell programming.

Also, you should be familiar with some common UNIX tools, including `sed`, `awk`, etc. However, if you are not familiar with all the commands used in this chapter you can always refer to the manpages or InfoExplorer.

7.1 Determining Configuration Commands

Changes in the configuration of products are normally made by using either of the following procedures:

- Update the configuration using commands
- Update the configuration by modifying configuration files

First we will concentrate on configuration commands.

The configuration procedure which we developed in the previous chapters of this book included more than the configuration of the NetView DM/6000 product itself. This is because for NetView DM/6000 to run we also need to configure related products, in our scenario namely the AIX base operating system and the SNA Server product.

Depending on what component we want to configure, the way of determining the necessary commands might be different.

7.1.1 Determining NetView DM/6000 Commands

All commands that we use to configure NetView DM/6000 are part of the NetView DM/6000 command-line interface.

A command is invoked by typing `nvdm` *command (parameters)*.

To get a list of available commands you can type `nvdm help` on your Software Distribution for AIX 3.1 server.

This will produce the following output:

```
Valid commands are

    acc         delf        hldrq       lsq         rld         unbld
    act         delpm       hldtg       lsrq        rstrq       uncat
    addgp       delprf      imp         lstg        rtrv        uninst
    addpm       delrq       inst        lsusr       rtrvf       upd
    addtg       deltg       inv         preqpln     send        updak
    addpln      delusr      log         prgq        sendf       updbs
    addprf      eraserq     ls          prtyq       start       updpln
    addusr      exec        lsak        relc        stat        updprf
    auth        execf       lsbs        relq        stattg      updpwd
    bld         execpln     lscf        relrq       stop        updrq
    cat         exp         lscm        reltg       svr         updtg
    del         help        lsgp        rem         troff       updusr
    delcm       hldc        lspln       rentg       tron        vercm
    delgp       hldq        lsprf       reset       unauth

Type nvdmgi at a graphics-capable command prompt to run the Graphical
Interface. Type nvdm command prompt to run an interactive command line
session.

Using the interactive command line session the following command can
be used in addition:
      ?       for help
      !       to activate the OS command shell
              (exit to leave the OS command shell)
      open    to open a connection with a NetView DM/6000 Server
      quit    to exit from the interactive command line session

To get more help for an individual command type nvdm help followed by
the command name. For example nvdm help ls gives more help on the ls
command.  If you are running in an interactive command line session
type just help or ?.
```

Figure 69. Software Distribution for AIX V3.1 Commands

For NetView DM/6000 Version 1.2 or previous versions, you have to type `nvdm` to get a list of all possible commands:

```
Valid commands are

      acc          delrq         log          relc         stattg
      act          deltg         ls           relq         stop
      addgp        exec          lsbs         reltg        svr
      addpm        exp           lscf         rem          troff
      addtg        help          lscm         rentg        tron
      auth         hldc          lsgp         reset        unauth
      bld          hldq          lsq          rld          uncat
      cat          hldtg         lsrq         rtrv         uninst
      del          imp           lstg         send         updbs
      delcm        inst          lsusr        start        updtg
      delgp        inv           prgq         stat         updusr
      delpm

If available on your system, type nvdmgi at a graphics-capable command
prompt to run the graphical interface.  To get more help for an individual
command type nvdm help followed by the command name. For example nvdm help
ls gives more help on the ls command.
```

Figure 70. NetView DM/6000 Commands

When configuring NetView DM/6000 we normally need two types of commands:

- Commands to determine the current configuration or status of NetView DM/6000
- Commands to modify the current configuration of NetView DM/6000

For example in the shell procedure `nvdm_configure_targets` which we developed previously we configure the local targets for a server.

To do so we need to find out if the target already exists or if we have to create a new one. The `nvdm lstg targetname` command can be used to find out if target *targetname* already exists.

If the target does not exist yet, the command will not return a return code being zero, so we know that the target does not exist.

If the target exists we will update the existing target using the `nvdm updtg` command.

The code fragment looks like the following:

```
nvdm lstg $i >/dev/null
if [ $? -ne 0 ]
then
  COMMAND="nvdm addtg $i"
else
  COMMAND="nvdm updtg $i"
fi
```

Note

It is important that you redirect the output of commands to `/dev/null` if they are used just to query the configuration. This prevents the output from appearing on the screen or in the log file.

The above example assumes that the target name is stored in the `$i` shell variable.

In general, `nvdm` commands to query the current configuration of NetView DM/6000 start with ls, such as `lstg`, `lsgrp`, etc. and commands to modify the configuration start with add or upd.

You can get a complete description of all `nvdm` commands either by consulting the *NetView Distribution Manager/6000 User's Guide SH19-5003* or by typing `man nvdm_`**`command`**, for example `man nvdm_lstg`.

To invoke configuration commands we normally will have to pass parameters, such as, for example, target names or flags.

Normally these parameters will be retrieved from the ODM database using the ODM access methods that we have developed previously.

We will discuss how parameters can be introduced to the ODM database in detail in 7.3, "Adjusting the Data Model" on page 121.

However, you always will have to decide whether you want to put parameters in the data base or hard-code them in the configuration script.

For example, the `-y` parameter used with the `nvdm addtg` or `nvdm updtg` command determines the target operating system for that target.

If you have only AIX targets in your network you might want to hard-code this parameter in your script by always using the flag as in `-y AIX`.

However, if you have other operating systems on your network too, you might want to store this parameter in the ODM like we did in our scenario.

7.1.2 Determining AIX Commands

In our configuration procedure we also had to use native AIX operating system commands, mainly when adding users to the AIX operating system.

A good way to determine the commands necessary to perform a certain task is using SMIT.

For example, if we want to know the command to add a user to the AIX operating system we can type:

```
smitty user
```

Note

In the above example, `user` is the appropriate fast path to get immediately to the SMIT section dealing with commands related to AIX users. Normally the fast path to get to a certain SMIT section is quite easy to guess, for example, to get to the section dealing with file systems you can type `smitty fs`.

However, if you do not know the appropriate fast path you can just type `smitty` to get to the main menu and then walk through the menus until you get to the right place.

The following panel will appear:

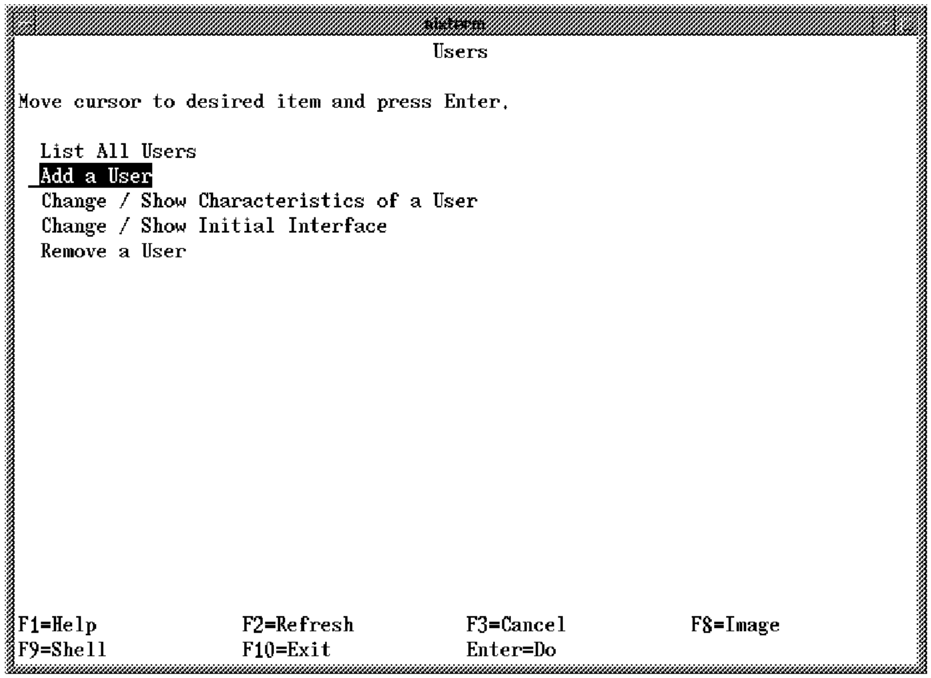
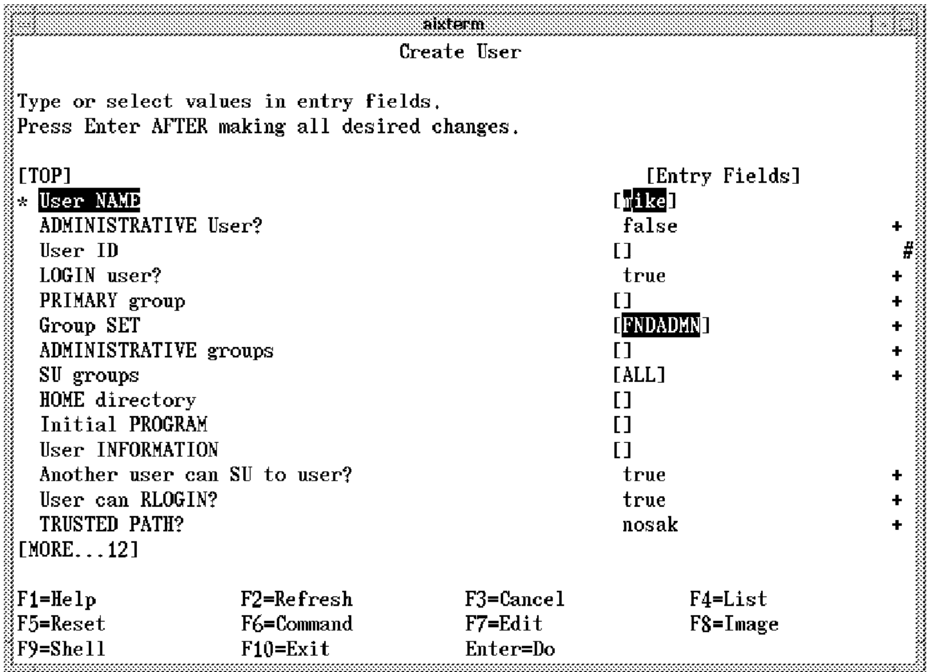


Figure 71. SMIT User Menu

We want to know the command to add a user to AIX, so we move the cursor to **Add a user** using the arrow down key and then press Enter.

This will pop up the following panel:



```
aixterm
                              Create User

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                 [Entry Fields]
* User NAME                                          [mike]
  ADMINISTRATIVE User?                                false              +
  User ID                                            []                  #
  LOGIN user?                                         true               +
  PRIMARY group                                      []                  +
  Group SET                                          [FNDADMN]           +
  ADMINISTRATIVE groups                              []                  +
  SU groups                                          [ALL]               +
  HOME directory                                     []
  Initial PROGRAM                                    []
  User INFORMATION                                   []
  Another user can SU to user?                        true               +
  User can RLOGIN?                                    true               +
  TRUSTED PATH?                                       nosak              +
[MORE...12]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 72. SMIT Create User Panel

We enter the parameters in the panel that we expect to be needed for the configuration.

In the example we do the following:

1. Enter `mike` in the User NAME field.
2. Enter `FNDADMN` in the Group SET field.

To see the command that SMIT will generate for that panel we press F6, which will result in the following screen:

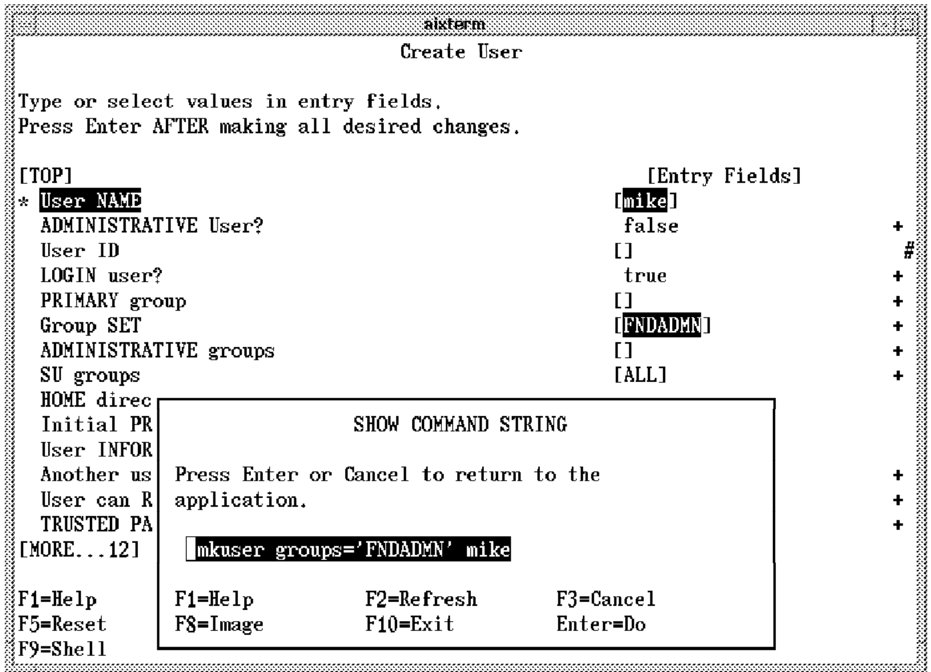


Figure 73. SMIT Show Command String Window

In the above example the command needed to add user mike to AIX, having FNDADMN in his group set is:

```
mkuser groups='FNDADMN' mike
```

We can now code the above statement in our configuration script. However, we will normally have to store parameters in shell variables, so the command could look like:

```
mkuser groups=$NVDMGRP $USERNAME
```

The above statement assumes that the shell variable `NVDMGRP` holds the group set and `USERNAME` holds the AIX user name. These variables could have been filled before, using the ODM access methods to query the ODM database.

Note

You should notice that in the configuration script we do not use the `mkuser` command to modify the group set of the user as shown in the above example.

In fact we use the `mkuser` command to create the user and then the `chuser` command to change the group set.

For configuration of the AIX operating system we will also use two types of commands, normally:

- Commands determining the current configuration of AIX
- Commands changing the current configuration of AIX

The following are also some general rules about command names in AIX:

- Commands adding an object to AIX start with either mk or cr, as in `mkuser` or `crfs`.
- Commands showing the current configuration of AIX start with ls, as in `lsfs`, `lsuser`, or `lsgroup`.
- Commands removing an object from AIX start with rm, as in `rmuser` or `rmfs`.
- Commands changing or updating the configuration of AIX start with ch, as in `chuser` or `chfs`.

Once you have found the command that you can use to perform the task you want, you can refer to the manpage of that command to get a list of all possible flags, for example:

```
man mkuser
```

This will also give you related information, for example, that the defaults for the `mkuser` command are stored in `/etc/security/user/mkuser.default`.

Information like this is often helpful to avoid coding errors, so if you do not know a command you want to use in full detail, you should always consult the manpage first.

7.1.3 Determining SNA Server Commands

SNA Server, like almost any IBM product for AIX, offers a SMIT interface to configure and control it. The fast path to get to the SNA Server section of SMIT is `sna`, so we type `smitty sna` to get to the following panel:

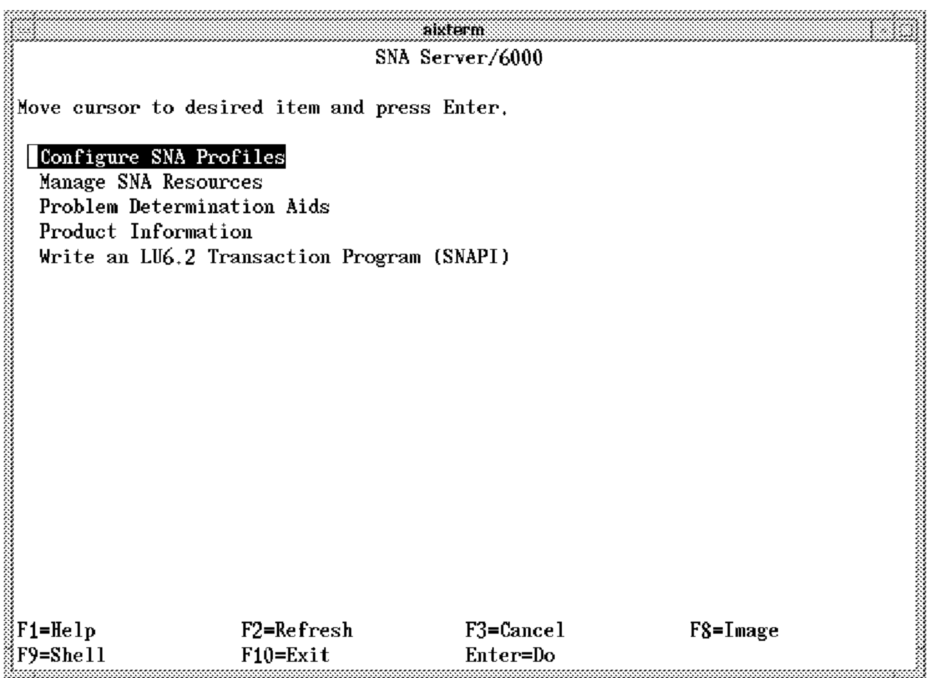


Figure 74. SMIT SNA Server/6000 Menu

For our purposes we will mainly use the sub menus Configure SNA Profiles and Manage SNA Resources.

First we will show an example of how to configure an SNA profile.

For that purpose we do the following:

1. Move the cursor to **Configure SNA Profiles** and press Enter. This will get us to the Configure SNA Profiles menu.
2. Move the cursor to **Advanced Configuration** and press Enter. This will get us to the Advanced Configuration menu.
3. Move the cursor to **Sessions** and press Enter.
4. Move the cursor to **LU 6.2** and press Enter.
5. Move the cursor to **LU 6.2 Local LU** and press Enter.
6. Move the cursor to **Add a profile** and press Enter.

This should get us to the Add LU 6.2 Local LU Profile panel:

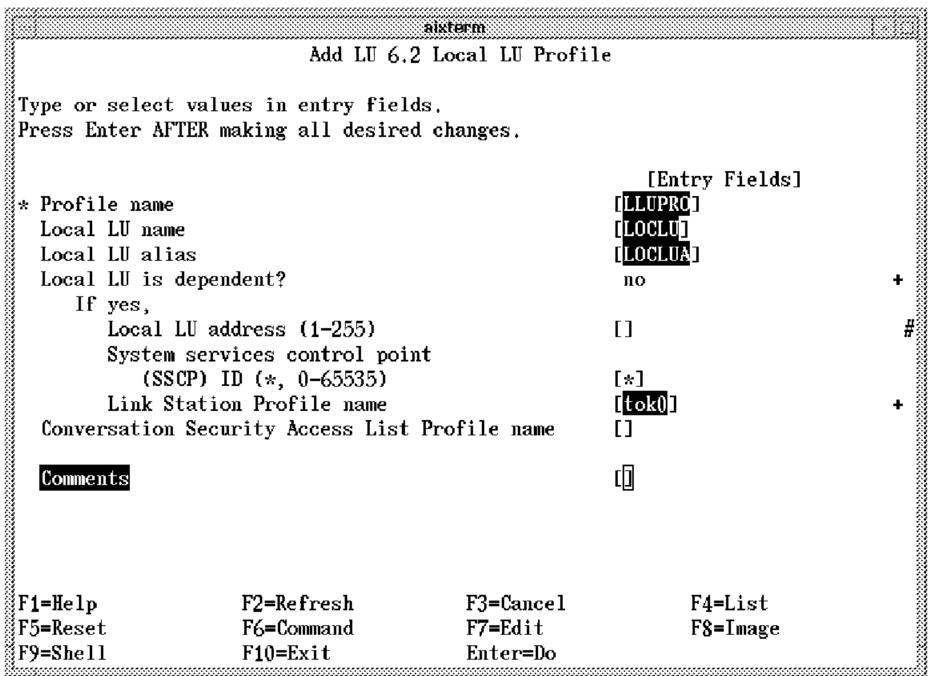


Figure 75. SMIT Add LU 6.2 Local LU Profile panel

We enter the values as shown in the above panel and then press F6.

This will pop up the following window:

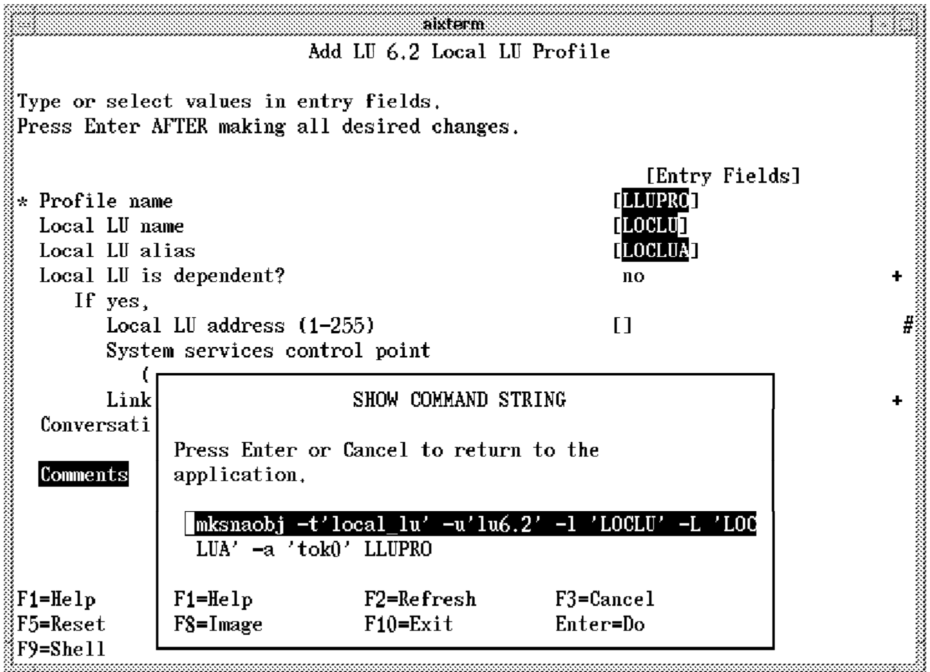


Figure 76. SMIT Show Command String Window

In the above example the command to configure the LU 6.2 Local LU profile is:

```
mksnaobj -t'local_lu' -u'lu6.2' -l 'LOCLU' -'LOCLUA' -a 'tok0'
LLUPRO
```

The parameters `-t'local_lu'` and `-u'lu6.2'` determine the profile we want to configure and can therefore be hard-coded in the script because they will not change.

In our example scenario we used the local LU name for the LU name field as well as for the LU alias field and the profile name itself, so - assuming that the local LU name is stored in the shell variable `LLUNAME` the final command would be:

```
mksnaobj -t'local_lu' -u'lu6.2' -l $LLUNAME -L $LLUNAME
-a $DEVICE $LLUNAME
```

Note

The above example also assumes that the Link Station Profile name is stored in the `DEVICE` shell variable.

The `mksnaobj` command is used to configure most of the SNA Server profiles. However, if the profile you want to create with `mksnaobj` already exists, the command will fail.

There is another command, `chsnaobj`, available to change existing profiles, where the parameters that need to be passed are exactly the same as for the corresponding `mksnaobj` command.

In our example configuration script we always try to configure a profile using `mksnaobj` first. If this command fails, we try the corresponding `chsnaobj` command, using the same parameters.

Besides configuring profiles we also need SNA Server commands to control the SNA Server product.

For example, we need to update the SNA Server configuration database after we have configured all profiles for the changes to become effective immediately.

To determine the necessary command we go back to the SNA Server Advanced Configuration menu and select **Verify Configuration Profiles**.

We use the Tab key to change the Update action field to dynamic_update and then press F6 to see the command:

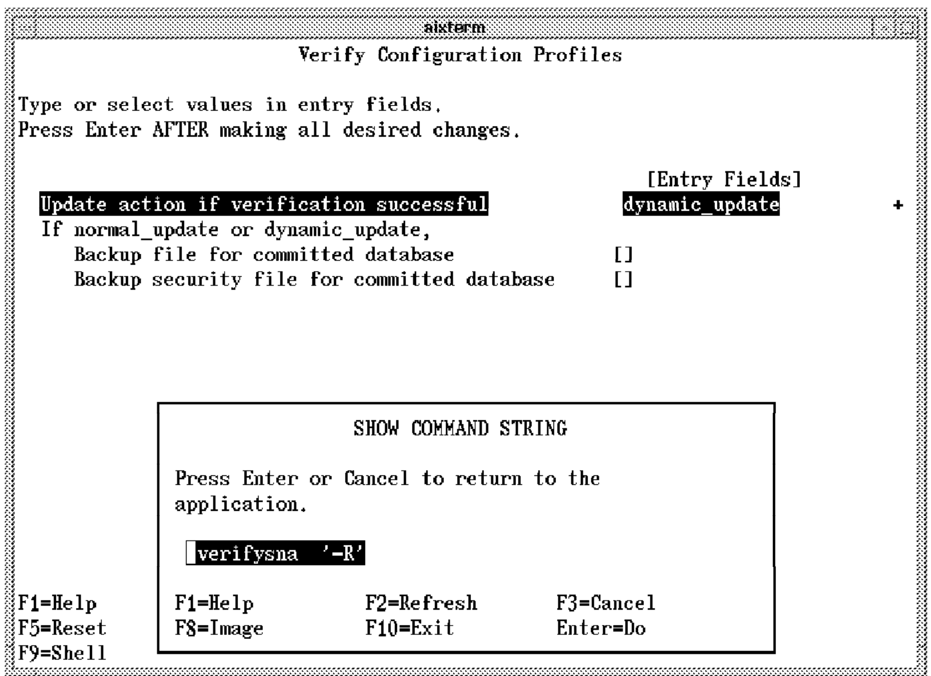


Figure 77. Verify Configuration Profiles Panel

As shown above, the command to dynamically update the SNA Server configuration is:

```
verifysna -R
```

7.2 Changing Configuration Files

Some configuration data for the products we need to configure is stored in flat ASCII files, therefore we have to modify these files to update the configuration.

Some of the files that we modified in our example were:

- The NetView DM/6000 base configuration file `nvdm.cfg`

- The SNA/DS connection files
- The SNA/DS routing table

- The /etc/services file

There are two ways to update configuration files:

1. Create a new configuration file
2. Update an existing configuration file

Creating a new configuration file is easier because you do not have to care about the current content of the file.

For example, when we change the SNA/DS routing table we always create a new file ignoring the current content.

A simple way to create a configuration file is to use the shell command echo and redirect its output to the file to be created, as in the following example:

```
echo "NETWORK PROTOCOL: TCP/IP

RS600015.*                  RS600015" >/usr/lpp/netviewdm/db/routetab
```

If we use shell variables instead of fixed strings, we can make the configuration more flexible:

```
echo "NETWORK PROTOCOL: TCP/IP

$SHORTNAME.*              $SHORTNAME" >$ROUTETAB
```

Note

The above example assumes the short name to be configured in the routing table is stored in the SHORTNAME shell variable and that the file name of the SNA/DS routing table is stored in the ROUTETAB shell variable.

In some cases we cannot ignore the current content of a file. Then we only want to change the sections of the file affected by the configuration update.

For example for setting an agents server we only need to change the entry for the SERVER field in the nvdm.cfg file keeping the rest of the file untouched.

For changing file contents we can use the UNIX stream editor sed. This tool allows us to find strings described by regular expression and replace them with new strings.

Note

If you are not familiar with sed or with pattern matching using regular expressions, you should consult the manpage for sed using the man sed command and the InfoExplorer sections dealing with regular expressions.

The following code fragment can be used to replace the entry for the SERVER field in the nvdm.cfg file:

```
CONFIG=/usr/lpp/netviewdm/db/nvdm.cfg
cp $CONFIG /tmp/config
sed "s/SERVER:.*/SERVER: rs60007" >$CONFIG
```

Using shell variables the code could look like the following:

```
CONFIG=/usr/lpp/netviewdm/db/nvdm.cfg
cp $CONFIG /tmp/config
sed "s/SERVER:.*/SERVER: $SERVER" >$CONFIG
```

The above example assumes that the name of the NetView DM/6000 server is stored in the SERVER shell variable.

7.3 Adjusting the Data Model

In this part we discuss how to change the data model used for NetView DM/6000 configuration.

A change in the data model will be needed if you want to significantly enhance the function of the existing configuration procedure.

If you just want to introduce a new global variable to the configuration procedure it is normally not necessary to change the data model. However, such changes normally require an update of the configuration script.

First, we discuss the case of introducing a new global variable.

7.3.1 Introducing New Global Variables

When designing our configuration data model we decided to store all parameters being unique in our entire software distribution in the nvdm_cfg_static class.

For example, we could want to introduce a new variable to store the path name for the NetView DM/6000 repository that should be used on all systems.

The new variable could simply be introduced by adding the following lines to the nvdm_cfg_static.odmadd file:

```
NAME  = "REPOS_DIR"
VALUE = "/usr/local/nvdm/repository"
```

After adding the lines the database can be updated by typing:

```
./build_db
```

Note

Of course you can also use `odme` to add the new variable by typing `odme nvdm_cfg_static`.

The variable can be retrieved from a shell script by using the ODM access methods developed before:

```
get_attribute nvdm_static NAME "REPOS_DIR" VALUE
REPOS=$VALUE
```

This will get the `VALUE` of the `REPOS_DIR` variable from the database and store it in the shell variable `REPOS`.

There are basically two reasons to introduce a new variable:

1. Making the script more flexible
2. Adding new functionality

If we just want to make the script more flexible, the only change in the script that is required is changing all occurrences where the parameter was hard-coded before with the variable.

In our example we have to change all lines where the name of the repository directory for NetView DM/6000 was hard-coded before.

In case of the repository directory this was not contained anywhere in the script before because we did not change the default entry for the `REPOSITORY` field in `nvdm.cfg`.

To add this we will just have to put in the following line somewhere in the main body of the script:

```
configure_nvdm_cfg "REPOSITORY" $REPOS
```

Note

The above example assumes that before the call is made the `REPOS` has been filled from the database.

If we did call `configure_nvdm_cfg` in the way shown above, this would cause the `sed` command within that procedure to fail, because the path name in the `REPOS` shell variable normally contains slashes (/) which are used as a delimiter in the `sed` command.

To avoid this, we precede every occurence of a slash in the class definition file `nvdm_cfg_static.odmadd` with a back-slash character.

The code to set the parameter then looks like:

```
get_attribute nvdm_cfg_static NAME REPOS_DIR VALUE
if [ "$VALUE" != "" ]
then
  VALUE=`echo "$VALUE"`
  configure_nvdm_cfg "REPOSITORY" $VALUE
fi
```

To add some functionality we could decide that we want to have the repository directory to be in an own file system.

To determine the command string for creating a file system we do the following:

1. Start SMIT by typing `smitty fs`.
2. Select **Add / Change / Show / Delete File Systems** and press Enter.
3. Select **Journaled File System** and press Enter.
4. Select **Add a Journaled File System** and press Enter.
5. Select **rootvg** and press Enter. This will get us to the Add a Journaled File System panel.
6. Enter `20000` in the SIZE of file system field.
7. Enter */test* in the MOUNTPOINT field.
8. Enter *yes* in the Mount AUTOMATICALLY at system restart field.
9. Press F6.

This will pop up the DISPLAY COMMAND STRING window of SMIT showing that the command to add a file system with the desired parameters is:

```
crfs -v jfs -g rootvg - a size=20000 -m /test -A yes -p rw -t no
```

Since we want the size of the file system to be variable we will store this in another ODM variable, by adding the following lines to the `nvdm_cfg_static.odmadd` file:

```
NAME  = "REPOS_SIZE"
VALUE = 50000
```

The code fragment to create the file system could look like the following:

```
REPOS=`grep "REPOSITORY" /usr/lpp/netviewdm/db/nvdm.cfg |
cut -d':' -f2`
get_attribute nvdm_cfg_static NAME REPOS_SIZE VALUE
if [ "$VALUE" = "" ]
then
  SIZE=20000
else
  SIZE=$VALUE
fi
crfs -v jfs -g rootvg -a size=$SIZE -m $REPOS
-A yes - p rw -t no
```

7.3.2 Changing the Data Model

We will now show an example in which we will change the data model. That means that we will change the ODM class definitions.

In the previous section we have introduced a new variable to contain the size of a file system to be created on every node for containing the repository directory of NetView DM/6000.

Since we stored this value in the class `nvdm_cfg_static` we could only use the same value for all nodes. However, it might be necessary to have different file system sizes on the different nodes. Also, some nodes might not need to have the repository directory to be put into an own file system.

In order to be able to have different file system sizes on every node and also to be able to decide whether the repository directory needs to be put into an own file system we introduce the following new attributes to the `nvdm_node` class:

- `repos_fs`: flag indicating if the repository directory has to be put into an own file system (either yes or no)
- `repos_size` : size in blocks of the file system to be created (only needed if `repos_fs` is set to yes)

In order to add these attributes to the ODM we have to change the class definition file:

```
#
# Create ODM class files for NVDM configuration DB
#

#
# the nvdm_groups class defines the target groups to be defined
# on a server
#

class nvdm_groups {
  char group_name[25];
  char description[25];
  char short_name[9];
  char node_name[25];
}

#
# the nvdm_node class describes the name (IP Hostname) and
# type (Server, Agent, Prep Site) of the node, where
# 0 : NVDM Server
# 1 : NVDM Agent
# 2 : NVDM Prep Site
# also included are attributes required for every node, like
# the name of the NVDM/6000 Server, etc.
#
# group_name is a link to the nvdm_groups class specifying
# the group this target belongs to

class nvdm_node {
  char  node_name[25];
  short node_type;
  char  short_name[9];
  char  target_os[12];
  char  description[25];
  char  contact_name[25];
  char  owning_manager[25];
  char  telephone_number[20];
  char  customer_name[20];
  char  repos_fs[4];
  long  repos_size;
  char  x_25_number[15];
  char  server_name[25];
  link  nvdm_groups nvdm_groups group_name group_name;
}

...
```

Figure 78. ODM Class Definition File config_db2.cre

Assuming that the class definition file is stored in `config_db2.cre` we can type the following command to create the ODM class files:

```
odmcreate -c config_db2
```

Now that the ODM class files have been changed we can also change our data definition file for the nvdm_node class.

We take the nvdm_node.odmadd file from our example scenario and add the attributes to reflect the following changes:

- The rs60007 server shall have the repository directory being in an own file system sized 100000 blocks.
- The rs60015 server shall have the repository directory being in an own file system sized 50000 blocks.
- The rs60004 shall not have the repository directory in an own file system.

The nvdm_node.odmadd file should then look like the following:

```
nvdm_node:
  node_name = "rs60007"
  node_type = 0
  short_name = "RS60007"
  target_os = "AIX"
  repos_fs = "yes"
  repos_size = 100000
  description = "ITSO Raleigh development"
  contact_name = "Stefan Uelpenich"
  owning_manager = "Wolfgang Geiger"
  telephone_number = "4711"
  customer_name = "IBM"
  x_25_number = ""
  server_name = "rs60007"
  group_name = "Group1"

nvdm_node:
  node_name = "rs600015"
  node_type = 0
  short_name = "RS600015"
  target_os = "AIX"
  repos_fs = "yes"
  repos_size = 50000
  description = "ITSO Raleigh test server"
  contact_name = "Stefan Uelpenich"
  owning_manager = "Wolfgang Geiger"
  telephone_number = "4711"
  customer_name = "IBM"
  x_25_number = ""
  server_name = "rs600015"
  group_name = "Group2"

nvdm_node:
  node_name = "rs60004"
  node_type = 1
  short_name = "RS60004"
  target_os = "AIX"
  repos_fs = "no"
  description = "ITSO Raleigh test client"
  contact_name = "Stefan Uelpenich"
  owning_manager = "Wolfgang Geiger"
  telephone_number = "4711"
  customer_name = "IBM"
  x_25_number = ""
  server_name = "rs600015"
  group_name = "Group1"
```

Figure 79. Data Definition File nvdm_node2.odmadd

Assuming that the data definitions are stored in `nvdm_node2.odmadd` we can add them by typing:

```
odmadd nvdm_node2.odmadd
```

Since we have cleared all the other ODM class files by calling the `odmcreate` command before, we also need to add the definitions for the other classes again:

```
odmadd nvdm_cfg_static.odmadd
odmadd nvdm_groups.odmadd
odmadd nvdm_queues.odmadd
odmadd nvdm_users.odmadd
odmadd nvdm_servers.odmadd
```

To add the file system we create another shell procedure within our configuration script:

```
#
# add file system for repository
# $1 = node name
#

add_fs_repos ()
{
 # get repository path
 REPOS=`grep "REPOSITORY" /usr/lpp/netviewdm/db/nvdm.cfg \
 | cut -d':' -f2`
 get_attribute nvdm_node node_name $1 repos_fs
 if [ "$VALUE" = "yes" ]
 then
   get_attribute nvdm_node node_name $1 repos_size
   if [ "$VALUE" = "" ]
   then
     SIZE=20000
   else
     SIZE=$VALUE
   fi
   print "NVDM CONFIG : Creating file system $REPOS."
   print "NVDM CONFIG : Size = $SIZE blocks."
   # first, save old files
   tar -cvf/tmp/save.tar $REPOS/.
   crfs -v jfs -g rootvg -a size=$SIZE -m $REPOS -A yes -p rw -t n
o
   mount $REPOS
   # restore files
   tar -xvf/tmp/save.tar $REPOS/.
 fi
}
```

Figure 80. add_fs_repos Shell Procedure

Explanation:
The script determines the path of the repository from the NetView DM/6000 `nvdm.cfg` file. Then the class `nvdm_node` is examined to detect whether this node needs to have its own file system for the repository.

If so, the size of that file system is determined from the `repos_size` attribute. If no size information is found, the size is set to a default value of `20000`.

Before creating the file system, the old data being in the repository path is saved to a `tar` archive. This is important, because as soon as the new file system is created and mounted over the path name of the repository, it will hide the old files.

After the file system has been created and mounted the shell procedure will restore the files from the `tar` archive to the newly created file system.

We run the modified configuration script again on `rs60007` by typing:

```
./config_nvdm rs60007
```

After the script has finished the new file system should have been added to the AIX operating system. We check this by typing `df` which produces the following output on `rs60007`:

```
Filesystem    Total KB    free %used   iused %iused Mounted on
/dev/hd4         16384    2904   82%    1137    27% /
/dev/hd9var      40960   20952   48%    1601    15% /var
/dev/hd2       1036288   35424   96%   37562    14% /usr
/dev/hd3         12288    2980   75%     173     4% /tmp
/dev/hd1          4096    3800    7%      44     4% /home
/dev/lv00        24576    3404   86%     195     3% /usr/lpp/netviewdm
/dev/lv01        53248   50260    5%      19     0% /usr/lpp/netviewdm/repos
```

Figure 81. Output from df Command

Note

In the above example the changes we made to the ODM definition were quite simple because we only added two new attributes to the nvdm_node class.

However, the changes become more difficult if you really change the data model, for example by introducing new classes or changing the relationship between classes.

Nevertheless, the procedure to implement these changes is the same as described above.

Chapter 8. Enhancing the Configuration Procedure

In this chapter we enhance the configuration procedure by introducing new features that were not included in the original procedure.

Unlike Chapter 7, “Customizing and Extending the Configuration Procedure” on page 109 where we give some general hints about how to customize and extend the procedure, we describe specific enhancements in detail in this chapter.

8.1 Configuring Intermediate Nodes

The project in which the configuration procedure described in this book was originally developed did not use intermediate nodes. Therefore the configuration of intermediate nodes was not included in the configuration procedure.

However, the intermediate node function of SNA/DS is very useful, so we will adapt the configuration procedure to support this feature.

Note

If you need a detailed description of the intermediate node concept you should consult the redbook *The NetView DM/6000 Cookbook*, GG24-4246.

Before we start to define configuration activities we will first have a look at an intermediate node scenario:

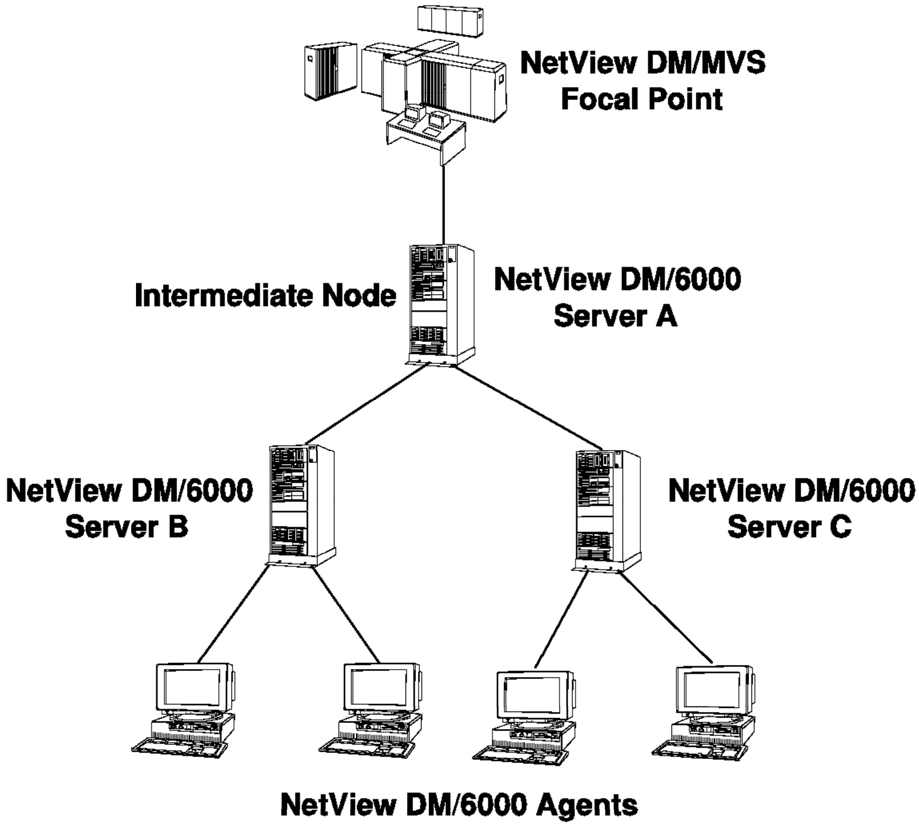


Figure 82. Intermediate Node Scenario

In the above example Server A acts as an intermediate node for connections between Servers B and C and the NetView DM/MVS focal point.

In order to enable the configuration procedure to configure a network containing intermediate nodes, the following configuration activities are affected:

- Configuration of SNA Server
- Configuration of the SNA/DS connection configuration files
- Configuration of the SNA/DS routing table

In our example configuration procedure we only configure SNA connections to a central NetView DM/MVS system. If we have SNA connections to, for example, other NetView DM/6000 servers we need to modify most of the configuration parts dealing with SNA.

We will not show how to do this in detail in this chapter. Nevertheless you should be aware of the fact, that this will also require significant modifications to the model.

In the data model we assume that SNA connections will only be needed to connect to one central MVS system. Therefore all parameters describing the SNA partner system are stored in the `nvdm_cfg_static` class. However, if we want to be able to configure SNA connections to any other system we would have to store this information in another class.

On a system that uses an intermediate node to connect to another system we only need a connection configuration file describing the SNA/DS connection to the intermediate node. The connections to other systems connected through the intermediate node are defined in the routing table.

On the intermediate node itself we need a connection configuration file for each system that is connected to the intermediate node using SNA/DS. The routing table on the intermediate node can be the same as the one used in the sample configuration procedure.

Since only NetView DM/6000 servers can have remote connections to other NetView DM servers, we only have to adjust the configuration parts dealing with the configuration of NetView DM/6000 servers.

8.1.1 Adjusting the Data Model

In our sample data model we have a class, `nvdm_queues` to define connections to remote destinations. Since remote connections in NetView DM/6000 are always based on SNA/DS, we use this class to configure SNA/DS connections, namely the connection configuration files and the routing table.

The `remote_server` attribute in this class determines the remote system to which we want to connect, where for an SNA connected system we use the partner LU name to determine the system and for TCP/IP connected systems we use the TCP/IP hostname.

In order to be able to configure connections using an intermediate node, we can add an attribute to this class that determines if an intermediate node is used to connect to the remote system. This attribute contains no value if we directly connect to the remote system, thus indicating that we do not want to use an intermediate node.

If we want to connect through an intermediate node, this attribute contains the short name of the intermediate node.

We will add the attribute `inter_node` to the `nvdm_queues` class to store the intermediate node information. In order to do so, we have to change the ODM class creation file to contain the new attribute. The following figure shows the affected part:

```
#
# the nvdm_queues class contains connections to
# remote servers
# e.g. a Focal Point or remote administrator
#
# Protocol must be "APPC" or "TCP/IP"
# if Protocol is TCP/IP the remote_server
# field must be filled with the IP hostname
# of the remote server
#
# This class will also be used to define
# The remote server as a remote target automatically
#

class nvdm_queues {
  link nvdm_node nvdm_node node_name node_name;
  char protocol[8];
  char remote_server[25];
  char focal_point[4];
  char inter_node[9];
}
```

Figure 83. Class Definition File

8.1.2 Adjusting the SNA/DS Connection Configuration Files

For the intermediate node itself, the creation of the SNA/DS connection configuration files can remain unchanged, assuming that all connections to remote systems are defined as instances of the `nvdm_queues` class. For the intermediate node the `inter_node` attribute contains no value.

In our example the objects in the `nvdm_queues` class for the intermediate node Server A could look as follows:

```
nvdm_queues:
     node_name     = "server_a"
     protocol      = "APPC"
     remote_server = "RA39TCF1"
     focal_point   = "yes"
     inter_node    = ""

nvdm_queues:
     node_name     = "server_a"
     protocol      = "TCP/IP"
     remote_server = "server_b"
     focal_point   = "no"
     inter_node    = ""

nvdm_queues:
     node_name     = "server_a"
     protocol      = "TCP/IP"
     remote_server = "server_c"
     focal_point   = "no"
     inter_node    = ""
```

Explanation:
The TCP/IP hostname of Server A is assumed to be `server_a`. The first entry describes the connection to the NetView DM/MVS focal point. Since we only configure SNA connections to MVS we only supply the LU name of NetView DM/MVS as the `remote_server` and determine the SNA network name from the `nvdm_cfg_static` object class.

The following entries define two TCP/IP connections to the other NetView DM/6000 servers, assuming that the TCP/IP hostnames of these servers are `server_b` and `server_c`.

For nodes connected through an intermediate node we have two types of connections:

- A connection to the intermediate node
- Connections to other systems using the intermediate node

For the connection to the intermediate node we have to create a connection configuration file whereas the connections to other systems through the intermediate node are defined in the SNA/DS routing table.

The objects in `nvdm_queues` for Server B could look as follows:

```
nvdm_queues:
     node_name     = "server_b"
     protocol      = "TCP/IP"
     remote_server = "server_a"
     focal_point   = "no"
     inter_node    = ""

nvdm_queues:
     node_name     = "server_b"
     protocol      = "APPC"
     remote_server = "RA39TCF1"
     focal_point   = "yes"
     inter_node    = "SERVERA"
```

Explanation:
The TCP/IP hostname of Server B is assumed to be `server_b`. We have one connection to the intermediate node using TCP/IP, assuming that the TCP/IP hostname of Server A is `server_a`. The connection to the focal point is made using Server A as an intermediate node indicated by the `inter_node` attribute set to the short name of Server A.

> **Note**
>
> You should notice that when we specify a remote TCP/IP server we use the TCP/IP hostname, whereas when we specify an intermediate node we use the short name. This is caused by the design of our data model.

The definitions for Server C look very similar, except that the hostname is `server_c`:

```
nvdm_queues:
     node_name     = "server_c"
     protocol      = "TCP/IP"
     remote_server = "server_a"
     focal_point   = "no"
     inter_node    = ""

nvdm_queues:
     node_name     = "server_c"
     protocol      = "APPC"
     remote_server = "RA39TCF1"
     focal_point   = "yes"
     inter_node    = "SERVERA"
```

In the configuration procedure that we have developed previously in this book, we use the shell procedure `configure_sna_ds_conn` to configure SNA/DS connection configuration files. This procedure calls either `configure_sna_ds_appc` to configure LU 6.2 connections or `configure_sna_ds_tcpip` to configure TCP/IP connections.

Since we do not need to configure a connection configuration file for connections using an intermediate node, we have to check the `inter_node` attribute in this procedure to determine if we have to create a connection configuration file.

The modified `configure_sna_ds_conn` procedure looks as follows:

```
#
# configure SNA/DS connection profiles
#
# $1 = IP Hostname of system to be configured
#

configure_sna_ds_conn ()
{

#
# perform SNA/DS configuration (connection profiles)
#

#
# remove demo profile CONNSNA,CONNTCP if existent
#
cd $SNA_DS_DIR
rm *

get_queues $1

if [ $NUM_QUEUE != 0 ]
then
  a=1
  for i in $PROTOCOL
  do
    print "NVDM CONFIG : Configuring $i connection"
    if [ "$i" != "APPC" -a  "$i" != "TCP/IP" ]
    then
      abort "Protocol is neither APPC nor TCP/IP. Exiting..."
    fi

    # determine if connection is made through an intermediate node

    INODE=`echo $REMOTE_SERVER | cut -d' ' -f"$a"`
    get_attribute_and nvdm_queues node_name $1 remote_server $INODE inter_node
    if [ "$VALUE" != "" ]
    then
      print "NVDM CONFIG : Remote connection to $INODE is made"
      print "              through intermediate node $VALUE."
      print "              No SNA/DS connection file is created."
    else

      if [ "$i" = "APPC" ]
      then
        configure_sna_ds_appc
      else
        REMSERV=`echo $REMOTE_SERVER | cut -d' ' -f "$a"`
        configure_sna_ds_tcpip $REMSERV
```

Figure 84 (Part 1 of 2). configure_sna_ds_conn Shell Procedure

```
      fi
    fi
    a=`expr $a + 1`
  done
fi
}
```

Figure 84 (Part 2 of 2). configure_sna_ds_conn Shell Procedure

Explanation:
We have replaced the remove (`rm`) statement to remove all files in the `/usr/lpp/netviewdm/dm/snads_conn` directory. In case a connection is changed to go through an intermediate node, this will erase the old connection configuration file for that connection.

Before we configure any connection configuration file, we check if the connection is made through an intermediate node. If so, no connection configuration file is created, since these connections are only defined in the routing table.

8.1.3 Adjusting the SNA/DS Routing Table

In the original configuration procedure we created a default entry for each remote connection where the connection field was set to the same name as the entry that was added.

In order to implement the intermediate node concept we have to create a routing table that sends all traffic through the intermediate node for connections where the `inter_node` field contains a value.

In the configuration procedure the shell procedure `configure_routetab` is used to create the SNA/DS routing table.

The modified `configure_routetab` procedure looks as follows:

```
#
# configure SNA/DS routing table
# $1 = IP Hostname
#

configure_routetab ()
{
 #
 # first, determine what network protocols we have
 #
 a=0
 b=0
 print "NVDM CONFIG : Configuring SNA/DS routing table."
 cd $SNA_DS_DIR
 HAVET=`grep PROTOCOL * | grep TCP/IP`
 if [ "$HAVET" != "" ]
 then
   print "NVDM CONFIG : System has TCP/IP connection to remote server."
   a=1
 fi

 HAVEA=`grep PROTOCOL * | grep APPC`
 if [ "$HAVEA" != "" ]
 then
   print "NVDM CONFIG : System has APPC connection to remote server."
   b=1
 fi

 if [ $a -eq 0 -a $b -eq 0 ]
 then
   print "NVDM CONFIG : There are no connections defined."
   return
 fi

 if [ $a -eq 1 -a $b -eq 1 ]
 then
   RPROT="BOTH"
 fi

 if [ $a -eq 1 -a $b -eq 0 ]
 then
   RPROT="TCP/IP"
 fi

 if [ $a -eq 0 -a $b -eq 1 ]
 then
   RPROT="APPC"
 fi
```

Figure 85 (Part 1 of 2). configure_routetab Shell Procedure

```
 print "NVDM CONFIG : Writing routing table to $SNA_DS_ROUTE"
 echo "NETWORK PROTOCOL: $RPROT

#
# SNA connections
#
" >$SNA_DS_ROUTE

#
# get all SNA Routes
#

 cd $SNA_DS_DIR
 SNA_R=`grep -p APPC * | grep "NEXT DSU" | cut -d':' -f2`
 if [ "$SNA_R" != "" ]
 then
   for i in $SNA_R
   do
     ONE=`echo $i | cut -d'.' -f1`
     TWO=`echo $i | cut -d'.' -f2`
     if [ "$TWO" = "*" ]
     then
       echo "$i    ANY  ANY  ANY  ANY  $ONE 5" >>$SNA_DS_ROUTE
     else
       echo "$i    ANY  ANY  ANY  ANY  $TWO 5" >>$SNA_DS_ROUTE
     fi
   done
 fi

 echo "
#
# TCP/IP connections
#
" >>$SNA_DS_ROUTE
 TCP_R=`grep -p TCP/IP * | grep "NEXT DSU" | cut -d':' -f2`
 if [ "$TCP_R" != "" ]
 then
   for i in $TCP_R
   do
     ONE=`echo $i | cut -d'.' -f1`
     echo "$ONE.*                    $ONE" >>$SNA_DS_ROUTE
   done
 fi
}
```

Figure 85 (Part 2 of 2). configure_routetab Shell Procedure

Explanation:
In the original version of configure_routetab we scanned the files in the `/usr/lpp/netviewdm/db/snads_conn` directory to gather information about the defined connections and then constructed the SNA/DS routing table from that information.

Since we do not create a connection configuration file for connections using an intermediate node we cannot use this approach anymore. Instead we use the information stored in the nvdm_queues class to retrieve information about the SNA/DS connections.

When we write a routing table entry we have to check for each connection if the connection is made using an intermediate node. If so, we have to specify the intermediate node in the CONNECTION field of the routing table. Otherwise we use the short name of the remote server.

In case we configure a TCP/IP connection we have to search the nvdm_node class for the short name of the remote_server, since the remote_server field in the nvdm_queues class contains only the TCP/IP hostname.

The SNA/DS routing tables generated for our example scenario will look as follows:

For Server A:

```
NETWORK PROTOCOL: BOTH

#
# SNA connections
#

USIBMRA.RA39TCF1   ANY   ANY   ANY   ANY   RA39TCF1     5

#
# TCP/IP connections
#

SERVERB.*                        SERVERB
SERVERC.*                        SERVERC
```

Figure 86. SNA/DS Routing Table (Server A)

The routing tables for Server B and Server C are identical:

```
NETWORK PROTOCOL: BOTH

#
# SNA connections
#

USIBMRA.RA39TCF1   ANY   ANY   ANY   ANY   SERVERA     5

#
# TCP/IP connections
#

SERVERA.*                        SERVERA
```

Figure 87. SNA/DS Routing Table (Server B and C)

8.2 Configuring NetView DM/MVS

In the configuration procedure developed in this book, we focused on the configuration of NetView DM/6000. In an environment where you have a NetView DM/MVS focal point it might also be desirable to configure NetView DM/MVS automatically.

We will not develop a complete procedure to fully configure NetView DM/MVS here. However, we will show an example of how to create a configuration procedure for NetView DM/MVS.

In the example we write a shell script that will produce the commands to configure the nodes attached to the NetView DM/MVS system automatically. For that purpose we will use the same ODM database that we also used to configure NetView DM/6000.

The procedure that we develop will produce an ASCII file containing the necessary MVS commands which can then be transferred to the MVS host.

The following figure shows the procedure:

```
#!/bin/ksh
#
# Generate Statements to configure
# NetView DM/MVS
#
# This script uses the ODM class nvdm_node and nvdm_servers
# to create automatic node definitions
#
# Author : Stefan Uelpenich / IBM Germany
# $Revision: 1.11 $
#
#

NODE_CLASS=nvdm_node
SERVER_CLASS=nvdm_servers

#
# variable field for job card creation
# they may also be put into the ODM,
# e.g. into nvdm_cfg_static
#

USERID=A47112
ACCOUNT="ACCT"
NAME=DSX
SIZE=6000K
CLASS=A
TIME=1440
LOADLIB="NDM.R5.NDMLOAD"
PW=DUMMY
FN="NETVIEW.R5"


#
#
# DATABASE ACCESS METHODS (ODM)
# these access methods may be replaced with
# access methods for any other database at
# a later time
#
#

#
# get list parameters from odm_class
# $1 = class name
# $2 = search field
# $3 = search field value
# $4 = attribute name
# The list of parameters is stored in the VALUE_LIST variable
```

Figure 88 (Part 1 of 5). Sample Procedure to Configure NetView DM/MVS

```
# The number of parameters is stored in VALUE_NUM
#

get_attribute_list ()
{
 VALUE_LIST=`odmget -q $2=$3 $1 | grep "$4 =" | cut -d'=' -f2 |\
 sed "s/\"//g" | cut -c2-79`
 VALUE_NUM=`odmget -q $2=$3 $1 | grep "$1:" | wc -l`
}

#
# get single parameters
# $1 = class name
# $2 = search field
# $3 = search field value
# $4 = attribute name
#

get_attribute ()
{
 VALUE=`odmget -q $2=$3 $1 | grep "$4 =" | cut -d '=' -f2 | sed "s/\"//g" |\
 cut -c2-79`
}

#
# get single parameters (AND)
# $1 = class name
# $2 = search field1
# $3 = search field value1
# $4 = search field2
# $5 = search field value2
# $6 = attribute name
#

get_attribute_and ()
{
 VALUE=`odmget -q "$2=$3 AND $4=$5" $1 | grep "$6 =" | cut -d '=' -f2 |\
 sed "s/\"//g" | cut -c2-79`
}


#
#
#  create job cards...
#
#


generate_servers ()
{
```

Figure 88 (Part 2 of 5). Sample Procedure to Configure NetView DM/MVS

```
 #
 # get all nodes that are defined as NVDM servers
 #
 get_attribute_list $NODE_CLASS node_type 0 node_name
 print "Number of servers: $VALUE_NUM"
 #
 # create job card header
 #
 echo "
//${USERID}A JOB
(${ACCOUNT}),${NAME},REGION=${SIZE},CLASS=${CLASS},TIME=${TIME},
//           MSGCLASS=9,PRTY=14,NOTIFY=${USERID}
//**
//JOBLIB  DD DSN=${LOADLIB},DISP=SHR
//**
//**
//GDSX101 EXEC PGM=DSXPREP,
//            PARM='FUNCTION=SUBMIT,USERID=${USERID},PASSWORD=${PW}'
//DSXPRINT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SNAP     DD  SYSOUT=*
//SYSUDUMP DD  DUMMY
//DSXDRD   DD  DSN=${FN}.DRD,DISP=SHR
//DSXLIB   DD  DSN=${FN}.LIB,DISP=SHR
//DSXLIBT  DD  DSN=${FN}.LIBT,DISP=SHR
//DSXTCF   DD  DSN=${FN}.TCF,DISP=SHR
//DSXHFDI  DD  DSN=${FN}.HFDI,DISP=SHR
//DSXHFDA  DD  DSN=${FN}.HFDA,DISP=SHR
//DSXGIX   DD  DSN=${FN}.GIX,DISP=SHR
//DSXGIXD  DD  DSN=${FN}.GIXD,DISP=SHR
//NDMRQF   DD  DSN=${FN}.RQF,DISP=SHR
//NDMRQFDA DD  DSN=${FN}.RQFDA,DISP=SHR
//SYSIN    DD  *
" > $outfile
 #
 # create an entry for each node to be created
 #
 for i in $VALUE_LIST
 do
    #
    # get necessary attributes from database
    #
    get_attribute $NODE_CLASS node_name $i short_name
    SHORT=$VALUE
    get_attribute $SERVER_CLASS node_name $i local_lu_name
    LUNAME=$VALUE
    get_attribute $NODE_CLASS node_name $i description
    DESC=$VALUE

printf "%-70s *\n" "$SHORT DEF NODE  NAME=$SHORT," >>$outfile
printf "%-70s *\n" "             NODETYPE=NDM6," >>$outfile
```

Figure 88 (Part 3 of 5). Sample Procedure to Configure NetView DM/MVS

```
printf "%-70s *\n" "             LUNAME=$LUNAME," >>$outfile
printf "%-70s *\n" "             LOGMOD=NVDMNORM,"  >>$outfile
printf "%-70s *\n" "             RGN=$SHORT," >>$outfile
printf "%-70s *\n" "             REN=$SHORT," >>$outfile
printf "%-70s *\n" "             LINETYPE=L," >>$outfile
printf "%-70s *\n" "             STATUS=P," >>$outfile
printf "%-70s *\n" "             CLASS=A0," >>$outfile
printf "%-70s *\n" "             SRVNAME=$SHORT," >>$outfile
printf "%-70s *\n" "             TIMZOFFS=0," >>$outfile
printf "%-70s\n" "             NOTE='$DESC'" >>$outfile

 done
}

generate_agents ()
{
 get_attribute_list $NODE_CLASS node_type 1 node_name
 print "Number of agents: $VALUE_NUM"
 for i in $VALUE_LIST
 do
    #
    # get necessary attributes from database
    #
    get_attribute $NODE_CLASS node_name $i short_name
    SHORT=$VALUE
    get_attribute $NODE_CLASS node_name $i server_name
    SVR=$VALUE
    get_attribute $SERVER_CLASS node_name $SVR local_lu_name
    LUNAME=$VALUE
    get_attribute $NODE_CLASS node_name $SVR short_name
    SVRSHORT=$VALUE
    get_attribute $NODE_CLASS node_name $i description
    DESC=$VALUE

printf "%-70s *\n" "$SHORT DEF NODE  NAME=$SHORT," >>$outfile
printf "%-70s *\n" "             NODETYPE=NDM6," >>$outfile
printf "%-70s *\n" "             LUNAME=$LUNAME," >>$outfile
printf "%-70s *\n" "             LOGMOD=NVDMNORM,"  >>$outfile
printf "%-70s *\n" "             RGN=$SVRSHORT," >>$outfile
printf "%-70s *\n" "             REN=$SHORT," >>$outfile
printf "%-70s *\n" "             LINETYPE=L," >>$outfile
printf "%-70s *\n" "             STATUS=P," >>$outfile
printf "%-70s *\n" "             CLASS=A0," >>$outfile
printf "%-70s *\n" "             SRVNAME=$SVRSHORT," >>$outfile
printf "%-70s *\n" "             TIMZOFFS=0," >>$outfile
printf "%-70s\n" "             NOTE='$DESC'" >>$outfile

 done
}
```

Figure 88 (Part 4 of 5). Sample Procedure to Configure NetView DM/MVS

```
#
# MAIN
#

print "NetView DM/MVS configuration generator."
print "Name of output file:"
read outfile

generate_servers
generate_agents
```

Figure 88 (Part 5 of 5). Sample Procedure to Configure NetView DM/MVS

Explanation:
Assuming that the shell script is stored in a file named `nvdm_mvs` we can invoke it by typing:

```
nvdm_mvs
```

The script will then ask for a file name of the file where the output will be placed. This file will contain a job card for MVS that can be used to define the nodes of our software distribution network to NetView DM/MVS.

Values that depend on the specific MVS environment are held in shell variables being set at the beginning of the script. For example, the shell variable `USERID` contains the MVS user ID.

Another way to store these values would be to put them in the ODM, preferably in the `nvdm_cfg_static` class.

The script contains two procedures, `configure_servers` and `configure_agents`. The `configure_servers` procedure is used to create the job card header and an entry for each server defined in the software distribution network. Information about servers is gathered from the `nvdm_node` and `nvdm_servers` ODM classes.

The `configure_agents` procedure is used to create an entry for each agent defined in the software distribution network. Since NetView DM/MVS is not connected to NetView DM/6000 agents directly but through a NetView DM/6000 server, we have to find out the appropriate server for each agent and then determine the short name and the LU name of the server.

These values are then used when defining the agent as a node to NetView DM/MVS.

We use the following example node definition to demonstrate the script:

```
nvdm_node:
   node_name = "nw13nvdm105"
   node_type = 0
   short_name = "NW13NVDM"
   target_os = "AIX"
   description = "NetView DM Server 1"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "1234"
   customer_name = "ITSO Raleigh"
   repos_fs = ""
   repos_size = 0
   x_25_number = ""
   server_name = "nw13nvdm105"
   group_name = "group1"

nvdm_node:
   node_name = "nw12adsm105"
   node_type = 1
   short_name = "NW12ADSM"
   target_os = "AIX"
   description = "Johnbergs ADSM Server"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "5678"
   customer_name = "ITSO Raleigh"
   repos_fs = ""
   repos_size = 0
   x_25_number = ""
   server_name = "nw13nvdm105"
   group_name = "group1"

nvdm_node:
   node_name = "nw18nvdm105"
   node_type = 0
   short_name = "NW18NVDM"
   target_os = "AIX"
   description = "NetView DM Server 2"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "1234"
   customer_name = "ITSO Raleigh"
   repos_fs = ""
   repos_size = 0
   x_25_number = ""
   server_name = "nw18nvdm105"
   group_name = "group2"
```

Figure 89 (Part 1 of 2). Sample Definition for nvdm_node Class

```
nvdm_node:
   node_name = "nw38r40p137"
   node_type = 1
   short_name = "NW3840P"
   target_os = "AIX"
   description = "Sample Client for AIX4"
   contact_name = "Stefan Uelpenich"
   owning_manager = "Wolfgang Geiger"
   telephone_number = "5678"
   customer_name = "ITSO Raleigh"
   repos_fs = ""
   repos_size = 0
   x_25_number = ""
   server_name = "nw13nvdm105"
   group_name = "group1"
```

Figure 89 (Part 2 of 2). Sample Definition for nvdm_node Class

Further, we use the following SNA definitions in the `nvdm_servers` class:

```
nvdm_servers:
   node_name = "nw13nvdm105"
   local_lu_name = "LUNDM13"
   pu_name = "PUNDM13"
   cp_name = "CPNDM13"
   xid = ""
   sna = "yes"

nvdm_servers:
   node_name = "nw18nvdm105"
   local_lu_name = "LUNDM18"
   pu_name = "PUNDM18"
   cp_name = "CPNDM18"
   xid = ""
   sna = "yes"
```

Figure 90. Sample Definition for nvdm_servers Class

Assuming that we have added the above definitions to the ODM, for example by using the `odmadd` command, we can start the script by typing:

```
nvdm_mvs
```

The output file for the above definitions will look as follows:

```
//A47112A JOB (ACCT),DSX,REGION=6000K,CLASS=A,TIME=1440,
//           MSGCLASS=9,PRTY=14,NOTIFY=A47112
//**
//JOBLIB  DD DSN=NDM.R5.NDMLOAD,DISP=SHR
//**
//**
//GDSX101 EXEC PGM=DSXPREP,
//            PARM='FUNCTION=SUBMIT,USERID=A47112,PASSWORD=DUMMY'
//DSXPRINT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SNAP     DD  SYSOUT=*
//SYSUDUMP DD  DUMMY
//DSXDRD   DD  DSN=NETVIEW.R5.DRD,DISP=SHR
//DSXLIB   DD  DSN=NETVIEW.R5.LIB,DISP=SHR
//DSXLIBT  DD  DSN=NETVIEW.R5.LIBT,DISP=SHR
//DSXTCF   DD  DSN=NETVIEW.R5.TCF,DISP=SHR
//DSXHFDI  DD  DSN=NETVIEW.R5.HFDI,DISP=SHR
//DSXHFDA  DD  DSN=NETVIEW.R5.HFDA,DISP=SHR
//DSXGIX   DD  DSN=NETVIEW.R5.GIX,DISP=SHR
//DSXGIXD  DD  DSN=NETVIEW.R5.GIXD,DISP=SHR
//NDMRQF   DD  DSN=NETVIEW.R5.RQF,DISP=SHR
//NDMRQFDA DD  DSN=NETVIEW.R5.RQFDA,DISP=SHR
//SYSIN    DD  *

NW13NVDM DEF NODE  NAME=NW13NVDM,                                          *
             NODETYPE=NDM6,                                                *
             LUNAME=LUNDM13,                                               *
             LOGMOD=NVDMNORM,                                              *
             RGN=NW13NVDM,                                                 *
             REN=NW13NVDM,                                                 *
             LINETYPE=L,                                                   *
             STATUS=P,                                                     *
             CLASS=A0,                                                     *
             SRVNAME=NW13NVDM,                                             *
             TIMZOFFS=0,                                                   *
             NOTE='NetView DM Server 1'
NW18NVDM DEF NODE  NAME=NW18NVDM,                                          *
             NODETYPE=NDM6,                                                *
             LUNAME=LUNDM18,                                               *
             LOGMOD=NVDMNORM,                                              *
             RGN=NW18NVDM,                                                 *
             REN=NW18NVDM,                                                 *
             LINETYPE=L,                                                   *
             STATUS=P,                                                     *
             CLASS=A0,                                                     *
             SRVNAME=NW18NVDM,                                             *
             TIMZOFFS=0,                                                   *
             NOTE='NetView DM Server 2'
```

Figure 91 (Part 1 of 2). Output File Created by nvdm_mvs Script

```
NW12ADSM DEF NODE  NAME=NW12ADSM,                                          *
             NODETYPE=NDM6,                                                *
             LUNAME=LUNDM13,                                               *
             LOGMOD=NVDMNORM,                                              *
             RGN=NW13NVDM,                                                 *
             REN=NW12ADSM,                                                 *
             LINETYPE=L,                                                   *
             STATUS=P,                                                     *
             CLASS=A0,                                                     *
             SRVNAME=NW13NVDM,                                             *
             TIMZOFFS=0,                                                   *
             NOTE='Johnbergs ADSM Server'
NW3840P DEF NODE  NAME=NW3840P,                                            *
             NODETYPE=NDM6,                                                *
             LUNAME=LUNDM13,                                               *
             LOGMOD=NVDMNORM,                                              *
             RGN=NW13NVDM,                                                 *
             REN=NW3840P,                                                  *
             LINETYPE=L,                                                   *
             STATUS=P,                                                     *
             CLASS=A0,                                                     *
             SRVNAME=NW13NVDM,                                             *
             TIMZOFFS=0,                                                   *
             NOTE='Sample Client for AIX4'
```

Figure 91 (Part 2 of 2). Output File Created by nvdm_mvs Script

8.3 Configuring NetView DM TCP/IP Ports

For communicating over TCP/IP NetView DM/6000 uses certain TCP/IP ports which need to be defined in the `/etc/services` file. This is normally done by editing this file, for example, by using the `vi` editor.

The following shell procedure can be used to add the ports needed by NetView DM/6000 automatically:

```
#
# check if TCP/IP ports for NetView DM/6000 are
# existing. If not, add them to /etc/services file
#
check_ports ()
{
 #
 # first, make a backup copy of /etc/services..."
 #
 cp /etc/services /etc/services.nvdm
 #
 # check for port NetViewDM-rcv
 #
 print "CONFIG NVDM : Checking NetViewDM-rcv port..."
 R=`grep NetViewDM-rcv /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM-rcv   731/tcp" >>/etc/services
 fi
 #
 # check for port NetViewDM-snd
 #
 print "CONFIG NVDM : Checking NetViewDM-snd port..."
 R=`grep NetViewDM-snd /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM-snd   730/tcp" >>/etc/services
 fi
 #
 # check for port NetViewDM6000
 #
 print "CONFIG NVDM : Checking NetViewDM6000 port..."
 R=`grep NetViewDM6000 /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/service
   echo "NetViewDM6000   729/tcp" >>/etc/services
 fi
}
```

Figure 92. check_ports Shell Procedure

Explanation:
The shell procedure checks for the following ports needed by NetView DM/6000:

- NetViewDM6000
- NetViewDM-snd
- NetViewDM-rcv

In case one of the ports is not defined in the `/etc/services` file, the procedure will add the ports, using the default TCP port number, for example, port `730` for `NetViewDM-snd`.

8.4 Configuring the root.cli File

When a NetView DM/6000 server or agent has been used with a different hostname before, there might occur a problem, because the product keeps the hostname in several configuration files.

The product configuration files, like `nvdm.cfg` are not a problem, because they are reconfigured when running the configuration procedure.

There is, however, a file called `/usr/lpp/netviewdm/uicfg/username/uicfg/username` for each user that has used the workstation before. This is a binary file which also contains the hostname of the workstation.

When you try to reconfigure NetView DM/6000 with a different hostname now, starting the product will fail, since the file still contains the old hostname.

To solve this problem we need to modify the hostname in the file and adjust it to the hostname currently used for that workstation.

The following C program can be used to perform this task:

```
/* create uicfg/xxx.cli file
   Author : Stefan Uelpenich/IBM Germany
*/

#include <stdio.h>
#include <string.h>
#include <fcntl.h>

static char cfgfile[]={ 0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0x04 };


main(argc,argv)
int argc;
char *argv[];
{
 FILE *hnd;
 if (argc != 2) {
   printf ("Syntax : %s hostname\n",argv[0]);
   exit (0);
 }
 printf("Create /usr/lpp/netviewdm/uicfg/root.cfg file...\n");
 hnd=fopen("/usr/lpp/netviewdm/uicfg/root.cli","wb+");
 strcpy(cfgfile,argv[1]);
 fwrite(cfgfile,1,65,hnd);
 fclose(hnd);
}
```

Figure 93. uicfg.c Program

Assuming that the source code is stored in a file `uicfg.c`, you can compile it by typing:

```
cc -o uicfg uicfg.c
```

The compiled program can then be invoked using the hostname to be configured as the command line argument, for example:

```
uicfg rs600012
```

This program should be included in the configuration script to make sure, that the product will start successfully, even if the hostname has changed and NetView DM/6000 has been used before on that workstation.

Note

The above program only adjusts the file for the `root` user. If you want to make the program more general you could add a parameter determining the user name and therefore adjust the files for all users.

Chapter 9. Configuring a Production Environment

We now develop some general concepts for applying the configuration procedure that we have created before to a real production environment.

That means that we show the procedures to configure a large number of RS/6000 systems from a central point of control.

We will concentrate on the following topics:

- Customizing the configuration procedure
- Testing the configuration procedure in a test environment
- Generate configuration data for the production environment
- Defining a roll-out strategy

9.1 Customizing the Configuration Procedure

As stated before the configuration procedure documented in this book shall be the base for the configuration procedure you will need in your specific environment. Therefore, it is most likely that it will need at least some sort of customization.

The customization might include:

- Selecting the configuration steps you need to perform
- Adding new functions to the configuration script
- Changing the data model

We have shown procedures to perform all of the above steps previously, so you should be able to create your own configuration procedure that specifically meets the requirements of your environment.

As soon as you think that customization is finished, you can test the procedure in a test environment.

9.2 Testing the Configuration Procedure

In order to test your procedure you should set up a test environment that is a model of your real target environment.

In the scenario used for developing our example configuration procedure we had, for example, two NetView DM/6000 servers, one NetView DM/6000 agent and one NetView DM/MVS focal point.

This could be the model for a target environment where you have some NetView DM/6000 servers connected to a NetView DM/MVS focal point and lots of agents connected to each server.

Some points you should consider when setting up a test environment include:

- What kind of node types do I have on my network?

- What kind of network protocols do I intend to use?
- What kind of networks will I have in my target environment?
- What is the number of systems on my network?
- Do I have remote administrators?
- Do I have a focal point system?
- What kind and level of operating systems do I have?

Your test environment should be as close to the real production environment as possible. The following list might give you some hints about what can be important:

- If you have different versions of the operating system in your target network you should have at least one machine with either of these levels in your test environment. For example, when you know that there will be agents on your network running AIX 4.1, you should have one agent in your test environment running AIX 4.1.
- If you have different machine types on your network, you should have one machine of each type in your test environment. For example, the target machines may differ in disk space, memory size or processor speed.
- If you have different network types in your target network you should at least test your procedure once using each network type. For example, if your test environment runs on a token-ring network and you know that your target network will be X.25, you can develop your procedure in the token-ring network. As soon as you are finished you should then test the procedure in an X.25 network before starting the roll-out.
- If you intend to use network protocols other than TCP/IP, you should carefully test the setup of these protocols. Whereas the configuration of TCP/IP is comparably easy, the configuration of SNA LU 6.2 is normally a lot more complicated.
- The number of systems on your network should be considered, for example, to get an idea of how to organize target groups, how to structure server levels, etc.

9.3 Generating Configuration Data for the Target Environment

In our example scenario we defined all configuration data manually. This worked fine in the test environment because we had only three systems to be configured.

However, if we have some hundreds, or even thousands of systems on our network, it would be very time consuming and error-prone to type in all configuration data manually.

We now show some techniques to generate configuration data automatically.

The goal should be to keep the information that has to be entered manually as minimal as possible.

One way to achieve this is to find out the core data describing your software distribution network. All configuration data needed to configure the software distribution network must either be contained in this core data or deductible from that data.

Note

What data is core data and what data can be deducted depends strongly on your specific environment and your specific requirements.

Deducing data from core data always reduces flexibility, which is not always desirable.

9.3.1 Creating ODM Definitions Automatically

In this section we show an example of how you can set up rules in your organization that allows you to generate node-specific information automatically. If this does not apply to your situation, skip this section.

Note

We assume that we have to create ODM definitions in this example since we used the ODM to store the configuration database in all the previously described examples.

However, if we used another database, we would have to create data definition files for that database.

For example, we replaced the ODM with DB2/6000 in Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219. In order to fill the configuration database stored in DB2/6000 automatically, we can either create SQL insert statements automatically with the procedures described below or use the migration tool described in 14.1, “ODM to DB2/6000 Conversion” on page 277 to migrate an existing ODM configuration database to DB2/6000 or use the graphical user interface presented in Chapter 15, “Modifying Configuration Data Using a Graphical User Interface” on page 293.

If we have a core data set containing the essential information about our software distribution network we can construct at least parts of the ODM data definition files automatically.

For that purpose we need a rule for every ODM attribute to be created describing how this attribute can be constructed from the core data.

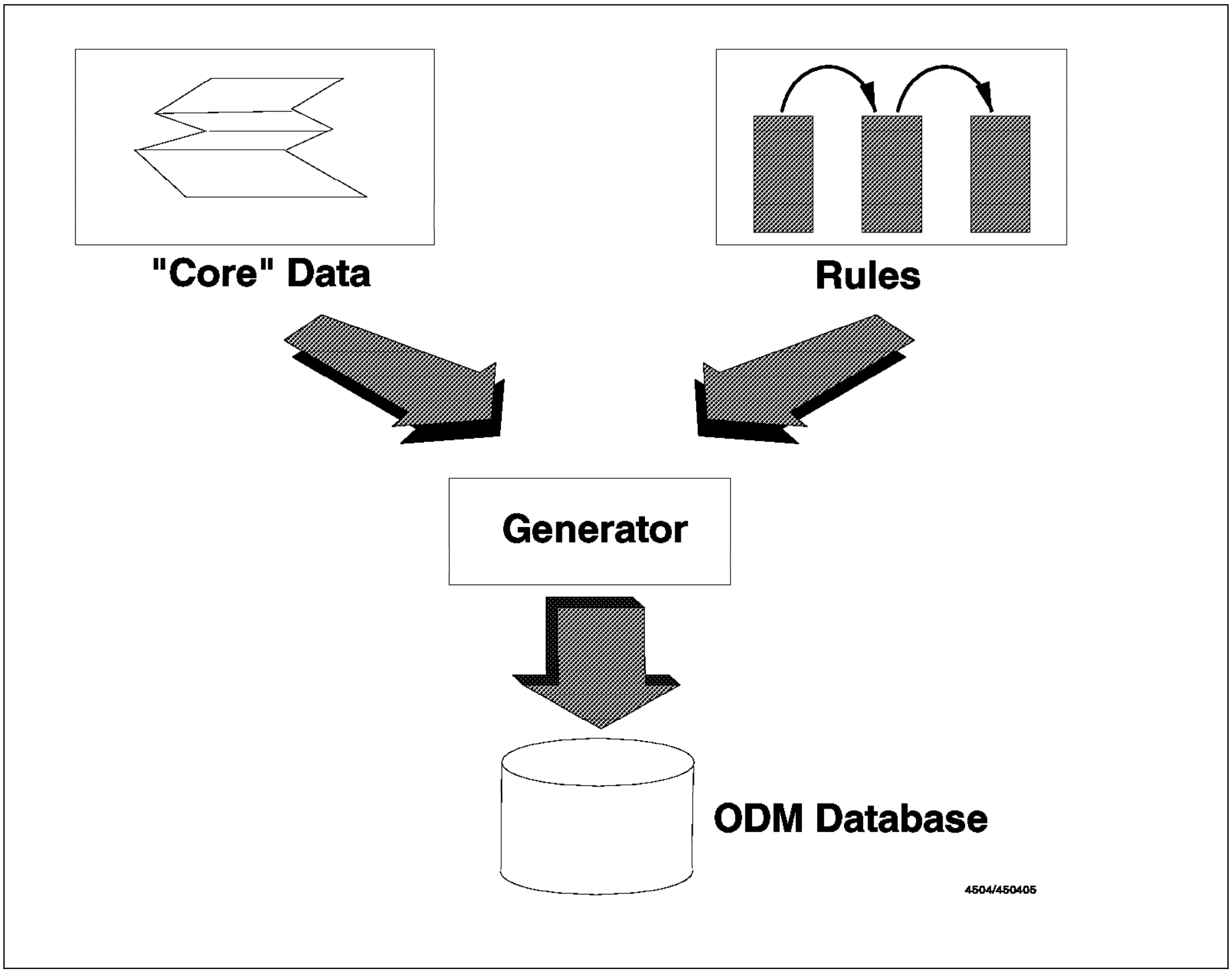


Figure 94. Generating ODM Classes Automatically

Let's start with a very simple example of core data:

We have an organization table, stored in an AIX file, containing the branch office structure of a company. In the first approach the table looks like the following:

```
h0001     h0001
h0001     b0007
h0001     b0011
h0002     h0002
h0002     b0001
h0002     b0003
h0003     h0003
h0003     b0008
h0003     b0009

...
```

Figure 95. Organization Structure File

Explanation:

Within the organization there are head offices and branch offices. The head offices are connected to the central headquarters, whereas each branch office is connected to a head office.

The above table contains the names of the head offices and branch offices and their relationships. Each branch office has a head office to which it is connected. A head office is always connected to itself, because it also acts as a branch office.

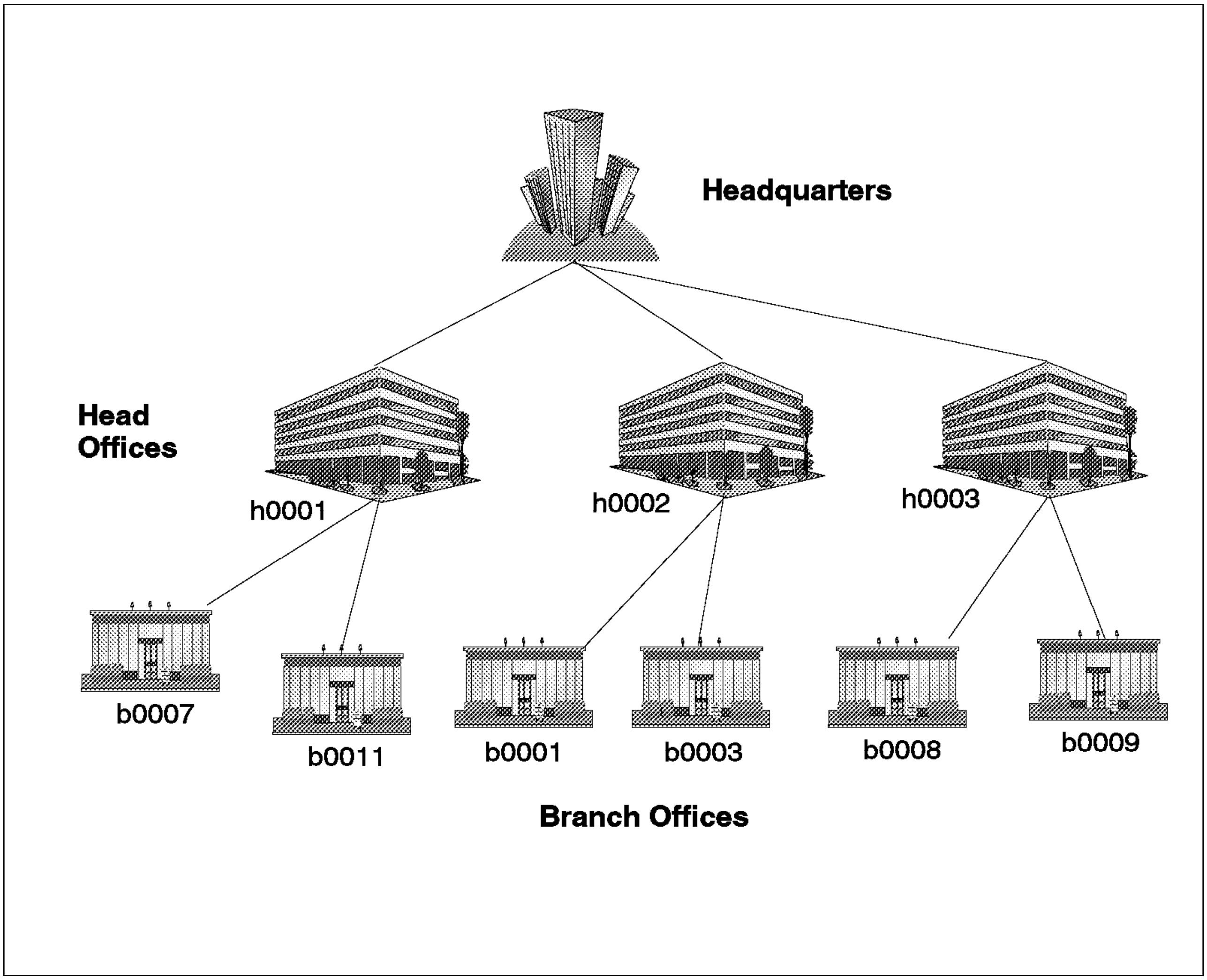


Figure 96. Example Corporate Structure

The names of head offices always start with an h whereas the names of branch offices always start with a b.

In this example we want to configure a software distribution network in the following way:

- All head offices act as NetView DM/6000 servers.
- All branch offices act as NetView DM/6000 agents.
- All NetView DM/6000 servers are connected to a central NetView DM/MVS system located at the headquarters using SNA.
- All targets connected to a head office form a target group.

- All targets will only have one user root which will have NetView DM/6000 administrator authority.

We now want to fill the ODM object classes with the configuration data for this network automatically. We use the object classes designed in 3.2, “Defining Object Classes” on page 12.

In order to do so we need rules to create the attributes for the objects we need to describe our network.

Note

The rules we use in this example are very simple, since we only want to show the basic principles. It is most likely that in a real environment the rules needed to create configuration data are more complex.

The following list contains the rules for our environment:

1. `nvdm_node` object class:
   - The `node_name` (IP hostname) is the same as the head office or branch office name.
   - The `node_type` can be derived from the office name. Since head offices are NetView DM/6000 servers, all offices having a name starting with h will have `node_type` 0, whereas all branch offices are agents and have `node_type` 1.
   - The `short_name` will be the office name converted into uppercase.
   - The `server_name` can be gathered directly from the table. The first column contains the head office which is also the corresponding NetView DM/6000 server. For a head office this field points to itself.
   - The `group_name` can be the name of the server since all nodes connected to a server also form a target group.
   - The `target_os` field will be set to AIX because all nodes we want to configure run the AIX operating system.
   - The optional fields, like `description`, `contact_name`, etc. can be left blank or filled with any useful values. For example, the core data table could also contain the name of the city where each office is located. Then the description field could be filled with that name.
2. `nvdm_groups` object class
   - The `node_name` is the same IP hostname as used in the `nvdm_node` object class.
   - The `group_name` is the name of the group to be created. Since all targets connected to one server form a target group we have to create one instance of this class for every server in our network. The `group_name` field contains the same value as the corresponding `group_name` field in the `nvdm_node` class.
   - We use the `group_name` converted to uppercase as the group `short_name`.
   - The `description` field is optional.
3. `nvdm_users` object class

- The node_name is the same IP hostname as used in the nvdm_node object class.
- The username is the AIX user that has to be configured as a NetView DM/6000 target user. Since we want to configure only the root user at each target we have to create one instance of this class for every node.
- The usergroup field will always be set to FNDADMN because we only define the root user as NetView DM/6000 administrator on each target.

4. nvdm_servers object class
   - The node_name is the same IP hostname as used in the nvdm_node object class.
   - In our example we assume that we have a very simple naming scheme for SNA parameters. For example, LU 6.2 names always start with LU followed by the office name converted to uppercase, like LUH0001. This value will be stored in local_lu_name. We need to create one instance of this class for each NetView DM/6000 server, because only the servers in the head offices are connected to the focal point using SNA.
   - The pu_name will always be PU followed by the office name, like PUH0001.
   - The cp_name will always be CP followed by the office name, like CP0001.
   - The xid field will be left blank, because we will use the control point XID.
   - The sna field will always be set to yes because we want SNA Server configuration to be performed on each node.
5. nvdm_queues object class
   - The node_name is the same IP hostname as used in the nvdm_node object class.
   - The protocol is always set to APPC since we only have a remote connection to the MVS focal point using APPC. One instance of this class has to be created for each NetView DM/6000 server.
   - The remote_server field always contains the LU 6.2 name of the focal point system, for example LUHQNVDM.
   - The focal_point field is always set to yes since we have only remote connections to the focal point system.

The above rules could now easily be translated into a simple Shell script that uses the core data set to create the ODM definition files automatically.

Anytime the core table is changed we only need to run the Shell script again to update the ODM database. For example, if a branch office is moved to another head office the change has to be performed only in the core data table.

Note

Similar considerations about generating data for the configuration database apply to the DB2/6000 scenario in Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219.

9.4 Defining a Roll-Out Strategy

The roll-out strategy is basically the process that is used to install and configure the workstations in the network.

We assume that we have adapted the configuration procedure for our specific environment and that we have also filled the configuration database with the data representing our software distribution network.

We now have to design the procedure to completely install and configure any system that is part of our network.

Depending on what software has to be installed on each system this procedure will be different for every specific environment. However, there are some general concepts that can be applied to any kind of software:

- The software has to be installed on the system.
- Software that needs configuration should be automatically configured.

All software that is needed on a system has to be installed on that system where there are several ways to do that. We will discuss the possibilities in 9.4.1, "Installing Software."

Software that needs configuration should be configured automatically on each system. This is especially important for networks containing a large number of systems.

In this book we have developed a configuration procedure to configure NetView DM/6000 automatically as well as some other components, such as SNA Server. You might want to implement your own configuration procedures for other products, too.

Here are some examples of products that could be configured by a configuration procedure:

- Adstar Distributed Storage Manager/6000 (ADSM/6000)
- Host Connection Program/6000 (HCON/6000)
- Distributed SMIT (DSMIT)
- AIX operating system parameters, such as users, file systems, adapters, etc.

9.4.1 Installing Software

All software that is required on a system has to be installed on that system first.

Depending on the type of software there are several ways to do that. The first software product that is needed on an RS/6000 system is the AIX base operating system.

You can install AIX, for example, by:

1. Using an AIX installation tape
2. Using a system backup (`mksysb`) tape
3. Using the NetView DM/6000 pristine installation feature

4. Using a network installation server

> **Note**
>
> If your systems come pre-installed from the IBM factory you might not need any of the above alternatives, because then all your systems already have a base operating system installed. However, with pre-installed systems you might also lose flexibility.

- AIX installation tape

  The classic way to install an AIX system is to use an AIX installation tape. Since this is very time-consuming and also requires permanent user-interaction this will not be your choice when you have to install dozens, or even hundreds of systems.

- System backup

  Another way is to install one system as a model and then install a system backup (`mksysb`) tape on the other workstations. This is a good choice if you have only one or a few types of systems on your network.

  Then you have to create a system backup image for each type of system. Using a system backup tape instead of an installation tape has the following advantages:

  – The installation of a system backup is a lot faster than the installation from an installation tape.

  – You can do a lot of customization on your model system before you create the system backup, and therefore, install this customization also on the systems that are installed with the system backup later.

- NetView DM/6000 pristine installation feature

  You have basically the same kind of installation as with a system backup tape. The difference is that in the pristine installation the system backup is installed from the network instead from a tape and that the whole process is triggered by NetView DM/6000.

  Since the pristine installation transmits the complete system backup over the network, and a system backup image is quite large, at least some hundred MB, this can only be performed in a LAN environment. However, in a typical production environment you will quite often have WAN connected systems.

- Network install server

  Using a network install server you can also install system backup images across a LAN network. In contrast to the pristine installation this process is not triggered by NetView DM/6000.

  Which of the above alternatives is appropriate for a specific environment depends mostly on the type of network:

  – In a LAN network normally the pristine installation is the best choice.

  – In a WAN network normally using system backup tapes is the best choice.

In some cases it may also be enough to use pre-installed systems. However, this is not very flexible. For example, if you have only a small amount of hard disk space available on your workstations, you may want to install only a minimal operating system. If your system comes pre-installed, you will normally not have the chance to do that.

Besides the base operating system you will normally have to install additional software packages. Of course you can install these packages on your model system before creating the system backup image.

As far as change management is concerned there is the following problem with this approach:

If you install software packages on the model system and then create a system backup image, these packages are not included in the change management history because they were not installed using NetView DM/6000. Since we want to track as much software as possible using the change management process this is not desirable.

The alternative is to install only the base operating system, for example using a system backup image, and then all additional packages using NetView DM/6000. However, this requires that you already have a working software distribution network before you can use NetView DM/6000. Hence, you will also need to install the NetView DM/6000 product, either server or agent, on each system.

This can be done, for example, by including the product in the system backup. The problem is, that NetView DM/6000 might need additional products in order to work (in our example SNA Server) so this must also be included in the system backup image.

Note

Software that was installed without using NetView DM/6000 should also be included in the change management history. A way to do that is to use the software inventory function of NetView DM/6000. How software inventory discovery can be implemented is explained in the redbook *NetView DM/6000 Agents and Advanced Scenarios*, GG24-4490.

9.4.2 Configuring Software Products

The configuration activities needed to configure a software product differ significantly depending on what product you need to configure.

There are products that need only very few configuration steps whereas other products might need a large number of configuration steps.

We have developed a configuration procedure for NetView DM/6000 in this book since automatic configuration of NetView DM/6000 is a quite complicated task.

Other products might need only a very limited customization. If the customization is the same for all systems, it can be included in the system backup, that is, it will be applied to the model system before the system backup is created. For example, if all systems need a modified `/etc/profile`, this customization can be performed on the model system.

If we have a configuration procedure, such as the one we have developed for NetView DM/6000 we have to distribute the configuration procedure as well as the configuration database to all systems that need to be configured.

In our example environment we have done this by transmitting all data across a LAN network. However, this might be a problem if we have WAN connected systems.

If we have a large number of systems to be configured, the ODM database that we use to store configuration information will be very large. This database has to be copied to all systems that have to be configured. If we have, for example, an X.25 network with a very limited bandwidth, it is very time consuming to transfer this data to all systems across the network.

If we decide to install the systems using a system backup it is a better approach to include the ODM database in the system backup then so that it does not have to be transmitted across the network.

Chapter 10. Pristine Installation

In this chapter we perform a pristine installation of an RS/6000 system on our network.

We combine the pristine installation procedure supplied with the NetView DM/6000 product with the configuration procedure we have developed previously in order to automatically create a NetView DM/6000 server.

10.1 Overview and Objective

With Version 1.2 a new function was introduced to the NetView DM/6000 product allowing the installation of RS/6000 systems "from scratch" triggered by NetView DM/6000.

Note

This chapter deals with the pristine installation feature available in NetView DM/6000 Version 1.2.1. If you are using Software Distribution for AIX you should refer to Chapter 16, “Cloning Systems Using Software Distribution for AIX 3.1” on page 317.

We do not explain the pristine installation in full detail here, since this is fully documented in the redbook *NetView DM/6000 Agents and Advanced Scenarios* GG24-4490. If you want to get detailed information about the pristine installation process, for example, about remote IPL, you should consult this redbook.

However, with PTF U436928 the pristine installation procedure was enhanced to allow for some new functionality.

The major enhancements are:

- You can now install backup images created with the `mksysb` command that are stored in files at the model workstation.
- The complete pristine installation process, including the preparation of the model workstation is now triggered by NetView DM/6000 change files.
- Sample change file profiles are supplied allowing for the easy customization for your specific environment.

We will use the pristine installation in this scenario to create a NetView DM/6000 server. This will include the following steps:

- Create an `mksysb` image of a NetView DM/6000 server and store it at the model workstation.
- Prepare the model workstation and the NetView DM/6000 server for the pristine installation process.
- Use the pristine installation procedure to install the NetView DM Agent/6000 product on the target workstation.
- Install the `mksysb` image on the target using NetView DM/6000 thus installing the NetView DM/6000 server code on the target.

- Use the configuration script `config_nvdm` to automatically configure the NetView DM/6000 server product at the target.

Note

It is important to understand that we will install a NetView DM/6000 server on the target workstation in this scenario although the pristine installation procedure itself covers only the installation of agents. However, the pristine installation procedure allows us to trigger the whole installation process using NetView DM/6000. For the above reasons the NetView DM/6000 server code is not contained in a change file but in the `mksysb` image stored on the model workstation.

10.2 The Pristine Installation Process

Before we start we will give a short overview of the pristine installation process.

Note

A description of the new pristine installation is available in the file `/usr/lpp/netviewdm6000/inst_U436928/lpp.README` at your server after you have installed the PTF U436928. Before you try the pristine installation you should read this file.

In order to perform the pristine installation we need to have a model workstation, a NetView DM/6000 server and a target that we want to install.

The model workstation will act as a network boot server for the target and also hold the `mksysb` image to be installed on the target.

The model can, however, also be the same machine as the NetView DM/6000 server, as in our scenario:

- We will have `rs60007` acting as the NetView DM/6000 server and as the model workstation.
- We will have `rs600015` acting as the target to be installed.

Note

Normally the model needs to be a CC client because the pristine installation procedure needs the files supplied with the NetView DM Agent/6000 in order to configure the target as an agent.

However, we can also use a server if we keep the install image of the agent on that server. How this can be done will be explained when preparing the model.

The target workstation will be configured to boot from the network using the model workstation as its boot server.

The boot image will contain a fully configured NetView DM/6000 agent enabling the target to act as a NetView DM/6000 agent right after it has been booted.

As soon as the NetView DM/6000 agent on the target is active, a change file can be installed from the server containing a script which will then install the `mksysb` image from the model to the target.

10.3 Prerequisites for Server and Model Workstation

The CC server needs to have the following software installed:

- AIX 3.2.5
- TCP/IP Version 2.1
- NetView DM/6000 including the Tools option

Note

It is especially important that the server has the NetView DM/6000 Tools option installed because this option includes the sample change file profiles to prepare the model and the server for the pristine installation.

The model workstation needs to have the following software installed:

- AIX 3.2.5
- TCP/IP Version 2.1
- Network File System (NFS)
- NetView DM Agent/6000

Since in our scenario the model workstation and the NetView DM/6000 server reside on the same machine, this machine needs to meet both of the above requirements.

In order to provide the agent code on the server the appropriate install image for the agent has to be in the `/usr/sys/inst.images` at the server.

Note

You cannot install a system backup for AIX 4.1 using the pristine installation feature of NetView DM/6000 Version 1.2.1. If you intend to clone AIX 4.1 systems or want to upgrade systems from AIX 3.2.5 to AIX 4.1 you must use Software Distribution for AIX. An example of how to use this feature is shown in Chapter 16, "Cloning Systems Using Software Distribution for AIX 3.1" on page 317. Please refer to this chapter if you intend to clone AIX 4.1 systems. Nevertheless you should continue reading this chapter completely since it contains information about the pristine installation process in general. This information is not repeated in Chapter 16, "Cloning Systems Using Software Distribution for AIX 3.1" on page 317.

The model workstation must also have enough disk space available:

- 16 MB of disk space for the first target to be free
- 8 MB of disk space for each additional target to be free

Furthermore, the model workstation needs disk space to hold the `mksysb` image that will be installed at the target. The size of this image depends on the system where you create the system backup but normally you should have at least 1 GB of free disk space.

10.4 Creating the System Backup Image

We will now create the system backup to be installed at the target as the first step in performing the pristine installation.

Before you create the system backup you must decide what you want to be installed on the target. In our scenario we want to install the following products:

- The AIX 3.2.5 base operating system
- NetView DM/6000 Version 1.2
- SNA Server Version 2.1

We need the NetView DM/6000 server product because we want to create a NetView DM/6000 server. This server is supposed to have SNA connections, so we also need the SNA Server product.

> **Note**
>
> You should notice that all the products you include in the system backup will not appear in the change management history of NetView DM/6000. If you want to avoid this, you can include only a minimal system in the `mksysb` image and install any additional products using NetView DM/6000. If you decide to include all products in the system backup, you can create an inventory discovery procedure to detect these products and add them to the software inventory. How to do this is described in the redbook *NetView DM/6000 Agents and Advanced Scenarios*, GG24-4490.

In order to create the system backup image we have to perform the following steps:

1. Install a system that can be used to create the system backup.
2. Create a file system on the model workstation to hold the system backup.
3. Export this file system to the target using NFS.
4. Transfer the system backup to the model workstation.

First we have to select a system which has all the software installed that we want to have in our system backup.

If we do not have a system with an appropriate configuration we will have to install the additional software products prior to creating the system backup.

In our example we create the system backup on `rs600015` which contains NetView DM/6000 server and SNA Server.

Note

You might want to customize the system where you create the system backup before initiating the `mksysb` command. For example, you can customize system profiles such as `/etc/profile`.

In our example it is particularly important, that we create a `/.rhosts` file on the machine we want to take the system backup from, containing the following line:

```
rs60007.itso.ral.ibm.com   root
```

This is required, because we want to run the configuration script on `rs600015` from `rs60007` to configure the NetView DM/6000 server after the pristine installation is completed.

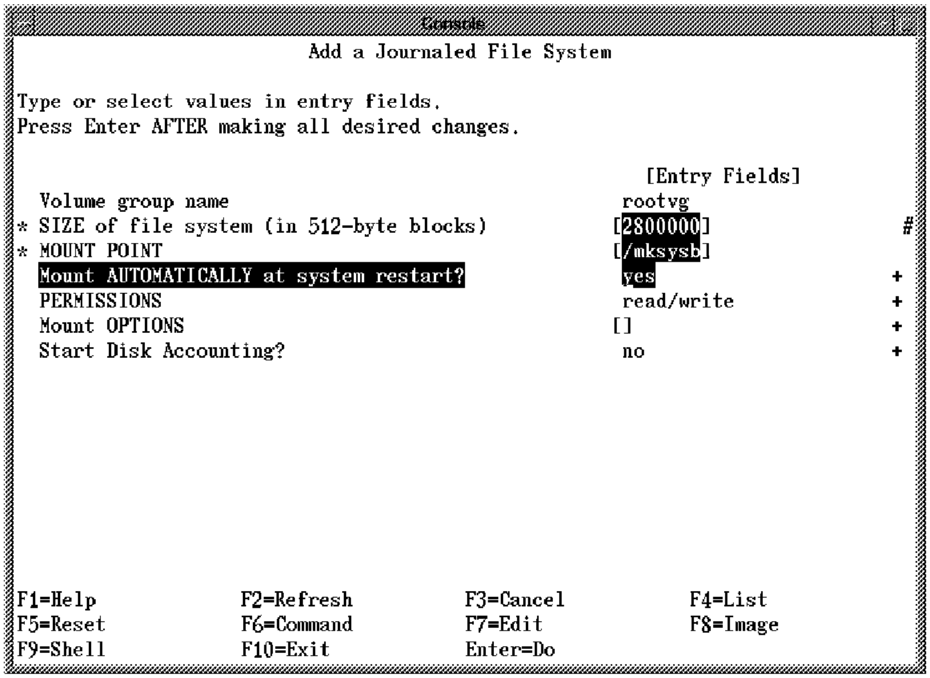
Before we create the system backup we will create the file system to hold the system backup on our model workstation and then export this file system to `rs600015`.

This enables us to mount this file system from `rs600015` and then write the system backup directly to the NFS mounted drive. Therefore we do not need to explicitly transfer the `mksysb` image to `rs60007` after we have created it on `rs600015`.

In order to create the file system we do the following:

1. Type `smitty crfs` to get to the appropriate SMIT fast path.
2. Select **Add a Journaled File System** and press Enter.
3. Select **rootvg** and press Enter.

The following panel will appear:



```
Console
                          Add a Journaled File System

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
  Volume group name                                    rootvg
* SIZE of file system (in 512-byte blocks)            [2800000]            #
* MOUNT POINT                                         [/mksysb]
  Mount AUTOMATICALLY at system restart?               yes                 +
  PERMISSIONS                                          read/write          +
  Mount OPTIONS                                       []                   +
  Start Disk Accounting?                               no                  +

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 97. SMIT Add a Journaled File System Panel

In our example we create a new file system `/mksysb` to be `2800000` blocks of size. We fill in the fields as shown above and then press Enter to create the file system.

> **Note**
>
> The file system that we want to create is fairly big, so it might exceed the maximum number of logical partitions allowed for a logical volume by default.

If the command to create the file system fails, we do the following:

1. Type `smitty chlv` to get to the appropriate SMIT fast path.
2. Select **Change a Logical Volume** and then press Enter.
3. Press F4, select the logical volume holding the file system and the press Enter.

The following panel will appear:

```
                        Change a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
* Logical volume NAME                                  lv02
  Logical volume TYPE                                 [jfs]
  POSITION on physical volume                          outer_middle        +
  RANGE of physical volumes                            minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  [32]                 #
    to use for allocation
  Allocate each logical partition copy                 yes                 +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during reorganization?  yes                 +
  Logical volume LABEL                                [/mksysb]
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]
  SCHEDULING POLICY for writing logical                parallel            +
    partition copies
  PERMISSIONS                                          read/write          +
[MORE...3]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 98. SMIT Change a Logical Volume Panel

Change the MAXIMUM NUMBER of LOGICAL PARTITIONS field to `512` and press Enter.

After that you can enlarge the file system using `smitty chfs`.

If the file system was created successfully we can mount it by typing:

```
mount /mksysb
```

We need to export the file system to `rs600015` using NFS in order to write the system backup created on `rs600015` to that file system.

To do so we can either use the SMIT fast path `smitty nfs` and then get to the appropriate panel or we type in the command:

```
mknfsexp -d /mksysb -t rw -r rs600015 -B
```

On `rs600015` we have to mount the file system using the following command:

```
mount rs60007:/mksysb /mksysb
```

After we have mounted the file system on `rs600015` we can create the system backup.

To do so we type `smitty mksysb` on `rs600015`.

The following panel will appear:

```
                         Backup The System

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
    WARNING:  Execution of the backup command will
              result in the loss of all material
              previously stored on the selected
              output medium. This command backs
              up only rootvg volume group.

  FORCE increase of work space if needed            yes               +
* Backup DEVICE or FILE                            [/mksysb/600015.image]
    (example: /dev/rfd0)







F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 99. SMIT Backup the System Panel

We enter the values as shown above and press Enter.

This will create a system backup of `rs600015` and store it in the file `/mksysb/600015.image`, which is physically located on `rs60007`.

Note

You must specify a file as the backup device, not a directory. If you specify a directory, the `mksysb` command will fail.

10.5 Preparing the Model Workstation and the Server

The first thing to do on the model workstation is to export the file system where the system backup is located to the target.

Since we already did this to create the system backup we can skip this step.

In order to configure the model workstation as a network boot server an example script is provided in `/usr/lpp/netviewdm/tool/NServ.cfg.template` at the server. We have to customize this script to use it for our specific environment.

Before we do this, we copy the template file to another file that we will customize afterwards:

```
cd /usr/lpp/netviewdm/tool
cp NServ.cfg.template Boot_Serv.config
```

Note

You should copy the file to Boot_Serv.config because this is the name used in the change file profile. If you want to use another name, you will have to modify profile.pristool to reflect this change.

We have to customize the Boot_Serv.config file to reflect our specific environment. In order to customize it we need the following parameters:

- The network adapter device used for network booting the target
- The IP hostname of the target
- The hardware address of the network adapter installed in the target
- The IP hostname of the server

The following figure shows the Boot_Serv.config file that we have modified for our scenario:

```
#!/bin/ksh
#     (C) COPYRIGHT IBM CORP 1993                                            #
#     ALL RIGHTS RESERVED                                                    #
#     LICENSED MATERIALS - PROPERTY OF IBM                                   #
#                                                                            #
#  Module:          NServ.cfg.template                                       #
#                                                                            #
#  Component:       Pristine machine installation tool                       #
#                                                                            #
#  Description:     AIX Korn shell script to configure Network Server        #
#                                                                            #
#
#                                                                            #
# This is a sample of the procedure that must be run on the CC Client        #
# that will act as the Network Server for the pristine installation.         #
#                                                                            #
# ROLES OF NETWORK SERVER:                                                   #
#     1. It is a model workstation in the clone installation procedure       #
#        (its hard disk is copied onto another workstation).                 #
#     2. Boot Server. It provides the network boot image for remote          #
#        clients.                                                            #
#     3. System backup images repository for the backup installation         #
#        procedure. The system backup images must reside in a directory      #
#        (that cannot be /usr or under /usr) and exported to the pristine    #
#        clients.                                                            #
#                                                                            #
# NOTE: The Network Server must have about 16Mb of free disk space for the   #
#       first CC client to be installed and about 8Mb for any additional     #
#       CC client.                                                           #
#                                                                            #
# Replace the following parameters with their actual values:                 #
# </dev/netdevice>: /dev/tok0, /dev/tok1, /dev/tok2, /dev/ent0, /dev/ent1,   #
#                   /dev/ent2                                                #
# <clientname>: pristine CC client name                                      #
# <hardwareaddress>: boot device MAC address                                 #
# <servername>: NetView DM/6000 name                                         #
# <gatewayIPaddr>: gateway IP address (optional)                             #
# <subnetmask>: subnetwork mask (optional, mandatory if gatewayIPaddr is     #
#               used)                                                        #
#                                                                            #
# EXAMPLE:                                                                   #
# /usr/lpp/netviewdm/script/fndnprel /dev/tok0 RISC02 08005A4FD558 RISC03 \  #
# -G 129.82.23.3 -S 255.255.255.0
#
#                                                                            #
#                                                                            #



ids=`id | sed -e "s/uid=\([0-9]*\).*gid=\([0-9]*\).*/\1 \2/"`
```

Figure 100 (Part 1 of 2). Boot_Serv.config File on rs60007

```
set -- `echo $ids`
GROUPs=$2
cp /usr/lib/dwm/dwm_functions /usr/lib/dwm/dwm_functions.$$
sed -e "s/\"0 0\"/\"0 $GROUPs\"/" /usr/lib/dwm/dwm_functions > /usr/lib/dwm/tt
mv /usr/lib/dwm/tt /usr/lib/dwm/dwm_functions
cd /usr/lpp/netviewdm/script

#
# Customize the following part
#

/usr/lpp/netviewdm/script/fndnprel  /dev/tok0 rs600015 \
  10005ab1d731 rs60007
rc=$?

#
# End of customizable part
#

mv /usr/lib/dwm/dwm_functions.$$ /usr/lib/dwm/dwm_functions

exit $rc
```

Figure 100 (Part 2 of 2). Boot_Serv.config File on rs60007

Normally the `Boot_Serv.config` is used in a change file which is then installed on the model workstation. For that purpose there is a change file profile `profile.pristool` which can be used to build the change file:

```
cd /usr/lpp/netviewdm/tool
nvdm bld profile.pristool
```

Since our NetView DM/6000 server also acts as the network boot server we do not need to execute this script using a change file but can directly execute it on `rs60007`:

```
cd /usr/lpp/netviewdm/tool
./Boot_Serv.config 2>&1 | tee bserv.log
```

Note

In order for the script to run successfully you have to supply the latest PTF level of the agent code in the `/usr/sys/inst.images` directory. At the time this book was written this was the file `netviewdm6000.1.0.2.1.U436929`. The preparation script will look exactly for that file name. You do not have to supply the agent code if your model workstation is a NetView DM/6000 agent because the script then accesses the agent code directly.

The following figure shows the log file produced for the above command:

```
Creating /export/install filesystem  8MB large.

New File System size is 16384
Creating /export/root filesystem  4MB large.

New File System size is 8192
Creating /export/nvdma filesystem  4MB large.

New File System size is 8192
Creating /export/nvdma/rs600015 directory.

Making the boot image...

bosboot: Boot image is 4262 512 byte blocks.
Making the INSTALL spot...
Creating the rs600015 client.

ln: /tftpboot/rs600015 exists.  Specify -f to remove.
Making NFS and exporting file systems: it may take some minutes.

New volume on /usr/sys/inst.images/netviewdm6000.1.0.
 Cluster 51200 bytes (100 blocks).
    Volume number 1
    Date of backup: Tue Apr 11 10:45:50 1995
    Files backed up by name
    User caminada
    files restored: 10
Creating diskettes files...
Creating extended display diskette...
    files restored: 32
Backing up to /export/install/dskt/ext
 Cluster 51200 bytes (100 blocks).
Volume 1 on /export/install/dskt/ext
Done     at Fri May 19 16:32:36 1995.
2300 blocks on 1 volume(s)
Creating display diskette...
    files restored: 32
Backing up to /export/install/dskt/disp
 Cluster 51200 bytes (100 blocks).
Volume 1 on /export/install/dskt/disp
Done     at Fri May 19 16:32:43 1995.
2400 blocks on 1 volume(s)
Creating install and maintenance diskette...
Backing up to /export/install/dskt/inst
 Cluster 51200 bytes (100 blocks).
Volume 1 on /export/install/dskt/inst
Done     at Fri May 19 16:32:52 1995.
2800 blocks on 1 volume(s)
    files restored: 2
```

Figure 101 (Part 1 of 2). bserv.log Log File

```
    files restored: 2
    files restored: 1
Populating /export/install with needed commands...
    files restored: 5
    files restored: 6
```

Figure 101 (Part 2 of 2). bserv.log Log File

The script will perform several tasks, allowing the machine on which it is executed to act as a network boot server.

This will include:

- Creating NFS file systems to be exported to the target
- Creating a boot image for the target
- Creating an install SPOT for the target
- Populating the exported file systems with the NetView DM/6000 agent code

Note

Under certain circumstances the `fndnprel` script might fail.

The script creates several diskette images and stores them in the file systems to be exported to the client. In case the files to be stored on the diskette do not fit onto one diskette, the command that is used to create the diskette will ask for a new diskette.

For example, the `mkinstdskt` command might need two diskettes if you have an FDDI adapter installed. Since the output of `mkinstdskt` and other commands is redirected to `/dev/null` within the `fndnprel` script, the message prompting the user to insert another diskette cannot be seen on the screen.

If the script does not continue for a long time, you should therefore remove the redirection to `/dev/null`.

The process of installing the `mksysb` image on the target will be triggered by a NetView DM/6000 change file.

There is a change file profile that can be used to create this change file which is located in `/usr/lpp/netviewdm/tool/profile.backup`. The only thing that needs to be customized in this change file profile is the name of the `mksysb` install image.

The following figure shows the change file profile modified for our specific environment:

```
GLOBAL NAME:                NVDM.BACKUP.REF.1
CHANGE FILE TYPE:           GEN
COMPRESSION TYPE:           LZW
PACK FILES:                 NO
SECURE PACKAGE:             NO
# Replace "/bck.images/sysbck1" with the full path of your backup
PRE-INSTALL:           /usr/lpp/netviewdm/work/fnd_sysbck_inst /mksysb/60001
POSTREQ COMMAND:           /call_shutc

OBJECT:
    SOURCE NAME:            /usr/lpp/netviewdm/script/fnd_sysbck_inst
    TARGET NAME:            /usr/lpp/netviewdm/work/fnd_sysbck_inst
    TYPE:                   FILE
    ACTION:                 COPY
    INCLUDE SUBDIRS:        NO
```

Figure 102. profile.backup File on rs60007

To build the change file we type:

```
cd /usr/lpp/netviewdm/tool
nvdm bld profile.backup
```

Note

You might want to have several system backups stored at your model workstation to install different kinds of clients. In that case you should build one change file for each system backup image. Do this by copying `profile.backup` to another file name and then customizing the `PRE-INSTALL` field. You also need to assign a new `GLOBAL NAME` since you cannot have multiple change files with the same name in your catalog.

10.6 Booting the Client

We will now boot the client from the network, using our NetView DM/6000 server as the network boot server.

Before doing so we have defined the target workstation `rs600015` as a local target to the NetView DM/6000 server `rs60007`

Note

In this example we assume that the target is able to perform a remote IPL. If you do not know how to determine whether your target workstation supports remote IPL or what to do if your target does not support remote IPL you should consult the redbook *NetView DM/6000 Agents and Advanced Scenarios*.

In order to boot the target we perform the following steps on the target workstation:

1. Turn the key switch to Secure position and switch on the system.
2. Wait until the three-digit LED displays `200`.

3. Turn the key switch to Service position, press the yellow reset button and wait until the boot MAIN MENU appear on the screen.

> **Note**
>
> If the boot MAIN MENU does not appear within a short time your target workstation probably does not support remote IPL.

4. Select **1** and press Enter. This will get you to the SELECT BOOT (STARTUP) DEVICE panel.
5. Select the type of network adapter you use for booting and press Enter. This will get you to the SET OR CHANGE NETWORK ADDRESSES panel.
6. Enter the IP addresses for the client and the boot server, then select **99** and press Enter. This will save the addresses and get you back to the main menu.
7. In the main menu select **6** and press Enter.
8. Turn the key switch to Normal position and press Enter.

Within a short period of time the STARTING SYSTEM (BOOT) panel will appear. After a few seconds the system will display the number of BOOTP and TFTP packets being transferred between the boot server and the target.

If it does not, something is wrong with the setup of either the network boot server or the target workstation. There are a lot of possible causes for the network boot not working correctly.

Most of them as well as possible solutions are also documented in the redbook *NetView DM/6000 Agents and Advanced Scenarios* GG24-4490. If you encounter any problems with the remote boot process you should consult either this redbook or the appropriate AIX system documentation.

After a short time the network boot of the target will be completed and the target displays a message that its NetView DM/6000 agent is ready and waiting for change requests.

10.7 Submitting the Install Request

Before we submit the install request to perform the pristine installation of the target we have to check if the NetView DM/6000 agent on the target is ready.

To do so we start the NetView DM/6000 graphical user interface on `rs60007` by typing:

```
nvdmgi &
```

In the catalog window we select **Windows** from the menu bar and then **Targets...** from the pull-down menu.

The following panel will appear:

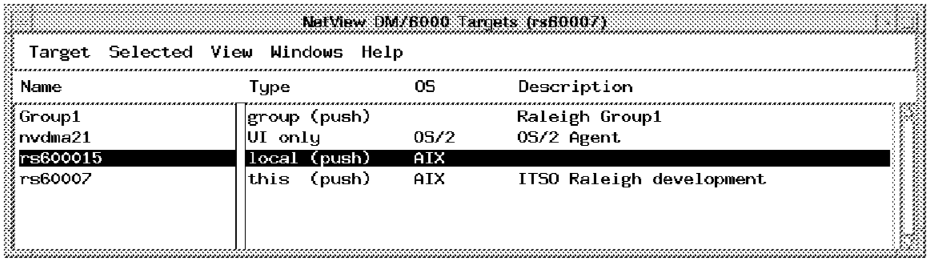


Figure 103. NetView DM/6000 Targets Window

We select the `rs600015` target. After that we select **Selected** from the menu bar and then **Status...** from the pull-down menu.

The following panel will appear:

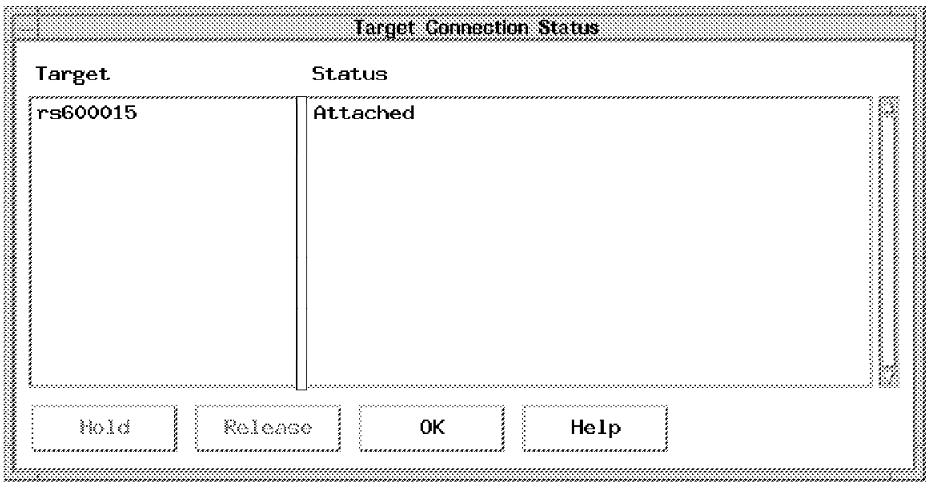


Figure 104. NetView DM/6000 Target Connection Status Window

The agent should have a status of 'Attached'. If it does not, the target failed to start the NetView DM/6000 agent.

In that case you should do the following:

- Watch the messages appearing on the screen when starting the target. This might give you some hints about the cause of the error.
- Check the NetView DM/6000 server log file `/usr/lpp/netviewdm/fndlog` on `rs60007`.

Note

In our scenario we encountered an error because the `/usr` file system was not correctly exported on `rs60007`. Therefore the client was denied access when trying to mount this filesystem from `rs60007` and the `fndpru` script which is used to boot the client was killed.

In order to export the `/usr` file system to `rs600015` we typed the following line at `rs60007`:

```
mknfsexp -d /usr -t rw -r rs600015
```

In order to start the installation of the `mksysb` we go to the catalog window of NetView DM/6000:

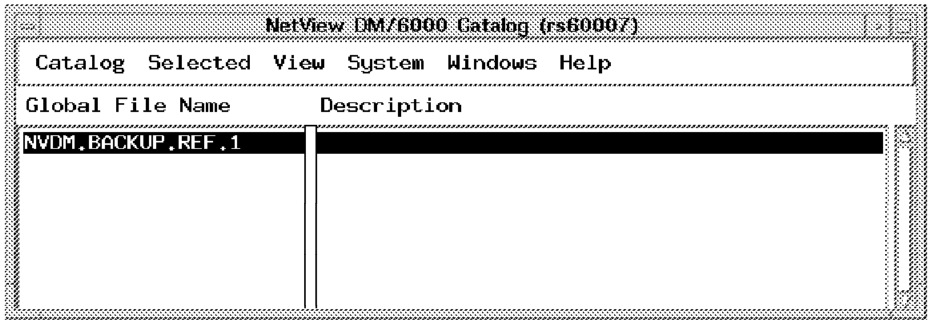


Figure 105. NetView DM/6000 Catalog Window on rs60007

We do the following:

1. Select the change file NVDM.BACKUP.REF.1.
2. Select **Selected** from the menu bar.
3. Select **Install...** from the pull-down menu.

The following panel will appear:

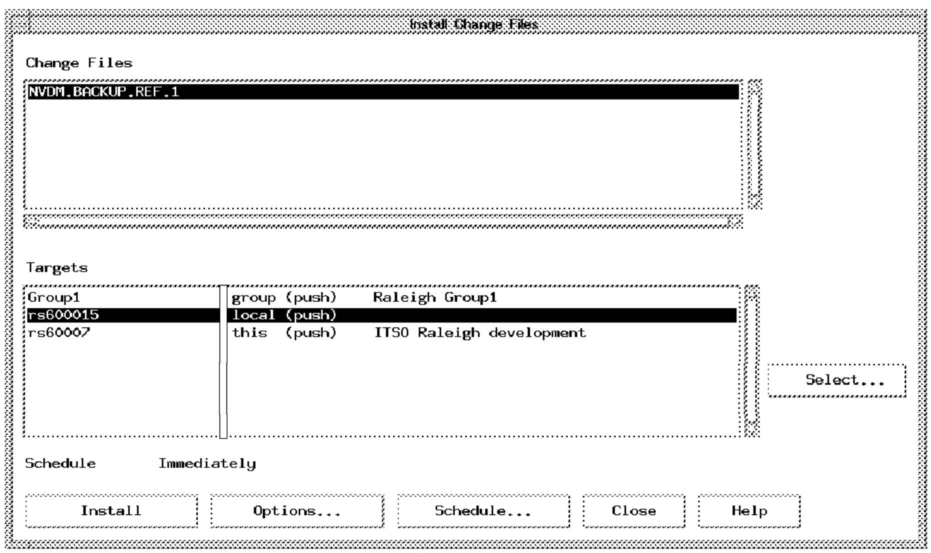


Figure 106. NetView DM/6000 Install Change Files Window

We select the target `rs600015` and then the **Install** push button. This will submit the installation request.

To watch the progress of the installation process we type the following command on `rs60007`:

```
tail -f /export/nvdma/rs600015/work/request.out
```

The following figure shows the `request.out` file that was produced for our example environment. This file is quite large, so we will show only excerpts:

```
Mounting device...
600015.image
/mksysb

Retrieving .fs.size from backup...

Retrieving file systems sizes from install media...

rootvg

Creating page logical volume(s).
hd6
hd61

Creating boot logical volume.
hd5

Creating dump logical volume.
hd7

Creating log logical volume.
hd8

Creating / logical volume.

Making the / file system...

Creating /usr logical volume.

Making the /usr file system...

Creating /home logical volume.

Making the /home file system...

Creating /tmp logical volume.

Making the /tmp file system...

Creating /usr/lpp/netviewdm logical volume.

Making the /usr/lpp/netviewdm file system...

Creating /var logical volume.

Making the /var file system...
USTAR format archive
./.fs.size
```

Figure 107 (Part 1 of 2). /export/nvdma/rs600015/work/request.out File

```
./var/
./var/adm
./var/adm/acct
./var/adm/cron

....


Copying device special files from RAM to hard disk...

Copying volume group maps from RAM to hard disk...
mkdir: cannot create /tmp/objrepos.inst.
/tmp/objrepos.inst: File exists

Copying ODM from RAM to hard disk...
9.24.104.76

Updating database with names of dynamically created special files..

Retrieving device configuration from previous system...

bosboot: Boot image is 11297 512 byte blocks.

 Licensed Materials - Property of IBM

 5756-03001
   (C) Copyright International Business Machines Corp. 1985, 1991.
   (C) Copyright AT&T 1984, 1985, 1986, 1987, 1988, 1989.
   (C) Copyright Graphic Software Systems Incorporated 1984, 1990, 1991.
   (C) Copyright KnowledgeSet Corporation 1990, 1991.
   (C) Copyright Open Software Foundation, Inc. 1989, 1990.
   (C) Copyright Massachusetts Institute of Technology 1985, 1986,
        1987, 1988, 1989.
   (C) Copyright Regents of the University of California 1980, 1982,
        1983, 1985, 1986, 1987, 1988, 1989.
   (C) Copyright Silicon Graphics, Inc. 1988, 1989, 1990.
   (C) Copyright SUN Microsystems, Inc. 1984, 1985, 1986, 1987, 1988&peri
   (C) Copyright TITN Inc. 1984, 1989.
   (C) Copyright Mentat Inc. 1990, 1991.

 All rights reserved.
 US Government Users Restricted Rights - Use, duplication or disclosure
 restricted by GSA ADP Schedule Contract with IBM Corp.

        Task completed...

                       fnd_sysbck is exiting now.
                       If the system booted in Maintenance Mode
                       turn the key in Normal Mode.
```

Figure 107 (Part 2 of 2). /export/nvdma/rs600015/work/request.out File

Warning

The `request.out` file produced when installing the change file to perform the pristine installation is located in the `/export/nvdma` file system which has been created by the preparation script `fndnprel`. By default this file system is created with a size of 4MB. However, if you have a large number of files in your `mksysb` image the `request.out` file may be too large for the file system. This is because every file extracted at the target is logged in `request.out`.

To make sure that the `/export/nvdma` file system will not run full you should enlarge it to be 8MB of size.

To monitor whether the install request has finished we go to the NetView DM/6000 Targets window and select target `rs600015`. Then we select **Selected** from the menu bar and then **Open** from the pull-down menu.

From the cascaded menu we select **History...**

The following panel will appear:

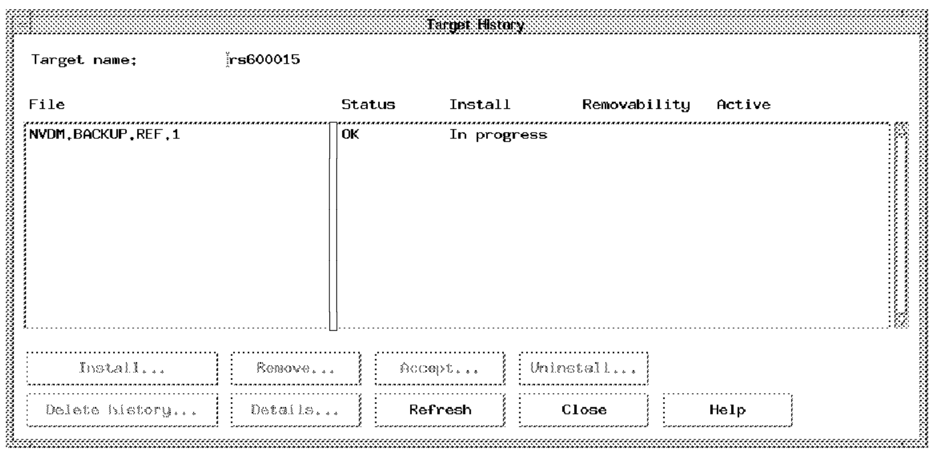


Figure 108. NetView DM/6000 Target History Window

As long as the status is 'In progress' the installation script is running.

Note

The pristine installation process will take some time to run. The time needed to perform the process depends mostly on the speed of the network you use and on the size of the `mksysb` image.

In our example, using a 4Mb token-ring network and an install image being approximately 1GB of size the installation process took about 2.5 hours to run.

10.8 Configuring the NetView DM/6000 Server

We will now run the configuration procedure to configure the NetView DM/6000 server product on `rs60007`.

The configuration data base we use for that purpose is the same that we have used in the scenario where we developed the configuration procedure.

Warning

You should be aware of the fact that the reconfiguration of a NetView DM/6000 server is a lot more complicated than the reconfiguration of an agent.

In this scenario we installed the NetView DM/6000 product on an RS/6000 system and then made a system backup image to contain all the products currently installed on that machine.

When the NetView DM/6000 server is installed it automatically configures the `nvdm.cfg` file to contain the current `hostname` in the `WORKSTATION NAME` and `SERVER` fields.

Further it creates an initial target record for the server, located in the file `/usr/lpp/netviewdm/db/target_config/`*servername*.

In our specific environment this is not a problem, because we took the system backup on `rs600015` and then installed `rs600015` "from scratch" using this image, so the `hostname` as well as the NetView DM/6000 server name did not change.

However, if you intend to install a lot of servers using the same system backup image, the `hostname` and the NetView DM/6000 server name will be different on every server.

The pristine installation script will automatically configure TCP/IP on the target system. This will include setting the `hostname` correctly.

What will happen then is that the `hostname` and the `WORKSTATION NAME` configured in `nvdm.cfg` do not match and that you can therefore neither start nor stop the NetView DM/6000 server.

For example if the server name on which the system was created was `rs6000xx` and the target was `rs600015` the `WORKSTATION NAME` and `SERVER` fields contain `rs6000xx` whereas the `hostname` is set to `rs600015` after the pristine installation.

Also the initial target record is stored in

```
/usr/lpp/netviewdm/db/target_config/rs6000xx
```

When we reconfigure the `nvdm.cfg` the `nvdm.cfg` file for `rs600015` the server will fail to start because there is no initial target record for `rs600015`.

To solve this problem we add the following code to our script to be executed before configuring the `WORKSTATION NAME`:

```
#-------------change--------------
# if we configure a server and want to change the
# WORKSTATION NAME we must stop the server before
# reconfiguring this field.
# To do so we must be sure that the hostname and
# the WORKSTATION NAME match
#
if [ "$NODE_TYPE" = "0" -o "$NODE_TYPE" = "2" ]
then
  # get current hostname
  OHN=`hostname`
  print "NVDM CONFIG : Current hostname of server is $OHN."
  # get WORKSTATION NAME currently configured
  HN=`grep "WORKSTATION NAME:" $CONFIG | cut -d':' -f2`
  print "NVDM CONFIG : Current WORKSTATION NAME of server is $HN."
  print "NVDM CONFIG : Stopping Server..."
  # make sure that both match
  hostname $HN
  nvdm stop -x
  print "NVDM CONFIG : Sleeping 10 seconds..."
  sleep 10
  # set back hostname
  print "NVDM CONFIG : Setting hostname to $OHN."
  hostname $OHN
  # also, we must be sure that there is an initial target record for
  # the servername configured
  ls /usr/lpp/netviewdm/db/target_config/$1 >/dev/null 2>&1
  if [ $? -ne 0 ]
  then
    echo "DESCRIPTION:     INITIAL TARGET CONFIGURATION RECORD
TARGET TYPE:     PUSH
TARGET OS:       AIX
RBAPI TRACE:     NONE
LOG LEVEL:       N
SHORT NAME:      SERVER
CM WINDOW START:           0 : 0
CM WINDOW STOP:            23:59
DISTRIBUTION WINDOW START: 0 : 0
DISTRIBUTION WINDOW STOP:  23:59

NUMBER OF PARMS: 0
NUMBER OF USERS: 1
USER: root" >/usr/lpp/netviewdm/db/target_config/$1
  fi
fi
#-----------end-of-change---------
```

Figure 109. Code to Change Server Settings

Before starting the NetView DM/6000 server configuration we have to make sure that the server is running, because this is a prerequisite for configuring targets, etc.

The following code will be inserted before the server configuration part in our script:

```
#-------------change--------------
# restart server, in case it is not already running
print "NVDM CONFIG : Restarting Server..."
nvdm start
#-----------end-of-change---------
```

Figure 110. Code to Restart NetView DM/6000 Server

With these changes made we run the configuration script to configure `rs600015`.

For that purpose the file `node_list` contains the following line:

```
rs600015
```

To run the configuration script and redirect the output to `netlog3` we type:

```
configure_network 2>&1 | tee netlog3
```

The log file produced for this command looks like the following:

```
**** CONFIGURING NETVIEW DISTRIBUTION MANAGER/6000 ****
** Creating tar archive
Size before compressing : 757760
** Crunching tar archive
Size after compressing : 214702
*** Processing node : rs600015
** Copy compressed archive
** Uncrunching compressed archive
** Extracting files from tar archive
Creating ODM DB ...
nvdm_groups
nvdm_node
nvdm_users
nvdm_cfg_static
nvdm_servers
nvdm_queues
Invoking configuration script...
NVDM CONFIG : --> Trying to configure node rs600015
NVDM CONFIG : Node type is 0 (0 = Server, 1 = Agent, 2 = Prep)
NVDM CONFIG : --> NVDM Base Node Configuration
NVDM CONFIG : Current hostname of server is rs600015.
NVDM CONFIG : Current WORKSTATION NAME of server is      rs6000xx.
NVDM CONFIG : Stopping Server...
rs6000xx
Trying to connect to default server (rs6000xx).
Connected to server rs6000xx.
NVDM CONFIG : Sleeping 10 seconds...
NVDM CONFIG : Setting hostname to rs600015.
rs600015
NVDM CONFIG : Setting nvdm.cfg (WORKSTATION NAME) to rs600015
NVDM CONFIG : Setting nvdm.cfg (SERVER) to rs600015
NVDM CONFIG : Setting nvdm.cfg (LOG FILE SIZE) to 250000
NVDM CONFIG : Setting nvdm.cfg (TCP/IP PORT) to 729
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Restarting Server...
Trying to connect to default server (rs600015).
Connected to server rs600015.
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Setting SNA Network Name to USIBMRA
NVDM CONFIG : Setting SNA Datalink Device to tok0
NVDM CONFIG : Setting SNA Remote Link Address to 400001240000
NVDM CONFIG : Setting SNA NVDM Mode Profile Name to NVDMNORM
NVDM CONFIG : Setting SNA NVDM Mode Name to NVDMNORM
NVDM CONFIG : Setting SNA TPN Profile Name (Send) to NVDMSND
NVDM CONFIG : Setting SNA TPN Profile Name (Receive) to NVDMRCV
NVDM CONFIG : Setting SNA Partner LU Name (MVS Host) to RA39TCF1
NVDM CONFIG : Setting SNA Side Info Profile Name (Send) to NVDMSIDS
NVDM CONFIG : Setting SNA Side Info Profile Name (Receive) to NVDMSIDR
NVDM CONFIG : Setting Solicit SSCP Field (yes|no) to yes
```

Figure 111 (Part 1 of 5). netlog3 Log File

```
NVDM CONFIG : Setting I-Field Size to 2042
NVDM CONFIG : Setting SNA Local SAP No. to 04
NVDM CONFIG : Setting Remote SAP No. to 04
NVDM CONFIG : Setting SNA Initiate Call Field (yes|no) to yes
NVDM CONFIG : Setting SNA Activate on start (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on normal termination (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on abnormal termination (yes|no) to yes
NVDM CONFIG : Setting SNA VTAM CP Name (for LU6.2 Location Profile) to RAK
NVDM CONFIG : Setting PU NAME for rs600015 to RA60015
NVDM CONFIG : Setting Local LU Name for rs600015 to RA60015B
NVDM CONFIG : Setting Control Point Name for rs600015to RA6015CP
NVDM CONFIG : Could not determine XID for rs600015 configu
ration.
NVDM CONFIG : Setting USE_CP_XID to yes
+ mk_qcinit -y token_ring -t appn_end_node -w USIBMRA -d RA6015CP
NVDM CONFIG : --> Configuring SNA
NVDM CONFIG : Adding DLC Device for tok0
NVDM CONFIG : Configuring SNA Initial Node Setup
+ chsnaobj -t control_pt -e USIBMRA -a RA6015CP -A RA6015CP
-N appn_end_node node_cp
NVDM CONFIG : Configuring SNA Control Point Profile
===============================================================================
Profile type 'control_pt' name 'node_cp' CHANGED.
===============================================================================
+ mksnaobj -t sna_dlc_token_ring -d tok0 -b yes -w yes -m 2042 -H 04
-c no -q 0 tok0
0105-0031 Profile type 'sna_dlc_token_ring' name 'tok0' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t sna_dlc_token_ring -d tok0 -b yes -w yes -m 2042 -H 04
-c no -q 0 tok0
NVDM CONFIG : Configuring SNA DLC Profile
===============================================================================
===============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
===============================================================================
Profile type 'sna_dlc_token_ring' name 'tok0' CHANGED.
===============================================================================
+ mksnaobj -t link_station -w token_ring -y tok0 -d 400001240000
-l 07100000 -s 04 -a yes -O yes -F yes -h yes -z yes -c yes RA60015
0105-0031 Profile type 'link_station_token_ring' name 'RA60015' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t link_station -w token_ring -y tok0 -d 400001240000
-l 07100000 -s 04 -a yes -O yes -F yes -h yes -z yes -c yes RA60015
NVDM CONFIG : Configuring SNA Link Station Profile
===============================================================================
===============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
===============================================================================
```

Figure 111 (Part 2 of 5). netlog3 Log File

```
+ mksnaobj -t local_lu -u lu6.2 -l RA60015B -L RA60015B RA60015B
0105-0031 Profile type 'local_lu_lu6.2' name 'RA60015B' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t local_lu -u lu6.2 -l RA60015B -L RA60015B RA60015B
Profile type 'link_station_token_ring' name 'RA60015' CHANGED.
===============================================================================
NVDM CONFIG : Configuring SNA Local LU Profile
===============================================================================
===============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
===============================================================================
+ mksnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N #CONNECT -m NVDMNORM NVDMNORM
0105-0031 Profile type 'mode' name 'NVDMNORM' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N #CONNECT -m NVDMNORM NVDMNORM
Profile type 'local_lu_lu6.2' name 'RA60015B' CHANGED.
===============================================================================
NVDM CONFIG : Configuring SNA Mode Profile
===============================================================================
===============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
===============================================================================
+ mksnaobj -t local_tp -n 21F0F0F7 -h yes -c basic -d 0 -P yes
-w /usr/lpp/netviewdm/bin/fndts -s none NVDMSND
0105-0031 Profile type 'local_tp' name 'NVDMSND' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t local_tp -n 21F0F0F7 -h yes -c basic -d 0 -P yes
-w /usr/lpp/netviewdm/bin/fndts -s none NVDMSND
+ mksnaobj -t local_tp -n 21F0F0F8 -h yes -c basic -d 0 -P yes
-w /usr/lpp/netviewdm/bin/fndtr -s none NVDMRCV
Profile type 'mode' name 'NVDMNORM' CHANGED.
===============================================================================
NVDM CONFIG : Configuring SNA TPN Profile (SEND)
===============================================================================
===============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
===============================================================================
Profile type 'local_tp' name 'NVDMSND' CHANGED.
===============================================================================
NVDM CONFIG : Configuring SNA TPN Profile (Receive)
===============================================================================
0105-0031 Profile type 'local_tp' name 'NVDMRCV' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t local_tp -n 21F0F0F8 -h yes -c basic -d 0 -P yes
-w /usr/lpp/netviewdm/bin/fndtr -s none NVDMRCV
+ mksnaobj -t partner_lu6.2 -p no -P USIBMRA.RA39TCF1 -O none
```

Figure 111 (Part 3 of 5). netlog3 Log File

```
-A RA39TCF1 RA39TCF1
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
Profile type 'local_tp' name 'NVDMRCV' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA LU6.2 Partner LU
================================================================================
0105-0031 Profile type 'partner_lu6.2' name 'RA39TCF1' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t partner_lu6.2 -p no -P USIBMRA.RA39TCF1 -O none
-A RA39TCF1 RA39TCF1
+ mksnaobj -t partner_lu6.2_location -P USIBMRA.RA39TCF1 -O USIBMRA.RAK
-m link_station -l RA60015B -s RA60015 RA39TCF1
0105-0031 Profile type 'partner_lu6.2_location' name 'RA39TCF1' already exists.
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
Profile type 'partner_lu6.2' name 'RA39TCF1' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA LU 6.2 Location Profile
================================================================================
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t partner_lu6.2_location -P USIBMRA.RA39TCF1 -O USIBMRA.RAK
-m link_station -l RA60015B -s RA60015 RA39TCF1
+ mksnaobj -t side_info -L RA6015CP -P USIBMRA.RA39TCF1 -m NVDMNORM
-d 21F0F0F7 -h yes NVDMSIDS
0105-0031 Profile type 'side_info' name 'NVDMSIDS' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t side_info -L RA6015CP -P USIBMRA.RA39TCF1 -m NVDMNORM
-d 21F0F0F7 -h yes NVDMSIDS
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
Profile type 'partner_lu6.2_location' name 'RA39TCF1' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA Side Info Profile (Send)
================================================================================
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ mksnaobj -t side_info -L RA60015B -P USIBMRA.RA39TCF1 -m NVDMNORM
-d 21F0F0F8 -h yes NVDMSIDR
0105-0031 Profile type 'side_info' name 'NVDMSIDR' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
+ chsnaobj -t side_info -L RA60015B -P USIBMRA.RA39TCF1 -m NVDMNORM
```

Figure 111 (Part 4 of 5). netlog3 Log File

```
-d 21F0F0F8 -h yes NVDMSIDR
Profile type 'side_info' name 'NVDMSIDS' CHANGED.
==============================================================================
NVDM CONFIG : Configuring SNA Side Info Profile (Receive)
==============================================================================
==============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
==============================================================================
Profile type 'side_info' name 'NVDMSIDR' CHANGED.
==============================================================================
NVDM CONFIG : Updating SNA Server...

verifysna command OK.
The profiles listed above have been dynamically updated successfully.
NVDM CONFIG : Configuring TCP/IP connection
NVDM CONFIG : Configuring SNA/DS connection configuration file.
NVDM CONFIG : (TCP/IP) for remote Server RS60007.
NVDM CONFIG : Configuring APPC connection
NVDM CONFIG : Configuring SNA/DS connection configuration file
/usr/lpp/netviewdm/db/snads_conn/RA39TCF1
NVDM CONFIG : Configuring SNA/DS routing table.
NVDM CONFIG : System has TCP/IP connection to remote server.
NVDM CONFIG : System has APPC connection to remote server.
NVDM CONFIG : Writing routing table to /usr/lpp/netviewdm/db/routetab
NVDM CONFIG : Saving target history for rs6000xx
NVDM CONFIG : Deleting Target rs6000xx from Server rs600015 configuration.
NVDM CONFIG : Defining Target rs600015 on server rs600015
NVDM CONFIG : Target already exists. Updating...
nvdm updtg rs600015 -s 'RS600015' -y 'AIX' -d 'ITSO Raleigh test server'
-q 'Stefan Uelpenich' -o 'Wolfgang Geiger' -t '4711' -r 'IBM'
WARNING: The Network ID of this domain has been changed to RS600015.
NVDM CONFIG : Adding Target Users...
NVDM CONFIG : Adding root User
NVDM CONFIG : Adding suelpen User
NVDM CONFIG : Defining Target rs60004 on server rs600015
nvdm addtg rs60004 -s 'RS60004' -y 'AIX' -d 'ITSO Raleigh test client'
-q 'Stefan Uelpenich' -o 'Wolfgang Geiger' -t '4711' -r 'IBM'
NVDM CONFIG : Adding Target Users...
NVDM CONFIG : Adding root User
NVDM CONFIG : Configuring Target Groups for rs600015
NVDM CONFIG : Adding group Group2
NVDM CONFIG : Defining remote target for rs60007
NVDM CONFIG : Defining remote target for RA39TCF1
NVDM CONFIG : RA39TCF1 will be configured as focal point.
+ eval nvdm addtg RA39TCF1 -m report_to -s RA39TCF1 -n USIBMRA -d 'NVDM_MVS'
+ nvdm addtg RA39TCF1 -m report_to -s RA39TCF1 -n USIBMRA -d NVDM_MVS
0513-029 The sna Subsystem is already active.
Multiple instances are not supported.
NVDM CONFIG : Releasing NVDM SNA communications.
NVDM CONFIG : !!! Configuration of Server completed successfully !!!
```

Figure 111 (Part 5 of 5). netlog3 Log File

After the script has been run the NetView DM/6000 server on `rs600015` is ready to be used.

Note

We could also specify the command `config_nvdm` to run the configuration script as a post-install script for the change file that performs the pristine installation of `rs600015`.

Chapter 11. Migrating the Procedure to Software Distribution for AIX V3.1

In this chapter we discuss the implications of using Software Distribution for AIX V3.1 instead of NetView DM/6000 V1.2.

We describe how using the new version affects the configuration procedure that we have developed in this book and modify the procedure to work with the new version.

We will focus only on configuration matters and will not discuss the new features of Software Distribution for AIX V3.1, for example, the plan facility.

11.1 Configuration Matters

As far as configuration is concerned we have to determine all changes that have been made in Version 3.1 that affect the configuration procedure that we have developed in this book.

The configuration procedure configures the product either by using the command line interface or by directly writing to product configuration files. Therefore, we have to deal with two types of changes:

- Changes in the command line interface
- Changes in configuration files

Changes in the command line interface include:

- New commands
- New functionality within commands
- New command line options

Changes in configuration files include:

- New configuration file names
- Modified configuration file structure

We will show all configuration procedures that need to be modified and explain the changes.

We will use the following commands that are new in version 3.1 or have been modified compared to previous versions:

- `nvdm addusr` to add users to Software Distribution for AIX
- `nvdm lsusr` to list users defined in Software Distribution for AIX
- `nvdm updusr` to update user definitions in Software Distribution for AIX
- `nvdm delusr` to delete users from Software Distribution for AIX
- `nvdm lsrq` to determine requests for a target

- `nvdm prgq` to purge a local queue for a target
- `nvdm delrq` and `nvdm eraserq` to delete requests
- `nvdm rentg` to rename a target
- `nvdm lsgp` to list target groups

11.1.1 Adding NetView DM Users to AIX

In order to enable an AIX user to use NetView DM, the user has to be defined twice:

1. The user has to be defined as a user of NetView DM/6000.
2. The user has to be defined to the AIX operating system itself.

> **Note**
>
> Whereas NetView DM/6000 V1.2 user authorization was implemented using AIX user groups, Software Distribution for AIX uses user authorization profiles to control access to Software Distribution for AIX User authorization profiles are implemented within the Software Distribution for AIX product and not in the AIX operating system.

The different user concepts will affect the configuration steps we have to perform for the different versions of the product.

The procedures to configure users in NetView DM/6000 V1.2 are described in 4.5, "Adding NetView DM/6000 Users to AIX" on page 39 and 4.9, "Configuring Local Targets" on page 68. The `add_users_aix` procedure is used to add users to AIX and assign them to the appropriate user groups and the procedure `nvdm_configure_targets` is used to assign users to targets.

In Software Distribution for AIX V3.1 we have to deal with the following major changes compared to V1.2:

- We do not need to assign AIX users to AIX users groups, but have to assign them the appropriate user authorization profile that is defined in Software Distribution for AIX.
- We cannot define users by updating the target information, because users are not contained in the target definition anymore. Instead we have to explicitly create Software Distribution for AIX and assign them to their targets.

We will use the `nvdm_users` object class to determine which users have to be created for which target. Unlike with Version 1.2, the `usergroup` field does not represent an AIX user group but the name of an authorization profile.

However, we can use the same data model that is used for previous versions of NetView DM.

With Version 3.1 there is a new command `nvdm addprf` to create user profiles. However, there are three standard authorization profiles delivered with the product that match the user groups previously used in Version 1.2. These are named exactly like the user groups used before:

- FNDUSER

- FNDBLD
- FNDADMN

We will only supply the code to configure the above standard profiles within this example, so you are only allowed to specify these values in the `usergroup` field.

However, if you want to be able to support custom authorization profiles you will have to add a new procedure that is able to add user authorization profiles. This procedure can be fed with values from a new object class that contains information about the custom user authorization profiles to be created.

In order to be able to reconfigure user definitions, we will first supply a procedure to delete the existing user profiles on a server. This procedure is called before the users are configured thus allowing to remove users that are not in the configuration database anymore.

The following figure shows the procedure:

```
#
# delete all users currently defined on that server
# the root user profile cannot be deleted
#
nvdm_delete_users ()
{
 #
 # determine all users that are defined on this server
 #
 USRLIST=`nvdm lsusr '*' | grep "User:" | cut -d':' -f2`
 for i in $USRLIST
 do
   if [ "$i" != "root" ]
   then
     print "NVDM CONFIG : Deleting existing user profile : $i"
     nvdm delusr $i -f
   fi
 done
}
```

Figure 112. nvdm_delete_users Shell Procedure (for Version 3.1)

The procedure `add_users_aix` will be used to add users to AIX as well as to Software Distribution for AIX. When called with target as the second command line argument, the procedure will only add the defined users to the AIX operating system in case they do not exist yet.

When called with server as the second command line argument, the procedure will create a new Software Distribution for AIX user. If this user already exists, the user definition is updated.

On a server, the `add_users_aix` procedure is called for each target defined for that server, thus defining all users for all targets.

The following figure shows the procedure:

```
#
# add user at OS level (AIX)
# $1 = IP Hostname
# $2 = Type: either "server" or "target"
#            use "target", when you want to add a user to AIX
#            add a target workstation; the user will always be
#            assigned group FNDADMN
#            use "server", when you want to add a user to AIX
#            add a server workstation; the user will be assigned
#            the appropriate usergroup defined in the database
#

add_users_aix ()
{
print "NVDM CONFIG : --> Adding AIX users for NVDM..."
get_attribute_list nvdm_users node_name $1 username
if [ $VALUE_NUM != 0 ]
then
  for i in $VALUE_LIST
  do
    #
    # First, add NVDM user to operating system...
    # check if user exists
    #
    lsuser $i 2>/dev/null 1>&2
    #
    # if not (RC 2 from lsuser command)
    #
    if [ $? = 2 ]
    then
      print "NVDM CONFIG : Adding user $i to AIX OS."
      mkuser $i
    fi
    #
    # only continue, if we are about to configure a server
    #
    if [ "$2" = "server" ]
    then
      get_attribute_and nvdm_users node_name $1 username $i usergroup
      GRP=$VALUE
      print "NVDM CONFIG : Authorization profile $GRP assigned to $i."
      nvdm lsusr $i 2>/dev/null
      #
      # if RC != 0 then user does not exist yet
      #
      if [ $? -ne 0 ]
      then
        nvdm addusr $i $GRP -t $1
```

Figure 113 (Part 1 of 2). add_users_aix Shell Procedure (for Version 3.1)

```
      else
        nvdm updusr $i $GRP -t $1
      fi
    fi
  done
fi
}
```

Figure 113 (Part 2 of 2). add_users_aix Shell Procedure (for Version 3.1)

11.1.2 Configuring SNA/DS Connection Profiles

We have to deal with the following changes as far as SNA/DS connection files are concerned:

- The name of the SNA/DS connection profile directory has changed.
- There are additional parameters available in an SNA/DS connection file.

Whereas in Version 1.2 the SNA/DS connection profiles are located in the `/usr/lpp/netviewdm/db/snads_conn` directory, the directory name in Version 3.1 is `/usr/lpp/netviewdm/db/snadscon`.

In the configuration script the directory name is held in the shell variable `SNA_DS_DIR` which is set at the beginning of the configuration script `config_nvdm`.

In Software Distribution for AIX Version 3.1 there is a new type of connection called server-to-server (STS) connection. Therefore there is a new field `TYPE` in each connection profile that determines the type of connection to be used (either SNA or STS).

Note

In the `TYPE` field SNA means that this is a normal SNA/DS connection and not an STS connection. It does not mean that the `PROTOCOL` has to be APPC. You can, of course, have `PROTOCOL` set to TCP/IP and `TYPE` set to SNA meaning that you configure an SNA/DS connection using TCP/IP.

We will slightly modify the procedures in 4.7, "Configuring SNA/DS Connection Profiles" on page 61 to reflect these changes.

The procedures `configure_sna_ds_appc` and `configure_sna_ds_tcpip` will both be modified to add the line

```
TYPE:          SNA
```

to each connection profile. In fact, this it not even necessary, since SNA is the default value for the `TYPE` field. However, we include it for ease of maintenance.

Also, we will add a check for the partner location to both procedures. If the short name for the partner cannot be determined, the procedures will not create the connection profile.

The following figure shows the `configure_sna_ds_appc` procedure:

```
#
# Configure SNA/DS connection configuration file (APPC)
#

configure_sna_ds_appc ()
{
 print "NVDM CONFIG : Configuring SNA/DS connection\
 configuration file $SNA_DS_DIR/$PARTNER"

 if [ "$PARTNER" = "" ]
 then
   print "NVDM CONFIG ERROR : APPC Partner LU not defined."
   print "NVDM CONFIG ERROR : Cannot create SNA/DS connection profile.
   return 1
 fi

 echo "PROTOCOL:              APPC
TYPE:                             SNA
SEND TP SYMBOLIC DESTINATION:     $SIDS
RECEIVE TP SYMBOLIC DESTINATION:  $SIDR
NEXT DSU:                         $SNA_NET.$PARTNER
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" > $SNA_DS_DIR/$PARTNER
}
```

Figure 114. configure_sna_ds_appc Shell Procedure (for Version 3.1)

The following figure shows the `configure_sna_ds_tcpip` procedure:

```
#
# Configure SNA/DS connection configuration file (TCP/IP)
# $1 = TCP/IP Hostname of remote system
#

configure_sna_ds_tcpip ()
{
 #
 # get short name of remote server
 #

 get_attribute nvdm_node node_name $1 short_name
 A=$VALUE
 print "NVDM CONFIG : Configuring SNA/DS connection configuration file."
 print "NVDM CONFIG : (TCP/IP) for remote Server $1."

 if [ "$A" = "" ]
 then
   print "NVDM CONFIG ERROR :  Could not determine short name for $1."
   print "NVDM CONFIG ERROR :  Please update nvdm_node class."
   return
 fi

 echo "PROTOCOL:                          TCP/IP
TYPE:                             SNA
REMOTE SERVER NAME:               $1
TCP/IP TIME-OUT:                  300
NEXT DSU:                         $A.$A
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" >$SNA_DS_DIR/$A
}
```

Figure 115. configure_sna_ds_tcpip Shell Procedure (for Version 3.1)

Note

The procedures to create connection files cannot be used to configure server-to-server (STS) connections which are a new feature in Version 3.1. However, you can adapt the procedures to support this feature.

11.1.3 Configuring Local Targets

The steps necessary to configure local targets are described in 4.9, “Configuring Local Targets” on page 68.

In order to work with Version 3.1, the procedures that are needed to configure local targets have to be adapted.

The first procedure `nvdm_delete_targets` is used to delete all local targets defined for a Software Distribution for AIX server.

Since SD4AIX. V3.1 maintains a local queue for each local target, the product will normally refuse to remove a target from the server configuration if there are still pending requests.

Therefore we first have to purge the queue used for the local target we want to remove by using the `nvdm prgq` command. However, we want to save information about all requests that are still in the queue. For that purpose the script will create a log file containing all requests currently in the queue, before the queue is purged and the target is deleted.

To be sure that all pending requests for a target are deleted, we use the `nvdm delrq` and `nvdm eraserq` commands to remove any single pending requests.

However, the `nvdm eraserq` command will sometimes refuse to erase a request the first time is called. When the command is called again, the request will be marked as Pending, delete requested and can be erased. Therefore we will call the `nvdm delrq` and `nvdm eraserq` commands twice.

The following figure shows the `nvdm_delete_targets` shell procedure:

```
#
# delete local targets from NVDM Server configuration
# $1 = Server IP Hostname
#

nvdm_delete_targets()
{
 #
 # get list of existing targets
 #

 TLIST=`nvdm lstg '*' | grep "Target:" | cut -d':' -f2`

 #
 # get list of all defined targets for this server
 #

 get_attribute_list nvdm_node server_name $1 node_name
 YLIST=$VALUE_LIST
 XLIST=""
 for i in $YLIST
 do
  XLIST=$XLIST" "`echo $i | cut -d'.' -f1`
 done

 #
 # delete all targets which are not defined for this server
 #

 for i in $TLIST
 do
  match=0
  for x in $XLIST
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    nvdm_save_history $i
    print "NVDM CONFIG : Deleting Target $i from Server $1 configuration."
    #
    # before a target can be deleted, we have to
    # discard all pending requests
    #
    PEND=`nvdm lsrq -w $i | grep "Request ID:" | cut -d':' -f2 | \
```

Figure 116 (Part 1 of 2). nvdm_delete_targets Shell Procedure (for Version 3.1)

```
          awk '{ print $3 }'`
    if [ "$PEND" != "" ]
    then
      print "NVDM CONFIG : Requests IDs $PEND for $i will be deleted."
      print "NVDM CONFIG : Information about pending requests for"
      print "NVDM CONFIG : Target $i will be written to $i.req"
      echo "The following requests were purged:" >$i.req
      for x in $PEND
      do
        nvdm lsrq -l $x >>$i.req
      done
    fi
    nvdm hldq $i
    nvdm prgq $i -f
    for x in $PEND
    do
      echo "y" >/tmp/yes
      nvdm delrq $x -f
      nvdm eraserq $x </tmp/yes
      sleep 2
      nvdm delrq $x -f
      nvdm eraserq $x </tmp/yes
    done
    nvdm deltg $i -f
  fi
 done
}
```

Figure 116 (Part 2 of 2). nvdm_delete_targets Shell Procedure (for Version 3.1)

The following figure shows an example of a log file that was created by `nvdm_delete_targets`. This file contains all requests for the deleted target.

```
The following requests were purged:
   Request ID:              rs600011 root 12 0
   SNA correlator:          rs600011 09/01/95 53257
   Submission time:         09/01/95 10:37:19
   Request type:            Ret inv
   Object:
   Status:                  Successful
   Error severity:          0
   Schedule time:           09/01/95 10:37:19
   Starting mode:           Released
   Priority:                No
   Application ID:          CLI
   Execution window:
    Execution time :        When received by target
    Expiration time:        Undefined
   Time Format     :        Local time at origin
   Termination target exit:
   Termination exit:

   Request ID:              rs600011 root 14 0
   SNA correlator:          rs600011 09/01/95 53259
   Submission time:         09/01/95 10:54:26
   Request type:            Install
   Object:                  HISTORY.REF.1
   Status:                  Successful
   Error severity:          0
   Schedule time:           09/01/95 10:54:26
   Starting mode:           Released
   Priority:                No
   Application ID:          CLI
   Execution window:
    Execution time :        When received by target
    Expiration time:        Undefined
   Time Format     :        Local time at origin
   Termination target exit:
   Termination exit:
```

Figure 117. rs600016.req Request Log File

The `nvdm_save_history` shell procedure has to be slightly modified to be compliant with the new file format for software inventory files in Version 3.1.

In Version 1.2 and previous releases the name of the change file to appear in the target software inventory had to be specified using the `PRODUCT` keyword in the software inventory file `fndswinv`.

The keyword was followed by the global name of the change file to be cataloged. In Version 3.1 the `GLOBAL NAME` keyword is used to specify change files in the software inventory file.

In order to work with Version 3.1, we change the `nvdm_save_history` procedure to be compliant with this change.

Further, we have extended the time the procedure waits for the install request for the inventory file to be completed to 15 seconds. You might need to adjust this value to your own environment.

The following figure shows the `nvdm_save_history` shell procedure:

```
#
# Save NVDM target history by creating software inventory
# file and copying it to corresponding node
# requires /.rhosts file on target
# $1 = target name
#
nvdm_save_history ()
{
 print "NVDM CONFIG : Saving target history for $1"
 #nvdm inv
 SLIST="`nvdm lscm -w $1 '*' | grep 'Global file name:' | cut -d':' -f2`"
 >/tmp/inv
 if [ "$SLIST" != "" ]
 then
   for o in $SLIST
   do
     print "NVDM CONFIG : Adding $o to software inventory file."
     print "GLOBAL NAME: "$o >>/tmp/inv
     print "DESCRIPTION: Target has been moved!" >>/tmp/inv
   done
   print "NVDM CONFIG : Copying inventory file $SW_INV to $1."
   echo "GLOBAL NAME:                   HISTORY.REF.1
CHANGE FILE TYPE:              GEN
COMPRESSION TYPE:              LZW
REBOOT REQUIRED:               NO
PACK FILES:                    NO
SECURE PACKAGE:                NO
OBJECT:
SOURCE NAME:               /tmp/inv
TARGET NAME:               /usr/lpp/netviewdm/fndswinv
TYPE:                      FILE
ACTION:                    COPY
INCLUDE SUBDIRS:           NO" >/tmp/hist.pro
   nvdm delcm HISTORY.REF.1 -w '*'
   nvdm uncat HISTORY.REF.1 -d -f
   nvdm bld /tmp/hist.pro -f
   nvdm inst HISTORY.REF.1 -w $1 -f -i
#
# we will sleep here for 15 secs to allow
# the CF to be sent to the target before
# it is deleted. You might need to adjust
# this value, especially if you are, for example,
# in a WAN environment
#
 fi
 print "NVDM CONFIG : Sleeping for 15 secs."
 sleep 15
}
```

Figure 118. nvdm_save_history Shell Procedure (for Version 3.1)

The `nvdm_configure_targets` procedure has to be modified to be compliant with the following changes:

- Users are not defined in the target definition anymore.
- There is a new flag `-b` available with the `nvdm addtg` command.

Since users are defined using the `nvdm addusr` command in Version 3.1, the code to define users in the target definition must be removed from the `nvdm_configure_targets` shell procedure.

The flag `-b` that has been added to the `nvdm addtg` defines the target type. We will set this flag to client for the agents to be defined.

> **Note**
>
> You must not set the `-b` flag when updating the target definition for a server.

On a Software Distribution for AIX server a target definition is automatically created for the server itself during installation. This target definition is called the Initial Target Record.

We will have to use the `nvdm updtg` command when updating the definitions for that target. However, the product will not allow to change the target type for the server itself, so if we specify the `nvdm updtg` command with the flag `-b server` the update will fail, regardless of the fact that this is the same type which is already defined. Hence, we will not use the `-b` flag, when configuring the target definition for the server.

> **Note**
>
> The return codes for the `nvdm lstg` command have changed in Version 3.1.

In Version 1.2 the command `nvdm lstg` targetname produced a return code that was not `0` in the `$?` shell variable if the target did not exist.

In Version 3.1 this command will produce a return code of `0` no matter if the target exists or not.

We did use the return code to check if the target already existed. Now we have to check for the message `FNDCL129E` that is returned by the command if the target does not exist.

The following figure shows the `nvdm_configure_targets` shell procedure:

```
#
# configure Targets for an NVDM/6000 Server
# $1 = Server IP Hostname
#

nvdm_configure_targets ()
{
 #
 # First, determine all Nodes which have this Server
 # defined as their NVDM/6000 server
 #

 # access database

 get_attribute_list nvdm_node server_name $1 node_name
 ATLIST=$VALUE_LIST
 TLIST=""
 for i in $ATLIST
 do
  TLIST=$TLIST" "`echo $i | cut -d'.' -f1`
 done

 count=0
 for i in $TLIST
 do
   count=`expr $count + 1`
   print "NVDM CONFIG : Defining Target $i on server $1"
   A=`nvdm lstg $i 2>&1 | grep FNDCL129E`
   #
   # if FNDCL129E not found then target exists already
   #

   if [ "$A" != "" ]
   then
     COMMAND="nvdm addtg $i"
   else
     COMMAND="nvdm updtg $i"
     print "NVDM CONFIG : Target already exists. Updating..."
   fi

   #
   # get required target attributes
   #

   huhn=`echo $ATLIST | cut -d' ' -f$count`

   for a in short_name target_os description contact_name\
 owning_manager telephone_number customer_name
```

Figure 119 (Part 1 of 2). nvdm_configure_targets Shell Procedure (for Version 3.1)

```
  do
    get_attribute nvdm_node node_name $huhn $a
    v=$VALUE
    if [ "$v" != "" ]
    then
      case $a in
        short_name)     COMMAND=$COMMAND" -s '$v'" ;;
        target_os)      COMMAND=$COMMAND" -y '$v'" ;;
        description)    COMMAND=$COMMAND" -d '$v'" ;;
        contact_name)   COMMAND=$COMMAND" -q '$v'" ;;
        owning_manager) COMMAND=$COMMAND" -o '$v'" ;;
        telephone_number) COMMAND=$COMMAND" -t '$v'" ;;
        customer_name)    COMMAND=$COMMAND" -r '$v'" ;;
      esac
    fi
  done

  if [ "$i" != "$1" ]
  then
    COMMAND=$COMMAND" -b client"
  fi
  echo $COMMAND
  eval $COMMAND
 done
}
```

Figure 119 (Part 2 of 2). nvdm_configure_targets Shell Procedure (for Version 3.1)

11.1.4 Configuring Remote Targets

In Version 3.1 the flags that can be used with the `nvdm addtg` command have slightly changed as far as the configuration of remote targets is concerned.

The following changes will affect the configuration procedure `nvdm_remote_targets`:

- We have to specify the type of the remote target using the `-b` flag. Since we only allow to configure other servers as remote targets we will always specify `-b server` with the command.
- The parameters available with the `-m` flag have changed. Instead of the type report_to we will have to use focal to configure the focal point system. The type remote is no longer existent, so we will use the default (Push) for other RS/6000 servers connected trough TCP/IP.
- For remote connections that use APPC we have to specify `-tp appc:` to define APPC as the protocol to be used for the remote connection.

Note

You might need to extend the nvdm_remote_targets procedure to support other features. For example, you can enhance the procedure to also allow the configuration of remote agents. By default, we configure remote RS/6000 servers as Push mode targets. In case you want to configure the remote system as a Remote Administrator (RA), you will have to define this system as a manager.

In both cases you will also have to modify the data model.

The following figure shows the nvdm_remote_targets shell procedure:

```
#
# configure Remote Targets
# $1 = IP Hostname
#

nvdm_remote_targets ()
{
 #
 # First, get all remote targets defined for this server
 # Remote Targets are determined by searching the nvdm_queues
 # class because any connection to a remote system requires a
 # queue

 get_attribute_list nvdm_queues node_name $1 remote_server

 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No remote targets defined"
   return
 fi

 for i in $VALUE_LIST
 do
   print "NVDM CONFIG : Defining remote target for $i"

   #
   # determine if system to be configured is a Remote Target or
   # a Focal Point
   #
   get_attribute_and nvdm_queues node_name $1 remote_server $i focal_point

   if [ "$VALUE" = "yes" ]
   then
     print "NVDM CONFIG : $i will be configured as focal point."
     # for the MVS focal point short name will be the same as node name
     # network id will be the SNA Network Name

set -x
     eval nvdm addtg $i -m focal -b server -s $i -n $SNA_NET \
 -d "'NVDM_MVS'" -tp appc:
set +x
   else
     # get short name for remote server from class nvdm_node
     get_attribute nvdm_node node_name $i short_name
     if [ "$VALUE" = "" ]
     then
       abort "No Short Name defined for $i in class nvdm_node. Exiting..."
     fi
```

Figure 120 (Part 1 of 2). nvdm_remote_targets Shell Procedure (for Version 3.1)

```
      RSHORT=$VALUE
      #
      # This remote server is assumed to be connected via TCP/IP
      # so, we set the network name to be the same as the short name
      #
      nvdm addtg $i -s $RSHORT -n $RSHORT -b server
    fi
  done
}
```

Figure 120 (Part 2 of 2). nvdm_remote_targets Shell Procedure (for Version 3.1)

11.1.5 Configuring Target Groups

There is a change in the output format of the `nvdm lsgp` command, which lists all target groups configured on a CC server. The following figures show the output format for both versions:

NetView DM/6000 Version 1.2:

```
nvdm lsgp '*'
```

```
Group                           Mode      Description

Group1                          Push
Group2                          Push
```

Figure 121. Output from nvdm lsgp (NetView DM/6000 V1.2)

Software Distribution for AIX Version 3.1:

```
nvdm lsgp '*'
```

```
Group:                 Group1
Mode:                  Push
Description:

Group:                 Group2
Mode:                  Push
Description:
```

Figure 122. Output from nvdm lsgp (Software Distribution for AIX V3.1)

As we use the command `nvdm lsgp` in the shell procedure `nvdm_delete_groups` when configuring target groups, we must modify the procedure to reflect the change.

The described output format difference affects the part of `nvdm_delete_groups` where we process the output from the above command in order to determine the existing target groups on the server. The following figure depicts the modification of the procedure due to the format change of the command `nvdm lsgp`:

```
#
# Delete all existing groups before adding groups from
# configuration database
# $1 = IP Hostname of server to be configured
#

nvdm_delete_groups ()
{
 #
 # determine existing groups
 #
 GP=`nvdm lsgp '*' | grep "Group:" | cut -c24-`
 #
 # determine list of defined groups
 #
 get_attribute_list nvdm_groups node_name $1 group_name
 XGP=$VALUE_LIST

 for i in $GP
 do
  match=0
  for x in $XGP
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    print "NVDM CONFIG : Deleting group $i from $1 configuration."
    nvdm delgp $i -f
  fi
 done
}
```

Figure 123. nvdm_delete_groups Shell Procedure (for Version 3.1)

11.1.6 Restarting Software Distribution for AIX

After a server has been configured it needs to be restarted. The shell procedure `restart_nvdm` is used for that purpose.

In order to detect whether the server is running we use the `nvdm stat` command. By examining the return code of this command, we can tell whether the server is running.

Whereas, in NetView DM/6000 V1.2 a return code of `121` indicates that the server is not running, in Software Distribution for AIX V3.1 the corresponding return code is `218`.

The following figure shows the shell procedure for Version 3.1:

```
restart_nvdm ()
{
 print "NVDM CONFIG : --> In order for the changes to become active"
 print "NVDM CONFIG :     NetView DM/6000 will be restarted on this node"

 #
 # determine if nvdm is running
 #

 nvdm stat 1>/dev/null 2>&1

 if [ $? = 218 ]
 then
   print "NVDM CONFIG : NVDM is not running. It will be started now."
   nvdm start
   nvdm start
 else
   print "NVDM CONFIG : Stopping NVDM."
   nvdm stop -x 1>/dev/null 2>&1
   s=1
   print "NVDM CONFIG : Restarting NVDM."
   while [ $s = 1 ]
   do
     print "NVDM CONFIG : Restarting NVDM."
     nvdm start
     nvdm stat
     if [ $? != 218 ]
     then
       s=0
     fi
   done
 fi
}
```

Figure 124. restart_nvdm Shell Procedure (for Version 3.1)

Warning

Starting the server might fail if you have defined APPC connections in your database without having SNA Server installed correctly. If SNA Server is not installed in the correct version or not installed at all, the Software Distribution for AIX server will fail to start if there are APPC connections defined.

11.1.7 Updating Server Information

When configuring a server we have found that changing the hostname of a machine can heavily influence the configuration of NetView DM. Therefore we have to be careful when reconfiguring a Software Distribution for AIX server that has already been configured with a different hostname.

For that purpose we added some additional code to the configuration script in Figure 109 on page 188 that was imbedded in the `nvdm_update_server` procedure in the configuration script.

For Version 1.2 this procedure creates an initial target record for the server in the `/usr/lpp/netviewdm/db/target_config` directory. For Version 3.1 this is not possible because of the following reasons:

- The storage method for target information has changed. Whereas in Version 1.2 there is one file for each target in the above mentioned directory, in Version 3.1 information about all targets is held in `/usr/lpp/netviewdm/db/trgcfg`.
- The initial target configuration is held in a binary file in Version 3.1 and not in an ASCII file as in Version 1.2. Therefore, it cannot be simply created by a shell script.

Instead we will use the `nvdm rentg` to rename the initial target record if necessary.

The following figure shows the shell procedure:

```
#
# update NVDM/6000 server definition
#

nvdm_update_server ()
{
#-------------change--------------
# if we configure a server and want to change the
# WORKSTATION NAME we must stop the server before
# reconfiguring this field.
# To do so we must be sure that the hostname and
# the WORKSTATION NAME match
#
if [ "$NODE_TYPE" = "0" -o "$NODE_TYPE" = "2" ]
then
  # get current hostname
  OHN=`hostname`
  print "NVDM CONFIG : Current hostname of server is $OHN."
  # get WORKSTATION NAME currently configured
  HN=`grep "WORKSTATION NAME:" $CONFIG | cut -d':' -f2`
  print "NVDM CONFIG : Current WORKSTATION NAME of server is $HN."
  print "NVDM CONFIG : Stopping Server..."
  # make sure that both match
  hostname $HN
  nvdm stop -x
  print "NVDM CONFIG : Sleeping 20 seconds..."
  sleep 20
  # set back hostname
  print "NVDM CONFIG : Setting hostname to $OHN."
  hostname $OHN
  # also, we must be sure that there is an initial target record for
  # the servername configured
  # we rename the initial target record to the new hostname
  nvdm rentg $HN $OHN -f
fi
#-----------end-of-change---------
}
```

Figure 125. nvdm_update_server Shell Procedure (for Version 3.1)

Chapter 12. Implementing the Configuration Data Model Using DB2/6000

In order to show the usability of the automatic configuration procedure for NetView DM/6000 in different database environments we implement the ODM data model from Chapter 3, “Designing a Data Model for Configuration Data” on page 11 using IBM Database 2 AIX/6000 (DB2/6000) in this chapter.

This direct porting might somehow seem awkward from the general viewpoint of SQL, which offers powerful means for data definition and manipulation. At the same time our configuration procedure uses only three quite primitive access procedures (see 4.2, “Database Access Procedures” on page 31). They define a clear database-independent interface, so the storing method of the configuration data can be exchanged transparently for the configuration activities of NetView DM/6000. This is what we demonstrate in this chapter.

This chapter is intended for system administrators who want to use DB2/6000 instead of ODM for keeping the configuration database. First, we present the basic advantages of DB2/6000 that give preference to the use of DB2/6000 over ODM in change and distribution management. With the intention of making your DB2/6000 configuration task easier, we point out the important steps of building your DB2/6000 server and clients in a Transmission Control Protocol/Internet Protocol (TCP/IP) network environment. Following the ODM data model we show the implementation and automatic creation of the configuration database as well as the appropriate database access procedures. We also provide an idea of how to design a data model that makes much better use of the DB2/6000 features regarding data integrity.

For a good understanding of this chapter we assume the reader has a basic knowledge of relational database concepts and some background in Structured Query Language (SQL).

12.1 Advantages of DB2/6000 over ODM

DB2/6000 has the following advantages over ODM concerning distribution and change management:

- DB2/6000 contains more powerful data definition methods such as indexing and enforcing referential integrity, defining package dependencies (for example, actions on the dependent object after deleting the parent object).
- DB2/6000 offers a big variety of data access methods: the full SQL apparatus of predicates, nested queries, views, joins, etc.
- The authorization mechanism of DB2/6000 gives you the possibility of defining access rights to users without regarding AIX authorities.
- The client/server approach of DB2/6000 ensures the transparent remote access to the database residing on the configuration server concurrently by several NetView DM/6000 targets, the necessity of distributing the whole configuration database in the CC domain no longer exists.

- The IBM Database 2 family of products for the IBM RISC System/6000 includes additional client support for OS/2 and DOS platforms, which enhances the capability for automatic installation and configuration of non-AIX targets.
- Moreover, the import and export utilities of DB2/6000 let you move data between a DB2/6000 node and DRDA-compliant databases (Distributed Relational Database Architecture). This allows the possibility of supplying configuration data to NetView DM/6000 agents that are not supported as DB2/6000 clients.
- Another possibility offered by DB2/6000 is it enables applications running on DOS, Windows, OS/2 and AIX workstations to access and update data on DRDA-compliant host database management systems like MVS, OS/400, VM and VSE. For that purpose you have to install IBM AIX Distributed Database Connection Services/6000 of the DB2/6000 product family.

12.2 General Steps in Installing and Configuring DB2/6000

If you have not installed DB2/6000 on your machine yet, the following sections provide instructions on how to perform the installation and configuration of the product, both on the server and on the clients. The description is oriented on our task to enable the machines in our network environment to access the configuration database residing on the configuration server. For more details, see the *DATABASE 2 AIX/6000 Installation Guide*, GC09-1570.

Note

We are using DB2/6000 Version 1.2 in this scenario.

12.2.1 The Overall Picture

Before starting the description of the installation and configuration steps for DB2/6000, we present a general overview of the structure of a DB2/6000 database network environment. Figure 126 on page 221 illustrates the principle of using DB2/6000 in client/server mode.

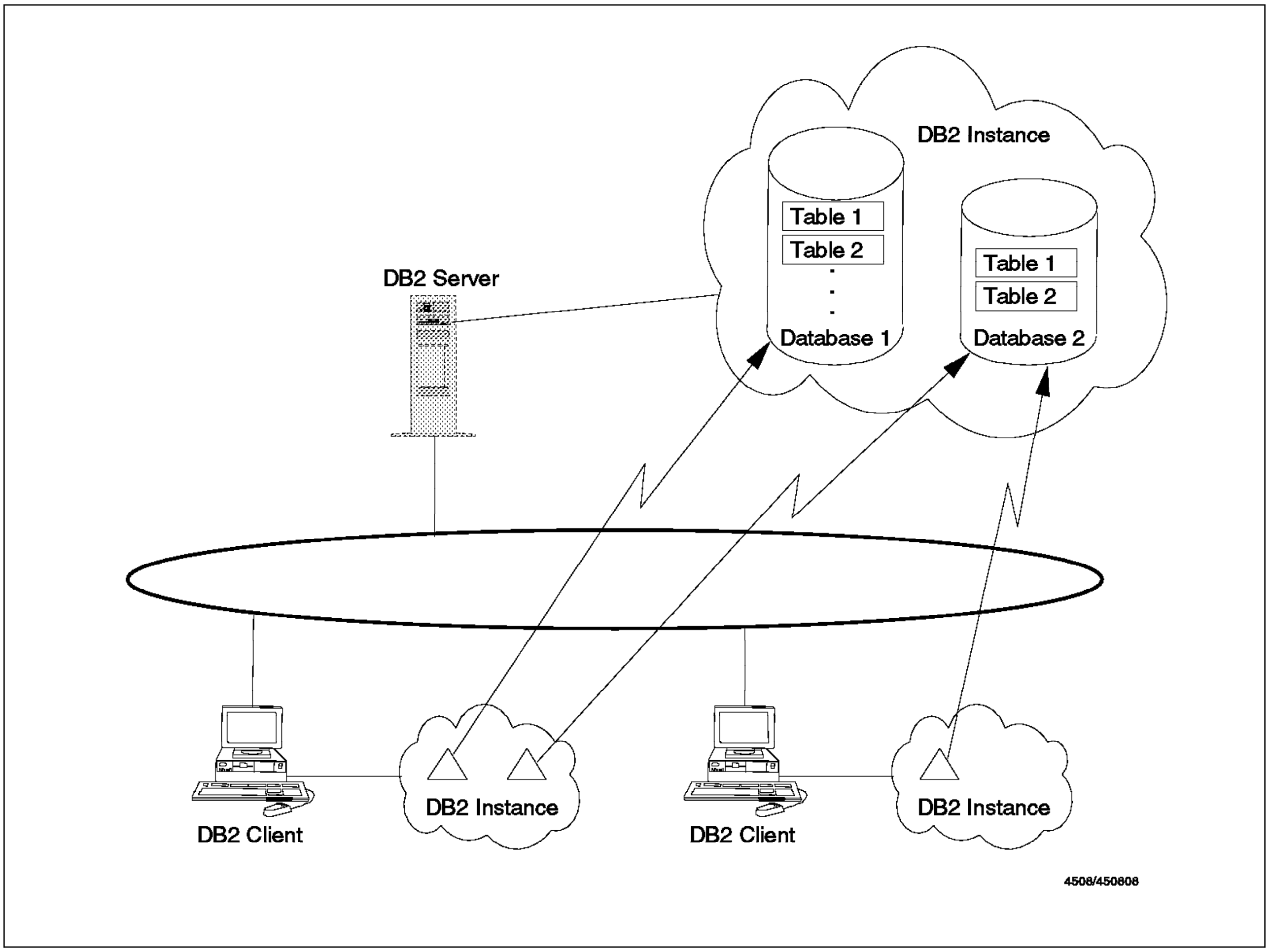


Figure 126. DB2/6000 Overview in a Network Environment

On each machine participating in the database network environment resides an *instance* of the database management system. The instance is created by `root` both on the database server and on the database clients. One instance manages several databases that contain many *tables*. On their part the tables consist of *rows* and *columns* as usual for relational databases.

Note

There may be more than one instance on a workstation, which is sensible (for example, in the case of running production and development in parallel). To avoid unnecessary confusion we omit this aspect because it leaves the scope of this book.

Physically the databases reside on the server. From each client instance exists a link to the desired database located on the server. This enables users on the clients to access databases transparently across the network.

There are three categories of database users:

instance owner

This is the database management system administrator (`sysadm`) that the DB2/6000 instance is assigned to. This assignment is done when `root` creates the instance. The instance owner has the highest authority over all databases in its instance.

database administrator

The database administrator (`dbadm`) has exclusive rights over a single database. This authority is granted by `sysadm` and is valid only within the specified database.

general user

The general database user can perform actions as granted by `sysadm` or by `dbadm` of the database.

12.2.2 Installing DB2/6000 on the Target Machine

Log in as `root` and use `smitty` to transfer the packages from the distribution medium to the target machine.

On the designated database server, where your configuration database will reside, you must install the following products from the DB2/6000 AIX family:

- IBM Database 2 AIX/6000, which contains a full-function relational database management system for the AIX operating system with the capability of a local database server and a remote database client in a network environment
- IBM AIX Database 2 Client Support/6000 (DB2 Client Support/6000), which provides remote client support, enabling the database server to accept requests from remote clients as well as local clients over the Transmission Control Protocol/Internet Protocol (TCP/IP)

Note

Client support for communications between the database server and the database clients over the Advanced Program-to-Program Communications (APPC) protocol is enabled after installing the additional SNA Support Feature of the DB2 Client Support/6000. In this case it is assumed that you have before installed and configured additional software supporting the LU 6.2 protocol (for example, SNA Server/6000).

On every AIX database client machine you must install IBM AIX Database 2 Software Developer's Kit/6000. This product enables applications to run on remote clients and contains a full development environment for client workstations including *interactive SQL*, *embedded SQL* and the *Call Level Interface*.

Note

Installing the IBM AIX Database 2 Client Application Enabler/6000 is not sufficient for our task to send queries from the remote database clients to the server holding the configuration database because this product provides only runtime support for applications but *does not* allow you to use interactive SQL.

There is also support for OS/2 and DOS database clients. To enable such clients to communicate with the AIX database server you must install on such machines the appropriate product, respectively IBM Database 2 Software Developer's Kit/2 or IBM Database 2 Software Developer's Kit/DOS.

12.2.3 Common Actions for Server and Client

1. Create an instance of the product.

   You must have `root` authority when performing this step.

   - Create an AIX user group that will be the instance owner group:

     ```
     mkgroup dbsysadm
     ```

   - Create an AIX user ID that will be the instance owner, that is it will have the highest database priority `sysadm`:

     ```
     mkuser pgrp=dbsysadm groups=dbsysadm home=/home/dbmsadm dbmsadm
     passwd dbmsadm (set the password for dbmsadm)
     ```

     Note

     The designated primary group of the instance owner becomes automatically the group of the database system administrator (`sysadm`) while creating the instance. Make sure that the instance owner has the correct primary group before running the instance creating script `db2instance` (in our example `dbsysadm`). Otherwise there exists the danger of inadvertent authorization to `sysadm` of members of `staff`, for example, which is the default primary group of AIX users without administrator rights.

   - Execute the `db2instance` command:

     ```
     /usr/lpp/db2_01_01_0000/instance/db2instance dbmsadm
     ```

     This command creates a directory `$HOME/sqllib` for `dbmsadm` that represents the database instance assigned to the user and defines its environment as instance owner.

   - Set up the database environment (for Bourne shell and Korn shell).

     Log in as `dbmsadm`. Edit the file `$HOME/sqllib/db2profile` and change the appropriate entries to the following:

     ```
     DB2INSTANCE=dbmsadm
     PATH=${PATH}:/home/dbmsadm/sqllib/bin:/home/dbmsadm/sqllib/adm
     PATH=${PATH}:/home/dbmsadm/sqllib/misc
     DB2DBDFT=NVDM_CFG
     ```

     Note

     The DB2DBDFT variable contains the name of the default database (default value SAMPLE). NVDM_CFG will be the name of the configuration database.

     Edit `$HOME/.profile` and add the call of `db2profile`:

```
# DB2/6000 settings
 . ./sqllib/db2profile
```

> **Note**
>
> This will call the script `db2profile` that sets the correct AIX environment variables and extends the command search directories in the global variable `PATH`. The settings becomes effective after the next login or after the execution of `.profile` (`. ./.profile`).

2. Enter the license information.

   Before you can use any of the products in the DB2/6000 family you must enter the NetLS license passwords. See *DATABASE 2 AIX/6000 Installation Guide*, GC09-1570 for more instructions about obtaining and registering of the license information into the file `/usr/lib/netls/conf/nodelock`.

3. Execute the `db2ln` command.

   This step creates links for libraries and include files for a particular version and release of the product:

   ```
   /usr/lpp/db2_01_01_0000/cfg/db2ln
   ```

4. Configure DB2/6000 to communicate over TCP/IP.

   In order to provide communication support over TCP/IP you must first have installed and configured the Base Operating System Network Facilities (BOSNET) both on the server and the clients.

   The configuration of DB2/6000 over TCP/IP include the following:

   - Ensure name resolution between server and clients.

     Make sure that both, server and client machines, know each other's host name. To check whether the respective host name (for example, `rs600012`) can be resolved issue the following:

     ```
     host rs600012
     ```

     In the case of success you will get an output like the following:

     ```
     rs600012.itso.ral.ibm.com is 9.24.104.124
     ```

     If the `host` query fails then check whether you are using a Domain Name Server (DNS) or you are resolving host names locally. Add an entry into your local `/etc/hosts` or let the DNS administrator add it for you into the `/etc/hosts` of the DNS similar to the following:

     ```
     9.24.104.124       rs600012.itso.ral.ibm.com     rs600012
     ```

   - Define the DB2/6000 communication ports.

Application programs communicate in TCP/IP networks over ports. To enable the connection between the DB2/6000 server and its clients you must specify two adjacent ports to the TCP/IP subsystem designated for the DB2/6000 communications. The ports must match on both sides, server and clients.

Log in as `root` and edit `/etc/services` to add the following two lines:

```
db2nvdmc      3700/tcp       # DB2 main connection port
db2nvdmi      3701/tcp       # DB2 interrupt port
```

The entry `db2nvdmc` is the *service name* that is used later for the configuration of the database system manager both on the server and on the client sites.

12.2.4 Further Server Configuration

1. Configure database manager for TCP/IP.

   The *service name* associated with the main connection port is used by the database manager to identify the port it will listen to. To enter this information into the database manager configuration file, log in as `dbmsadm` and use the following command from the shell command line:

   ```
   db2 update database manager configuration using svcename db2nvdmc
   ```

   TCP/IP support is generally enabled after issuing the next `db2start` command from the Command Line Processor `db2`.

2. Create a new file system for the database.

   As the creation of a new database requires about 12 MB storage on the database server, we recommend holding the configuration database in a separate file system mounted under the home directory of the instance owner. You must have `root` authority to be able to do that, execute the following commands:

   ```
   mkdir /home/dbmsadm/databases
   crfs -v jfs -g rootvg -a size=40000 -m /home/dbmsadm/databases -A yes -p rw
   mount /home/dbmsadm/databases
   chown dbmsadm:dbsysadm /home/dbmsadm/databases
   ```

   Note

   Since you execute the above commands as `root`, do not forget to change the ownership (`chown`) of the created directory to the instance owner (`dbmsadm`). Otherwise, it cannot create the database directory where the configuration database will physically reside.

3. Add two additional AIX users (optional).

   In order to provide a secure database network environment you should create respectively catalog the configuration database with server authentication type (see 12.3.3, “Authentication Types and Security Considerations” on page 247).

For this reason you need to create the following two database users on the server to make the authentication:

dbcfgadm This database user has update rights for all tables of the configuration database. It corresponds to the NetView DM/6000 FNDADMN authority and is allowed to alter the NetView DM/6000 configuration by changing the data in the tables. The responsibility of changing the data model (that is, creating and dropping tables) or creating and dropping the whole database is reserved to the instance owner `dbmsadm`.

> **Note.**
>
> Even the instance owner `dbmsadm` is allowed to create and drop a database *only* locally on the server.

dbcfgusr This user is able to connect to the configuration database and select its tables. It corresponds to the NetView DM/6000 FNDBLD and FNDUSER privileges, that cannot change the configuration and are not allowed to perform administrative work.

Log in as `root` and execute the following commands to create the AIX users:

```
mkuser home=/home/dbcfgadm dbcfgadm
mkuser home=/home/dbcfgusr dbcfgusr
passwd dbcfgadm        (enter password for dbcfgadm)
passwd dbcfgusr        (enter password for dbcfgusr)
```

Edit the `.profile` of both users to add the following DB2/6000 lines similar to the change of `dbmsadm`'s `.profile`:

```
# DB2/6000 settings
 . /home/dbmsadm/sqllib/db2profile
```

12.2.5 Further Client Configuration

1. Configure database manager.

   To make the database server known to the client database manager, log in as `dbmsadm` and execute the following command from the shell command prompt:

   ```
   db2 catalog tcpip node rs12db remote rs600012 server db2nvdmc
   ```

   The arguments of the above command have the following meaning:

   `rs12db`: Name of the TCP/IP node used by the database manager when cataloging the database (see next step)

   `rs600012`: Host name of the remote database server machine

   > **Note.**
   >
   > If the database server is not in your TCP/IP domain (in the case of DNS) you must specify the fully qualified server name not just its alias, for example `rs600012.itso.ral.ibm.com`.

db2nvdmc**:** Service name bound to the designated main connection TCP/IP port

> **Note**
>
> There are two possible ways of entering commands to the Command Line Processor db2:
>
> - Call db2 from the shell command prompt and then enter SQL commands until you type quit or terminate. While the former exit leaves the connection to a database open, the latter closes it.
> - Call the SQL commands directly from the shell command prompt by prefixing them with db2. The connection to the database remains open (until you call db2 terminate). As the shell evaluation rules apply here, you can use variables and quote shell specific symbols (like *, " or ').
>
> As SQL is not case-sensitive, it is of no importance whether you use small or capital letters entering the commands for the DB2/6000 Command Line Processor. For the sake of uniformity we show SQL commands with small letters throughout this book.

2. Catalog the remote configuration database NVDM_CFG

   Now you must define the configuration database to the client as a remote database. You can do that using the catalog database command of the DB2/6000 Command Line Processor:

```
db2 'catalog database nvdm_cfg at node rs12db authentication client \
     with "NetView DM/6000 Configuration Database"'
```

> **Note**
>
> The authentication parameter specifies the user authentication type of DB2/6000. The default value is server. The cataloging of the database on the server is made implicitly when creating the database. The authentication method on both sides, server and client, must match to establish the connection between them. See 12.3.3, “Authentication Types and Security Considerations” on page 247 for the security considerations related to the authentication methods.

12.3 Depicting the Data Model for the Configuration Data in DB2/6000

Based on the data model from Chapter 3, “Designing a Data Model for Configuration Data” on page 11 we now present the structure of the NetView DM/6000 configuration database as defined by means of DB2/6000. First we describe the direct porting of the ODM data model to DB2/6000, so that the automatic configuration script is not affected at all. That means that you can run exactly the same script and either use ODM or DB2/6000 as your configuration database. In order to make the database access fully transparent for the configuration procedure, we do not exploit the means of relational design and implementation to a large extent.

At the end of this section we propose an improved data model that makes use of the advanced data definition techniques of DB2/6000 Structured Query Language (SQL). However, this data model requires appropriate changes in the configuration script, as the data there is restructured to better represent the referential integrity of the configuration data.

12.3.1 Porting of the ODM Data Model to DB2/6000

Figure 127 shows the direct translation of the ODM data model to DB2/6000, taking into account the best possible way to depict the referential dependencies of the configuration data. To compare this data model with the ODM definition refer to Figure 3 on page 12 in Chapter 3, “Designing a Data Model for Configuration Data” on page 11.

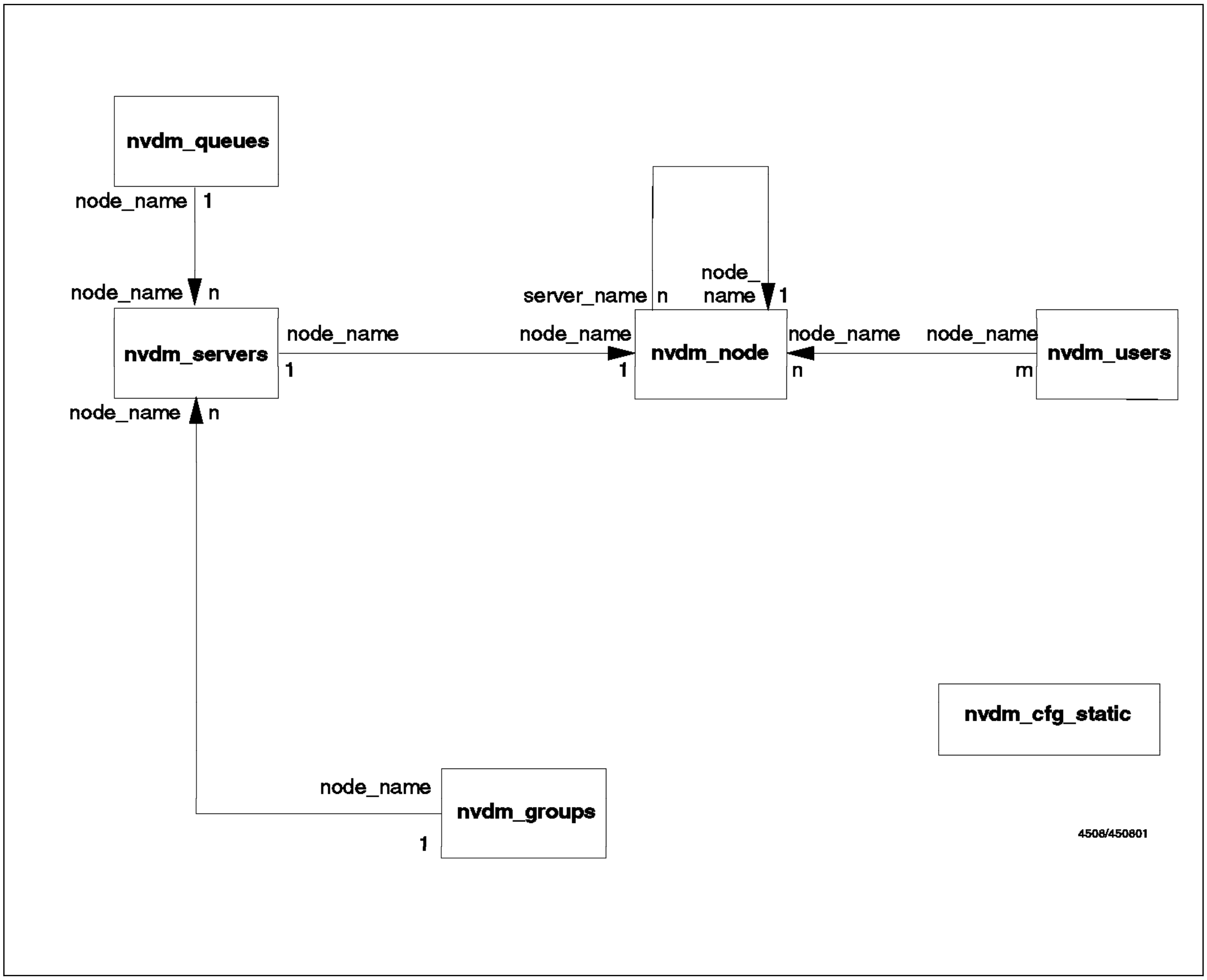


Figure 127. Direct Porting of the ODM Data Model to DB2/6000

Note

In connection with DB2/6000 we use the terminology of relational database systems. We use the terms *table* instead of ODM class, *column* instead of ODM attribute and *row* instead of ODM object.

For each table in the configuration database we define a *primary key*. This is a unique descriptor of the rows contained in the particular table and consist of one or more non-nullable columns. For example, `node_name` specifies unambiguously each row of the table `nvdm_node` while `node_name` and `username` form the primary key of table nvdm_users (compare with Chapter 3, "Designing a Data Model for Configuration Data" on page 11).

Foreign keys are columns in a table that constitute the primary key of another table. In our example the column node_name in nvdm_users is a foreign key because it has the same meaning as node_name that is the primary key of nvdm_node.

In this manner the couple foreign key and primary key define a relationship between the tables they are contained in. Figure 127 on page 228 shows the types of the relationships between the configuration tables: one-to-one, many-to-one or many-to-many. For example, while the relationship between nvdm_servers and nvdm_node is a one-to-one relationship (one server definition can correspond at the most to one node definition, and vice versa); there may exist many users on one node and one user may also be registered at more than one machine (many-to-many relationship between nvdm_node and nvdm_users).

Considering the relationship between nvdm_users and nvdm_node it is obvious that the existence of rows in the former table is only sensible when a row for the appropriate node exists in the latter table. The guaranteeing of this semantical relationship of the data in the database is called *data integrity*. Data integrity is defined by *referential constraints*.

The arrows in Figure 127 on page 228 show the referential constraints of the foreign keys. They are labelled with the names of the foreign keys and primary keys involved in the relationships between the tables. See the following figure for explanation:

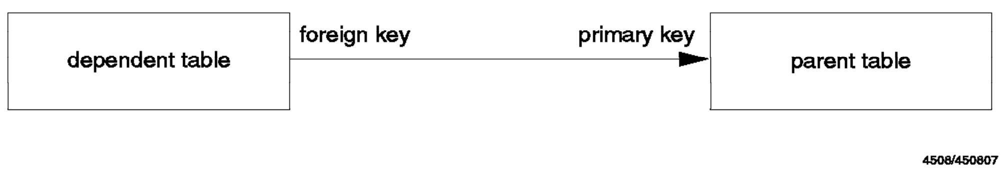


Figure 128. Referential Constraints between Tables

For example, the referential constraint between nvdm_users and nvdm_node defines the former table as *dependent* from the *parent table* nvdm_node.

In this case the constraint is defined as on delete cascade, which makes the independent existence of a user impossible on a machine without the definition of the latter as node in the nvdm_node table.

If we assume that the link construct in ODM represents referential integrity of data, there are two differences between the DB2/6000 definition the ODM definition (compare with Figure 3 on page 12):

- The reference from group_name of nvdm_node to group_name of nvdm_groups is not shown on Figure 127 on page 228.
- The foreign key server_name in nvdm_node points to the same table instead of referencing table nvdm_servers.

Note

In fact, ODM offers the link construct but it is only a syntactical feature that helps to define attributes with the same characteristics from the same data type. ODM cannot guarantee any referential integrity of the stored data.

The reason for these restrictions is the strict checking in DB2/6000 for reflexive dependencies during the data definition. Such referential dependencies arise in the following cases:

direct: Between `nvdm_node` and `nvdm_servers`

indirect: Between `nvdm_node` and `nvdm_groups` (the circle is closed over `nvdm_servers`)

The following figure depicts all defined link constructs in the ODM model that are not allowed to be defined as referential constraints in DB2/6000 (the broken lines depict the relationships not included in our DB2/6000 data model):

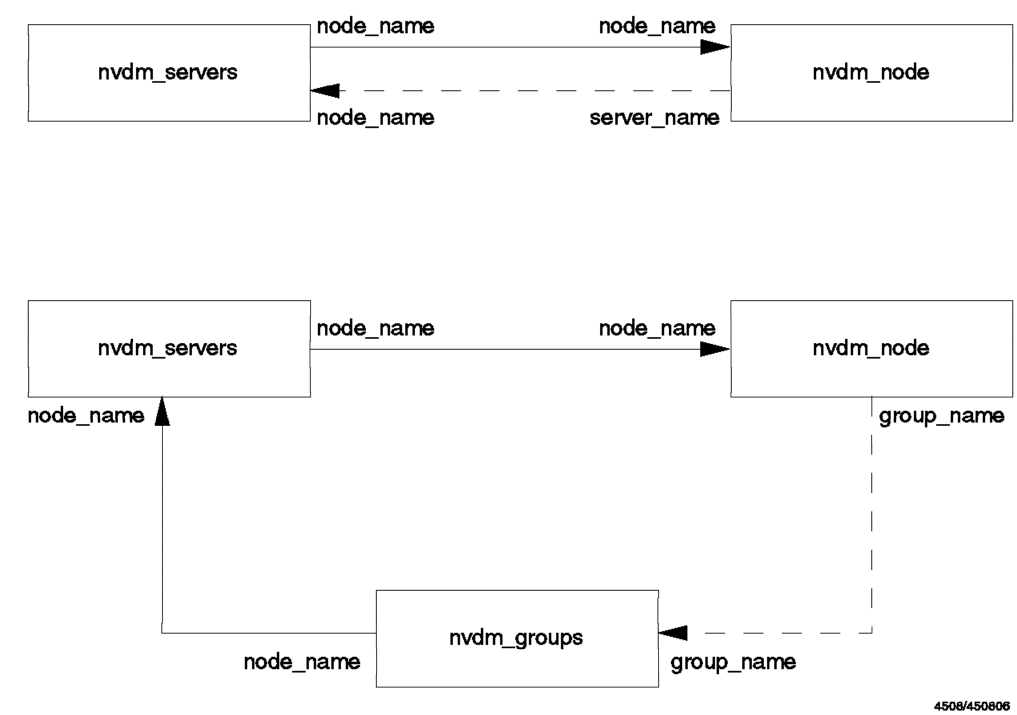


Figure 129. Reflexive Referential Dependences Derived from the ODM Data Model

Actually, the two relationships above define, together with the depicted dependencies in Figure 127 on page 228, a direct reflection between `nvdm_node` and `nvdm_servers` and an indirect reflection between `nvdm_node` and `nvdm_groups` (circular dependence between `nvdm_groups`, `nvdm_servers` and `nvdm_node`).

The advantage of the strict checking of the dependencies in DB2/6000 is the static monitoring of the referential integrity of the data, guaranteed by the database management system itself. That is, a sophisticated data model that makes use of the data definition features of DB2/6000 can move the responsibility of keeping the data integrity from the database administrator to the database management system.

12.3.2 Creating and Recreating the Configuration Database

In order to enable the automatic creation of the configuration database we describe a similar procedure to the script `build_db` from Chapter 3, "Designing a Data Model for Configuration Data" on page 11. We also named the script `build_db` as a replaceable part of the database procedures belonging to the automatic NetView DM/6000 configuration. It is shown in Figure 130 on page 232 and *must* be executed on the database server by the instance owner (`dbmsadm`). This requirement makes sense because of the following considerations:

- DB2/6000 restricts even the system owner from executing some management operations from a client machine. That includes `db2start`, `create database` and `drop database` operations needed by the script `build_db`.
- As the full name of a database table contains also the name of its creator (for example, `dbmsadm.nvdm_node`), the requirement that `dbmsadm` must create all configuration tables allows other authorized database users to refer to them unambiguously. Otherwise, if other database users are allowed to create tables, each time the creator part of the table name will be different.

```
#
#
# procedure for building the DB2/6000 configuration database
#
#


# creating / recreating the configuration database
#-------------------------------------------------
. ./db_create


# table definitions
#------------------
db2 -sf ./db_model.sql
if [ $? -ne 0 ]
then
  exit 1
fi


# comments
#---------
db2 -sf ./db_comment.sql
if [ $? -ne 0 ]
then
  exit 1
fi


# authorizations
#---------------
db2 -sf ./db_authorize.sql
if [ $? -ne 0 ]
then
  exit 1
fi


# inserting data
#---------------
db2 -sf ./db_import.sql
if [ $? -ne 0 ]
then
  exit 1
fi

print "\n\nDATABASE CONFIG: Database NVDM_CFG built SUCCESSFULLY!!!\n\n"
exit 0
```

Figure 130. Building the Configuration Database NVDM_CFG (Script build_db)

The script is made of different blocks that perform the following steps:

- Creating the database
- Defining the configuration tables

- Adding comments to the database objects
- Granting authorizations
- Inserting data into the tables

In the following we describe the steps in detail.

12.3.2.1 Creating the Database

This first task is performed by the script `db_create`, which is shown in Figure 131 on page 234.

```
#
#
# procedure for creating / recreating the NetView DM/6000
# configuration database
#
#


# authentication type (SERVER or CLIENT)
AUTH=CLIENT

# configuration database name (this name occurs in db_model.sql too)
DBNAME=$DB2DBDFT
DBDIR=/home/dbmsadm/databases


# start DB2/6000 manager
print "\nDATABASE CONFIG: starting the database manager"
set -x
db2 db2start
set +x

# get SQL state after connecting to the database
SQLCODE=`db2 -ec +o connect to $DBNAME`

case $SQLCODE in
# database exits (SQLCODE = 0): recreate it
  "0")      print "\nDATABASE CONFIG: dropping the old database"
            set -x
            db2 force application all
            set +x
            sleep 20
            set -x
            db2 drop database $DBNAME
            set +x
            print "\nDATABASE CONFIG: recreation of the database"
            ;;
# database does not exist: just create it
  "-1013" | "-1031")
            print "\nDATABASE CONFIG: creation of the database"
            ;;
# else
  "*")      print "\nDATABASE CONFIG: unknown error while creating/altering \
configuration database"
          exit 1 ;;
esac

db2 create database $DBNAME on $DBDIR \
    authentication $AUTH \
    with \"NVDM configuration database\"
db2 connect >/dev/null 2>&1   # necessary after FORCE
```

Figure 131. Creating and Recreating the Configuration Database NVDM_CFG (Script db_c

First, it starts the database manager (`db2start`). Depending on whether or not the configuration database already exists, it is created or recreated with the name

NVDM_CFG. In the case of recreation, first all users are forced to disconnect (`force application`) and the database is dropped.

> **Note**
>
> After using `force application` you can get the following message when trying to connect to the database (issuing `connect to` from the Command Line Processor `db2` or calling it in the case of implicit connect):
>
> ```
> SQL1224N: A database agent could not be started to service a request,
> or was terminated as a result of a database system shutdown or
> a force command.
> ```
>
> Issue a new connect request to get connected to the database.

The common step in both cases is the `create database` operation with authentication type client (see the value of the variable AUTH). Section 12.3.3, "Authentication Types and Security Considerations" on page 247 explains the use of authentication types in detail.

12.3.2.2 Defining the Configuration Tables

Figure 132 on page 236 shows the file `db_model.sql` with the DB2/6000 commands that create the tables of the configuration database. We are not going to describe again the meaning of the particular columns, but only point out some important details concerning the implementation in DB2/6000. The comments inserted into the database in the next section describe the particular meaning of the columns.

As shown in Figure 127 on page 228 the table `nvdm_node` is the parent table for `nvdm_users` and `nvdm_servers`. The latter itself is the parent table for `nvdm_groups` and `nvdm_queues`. Therefore, `nvdm_node` is the parent table of all of these tables. A row cannot be inserted into the dependent tables before it is inserted into `nvdm_node`, and vice versa, a deletion of a row in `nvdm_node` leads to the deletion of all related rows in the dependent tables (see the referential integrity considerations on page 229).

```
------------------------------------------------------------------------
--
-- Data Model Definition (DB2/6000)
--
------------------------------------------------------------------------


-- connecting the database
--------------------------
connect to nvdm_cfg


-- creation of NVDM tables
--------------------------

-- the nvdm_node table describes the name (IP Hostname) and
-- type (Server, Agent, Prep Site) of the node, where
-- 0 : NVDM Server
-- 1 : NVDM Agent
-- 2 : NVDM Prep Site
-- also included are attributes required for every node, like
-- the name of the NVDM/6000 Server, etc.
--
-- group_name is a link to the nvdm_groups table specifying
-- the group this target belongs to

create table nvdm_node \
    (node_name char(24) not null primary key, \
     node_type char(1) not null, \
     short_name char(8) not null, \
     target_os char(11), \
     description char(24), \
     contact_name char(24), \
     owning_manager char(24), \
     telephone_number char(19), \
     customer_name char(19), \
     repos_fs char(3), \
     repos_size char(20), \
     x_25_number char(14), \
     server_name char(24) not null, \
     group_name char(24), \
     foreign key r_server (server_name) \
       references nvdm_node \
       on delete cascade )
```

Figure 132 (Part 1 of 3). Database Table Definitions (Script db_model.sql)

```
-- nvdm_users is a table containing the users
-- for a target. this relation will be used on
-- servers and targets to define users

create table nvdm_users \
    (node_name char(24) not null, \
     username char(8) not null, \
     usergroup char(11) not null, \
     primary key (node_name,username), \
     foreign key r_node (node_name) \
       references nvdm_node \
       on delete cascade )

-- the nvdm_servers table contains parameters only
-- needed to configure NVDM/6000 Servers

create table nvdm_servers \
    (node_name char(24) not null primary key, \
     local_lu_name char(12), \
     pu_name char(8), \
     cp_name char(8), \
     xid char(8), \
     sna char(3), \
     foreign key r_node (node_name) \
       references nvdm_node \
       on delete cascade )

-- the nvdm_groups table defines the target groups to be defined
-- on a server

create table nvdm_groups \
    (node_name char(24) not null, \
     group_name char(24) not null, \
     description char(24), \
     short_name char(8) not null, \
     primary key (node_name, group_name), \
     foreign key r_node (node_name) \
       references nvdm_servers \
       on delete cascade )
```

Figure 132 (Part 2 of 3). Database Table Definitions (Script db_model.sql)

```
-- the nvdm_queues table contains connections to
-- remote servers
-- e.g. a Focal Point or remote administrator
--
-- Protocol must be "APPC" or "TCP/IP"
-- if Protocol is TCP/IP the remote_server
-- field must be filled with the IP hostname
-- of the remote server
--
-- This table will also be used to define
-- The remote server as a remote target automatically

create table nvdm_queues \
    (node_name char(24) not null, \
     remote_server char(24) not null, \
     protocol char(7), \
     focal_point char(3), \
     inter_node char(8), \
     primary key (node_name, remote_server), \
     foreign key r_node (node_name) \
       references nvdm_servers \
       on delete cascade )

-- nvdm_cfg_static contains all parameters being
-- unique for all targets

create table nvdm_cfg_static \
    (name char(19) not null primary key, \
     value char(127))


-- commit work and quit
-----------------------

commit work
quit
```

Figure 132 (Part 3 of 3). Database Table Definitions (Script db_model.sql)

The table `nvdm_cfg_static` has no relations to the other tables.

All columns of the configuration tables have the type char. One can argue that in some places it is better to use INT or SMALLINT data types (for example, for the column `node_type` in `nvdm_node`). We chose only character types to represent the configuration data, even when sacrificing some space for the internal data representation. Our intention was to keep the data access procedures (see 12.4, "Database Access Procedures" on page 253) as simple as possible. As shell scripts process string variables in the case of using non-character data types in DB2/6000, you must provide the appropriate data conversion in the access procedures depending on the column type.

Note

Comparing the lengths of the ODM attributes (see Figure 4 on page 18) and the lengths of the respective DB2/6000 columns, they differ in one character. This is because ODM needs one character more for the end-of-string symbol.

12.3.2.3 Adding Comments to the Database Objects

After creating the database tables, we add appropriate comments to the database objects. The following figure is showing the script `db_comment.sql`, which is called after `db_model`.

```
--------------------------------------------------------------------------
--
-- Comments on NetView DM/6000 Configuration Tables and Columns
--
--------------------------------------------------------------------------

-- Table nvdm_node
------------------
comment on table nvdm_node \
  is 'NetView DM/6000 nodes in the distribution network'
comment on column nvdm_node.node_name \
  is 'IP host name of the node (primary key)'
comment on column nvdm_node.node_type \
  is 'Node type (0=server, 1=agent, 2=preparation site, not null)'
comment on column nvdm_node.short_name \
  is 'Target short name (not null)'
comment on column nvdm_node.target_os \
  is 'Target operating system (AIX assumed if null)'
comment on column nvdm_node.repos_fs \
  is 'Flag indicating if the repository directory has to be put in an own \
file system (yes/no)'
comment on column nvdm_node.repos_size \
  is 'Size in blocks of the file system to be created (when repos_fs is set \
to yes'
comment on column nvdm_node.x_25_number \
  is 'If null configure SNA profiles to use the control point XID instead'
comment on column nvdm_node.server_name \
  is 'Name of the NetView DM/6000 server for this target (not null, \
foreigh key to nvdm_node)'
comment on column nvdm_node.group_name \
  is 'Target group the node belongs to'
comment on column nvdm_node.config_db \
  is 'Configuration database support (ODM or DB2, ODM if null)'

-- Table nvdm_users
-------------------
comment on table nvdm_users \
  is 'NetView DM/6000 users'
comment on column nvdm_users.node_name \
  is 'IP host name of the node (primary key, foreign key to nvdm_node)'
comment on column nvdm_users.username \
  is 'AIX user name (primary key)'
comment on column nvdm_users.usergroup \
  is 'AIX user group'
```

Figure 133 (Part 1 of 2). Script db_comment.sql for Adding Comments to the Database Objects

```
-- Table nvdm_servers
---------------------
comment on table nvdm_servers \
  is 'NetView DM/6000 servers'
comment on column nvdm_servers.node_name \
  is 'Server name (primary key, foreign key to nvdm_node)'
comment on column nvdm_servers.local_lu_name \
  is 'LU6.2 name'
comment on column nvdm_servers.pu_name \
  is 'SNA physical unit name'
comment on column nvdm_servers.cp_name \
  is 'SNA control point name'
comment on column nvdm_servers.xid \
  is 'XID of the server node'
comment on column nvdm_servers.sna \
  is 'Flag indicating whether this node uses SNA connection (yes/no)'

-- Table nvdm_groups
--------------------
comment on table nvdm_groups \
  is 'NetView DM/6000 target groups'
comment on column nvdm_groups.node_name \
  is 'Server name managing the target group \
(primary key, foreign key to nvdm_node)'
comment on column nvdm_groups.group_name \
  is 'Target group name (primary key)'
comment on column nvdm_groups.short_name \
  is 'Short name of the group (not null)'

-- Table nvdm_queues
--------------------
comment on table nvdm_queues \
  is 'SNA/DS queues'
comment on column nvdm_queues.node_name \
  is 'Server name (primary key, foreign key to nvdm_node)'
comment on column nvdm_queues.remote_server \
  is 'Short name of remote server'
comment on column nvdm_queues.protocol \
  is 'Communication protocol (APPC or TCP/IP)'
comment on column nvdm_queues.focal_point \
  is 'Flag indicating whether the remote node is a focal point'
comment on column nvdm_queues.inter_node \
  is 'Short name of intermediate node if present'

-- Table nvdm_cfg_static
------------------------
comment on table nvdm_cfg_static \
  is 'Common distribution network information'
comment on column nvdm_cfg_static.name \
  is 'Global parameter name'
comment on column nvdm_cfg_static.value \
  is 'Parameter value'
```

Figure 133 (Part 2 of 2). Script db_comment.sql for Adding Comments to the Database Objects

12.3.2.4 Granting Authorizations

The script `build_db` calls the Command Line Processor `db2` with the file `db_authorize.sql` to perform the user authorization task. This file contains the `grant` commands for the Command Line Processor as shown in the following figure:

```
---------------------------------------------------------------------
--
-- User Authorization
--
---------------------------------------------------------------------



grant connect on database to dbcfgadm,dbcfgusr,root

grant all on nvdm_node to dbcfgadm
grant all on nvdm_users to dbcfgadm
grant all on nvdm_servers to dbcfgadm
grant all on nvdm_groups to dbcfgadm
grant all on nvdm_queues to dbcfgadm
grant all on nvdm_cfg_static to dbcfgadm

grant select on nvdm_node to dbcfgusr,root
grant select on nvdm_users to dbcfgusr,root
grant select on nvdm_servers to dbcfgusr,root
grant select on nvdm_groups to dbcfgusr,root
grant select on nvdm_queues to dbcfgusr,root
grant select on nvdm_cfg_static to dbcfgusr,root
```

Figure 134. Database User Authorizations (Script db_authorize.sql)

The instance owner `dbmsadm` authorizes the users `dbcfgadm`, `dbcfgusr` and `root` to connect to the database NVDM_CFG. The users `dbcfgusr` and `root` can only select data from the configuration tables (owned by `dbmsadm`). The user `dbcfgadm` is granted all rights over the configuration tables except creating and dropping tables (see considerations on page 225 when creating the AIX users). It corresponds to the FNDADMN authority in NetView DM/6000.

12.3.2.5 Inserting Data into the Tables

There are several different ways of inserting data into the configuration database (see Chapter 15, “Modifying Configuration Data Using a Graphical User Interface” on page 293). At this place in the script `build_db` you can use the desired insert method calling an appropriate script (see Chapter 14, “Converting the Data Model between ODM and DB2/6000” on page 275 for an example).

In our case we use the import possibility of DB2/6000. The data is prepared in ASCII files in a particular format and then put into the database by the SQL command `import`.

Although this method seems uncomfortable, it is justified by the following:

- As the file format corresponds to DRDA standards, such files can be used to transfer data between DRDA-compliant databases.
- The files can be used as an intermediate step in converting data from DB2/6000 into non-DRDA-compliant databases such as ODM (see Chapter 14, “Converting the Data Model between ODM and DB2/6000” on page 275).
- Such files can be easily generated automatically from other applications (for example, in the case of large distribution networks where it is too awkward to enter the configuration data manually, see Chapter 9, “Configuring a Production Environment” on page 155).

The data imports into the NVDM_CFG database are done by the script `db_import.sql`, which contains `import` statements for each table and is called in `build_db` by the Command Line Processor. The following figure shows the data import file:

```
----------------------------------------------------------------------
--
-- Import Data
--
----------------------------------------------------------------------



import from NVDM_NODE.del of del insert into nvdm_node
import from NVDM_USERS.del of del insert into nvdm_users
import from NVDM_SERVERS.del of del insert into nvdm_servers
import from NVDM_GROUPS.del of del insert into nvdm_groups
import from NVDM_QUEUES.del of del insert into nvdm_queues
import from NVDM_CFG_STATIC.del of del insert into nvdm_cfg_static
```

Figure 135. Import of Data for the Tables of NVDM_CFG (Script db_import)

The data for each table of the NVDM_CFG database is contained in files with names made of the name of the respective table in upper-case and an extension `del`. For example the appropriate import file for the table `nvdm_node` is `NVDM_NODE.del`. The following figure shows the data for the table `nvdm_node`:

```
"rs600012","0","RS600012","AIX","ITSO Raleigh DB2 server","Plamen Kiradjiev",
          "Wolfgang Geiger","2377","IBM","yes",,,"rs600012",
"rs60004","0","RS60004",,"dummy server",,,,,,,,"rs60004","Group1"
"rs600011","0","RS600011","AIX","ITSO Raleigh test server","Stefan Uelpenich",
          "Wolfgang Geiger","4711","IBM",,"0",,"rs600011","Group1"
"rs60005","1","RS60005","AIX","dummy agent",,,,,,"0",,"rs600011","Group1"
"rs600077","1","RS600077","AIX","dummy agent",,,,,,,,"rs60004","Group1"
```

Figure 136. Import Data File for Table nvdm_node

The data is stored in the DEL (delimited ASCII) file format with commas (,) as column delimiters. Null values, if allowed, are provided by entering two adjacent commas.

The following is the output of the script `build_db` in the case when creating the database is a success:

```
DATABASE CONFIG: starting the database manager
+ db2 db2start
SQL1026N  The database manager is already active.

DATABASE CONFIG: creation of the database
DB20000I  The CREATE DATABASE command completed successfully.

   Database Connection Information

 Database product       = DB2/6000 1.2.0
 SQL authorization ID   = DBMSADM
 Local database alias   = NVDM_CFG


DB20000I  The SQL command completed successfully.

  ...

DB20000I  The SQL command completed successfully.

SQL3109N  The Import utility is beginning to import data from file
"NVDM_NODE.del".

SQL3110N  The Import utility has completed processing.  "3" rows were read
from the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "3".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "3" rows were processed from the input file.  "3" rows were
successfully inserted into the table.  "0" rows were rejected.


SQL3109N  The Import utility is beginning to import data from file
"NVDM_USERS.del".

SQL3110N  The Import utility has completed processing.  "8" rows were read
from the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "8".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "8" rows were processed from the input file.  "8" rows were
successfully inserted into the table.  "0" rows were rejected.


SQL3109N  The Import utility is beginning to import data from file
"NVDM_SERVERS.del".

SQL3110N  The Import utility has completed processing.  "2" rows were read
from the input file.
```

Figure 137 (Part 1 of 2). Database Creation Log Output

```
SQL3221W  ...Begin COMMIT WORK. Input Record Count = "2".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "2" rows were processed from the input file.  "2" rows were
successfully inserted into the table.  "0" rows were rejected.

SQL3109N  The Import utility is beginning to import data from file
"NVDM_GROUPS.del".

SQL3110N  The Import utility has completed processing.  "1" rows were
read from the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "1".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "1" rows were processed from the input file.  "1" rows were
successfully inserted into the table.  "0" rows were rejected.

SQL3109N  The Import utility is beginning to import data from file
"NVDM_QUEUES.del".

SQL3110N  The Import utility has completed processing.  "0" rows were
read from the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "0".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "0" rows were processed from the input file.  "0" rows were
successfully inserted into the table.  "0" rows were rejected.

SQL3109N  The Import utility is beginning to import data from file
"NVDM_CFG_STATIC.del".

SQL3110N  The Import utility has completed processing.  "25" rows were
read from the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "25".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "25" rows were processed from the input file.  "25" rows were
successfully inserted into the table.  "0" rows were rejected.

DATABASE CONFIG: Database NVDM_CFG built SUCCESSFULLY!!!
```

Figure 137 (Part 2 of 2). Database Creation Log Output

12.3.3 Authentication Types and Security Considerations

In this section we consider some specifics of the two possible authentication types in DB2/6000: server and client. It will help you to get an impression of how user authentication is done in DB2/6000 and what impact the chosen authentication type has on the remote database access and on the NetView DM/6000 configuration security.

12.3.3.1 Authentication Types in DB2/6000

There are two ways of using the configuration database: with server authentication or with client authentication. As the given authentication type has to be identical on both the server and the client sites the following actions have to be performed:

Server For general use set the appropriate value for the variable AUTH in the script `build_db`. When `build_db` is executed the database is created, respectively recreated with the desired authentication type.

For temporary change of the authentication, you must uncatalog the database and then catalog it again with the desired authentication type. You can do this only as the instance owner `dbmsadm`. To change the authorization from the default server to client, execute the following commands from the Command Line Processor:

```
uncatalog database nvdm_cfg
catalog database nvdm_cfg on /home/dbmsadm/databases \
authentication client
```

Note

Before uncataloging the database, you can run `list database directory` to get information about the path where the database resides.

The default authentication type for DB2/6000 is server. So if you omit the authentication parameter, the database is cataloged with server authentication.

Client To change the authentication type on the client site, you must uncatalog and catalog the database with the new authentication type. The actions are similar to the temporary change of the authentication type on the server, except that you must specify the TCP/IP node instead of database directory (see 12.2.5, “Further Client Configuration” on page 226 for details):

```
uncatalog database nvdm_cfg
catalog database nvdm_cfg at node rs12db authentication client
```

Only the instance owner has the permission of cataloging and uncataloging databases.

Note

It is important that the given authentication type matches on both the server and its clients. Otherwise the following error massage is generated:

```
SQL1401N  Authentication types do not match.
```

12.3.3.2 Comparison between the Two Types of Authentication in DB2/6000

The two types of authentication provide the two extremes considering the comfort of remote access and the database security. The following is a comparison of the two authentication types with regard to both of the criteria above:

Authentication type SERVER

- Remote access

  To access the remote database from a client, you must first issue the command `connect`, for example:

  ```
  connect to nvdm_cfg user dbcfgusr
  ```

  Then you are prompted to enter the password of the given user name. The authentication is made on the server, which means that such an AIX user *must* be registered on the server. Therefore the remote user must *explicitly* connect to the database.

- Security

  From both types the server authentication provides the higher level of security because every remote user must have a valid AIX user account and password on the server. Of course, the general network security considerations, like gaining the password while transmitting it to the server, apply here.

For the case of using server authentication, we defined two AIX users on the database server `dbcfgadm` and `dbcfgusr` (see page 225) and granted them appropriate authorities (see 12.3.2.4, “Granting Authorizations” on page 242). To obtain the desired priority level a remote user should log in as `dbcfgadm`, for NetView DM/6000 administration tasks, respectively as `dbcfgusr`, for simple selections for FNDBLD or FNDUSER task

Authentication type CLIENT

- Remote access

  The authentication on the client provides a very comfortable way to access the remote database from the client machine. Here the authentication is made locally on the client and is based on the local user account the current AIX user is registered under. Moreover, you can set the default database name in the variable DB2DBDFT from the script `db2profile` to automatically connect to the desired database after issuing `db2`.

This way is very convenient for the database access from shell scripts because there is no need of interaction while running the script. In our scenario, we granted `root` the authority for connecting the NVDM_CFG database and querying its tables. With this type of authentication, a `root` on the client can obtain implicitly the same database authorities as the `root` on the server just after calling the Command Line Processor `db2`, provided that DB2DBDFT is set to the desired default database. Otherwise, it only has to call `connect to nvdm_cfg` but user and password do not need to be supplied. The database user is accepted by the server after passing the authentication on the client since it is authorized to use the database.

Note

The database user does not need to be registered as an AIX user on the database server machine.

- Security

  At this place arises a big security hole in using the configuration database.

  NetView DM/6000 defines generally three groups of users: FNDADMN, FNDBLD and FNDUSER. By default, only the FNDADMN users have the authority of configuring and administrating NetView DM/6000.

  The automatic configuration procedure uses the NVDM_CFG database, among other things, to create the NetView DM/6000 users on the targets. It is quite possible that some `root`'s in the CC domain do not have FNDADMN rights, while other users on the same machines are authorized to change the NetView DM/6000 configuration.

  Moreover, some machines might not contain FNDADMN users at all, but every `root` can identify himself as any user on his local machine. This includes the instance owner which is presented on every client machine. That is, the `root` user on the database client, whether or not it is authorized, *can grab* database rights that he is not entitled to. Hence he can gain full control of NetView DM/6000 through the NVDM_CFG configuration database.

  Note

  This contradicts the security concepts of NetView DM/6000 where the authorization of the users on the targets is made on the CC server based on their privileges (FNDADMN, FNDBLD or FNDUSER) defined also on the server. We can speak here about server type authorization in similarity to the database notion.

We can draw the following conclusions after the considerations above:

- Use server authentication to ensure a higher level of security while configuring NetView DM/6000. This approach is reasonable in small networks because of the need to type passwords during the configuration of each remote database client.

- Use client authentication in large trusted networks to achieve a higher level of automation of the configuration.

Note.

Since we assumed our test environment is trusted, we defined client authentication (see Figure 130 on page 232). Another reason for this choice is our focus on the *automation* in configuring NetView DM/6000 in a large software distribution network.

When stressing more the secure aspect of the process, you must alter the AUTH variable definition in the script `build_db`, and catalog the database on the clients with authentication type SERVER (see previous section).

12.3.4 An Improved Data Model of the Configuration Database

This section is written for people who want to get an impression of a slightly different, more database-oriented approach of modelling the NetView DM/6000 configuration. Other reader, could continue with 12.4, "Database Access Procedures" on page 253, and come to this section later.

In 12.3.1, "Porting of the ODM Data Model to DB2/6000" on page 228, we presented the direct porting of the ODM data model from Chapter 3, "Designing a Data Model for Configuration Data" on page 11 with the goal to be able to exchange the database part of the automatic configuration procedure without affecting the code of the configuration script `config_nvdm`.

We pointed out some weaknesses of the designed ODM model with regard to defining the data integrity of the configuration database. They especially originate from the reflexive dependencies between tables (see 12.3, "Depicting the Data Model for the Configuration Data in DB2/6000" on page 227 for details).

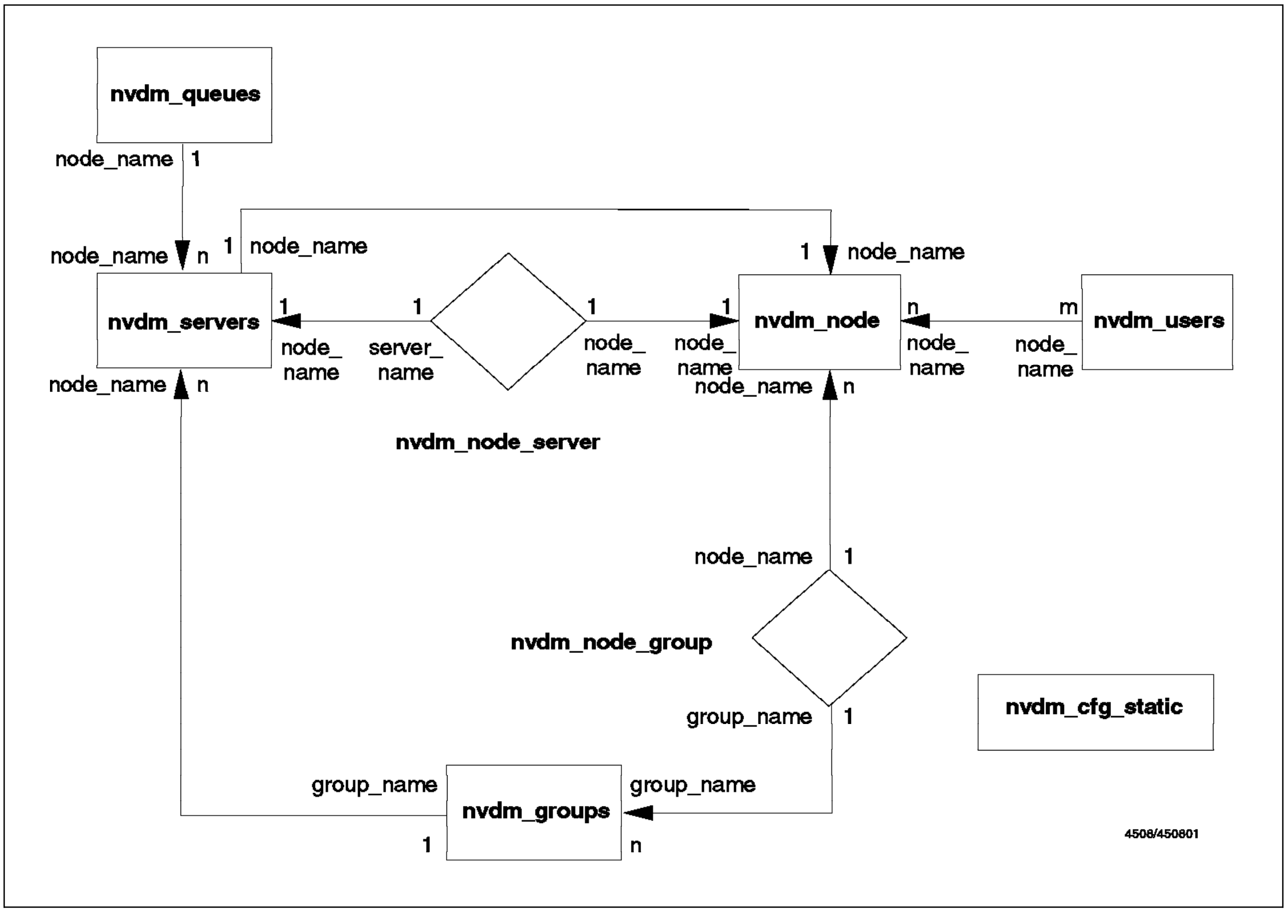


Figure 138. An Improved Data Model for NVDM_CFG

In this section we propose an improved database design that makes use of the advanced DB2/6000 data definition features for ensuring referential integrity of data (see Figure 138). We do not show the implementation of this database design. Our intention is to show how to use DB2/6000, designing a NetView DM/6000 configuration data model that passes the task of monitoring data integrity from the NetView DM/6000 administrator to the DB2/6000 database management system.

As ODM does not provide such powerful tools for defining and monitoring referential integrity of data, this aspect was omitted when defining the ODM data model.

To avoid the reflexive referential dependencies between the tables, `nvdm_node`, `nvdm_servers` and `nvdm_groups` (compare with Figure 127 on page 228) we introduce two new tables that express the table relationships not depicted in Figure 127 on page 228:

- **nvdm_node_server**: Defines the relationship from client in `nvdm_node` to server in `nvdm_servers` and its columns consist of the primary keys of the both tables.
- **nvdm_node_group**: Defines the membership of a node in `nvdm_node` to a group in `nvdm_group` and its columns consist of the primary keys of the both tables.

In the original tables `nvdm_node`, `nvdm_servers` and `nvdm_groups` we leave only the columns with specific information about the appropriate item (node, server or group). We call these tables *basic* and depict them in Figure 138 with a rectangle.

The two new tables are depicted as rhombs and called *relationship* tables.

The following list shows the changed tables with highlighting differences:

- table nvdm_node
  - node_name (primary key)
  - node_type
  - short_name
  - description
  - contact_name
  - owning_manager
  - telephone_number
  - customer_name
  - target_os
- table nvdm_servers
  - node_name (primary key)
  - local_lu_name
  - pu_name
  - cp_name
  - xid
  - sna
- table nvdm_groups
  - node_name (primary key)
  - group_name (primary key)
  - short_name
  - description
- table **nvdm_node_server**
  - **node_name** (foreign key referencing `nvdm_node`)
  - **server_name** (foreign key referencing `nvdm_servers`)
- table **nvdm_node_group**
  - **node_name** (foreign key referencing `nvdm_node`)
  - **group_name** (foreign key referencing `nvdm_groups`)

Note

Defining the relationship between `nvdm_node` and `nvdm_node_group` as a many-to-one relationship, we are now allowed to have a node belonging to different groups. This is generally possible in NetView DM/6000 but the former model did not represent it (see Chapter 3, “Designing a Data Model for Configuration Data” on page 11).

The referential constraints depicted in Figure 127 on page 228 are kept. The table `nvdm_node` remains the parent of all tables except `nvdm_cfg_static`. The new relationships added by the tables `nvdm_node_server` and `nvdm_node_group`, reflect the constraints between the basic tables in the other direction preventing the risk of reflection.

Referring to the new data model, as the deletion of a node in `nvdm_node` triggers the deletion of the appropriate rows in all other dependent tables (except `nvdm_cfg_static`), so do the opposite referential constraints work. When deleting a server entry in `nvdm_server`, all related rows in `nvdm_node_server` are affected, there will be no data remaining that determine a non-existing server as the server of a NetView DM/6000 node. In a similar way the same relationship is valid for the group membership of a node.

Note

The inserting of data is also dependent on the referential constraints. The principle here is the same: no data can exist in dependent tables without the related data in the parent table.

This approach guarantees the clean DB2/6000 implementation of the referential integrity of the NetView DM/6000 configuration database with the aid of foreign keys.

The given proposal in this section does not claim to be the ideal alternative of the NetView DM/6000 configuration data model. It gives just an idea of how to integrate the notion of data integrity into the the DB2/6000 database.

Note

If you decide to implement such a different data model for the configuration database you must adjust all database access procedure calls in the configuration script `config_nvdm` to the new model, as their arguments use table respectively column names. It is also quite possible that some changes in the logic of the configuration procedure are necessary.

12.4 Database Access Procedures

The configuration database NVDM_CFG created and the NetView DM/6000 configuration data imported, we now describe the implementation of the access procedures in DB2/6000 providing the same interface as the ODM database access procedures.

The three interfaces between the automatic configuration script and the configuration database are represented by the following procedures:

- get_attribute
- get_attribute_list
- get_attribute_and

Figure 139 on page 254 shows their implementation using DB2/6000.

```
#
#
# DATABASE ACCESS METHODS (DB2)
#
#



# database owner name
#--------------------
DBOWNER=dbmsadm


#
# connect to the configuration database
#-------------------------------------
print "DB2/6000 : Connect to configuration database"
db2 connect



#
# get data output from SQL (extract SQL header and trailer)
# $1: select clause
# $2: tables (from clause)
# $3: conditions (where clause)
#---------------------------------------------------------
get_data()
{
  WHERE="$3"
  if [ "$WHERE" = "" ]
  then
    WHERE="1=1"
  fi
  SELECT="$1"
  db2 select "$SELECT" from $2 where "$WHERE" | awk '
        BEGIN   {
                  inlist = 0
                }
   /^SQL[0-9][0-9][0-9][0-9][N,C]/ {
           cmd = sprintf("exec 1>&2;echo DB2/6000 : %s",$0)
           system(cmd)
         }
        /^-+/   {
                  inlist = 1
                  next
                }
```

Figure 139 (Part 1 of 3). Database Access Procedures for the Database NVDM_CFG (DB2/6000)

```
        /^$/    {
                  if (inlist == 1) inlist++
                  next
                }
        inlist == 1 {
          gsub(/ *$/,"")
                  print
                }
'

}




#
# get list of selected column values from a DB2 table
# $1 = table name
# $2 = search column name
# $3 = search column value
# $4 = output column name
# The list of selected column values is stored in the VALUE_LIST variable
# The number of selected values is stored in VALUE_NUM
#-------------------------------------------------------------------------
get_attribute_list ()
{
  VALUE_LIST=`get_data "$4" $DBOWNER.$1 "$2 = '$3'"`
  VALUE_NUM=`echo "$VALUE_LIST" | wc -w | sed 's/ //g'`
}

#
# get single select value
# $1 = table name
# $2 = search column name (must be the primary key of the table)
# $3 = search column value
# $4 = output column name
#---------------------------------------------------------------
get_attribute ()
{
  VALUE=`get_data "$4" $DBOWNER.$1 "$2 = '$3'"`
}
```

Figure 139 (Part 2 of 3). Database Access Procedures for the Database NVDM_CFG (DB2/6000)

```
#
# get single select value (AND)
# $1 = table name
# $2 = search field1
# $3 = search field value1
# $4 = search field2
# $5 = search field value2
# $6 = output column name
# field1 and field2 must constitute the primary key of the table
#-----------------------------------------------------------------
get_attribute_and ()
{
  VALUE=`get_data "$6" $DBOWNER.$1 "$2 = '$3' and $4 = '$5'"`
}
```

Figure 139 (Part 3 of 3). Database Access Procedures for the Database NVDM_CFG (DB2/6000)

There is a need of some automatic editing of the output from a DB2/6000 SQL query. Generally, the output of an SQL query consist of the following three parts:

header: With the column names and separating lines

data: The retrieved data

trailer: Selected row count summary

We take as example the following SQL query:

```
SELECT user_name FROM nvdm_users WHERE nvdm_node='rs600012'
```

This query is generated by the following call:

```
get_attribute_list nvdm_users node_name rs600012 user_name
```

The SQL output looks like:

```
USERNAME
---------
plamen
root
stefan

  3 record(s) selected.
```

In order to enable the further processing of the pure retrieved data we must cut the header and trailer information from the SQL query output. This task is performed by the procedure `get_data`. It is called with three arguments representing the three parts of the SQL query:

$1: `select` clause, containing the desired output columns

$2: `from` clause with the queried tables

$3: `where` clause containing the selection predicates as well as the additional parts of the SQL query, like `order` and `group`

The arguments must obey the standard shell evaluation rules. For example, you must quote the asterisk (*) in the select clause like the following:

```
get_data \* nvdm_node "server_name = 'rs60004'"
```

We recommend you enclose the third argument in double-quotes (") like in the given example. The procedure `get_data` returns the pure selected data row by row by applying the editing features of `awk`. Moreover, it checks whether an error message is returned by the database management system and redirects it to standard error.

With the help of this procedure the code for the database access procedures look very simple. You must just build the appropriate call of `get_data` and assign the returned value to the variables VALUE, respectively VALUE_LIST. Especially by `get_attribute_list`, it is also required to set VALUE_NUM to the number of selected rows. This task is performed by counting the list members in VALUE_LIST with the aid of `wc`.

Note

The procedure `get_data` is more powerful and general than the database access procedures used from the NetView DM/6000 configuration procedure. In Chapter 14, "Converting the Data Model between ODM and DB2/6000" on page 275 we show another application of this help procedure, which makes use of the `order` part of the SQL query.

Chapter 13. Testing the Automatic Configuration Procedure with Software Distribution for AIX V3.1 with DB2/6000

After we described the migration of the automatic configuration procedure from Software Distribution for AIX Version 1.2 to Software Distribution for AIX Version 3.1 in Chapter 11, “Migrating the Procedure to Software Distribution for AIX V3.1” on page 197 and ported the data model from ODM in DB2/6000 in Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219, we now test the script `config_nvdm` applying both enhancements.

In this chapter we show a more complicated scenario than our former test environment from Chapter 2, “Base of Automated Configuration” on page 5. Our network environment for the current test is depicted in Figure 140.

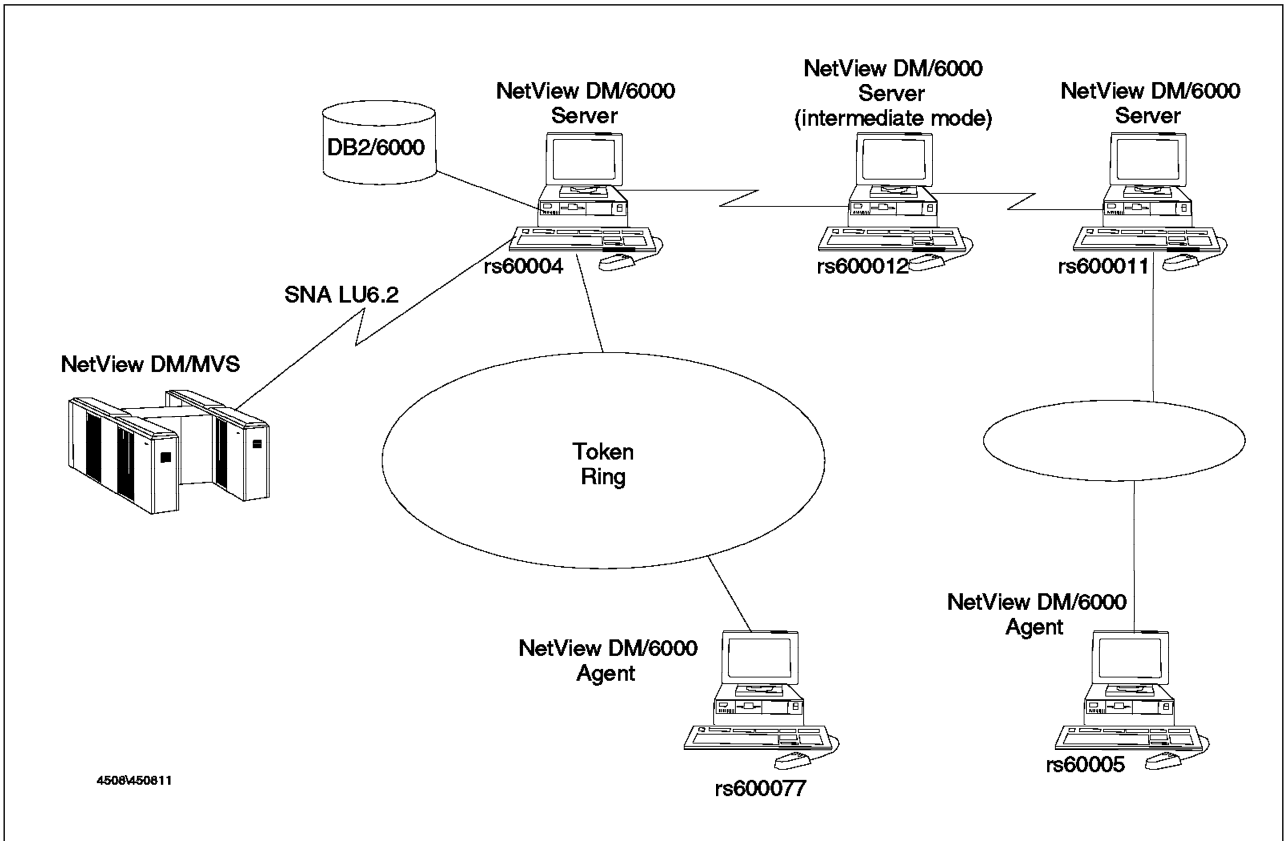


Figure 140. Scenario for Automatic Configuration of NetView DM/6000 V3.1 with DB2/60

We configure the NetView DM/6000 server `rs60004`. It is the CC server for the node `rs600077` and is connected with a NetView DM/MVS by an SNA/DS connection over the SNA LU6.2 protocol. Besides, our server is connected over SNA/DS based on TCP/IP with another server `rs600011` from a different network over the intermediate node `rs600012`. In this way we also show the configuration of an SNA/DS connection over TCP/IP through an intermediate node (refer to Chapter 8, “Enhancing the Configuration Procedure” on page 131).

13.1 Prerequisites for Node Configuration

In order to prepare our test environment for running the automatic configuration procedure, we performed the following tasks:

- Installation of the AIX 3.2.5 operating system on all machines in our test network environment
- Installation of Software Distribution for AIX Server Version 3.1 on `rs60004`, `rs600011` and `rs600012`
- Installation of Software Distribution for AIX Agent Version 3.1 on `rs600077` and `rs60005`
- Installation of SNA Server Version 2.1 on `rs60004`
- Configuration of TCP/IP on all AIX sites in our network over the token-ring adapter (inclusively enabling the TCP/IP name resolution for all machines)
- Configuration of `rs60004` as a DB2/6000 Version 1.2 server (for details see Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219)
- Configuration of all other AIX machines as DB2/6000 Version 1.2 clients (for details see Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219)

13.2 Starting the Configuration

In order to execute the automatic configuration procedure the DB2/6000 database must have been previously filled. See 12.3.2, "Creating and Recreating the Configuration Database" on page 231 about creating the DB2/6000 configuration database. We first present the contents of our configuration database. The output is done from SQL `select` statements by making some projections on the relevant columns in order to let the selected data fit in the page. The following figures depict the script `select_db` generating the SQL reports and its output.

```
#
#
# select statements for the relevant data in the NetView DM/6000
# configuration database
#
#


db2 select node_name, node_type, short_name, target_os, server_name \
from dbmsadm.nvdm_node > nvdm_node

db2 select \* from dbmsadm.nvdm_servers > nvdm_servers.select

db2 select \* from dbmsadm.nvdm_groups  > nvdm_groups.select

db2 select \* from dbmsadm.nvdm_queues > nvdm_queues.select

db2 select \* from dbmsadm.nvdm_users > nvdm_users.select

db2 select \* from dbmsadm.nvdm_cfg_static > nvdm_cfg_static.select
```

Figure 141. Script select_db for Generating Reports from the NetView DM/6000 Configuration Database

```
NODE_NAME                 NODE_TYPE SHORT_NAME TARGET_OS    SERVER_NAME
------------------------- --------- ---------- ------------ -------------------------
rs600012                  0         RS600012   AIX          rs600012
rs60004                   0         RS60004    -            rs60004
rs600011                  0         RS600011   AIX          rs600011
rs60005                   1         RS60005    AIX          rs600011
rs600077                  1         RS600077   AIX          rs60004

  5 record(s) selected.
```

Figure 142. Contents of DB2/6000 Table nvdm_node

```
NODE_NAME                 LOCAL_LU_NAME PU_NAME   CP_NAME   XID       SNA
------------------------- ------------- --------- --------- --------- ----
rs600012                  A             B         C         -         no
rs60004                   RA62224B      RA62224   RA62224A  -         yes
rs600011                  RA60011B      RA600011  RA6011CP  -         no

  3 record(s) selected.
```

Figure 143. Contents of DB2/6000 Table nvdm_servers

```
NODE_NAME                 GROUP_NAME                DESCRIPTION               SHORT_NAME
------------------------- ------------------------- ------------------------- ----------
rs600011                  Group1                    Raleigh Group1            GROUP1

  1 record(s) selected.
```

Figure 144. Contents of DB2/6000 Table nvdm_groups

```
NODE_NAME                 REMOTE_SERVER             PROTOCOL FOCAL_POINT INTER_NODE
------------------------- ------------------------- -------- ----------- ----------
rs600011                  rs600012                  TCP/IP   no          -
rs600011                  rs60004                   TCP/IP   no          RS600012
rs60004                   rs600011                  TCP/IP   no          RS600012
rs60004                   rs600012                  TCP/IP   no          -
rs60004                   RA39TCF1                  APPC     yes         -

  5 record(s) selected.
```

Figure 145. Contents of DB2/6000 Table nvdm_queues

```
NODE_NAME                 USERNAME  USERGROUP
------------------------- --------- ------------
rs600012                  root      FNDADMN
rs600012                  stefan    FNDBLD
rs600012                  plamen    FNDBLD
rs600011                  root      FNDBLD
rs60004                   plamen    FNDUSER
rs60004                   wolfgang  FNDBLD
rs600011                  hugo      FNDUSER
rs600011                  stefan    FNDADMN
rs600077                  root      FNDADMN
rs600077                  stefan    FNDUSER

  10 record(s) selected.
```

Figure 146. Contents of DB2/6000 Table nvdm_users

```
NAME                 VALUE
-------------------- ---------------------------------------------
VTAM_CP_NAME         RAK
SOLICIT_SSCP         yes
I_FIELD_SIZE         2042
LOCAL_SAP            04
REMOTE_SAP           04
INITIATE_CALL        yes
ACTIVATE_START       yes
RESTART_NORMAL       yes
RESTART_ABNORMAL     yes
RESTART_NVDM         no
REM_LINK_ADDR        400001240000
SNA_NET_NAME         USIBMRA
DATALINK_DEVICE      tok0
MODE_PROF_NAME       NVDMNORM
MODE_NAME            NVDMNORM
MAX_RU_SIZE          2048
TPN_PROF_NAME_SND    NVDMSND
TPN_PROF_NAME_RCV    NVDMRCV
RTPN_PROF_NAME_SND   NVDMSND
RTPN_PROF_NAME_RCV   NVDMRCV
PARTNER_LU_NAME      RA39TCF1
SIDE_INFO_PROF_SND   NVDMSIDS
SIDE_INFO_PROF_RCV   NVDMSIDR
TCPIP_PORT           729
NVDM_LOG_SIZE        250000

  25 record(s) selected.
```

Figure 147. Contents of DB2/6000 Table nvdm_cfg_static

Similar to Chapter 5, "Testing the Automatic Configuration Script" on page 83 we want to keep track of the execution of the configuration procedure. In order to do this we use the following statement:

```
config_nvdm rs60004 2>&1 | tee logfile
```

The configuration procedure for NetView DM/6000 with DB2/6000 support is listed in Appendix A, "The Configuration Script Listings" on page 331.

The following figure shows the log file contents after the successful execution of our configuration procedure:

```
NVDM CONFIG : Extracted hostname ... rs60004
DB2/6000 : Connect to configuration database

   Database Connection Information

 Database product       = DB2/6000 1.2.0
 SQL authorization ID   = ROOT
 Local database alias   = NVDM_CFG

===============================================================================
Software distribution network configuration script
$Revision: 1.1 $
IP Hostname = rs60004
Name resolution = rs60004.itso.ral.ibm.com is 9.24.104.27
===============================================================================
NVDM CONFIG : --> Trying to configure node rs60004
NVDM CONFIG : Node type is 0 (0 = Server, 1 = Agent, 2 = Prep)
NVDM CONFIG : --> NVDM Base Node Configuration
NVDM CONFIG : Current hostname of server is rs60004.
NVDM CONFIG : Current WORKSTATION NAME of server is      rs60004.
NVDM CONFIG : Stopping Server...
rs60004
Trying to connect to default server (rs60004).
Connected to server rs60004.
NVDM CONFIG : Sleeping 20 seconds...
NVDM CONFIG : Setting hostname to rs60004.
rs60004
FNDCL021E: The new target name is already in use.
NVDM CONFIG : Setting nvdm.cfg (WORKSTATION NAME) to rs60004
NVDM CONFIG : Setting nvdm.cfg (SERVER) to rs60004
NVDM CONFIG : Setting nvdm.cfg (LOG FILE SIZE) to 250000
CONFIG NVDM : Checking NetViewDM-rcv port...
CONFIG NVDM : Checking NetViewDM-snd port...
CONFIG NVDM : Checking NetViewDM6000 port...
NVDM CONFIG : Setting nvdm.cfg (TCP/IP PORT) to 729
NVDM CONFIG : Resetting root.cli ... (rs60004)
Create /usr/lpp/netviewdm/uicfg/root.cfg file...
NVDM CONFIG : Restarting Server...
NVDM CONFIG : --> In order for the changes to become active
NVDM CONFIG :     NetView DM/6000 will be restarted on this node
NVDM CONFIG : NVDM is not running. It will be started now.
FNDCL232E: Unable to start the system as the D&CC Agent is shutting down.
Trying to connect to default server (rs60004).
Connected to server rs60004.
Trying to connect to default server (rs60004).
Connected to server rs60004.
NVDM CONFIG : Setting SNA Network Name to USIBMRA
NVDM CONFIG : Setting SNA Datalink Device to tok0
NVDM CONFIG : Setting SNA Remote Link Address to 400001240000
```

Figure 148 (Part 1 of 7). Log File Contents After Configuration Procedure

```
NVDM CONFIG : Setting SNA NVDM Mode Profile Name to NVDMNORM
NVDM CONFIG : Setting SNA NVDM Mode Name to NVDMNORM
NVDM CONFIG : Setting SNA TPN Profile Name (Send) to NVDMSND
NVDM CONFIG : Setting SNA TPN Profile Name (Receive) to NVDMRCV
NVDM CONFIG : Setting SNA Partner LU Name (MVS Host) to RA39TCF1
NVDM CONFIG : Setting SNA Side Info Profile Name (Send) to NVDMSIDS
NVDM CONFIG : Setting SNA Side Info Profile Name (Receive) to NVDMSIDR
NVDM CONFIG : Setting Solicit SSCP Field (yes|no) to yes
NVDM CONFIG : Setting I-Field Size to 2042
NVDM CONFIG : Setting SNA Local SAP No. to 04
NVDM CONFIG : Setting Remote SAP No. to 04
NVDM CONFIG : Setting SNA Initiate Call Field (yes|no) to yes
NVDM CONFIG : Setting SNA Activate on start (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on normal termination (yes|no) to yes
NVDM CONFIG : Setting SNA Restart on abnormal termination (yes|no) to yes
NVDM CONFIG : Setting SNA VTAM CP Name (for LU6.2 Location Profile) to RAK
NVDM CONFIG : Setting PU NAME for rs60004 to RA62224
NVDM CONFIG : Setting Local LU Name for rs60004 to RA62224B
NVDM CONFIG : Setting Control Point Name for rs60004to RA62224A
NVDM CONFIG : Could not determine XID for rs60004 configu
ration.
NVDM CONFIG : Setting USE_CP_XID to yes
NVDM CONFIG : --> Configuring SNA
NVDM CONFIG : Exporting existing SNA profiles to /tmp/sna.org ...
NOTE: The committed database does not contain default
      template profiles; none will be exported.
Configuration file '/tmp/sna.org' exported.
NVDM CONFIG : Adding DLC Device for tok0
NVDM CONFIG : Configuring SNA Initial Node Setup
+ mk_qcinit -y token_ring -t appn_end_node -w USIBMRA -d RA62224A
NVDM CONFIG : Configuring SNA Control Point Profile
===============================================================================
+ chsnaobj -t control_pt -e USIBMRA -a RA62224A -A RA62224A -N appn_end_node node_cp
Profile type 'control_pt' name 'node_cp' CHANGED.
===============================================================================
NVDM CONFIG : Configuring SNA DLC Profile
===============================================================================
+ [ sna_dlc_token_ring = sna_dlc_x.25 ]
+ mksnaobj -t sna_dlc_token_ring -d tok0 -b yes -w yes -m 2042 -H 04 -c no -q 0 tok0
0105-0031 Profile type 'sna_dlc_token_ring' name 'tok0' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
===============================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
===============================================================================
+ [ sna_dlc_token_ring = sna_dlc_x.25 ]
+ chsnaobj -t sna_dlc_token_ring -d tok0 -b yes -w yes -m 2042 -H 04 -c no -q 0 tok0
Profile type 'sna_dlc_token_ring' name 'tok0' CHANGED.
```

Figure 148 (Part 2 of 7). Log File Contents After Configuration Procedure

```
================================================================================
NVDM CONFIG : Configuring SNA Link Station Profile
================================================================================
+ [ token_ring = x.25 ]
+ mksnaobj -t link_station -w token_ring -y tok0 -d 400001240000 -l 07100000 -s 04 -a yes \
  -O yes -F yes -h yes -z yes -c yes RA62224
0105-0031 Profile type 'link_station_token_ring' name 'RA62224' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ [ token_ring = x.25 ]
+ chsnaobj -t link_station -w token_ring -y tok0 -d 400001240000 -l 07100000 -s 04 -a yes \
  -O yes -F yes -h yes -z yes -c yes RA62224
Profile type 'link_station_token_ring' name 'RA62224' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA Local LU Profile
================================================================================
+ mksnaobj -t local_lu -u lu6.2 -l RA62224B -L RA62224B RA62224B
0105-0031 Profile type 'local_lu_lu6.2' name 'RA62224B' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t local_lu -u lu6.2 -l RA62224B -L RA62224B RA62224B
Profile type 'local_lu_lu6.2' name 'RA62224B' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA Mode Profile
================================================================================
+ mksnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N #CONNECT -m NVDMNORM NVDMNORM
0105-0031 Profile type 'mode' name 'NVDMNORM' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N #CONNECT -m NVDMNORM NVDMNORM
Profile type 'mode' name 'NVDMNORM' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA TPN Profile (SEND)
================================================================================
+ mksnaobj -t local_tp -n 21F0F0F7 -h yes -c basic -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndts \
  -s none NVDMSND
0105-0031 Profile type 'local_tp' name 'NVDMSND' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t local_tp -n 21F0F0F7 -h yes -c basic -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndts \
  -s none NVDMSND
Profile type 'local_tp' name 'NVDMSND' CHANGED.
```

Figure 148 (Part 3 of 7). Log File Contents After Configuration Procedure (Part 1 of 2)

```
================================================================================
NVDM CONFIG : Configuring SNA TPN Profile (Receive)
================================================================================
+ mksnaobj -t local_tp -n 21F0F0F8 -h yes -c basic -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndtr \
  -s none NVDMRCV
0105-0031 Profile type 'local_tp' name 'NVDMRCV' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t local_tp -n 21F0F0F8 -h yes -c basic -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndtr \
  -s none NVDMRCV
Profile type 'local_tp' name 'NVDMRCV' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA LU6.2 Partner LU
================================================================================
+ mksnaobj -t partner_lu6.2 -p no -P USIBMRA.RA39TCF1 -O none -A RA39TCF1 RA39TCF1
0105-0031 Profile type 'partner_lu6.2' name 'RA39TCF1' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t partner_lu6.2 -p no -P USIBMRA.RA39TCF1 -O none -A RA39TCF1 RA39TCF1
Profile type 'partner_lu6.2' name 'RA39TCF1' CHANGED.
================================================================================
NVDM CONFIG : Configuring SNA LU 6.2 Location Profile
================================================================================
+ mksnaobj -t partner_lu6.2_location -P USIBMRA.RA39TCF1 -O USIBMRA.RAK \
  -m link_station -l RA62224B -s RA62224
RA39TCF1
0105-0031 Profile type 'partner_lu6.2_location' name 'RA39TCF1' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t partner_lu6.2_location -P USIBMRA.RA39TCF1 -O USIBMRA.RAK \
  -m link_station -l RA62224B -s RA62224
RA39TCF1
Profile type 'partner_lu6.2_location' name 'RA39TCF1' CHANGED.
```

Figure 148 (Part 4 of 7). Log File Contents After Configuration Procedure (Part 2 of 2)

```
================================================================================
NVDM CONFIG : Configuring SNA Side Info Profile (Send)
================================================================================
+ mksnaobj -t side_info -L RA62224A -P USIBMRA.RA39TCF1 -m NVDMNORM -d 21F0F0F7 -h yes NVDMSIDS
0105-0031 Profile type 'side_info' name 'NVDMSIDS' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t side_info -L RA62224A -P USIBMRA.RA39TCF1 -m NVDMNORM -d 21F0F0F7 -h yes NVDMSIDS
0105-0102 Both the partner LU alias and fully qualified partner LU name cannot be specified in
          profile type 'side_info' name 'NVDMSIDS'.
0105-0124 Unable to change profile type 'side_info' name 'NVDMSIDS'.
```

Figure 148 (Part 5 of 7). Log File Contents After Configuration Procedure

```
================================================================================
NVDM CONFIG : Configuring SNA Side Info Profile (Receive)
================================================================================
+ mksnaobj -t side_info -L RA62224B -P USIBMRA.RA39TCF1 -m NVDMNORM -d 21F0F0F8 -h yes NVDMSIDR
0105-0031 Profile type 'side_info' name 'NVDMSIDR' already exists.
0105-0025 mksnaobj command failed.
+ RC=255
================================================================================
NVDM CONFIG RECOVER : Profile already existed. Changing existing one ...
================================================================================
+ chsnaobj -t side_info -L RA62224B -P USIBMRA.RA39TCF1 -m NVDMNORM -d 21F0F0F8 -h yes NVDMSIDR
0105-0102 Both the partner LU alias and fully qualified partner LU name cannot be specified in
          profile type 'side_info' name 'NVDMSIDR'.
0105-0124 Unable to change profile type 'side_info' name 'NVDMSIDR'.
================================================================================
NVDM CONFIG : Updating SNA Server...
WARNING: More than one Side Information Profile was found
         to represent the same local LU or CP alias 'RA62224B', partner
         LU name 'USIBMRA.RA39TCF1', and mode name 'NVDMNORM'.
         This may cause an unintended Side Information Profile name to
         be used to identify an active session using those same values.

verifysna command OK.
The profiles listed above have been dynamically updated successfully.
NVDM CONFIG : Configuring TCP/IP connection
NVDM CONFIG : Remote connection to rs600011 is made
              through intermediate node RS600012.
              No SNA/DS connection file is created.
NVDM CONFIG : Configuring TCP/IP connection
NVDM CONFIG : Configuring SNA/DS connection configuration file.
NVDM CONFIG : (TCP/IP) for remote Server rs600012.
NVDM CONFIG : Configuring APPC connection
NVDM CONFIG : Configuring SNA/DS connection configuration file
              /usr/lpp/netviewdm/db/snadscon/RA39TCF1
NVDM CONFIG : Configuring SNA/DS routing table.
NVDM CONFIG : System has TCP/IP connection to remote server.
NVDM CONFIG : System has APPC connection to remote server.
NVDM CONFIG : Writing routing table to /usr/lpp/netviewdm/db/routetab
NVDM CONFIG : Saving target history for RA39TCF1
FNDCL131E: The target specified is not local.
NVDM CONFIG : Sleeping for 15 secs.
NVDM CONFIG : Deleting Target RA39TCF1 from Server rs60004 configuration.
NVDM CONFIG : Saving target history for rs600011
NVDM CONFIG : Sleeping for 15 secs.
NVDM CONFIG : Deleting Target rs600011 from Server rs60004 configuration.
FNDCLC73E: The filters specified do not match any queues.
FNDCLC73E: The filters specified do not match any queues.
NVDM CONFIG : Saving target history for rs600012
NVDM CONFIG : Sleeping for 15 secs.
NVDM CONFIG : Deleting Target rs600012 from Server rs60004 configuration.
```

Figure 148 (Part 6 of 7). Log File Contents After Configuration Procedure

```
NVDM CONFIG : Defining Target rs60004 on server rs60004
NVDM CONFIG : Target already exists. Updating...
nvdm updtg rs60004 -s 'RS60004' -d 'dummy server'
WARNING: The Domain Address has been changed to RS60004.
NVDM CONFIG : Defining Target rs600077 on server rs60004
NVDM CONFIG : Target already exists. Updating...
nvdm updtg rs600077 -s 'RS600077' -y 'AIX' -d 'dummy agent' -b client
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Authorization profile FNDUSER assigned to plamen.

User:                         plamen
Authorization Profile:        FNDUSER
NVDM CONFIG : Authorization profile FNDBLD assigned to wolfgang.

User:                         wolfgang
Authorization Profile:        FNDBLD
NVDM CONFIG : --> Adding AIX users for NVDM...
NVDM CONFIG : Authorization profile FNDADMN assigned to root.

User:                         root
Authorization Profile:        FNDADMN
FNDCLD07E: The root user configuration cannot be updated.
NVDM CONFIG : Authorization profile FNDUSER assigned to stefan.

User:                         stefan
Authorization Profile:        FNDUSER
NVDM CONFIG : Deleting group Mode: from rs60004 configuration.
FNDCL523W: Mode: is not a configured group.
NVDM CONFIG : Configuring Target Groups for rs60004
NVDM CONFIG : No groups defined
NVDM CONFIG : Defining remote target for rs600011
NVDM CONFIG : Defining remote target for rs600012
NVDM CONFIG : Defining remote target for RA39TCF1
NVDM CONFIG : RA39TCF1 will be configured as focal point.
+ eval nvdm addtg RA39TCF1 -m focal -b server -s RA39TCF1 -n USIBMRA -d 'NVDM_MVS' -tp appc:
+ nvdm addtg RA39TCF1 -m focal -b server -s RA39TCF1 -n USIBMRA -d NVDM_MVS -tp appc:
0513-029 The sna Subsystem is already active.
Multiple instances are not supported.
NVDM CONFIG : Releasing NVDM SNA communications.

===============================================================================
NVDM CONFIG : !!! Configuration of Server completed successfully !!!
===============================================================================
```

Figure 148 (Part 7 of 7). Log File Contents After Configuration Procedure

At the beginning of the log file you can find the DB2/6000 database connect information. It contains the DB2/6000 version, the database user we are authenticated as (`root`) when running the script and the database name NVDM_CFG, which we configured as the default connection (see the DB2/6000 installation and configuration instructions in Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219). There is no difference in the database access when running the queries from a DB2/6000 client (in our case the NetView DM/6000 configuration server and the DB2/6000 are located on the same machine, `rs60004`).

The transparency in the database access from the configuration script `config_nvdm` can be recognized when comparing the above log file with the output from Chapter 5, “Testing the Automatic Configuration Script” on page 83. In spite of the different NetView DM/6000 configuration environment, you would not be able to determine whether ODM or DB2/6000 is used as a database platform, if we had not intentionally added the DB2/6000 connect database output into the log information.

13.3 Checking the NetView DM/6000 Configuration

In order to check the correctness of our automatic configuration, we execute some Software Distribution for AIX inquiry commands in this section. You can compare the output from the NetView DM/6000 configuration inquiry commands to the DB2/6000 database contents presented in the previous section.

First, we list the configured targets on our server `rs60004` by issuing the following command:

```
nvdm lstg '*'
```

Its output is shown in the following figure:

```
Target:                 RA39TCF1
Mode:                   Focal
Description:            NVDM_MVS
Type:                   SERVER

Target:                 rs600011
Mode:                   Push
Description:
Type:                   SERVER

Target:                 rs600012
Mode:                   Push
Description:
Type:                   SERVER

Target:                 rs60004
Mode:                   Push
Description:            dummy server
Type:                   SERVER

Target:                 rs600077
Mode:                   Push
Description:            dummy agent
Type:                   CLIENT
```

Figure 149. Configured Targets on the Server rs60004

The node `rs600077` is configured as a local target, while `rs600012`, `rs600011` and the host are configured as remote targets.

According to the intermediate node configuration discussed in Chapter 8, “Enhancing the Configuration Procedure” on page 131, connection queues are created only for the adjacent servers (in our example, the host and rs600012). Hence, we see the following output after executing nvdm stat:

```
SNADS: Released
XFER : Released
CMD  : Released

Target          Type    Connection        Entries   Q Status   Tg Status

RA39TCF1        snads   RA39TCF1          0         Held       Released
rs600012        snads   RS600012          0         Held       Released
```

Figure 150. Connection Queues configured on rs60004

The method used to reach the remote server rs600011 over the intermediate node rs600012 is represented in the file /usr/lpp/netviewdm/db/routetab as follows:

```
NETWORK PROTOCOL: BOTH

#
# SNA connections
#

USIBMRA.RA39TCF1   ANY  ANY  ANY  ANY  RA39TCF15

#
# TCP/IP connections
#

RS600011.*     RS600012
RS600012.*     RS600012
```

Figure 151. Contents of Routing Information File /usr/lpp/netviewdm/db/routetab

In order to list the Software Distribution for AIX user profiles configured on our server rs60004, we execute the following command:

```
nvdm lsusr '*'
```

It generates the following output:

```
User:                          plamen
Authorization Profile:         FNDUSER

User:                          root
Authorization Profile:         FNDADMN

User:                          stefan
Authorization Profile:         FNDUSER

User:                          wolfgang
Authorization Profile:         FNDBLD
```

Figure 152. Locally Configured User Profiles on rs60004

When comparing with the contents of the DB2/6000 table nvdm_users from the previous section, one can recognize that only users from the CC domain are registered on our Software Distribution for AIX server rs60004. For example, the user hugo on the remote server rs600011 is not in the list, although there is a target entry for this node.

Chapter 14. Converting the Data Model between ODM and DB2/6000

This chapter describes the method of switching between the two presented database scenarios for storing the Software Distribution for AIX configuration database. It is addressed to administrators of change and distribution networks who are using different database platforms and want to enable the transformation of the configuration data in both directions.

We present the conversion from ODM to DB2/6000 as well as the transformation of configuration data from DB2/6000 to ODM, although our claim is not to achieve symmetry in both conversion processes. Rather we are guided by two practical tasks that require conversion of the Software Distribution for AIX configuration data:

- Migrating the ODM configuration data to DB2/6000

  As we presented in Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219, there are many advantages of using DB2/6000 for the Software Distribution for AIX configuration data. Because of the wide availability of ODM as part of the AIX base operating system, it is quite possible that you started configuring your Software Distribution for AIX environment with an ODM data model. In order to benefit from the more powerful capabilities of DB2/6000 previously mentioned, install the relational database system and use it further for storing the Software Distribution for AIX configuration data. But the data must be automatically transferred into the new database environment as you intend to use the current Software Distribution for AIX configuration. This method of transferring the whole ODM database to DB2/6000 is described in 14.1, “ODM to DB2/6000 Conversion” on page 277.

- Configuring Software Distribution for AIX agents without DB2/6000 support from a configuration server based on DB2/6000

  In spite of the benefits of using DB2/6000 over ODM, in particular in large distribution networks, it might not be feasible to install DB2/6000 client software on all machines, only for Software Distribution for AIX purposes. One conceivable scenario of using both platforms, DB2/6000 and ODM, in a distribution network is depicted in Figure 153 on page 276.

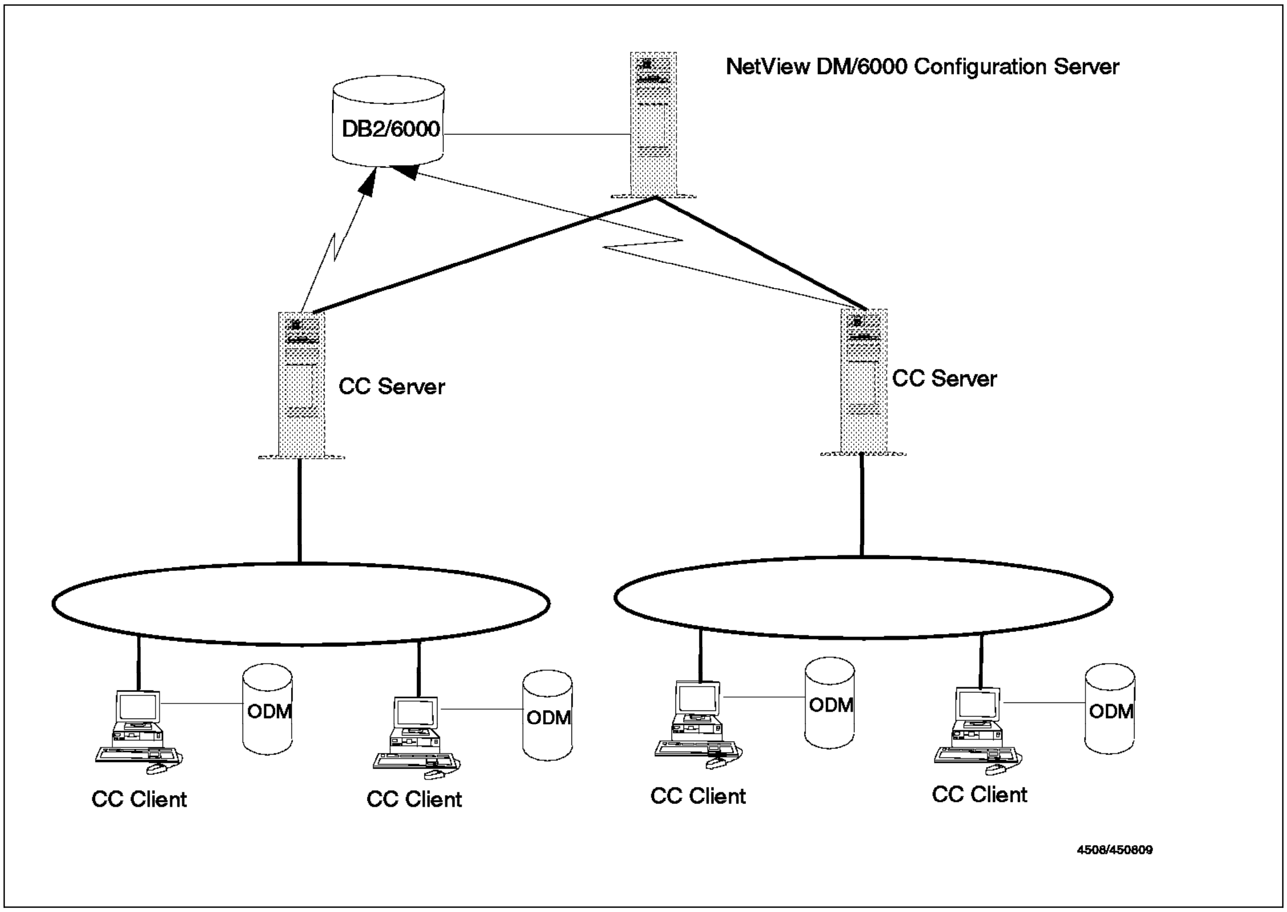


Figure 153. Mixed Use of DB2/6000 and ODM in a Distribution Network

In this case the configuration server holds the DB2/6000 database (that is, it is at the same time database server). The CC servers constitute DB2/6000 clients that access the Software Distribution for AIX configuration database remotely. The CC clients have not installed any DB2/6000 client software, so they are using ODM for storing the configuration data.

Therefore, there is a need to convert configuration data from the DB2/6000 database to ODM and transfer it from the CC server to its clients in the domain. In contrast to 5.3, "Automating the Configuration Process" on page 89, where the whole configuration database is sent to each machine, we describe in 14.2, "Extracting CC Domain Configuration from DB2 to ODM" on page 282 a method for extracting only the relevant data for a particular site, which consists of the information related to the enclosing CC domain. At the end of the chapter we present an automatic distribution and remote configuration procedure for Software Distribution for AIX networks with mixed database support.

Figure 154 on page 277 depicts both conversion processes. While the procedure `odm2db2` moves the whole configuration data from ODM to DB2/6000, the other direction, performed by `db22odm`, needs a machine name as argument to extract only the particular domain information from the DB2/6000 database.

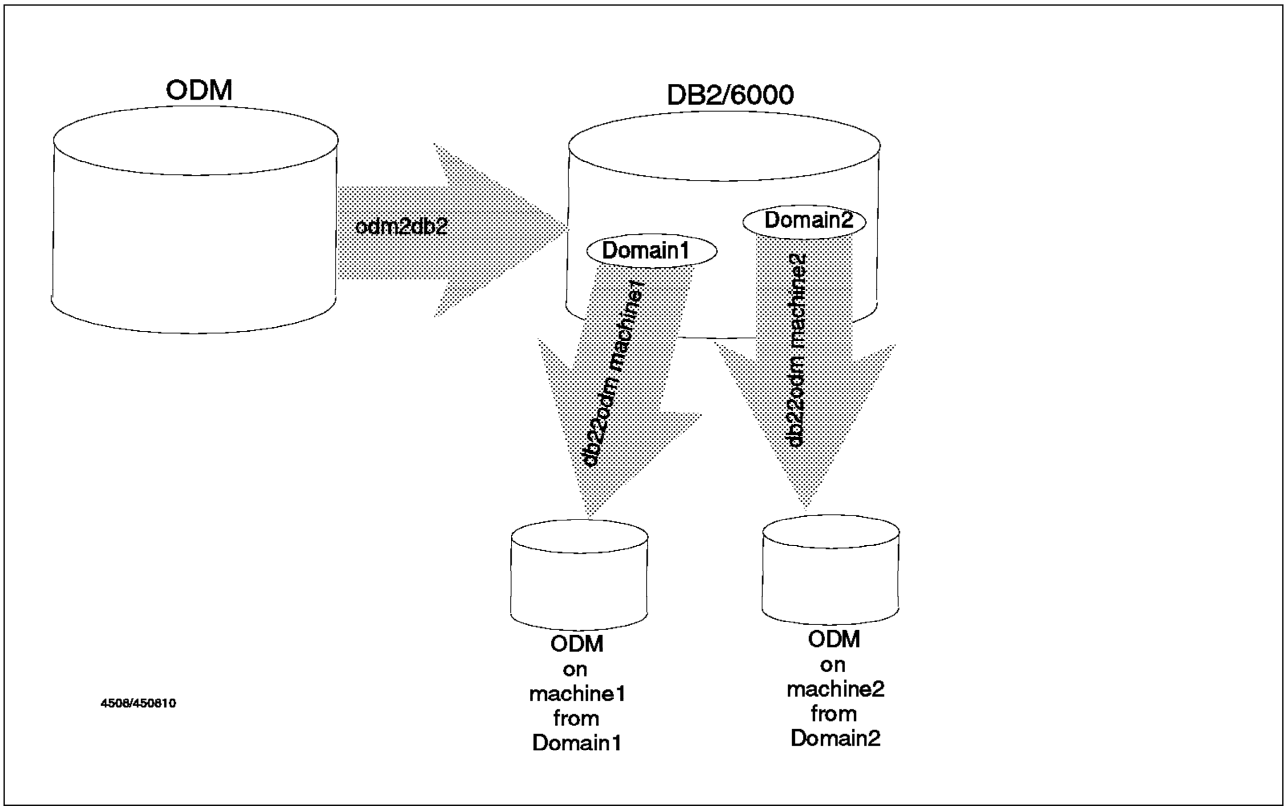


Figure 154. Configuration Data Conversion Processes

14.1 ODM to DB2/6000 Conversion

The first conversion task is performed by the procedure `odm2db2`, which is shown in Figure 155 on page 278. As depicted in Figure 154, this procedure takes all ODM contents related to the Software Distribution for AIX configuration and transforms them into DB2/6000.

```
#!/bin/ksh
#
#
# procedure for converting NetView DM/6000 configuration
# from ODM to DB2/6000
#
#



# convert an ODM class output from odmget to an DB2/6000 insert operator
# $1: ODM class name
#------------------------------------------------------------------------

convert_class()
{

  odmget $1 | awk -v class=$1 '
    BEGIN     {
            FS = " = "
            columns = ""
            values = ""
            q = sprintf("%c",39)
            printf "connect\n"
          }
    $0 == class":" {
            if (columns != "")
              printf "insert into %s (%s) values (%s)\n", class, columns, values
            columns = ""
            values = ""
          }
    !/^$/ && $0 != class":" {
            gsub("\t","",$1)
            gsub("\"","",$2)
            if ($2 == "") $2 = "null"
            else $2 = q $2 q
            if (columns == "") {
              columns = $1
              values = $2
            }
            else {
              columns = columns "," $1
              values = values "," $2
            }
          }
    END  {
            printf "insert into %s (%s) values (%s)\n", class, columns, values
          }
  '
}
```

Figure 155 (Part 1 of 2). ODM-to-DB2/6000 Conversion Script odm2db2

```
# execute convert_class for each NetView DM/6000 ODM class
# and send the output to the DB2/6000 Command Line Processor
# (the execution order of the insert statements is important
#  because of the defined referential constraints)
#------------------------------------------------------------

convert_class nvdm_node
convert_class nvdm_servers
convert_class nvdm_groups
convert_class nvdm_queues
convert_class nvdm_users
convert_class nvdm_cfg_static
```

Figure 155 (Part 2 of 2). ODM-to-DB2/6000 Conversion Script odm2db2

The principle the script `odm2db2` works is the following:

1. Retrieve data from ODM for a particular class with the aid of `odmget`
2. Process the output from the ODM query to build an SQL `insert` statement
3. Apply 1 and 2 to all Software Distribution for AIX configuration classes

The first two tasks are implemented in the procedure `convert_class`. Using `awk` we transform the output from the `odmget` command into an `insert` statement. In order to demonstrate this we take the following ODM object, returned as part of the query `odmget nvdm_users`:

```
 ...

nvdm_users:
     node_name = "rs60004"
     username = "root"
     usergroup = "FNDADMN"

 ...
```

After applying `convert_class` to that input, we get the following `insert` statement as output:

```
insert into nvdm_users (node_name,username,usergroup) \
  values ('rs60004','root','FNDADMN')
```

The following must be considered with regards to the syntactical transformations from ODM to DB2/6000:

- The attribute definition order of the ODM classes does not need to be the same as the column definition order in the appropriate DB2/6000 tables. Hence, we use the long variant of the SQL `insert` statement (that is, with listing the column names).

- String values in ODM are represented by double quotes ("), while in DB2/6000 they are enclosed in single quotes (').
- Null values in ODM appear as empty strings (""), while in the `insert` statement you must provide the `null` value.

After defining the syntactical transformation procedure `convert_class`, we apply it to all ODM classes related to Software Distribution for AIX.

Note

One can argue that the ODM class names could be obtained by the following commands:

```
cd /etc/objrepos
ls nvdm*
```

The reason why we fix the classes `convert_class` is applied to is because of the referential dependencies between the DB2/6000 tables in the model (see Figure 127 on page 228). There must be some knowledge about the data model in the script to determine the order of inserting the data into the tables.

Of course, you can make the script in this respect generic by retrieving this meta information from the system tables. For the sake of simplicity and understandability and as our main emphasis is not put on generic features of the database operations, we left this aspect out of our implementation.

The standard output of `odm2db2` consists of all the `insert` statements needed to fill the DB2/6000 configuration tables. This means that the user has the following possibilities of redirecting the output from the script:

- Redirect to a file:

  ```
  odm2db2 > db_insert.sql
  ```

  For example, you can send the file to another machine and execute it there with the Command Line Processor (for example, `db2 -f db_insert.sql`). This is the case when you migrate the Software Distribution for AIX configuration data from the ODM on one machine to DB2/6000 on another machine.

- Redirect to the Command Line Processor:

  ```
  odm2db2 | db2 -s
  ```

  This is the way we use the conversion procedure in our scenario. We slightly modified our script `build_db` for the creation and re-creation of the Software Distribution for AIX configuration database in DB2/6000 by exchanging the execution of the import script `db_import.sql` with the execution of the ODM-to-DB2/6000 conversion.

The new script is shown in Figure 156 on page 281 and named `build_db_odm` to distinguish from the former. The only difference from `build_db` is in the insertion of the data where the line is highlighted.

```
#
# procedure for building the DB2/6000 configuration database
# (converting data from the ODM database)
#


# creating / recreating the configuration database
#-------------------------------------------------
. ./db_create


# table definitions
#------------------
db2 -sf ./db_model.sql
if [ $? -ne 0 ]
then
  exit 1
fi


# comments
#---------
db2 -sf ./db_comment.sql
if [ $? -ne 0 ]
then
  exit 1
fi


# authorizations
#---------------
db2 -sf ./db_authorize.sql
if [ $? -ne 0 ]
then
  exit 1
fi


# inserting data
#---------------
odm2db2 | db2 -s
if [ $? -ne 0 ]
then
  exit 1
fi

print "\n\nDATABASE CONFIG: Database NVDM_CFG built SUCCESSFULLY!!!\n\n"
exit 0
```

Figure 156. Creating the DB2/6000 Database from ODM (Script build_db_odm)

14.2 Extracting CC Domain Configuration from DB2 to ODM

The extracting of the domain configuration related to a specific Software Distribution for AIX node from the DB2/6000 database is done by the script `db22odm`, which is shown in Figure 157.

```
#!/bin/ksh
#
#
# procedure for building the ODM configuration database from
# the DB2/6000 database for a specific NetView DM/6000 CC domain
# $1: node name
#
#




# check for $1
if [ "$1" = "" ]
then
  print "Please give a node name as argument"
  exit 1
fi


# database access procedure definition
#------------------------------------
. ./DB2





# convert DB2/6000 table into ODM object definition file
# $1: table name
# $2: select predicates (SQL format)
# $3: columns case: ( u[pper] / l[ower] )
#-------------------------------------------------------

convert_table()
{
  WHERE=$2
  if [ "$WHERE" = "" ]
  then
    WHERE="1=1"
  fi

  TBNAME=`echo $1|awk '{name = toupper($0); print name}'`
  COLS=`get_data name,colno sysibm.syscolumns "tbname = '$TBNAME' order by colno" | awk '{print
$1}'`
  COLCNT=`echo $COLS | wc -w | sed 's/ //g'`
  LENGTHS=`get_data length,colno sysibm.syscolumns "tbname = '$TBNAME' order by colno" | awk '{print
$1}'`
  COLS=`echo $COLS`
  LENGTHS=`echo $LENGTHS`
```

Figure 157 (Part 1 of 3). Extracting Domain Configuration Related to a Specific Host (Script db22odm)

```
  get_data \* $DBOWNER.$1 "$WHERE" | awk \
    -v case=$3 -v class=$1 -v colstr="$COLS" -v colcnt=$COLCNT -v lenstr="$LENGTHS" '
        BEGIN   {
           if (case   /^l/) colstr=tolower(colstr)
           split(colstr,columns)
           split(lenstr,lengths)
           for (i=1;i<=colcnt;i++) {
             cl = length(columns[i])
             if (cl > lengths[i]) lengths[i] = cl
           }
         }
         {
           n=1
           printf "%s:\n", class
           for (i=1;i<=colcnt;i++) {
             l = lengths[i]
             v = substr($0,n,l)
             n = n + l + 1
             sub(/^ */,"",v)
             sub(/ *$/,"",v)
             sub(/^-$/,"",v)
             if (v !  /^[0-9]+$/ ) v = "\"" v "\""
             printf "\t%s = %s\n" , columns[i], v
           }
         }
'

}



# connect to the database
#---------------------------
db2 connect >/dev/null 2>&1


# get the server of the node
#--------------------------------------------
SERVER=`get_data server_name $DBOWNER.nvdm_node "node_name = '$1'"`
if [ "$SERVER" = "" ]
then
  print "DATABASE CONVERT: node $1 has no server entry."
  exit 1
fi
```

Figure 157 (Part 2 of 3). Extracting Domain Configuration Related to a Specific Host (Script db22odm)

```
# get all nodes in the CC domain
#-------------------------------
NODES=`get_data node_name $DBOWNER.nvdm_node "server_name = '$SERVER'"`
NODES=\'`echo $NODES|sed "s/ /','/g"`\'



# building the ODM object definition output and inserting the data
#------------------------------------------------------------------
  print "DATABASE CONVERT : creating file nvdm_node.odmadd"
  convert_table nvdm_node "server_name = '$SERVER'" "l" > nvdm_node.odmadd
  print "DATABASE CONVERT : creating file nvdm_servers.odmadd"
  convert_table nvdm_servers "node_name = '$SERVER'" "l" > nvdm_servers.odmadd
  print "DATABASE CONVERT : creating file nvdm_groups.odmadd"
  convert_table nvdm_groups "node_name = '$SERVER'" "l" > nvdm_groups.odmadd
  print "DATABASE CONVERT : creating file nvdm_queues.odmadd"
  convert_table nvdm_queues "node_name = '$SERVER'" "l" > nvdm_queues.odmadd
  print "DATABASE CONVERT : creating file nvdm_users.odmadd"
  convert_table nvdm_users "node_name in ($NODES)" "l" > nvdm_users.odmadd
  print "DATABASE CONVERT : creating file nvdm_cfg_static.odmadd"
  convert_table nvdm_cfg_static "" "u" > nvdm_cfg_static.odmadd


exit 0
```

Figure 157 (Part 3 of 3). Extracting Domain Configuration Related to a Specific Host (Script db22odm)

In contrast to the ODM-to-DB2/6000 database conversion where the whole Software Distribution for AIX configuration is affected (see 14.1, “ODM to DB2/6000 Conversion” on page 277), the script `db22odm` requires a node name as an argument. Based on the node entry, the procedure determines all of the domain relevant data that must be transferred from ODM to DB2/6000.

The following presents the domain relevant data for a given node argument NODE (see Figure 132 on page 236 for the database definition):

- The server SERVER from NODE (the row from `nvdm_servers` corresponding to the `server_name` entry for NODE in `nvdm_node`)
- All targets NODES with the server SERVER (all rows from `nvdm_node` with `server_name = SERVER`)
- All target groups maintained on SERVER (all rows from `nvdm_groups` with `node_name = SERVER`)
- All server-to-server connections to be configured on SERVER (all rows from `nvdm_queues` with `node_name = SERVER`)
- All users on the targets determined above (all rows from `nvdm_users` with `node_name` equal to any target from the set NODES)
- All network common data (the whole `nvdm_cfg_static` table)

First, the script `db22odm` determines the server of the given node argument (`$1`) SERVER and the set NODES of all targets managed on that server.

Note

The obtained node name of the server matches the name of the argument (`$1`) in the case when the latter is a Software Distribution for AIX server. This fact does not affect our algorithm of determining the domain relevant data.

Thereafter, the procedure `convert_table` is applied on the configuration tables creating ODM object data files as required by the AIX command `odmadd`. The further use of these files in a concrete application is described in 14.3, "Remote Software Distribution for AIX Configuration with Different Database Support" on page 286.

The procedure `convert_table` is the opponent of `convert_class` (see 14.1, "ODM to DB2/6000 Conversion" on page 277) and performs the syntactical conversion of the SQL output into the format of an ODM object definition, which is the same as provided by the AIX command `odmget` (see the previous section). The additional capability of `convert_table` consists of the ability to provide selection criteria for the processed table. In general, it applies the procedure `get_data` (see Figure 139 on page 254) and builds the appropriate output in `odmadd` format with the aid of `awk`.

In order to enable the automatic editing of the selected results, the procedure first determines the column lengths of the inspected table. This is done by applying `get_data` on the system column `sysibm.syscolumns`, which contains the column definition data for each DB2/6000 table. Here we make use of the `order` clause of the SQL `select` statement to build two ordered lists, COLS and LENGTHS, containing the column names and their respective lengths. They are passed to `awk` where each SQL result row is split into the column values according to the column lengths.

Note

When the length of the column name is greater than the maximum length of the column value, the value output length does not match the column length information from `sysibm.syscolumns`.

SQL formats the output according to the maximum of both length values. This case is treated in the BEGIN part of the `awk` operation.

During the syntactical transformation, attention must be paid to the following format specifics:

- The ODM object definition format requires the enclosing of strings in double quotes ("), while integer values are assigned unchanged. But the value types are not distinguishable from the SQL output. As the ODM configuration model uses both types (see Chapter 3, "Designing a Data Model for Configuration Data" on page 11), we leave the numerical values unchanged and put all other values in double quotes.
- The null values appearing in the SQL output as dashes (-) must be converted into the ODM representation for null values: the empty string ("").

14.3 Remote Software Distribution for AIX Configuration with Different Database Support

Similar to 5.3, “Automating the Configuration Process” on page 89 we now want to enable the automatic remote configuration of the Software Distribution for AIX nodes from the configuration server. While the procedure `configure_network` treats only the case of ODM configuration database, we present in this section an enhanced procedure `configure_network_univ` that covers both cases of database use. It is shown in Figure 158.

```
#!/bin/ksh
#
# Copy Configuration to all Nodes and execute configuration script
# Depending upon whether the particular node is using an ODM or DB2/6000
# configuration database the appropriate configuration script is sent
# to it. The latter information is contained in the new nvdm_node field
# config_db ("DB2" or "ODM")
# For this to work each system to be configured has to have
# an entry for the central installation system in it's /.rhosts file
# The script requires two parameters:
#   $1: local path of the DB2/6000 version of the automatic configuration procedure
#   $2: local path of the ODM version of the automatic configuration procedure
# Author : Plamen Kiradjiev
#

print "**** CONFIGURING NETVIEW DISTRIBUTION MANAGER/6000 ****"


MYDIR=`pwd`
DB2PATH=$1
ODMPATH=$2


# determine the version used by the initiating site
case $MYDIR in
  $DB2PATH) INITVERSION=DB2
        . $DB2PATH/DB2 ;;
  $ODMPATH) INITVERSION=ODM
        . $ODMPATH/ODM
        print "** Rebuilding the ODM database"
        ./rebuild_db ;;
  *)        print "CONFIG NVDM: Cannot determine the version (DB2 or ODM) of the initiating machine."
        print "            enter \$1: DB2/6000 version full path"
        print "            enter \$2: ODM version full path"
            exit 1  ;;
esac
```

Figure 158 (Part 1 of 3). Automatic Remote Configuration of NetView DM/6000 (Script configure_network_univ)

```
LIST=`cat node_list`
for i in $LIST
do
   print "*** Processing node : $i"
   get_attribute nvdm_node node_name $i config_db
   VERSION=$VALUE
   if [ "$VERSION" = "" ]
   then
     VERSION=ODM
   fi
   case $INITVERSION in
# conversion from DB2/6000 to ODM if necessary
     DB2) if [ ! $VERSION = DB2 ]
      then
        print "** Extracting domain relevant data for $i from DB2/6000"
        cd $DB2PATH
        ./db22odm $i
        ls $DB2PATH/*odmadd|xargs -i basename {}|xargs -i mv -f {} $ODMPATH/{}
      fi
      ;;
# in the case of using ODM the ODM database is used in both cases
     ODM) VERSION=ODM
      ;;
   esac
# build appropriate tar archive and compress it
   print "** Creating tar archive for the $VERSION version"
   VPATH=$`echo $VERSION`PATH
   cd `eval echo $VPATH`
   tar -chvf/tmp/nvdm_$VERSION.tar . >/dev/null
   SIZE=`ls -l /tmp/nvdm_$VERSION.tar | awk '{ print $5 }'`
   print "Size before compressing : $SIZE"
   print "** Crunching tar archive"
   rm /tmp/nvdm_$VERSION.tar.Z 2>/dev/null
   compress /tmp/nvdm_$VERSION.tar
   SIZE=`ls -l /tmp/nvdm_$VERSION.tar.Z | awk '{ print $5 }'`
   print "Size after compressing : $SIZE"
# send the code to the desired machine, unpack and execute it there
   print "** Copy compressed archive"
   rcp /tmp/nvdm_$VERSION.tar.Z $i:/tmp/nvdm.tar.Z
   print "** Uncrunching compressed archive"
   rsh $i uncompress -f /tmp/nvdm.tar
   print "** Extracting files from tar archive"
   rsh $i "cd /tmp ; tar -xvf/tmp/nvdm.tar 1>/dev/null 2>&1"
   if [ $VERSION = ODM ]
   then
     print "** Creating ODM DB ..."
     rsh $i /tmp/build_net_db2
   fi
   print "** Invoking configuration script..."
   rsh $i ". ./.profile; /tmp/config_nvdm $i"
done
```

Figure 158 (Part 2 of 3). Automatic Remote Configuration of NetView DM/6000 (Script configure_network_univ)

```
cd $MYDIR

# rebuilding the ODM database in the case $INITVERSION = ODM
if [ $INITVERSION = ODM ]
then
  print "*** Rebuilding ODM database"
  ./rebuild_db
fi


exit 0
```

Figure 158 (Part 3 of 3). Automatic Remote Configuration of NetView DM/6000 (Script configure_network_univ)

For the purpose of distinguishing between remote nodes with different configuration storage methods, we introduce a new column (attribute) `config_db` for the DB2/6000 table (ODM class). Its value can be either DB2 or ODM.

Note

Nodes containing null values in this column (attribute) are assumed to support ODM. Thereby, the compatibility with configuration data from the former data models is guaranteed (see Chapter 3, "Designing a Data Model for Configuration Data" on page 11 and Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219).

The following shows the new `nvdm_node` definitions:

ODM: in file `config_db2_remote.cre`

```
class nvdm_node {
  char  node_name[25];
  short node_type;
  char  short_name[9];
  char  target_os[12];
  char  description[25];
  char  contact_name[25];
  char  owning_manager[25];
  char  telephone_number[20];
  char  customer_name[20];
  char  repos_fs[4];
  long  repos_size;
  char  x_25_number[15];
  char  server_name[25];
  link  nvdm_groups nvdm_groups group_name group_name;
  char config_db[4];
}
```

DB2/6000: in file db_model2.sql

```
create table nvdm_node \
    (node_name char(25) not null primary key, \
     node_type char(1) not null, \
     short_name char(9) not null, \
     target_os char(12), \
     description char(25), \
     contact_name char(25), \
     owning_manager char(25), \
     telephone_number char(20), \
     customer_name char(20), \
     repos_fs char(4), \
     repos_size char(20), \
     x_25_number char(15), \
     server_name char(25) not null, \
     group_name char(25), \
     config_db char(3), \
     foreign key r_server (server_name) \
       references nvdm_node \
       on delete cascade )
```

Considering the database support on the configuration server and the remote node, the following combinations are possible:

- ODM on the configuration server, ODM on the remote node

  This case is the same as in 5.3, "Automating the Configuration Process" on page 89 where the whole ODM database is sent to the remote node.

- ODM on the configuration server, DB2/6000 on the remote node

  This case is treated in the same way as the case above by ignoring the DB2/6000 capability of the remote node.

  Note

  It could be conceivable to make conversion in this database combination too, for example when the configuration server does not support DB2/6000 but the CC servers do (see Figure 153 on page 276). Because of the rareness of this case we do not consider it.

- DB2/6000 on the configuration server, DB2/6000 on the remote note

  No configuration data is transferred in this case. The remote node acts as a DB2/6000 client and queries the database remotely from the configuration server.

- DB2/6000 on the configuration server, ODM on the remote node

  Here we apply the DB2/6000-to-ODM conversion described in 14.2, "Extracting CC Domain Configuration from DB2 to ODM" on page 282. In contrast to the ODM-ODM combination, we extract the domain relevant data for the particular node and transfer only this part of the configuration database to the ODM of the remote node.

The script configure_network_univ requires two arguments that represent the paths for the DB2/6000 respectively ODM version of the automatic configuration procedure, locally on the initiating machine. Depending on from which of both paths the remote configuration script `configure_network_univ` is called, the variable INITVERSION is set to DB2 or ODM. The remote node database support is queried from the configuration database (field `config_db` in `nvdm_node`) and stored in the variable VERSION.

Both variables, INITVERSION and VERSION, determine the particular database version combination as explained above. According to INITVERSION, the appropriate database access procedure definitions are included in the script (from the file DB2 or ODM). Based on VERSION, the appropriate version of the automatic configuration procedure undergoes the following steps for each remote node:

- Building a `tar` archive and compressing it locally on the initiating machine (the configuration server)

  In the DB2/6000-to-ODM case we first apply `db22odm` to extract the domain configuration information for the particular node and include the created ODM object definition files (with extension `odmadd`) into the `tar` archive.

- Sending the package to the remote node to be configured (`rcp`)

  In order to be able to execute the `rcp` and the following `rsh` commands, you must ensure the presence of the configuration server entry in the `.rhosts` file in `root`'s home directory on each remote node (see 5.3, "Automating the Configuration Process" on page 89).

- Decompressing and unarchiving on the remote node (`rsh`)
- Initiating the configuration procedure on the remote node (`rsh`)

  In the case of the ODM version on the remote node for the NetView DM/6000 configuration, the creation of the ODM database is required before running the configuration script `config_nvdm`. For the DB2/6000-DB2/6000 combination, no database creation is needed, provided that it is configured as a DB2/6000 client (see installation instructions in Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219).

Figure 159 on page 291 shows a sample output from executing the script `configure_network_univ` for two remote machines, `rs60004` and `rs600011`. According to 5.3, "Automating the Configuration Process" on page 89, we maintain a file `node_list` in each version directory on the initiating site with the names of the remote machines to be configured.

In our example, the initiating machine is using DB2/6000 for storing the configuration data. The node `rs60004` is configured as a DB2/6000 client, while `rs600011` supports only ODM (with respect to the values of `config_db` in the appropriate `nvdm_node` rows).

The script for automatic remote NetView DM/6000 configuration is called in the following way:

```
cd /4508code/db_version_2
 ./configure_network_univ /4508code/db_version_2 /4508code/version_2 2>& \
    | tee logfile_net
```

Note

It is important to change the current directory to the appropriate version directory (in our example, `/4508code/db_version_2` for the DB2/6000 version on the initiating machine) because the script determines the local database version based on the current directory from where it is called.

The two arguments represent the directories where we store the particular versions of the configuration procedure on our sample initiating machine.

```
**** CONFIGURING NETVIEW DISTRIBUTION MANAGER/6000 ****
*** Processing node : rs60004
** Creating tar archive for the DB2 version
Size before compressing : 194560
** Crunching tar archive
Size after compressing : 57013
** Copy compressed archive
** Uncrunching compressed archive
** Extracting files from tar archive
** Invoking configuration script...

 ...

*** Processing node : rs600011
** Extracting domain relevant data for rs600011 from DB2/6000
DATABASE CONVERT : creating file nvdm_node.odmadd
DATABASE CONVERT : creating file nvdm_servers.odmadd
DATABASE CONVERT : creating file nvdm_groups.odmadd
DATABASE CONVERT : creating file nvdm_queues.odmadd
DATABASE CONVERT : creating file nvdm_users.odmadd
DATABASE CONVERT : creating file nvdm_cfg_static.odmadd
** Creating tar archive for the ODM version
Size before compressing : 122880
** Crunching tar archive
Size after compressing : 42193
** Copy compressed archive
** Uncrunching compressed archive
** Extracting files from tar archive
** Creating ODM DB ...
nvdm_groups
nvdm_node
nvdm_users
nvdm_cfg_static
nvdm_servers
nvdm_queues
** Invoking configuration script...

 ...
```

Figure 159. Output Log for the Execution of configure_network_univ

Chapter 15. Modifying Configuration Data Using a Graphical User Interface

We described in Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219 a procedure for the automatic generation of the Software Distribution for AIX configuration database under DB2/6000. There we took the data from import files in DEL format. This approach might be convenient for tasks such as exchanging data between DRDA-compliant platforms, converting data between different database systems or automating the data generation from user scripts. But there is no possibility for the NetView DM/6000 system administrator to access respectively change configuration data in a sensible way.

The following table summarizes the different approaches we have used to access the NetView DM/6000 configuration database in ascending order with regard to their user friendliness:

Table 2. Comparison between Different Access Approaches to the Configuration Data

Data Access Approach	Advantages	Disadvantages
Import Files	• Data transfer between DRDA-compliant databases • Data conversion between non-DRDA-compliant databases • Automatic database generation from external applications	• Not suited for data access • Not recommended for data update • Data model knowledge required
ODM-to-DB2/6000 Conversion	• Conceivable in the case of porting the configuration data from ODM to DB2/6000 • `odme` can be used to prepare the data before converting	• One-way approach • Not suited for the work with DB2/6000 after the conversion
Interactive SQL	• Most powerful approach for querying as well as updating data • Full SQL power available • Suited to be called from application scripts	• SQL skill required • Data model knowledge required • Not convenient for numerous single row updates
Visualizer Query/6000	• The most user friendliest • Plenty of different ways for browsing data (queries, views, reports) • Close to SQL in power (allows even model changing)	• Not possible to define user actions • Referential dependences of the data model hidden for the "conventional" database user • Not possible to build own model dependent forms

The presented data access approaches have been used in different contexts throughout this book. For the automatic creation of the Software Distribution for AIX configuration database we employed the import files. In Chapter 14, “Converting the Data Model between ODM and DB2/6000” on page 275 the second and the third approach apply. In the case of porting the configuration database from ODM to DB2/6000, `odme` can be used to perform some preparation changes. Thereafter, SQL `insert` statements are generated and executed from the script `build_db_odm` (see Figure 156 on page 281).

The Command Line Processor `db2` is extensively used by the scripts for database creation, access and conversion respectively, by the database access procedures (see Chapter 12, “Implementing the Configuration Data Model Using DB2/6000” on page 219). interactive SQL offers a good means for global changes of the configuration data in the following manner:

```
update nvdm_users set usergroup = 'FNDADMN' where username = 'wolfgang'
```

The above SQL statement sets the user group of the user `wolfgang` to FNDADMN on all targets in the distribution network.

In contrary for single row updates, the Command Line Processor appears a bit awkward to a system administrator, who is required to reenter a long SQL statement for each particular row to be changed. In this case a graphical user interface such as the Visualizer Query/6000 is more suited. Rather, it offers a convenient means for browsing and updating data, report generation, and even data model changes.

Although being so powerful and user friendly, the Visualizer Query/6000 could not fulfill our requirements to supply the system administrator with a tool that supports the referential integrity of the Software Distribution for AIX configuration model. Especially, it should provide a consistent way of inserting data into the database tables without having to worry about the existing dependencies between them. At the same time it must be user friendly and comfortable enough, so that the system administrator can abstract from the SQL level and the data model.

In this chapter we present a tool, that we implement in order to supply the Software Distribution for AIX system administrator with a convenient access to the configuration database as well as with a model-dependent, guided way of inserting data into the configuration tables. In the first part of the chapter we describe the appearance and use of the graphical interface to the Software Distribution for AIX configuration database. The second part deals with the design principles and some implementation details of the tool as well as pointing out its extensibility.

As the graphical interface is written in C, using on the one hand the Call Level Interface of DB2/6000 and on the other hand the Motif 1.2 libraries, we assume the reader interested in the implementation specifics of our interface has a particular background in these topics. We recommend the following reference literature:

- *DB2 Call Level Interface Guide and Reference (DATABASE 2)*, SC09-1626 - for the Call Level Interface functions of DB2/6000
- *AIXwindows Programming Guide*, SC23-2632 - for the Motif programming principles.

Appendix C, “Source Code of the Graphical User Interface” on page 419 contains the source code of the graphical user interface to the NetView DM/6000 configuration database.

Other readers, after getting acquainted with the usage of the graphical interface, can skip the implementation description without loss of information.

15.1 Using the Graphical Interface for Changing Configuration Data

The package of the graphical user interface to the Software Distribution for AIX configuration database consists of the following files:

- `dbAccess.h`: generic database access interface file li.`dbAccessDB2.c`: DB2/6000 access implementation file
- `dbFrames.h`: database frames interface file
- `dbFrames.c`: database frames implementation file
- `makefile`: make file for compiling the application
- `uicfgdb`: main program

In order to enable the compilation of the graphical user interface, the following software must have been installed on your workstation:

- C for AIX Compiler
- IBM AIX DATABASE 2 Software Developer's Kit/6000
- AIXwindows Application Development Toolkit Motif

After executing `make` the graphical user interface is compiled and ready to use. The produced executable file is `uicfgdb`.

Note

In order to enable other AIX users to call the graphical user interface, you should set the appropriate executable rights in the following manner:

```
chmod +x uicfgdb
```

Although the above command allows all AIX users to execute `uicfgdb`, there is hardly any risk of unauthorized database access since DB2/6000 controls the authorizations by itself.

According to our scenario from Chapter 12, "Implementing the Configuration Data Model Using DB2/6000" on page 219, you should run the graphical interface to the Software Distribution for AIX configuration database as the instance owner `dbmsadm` in order to be able to make changes. The user `root` has only read rights on the configuration tables. The tool is based on the client authentication type, so that the connection to the configuration database is made transparently without asking for a user name and a password.

After executing `uicfgdb`, the following window appears on the screen:

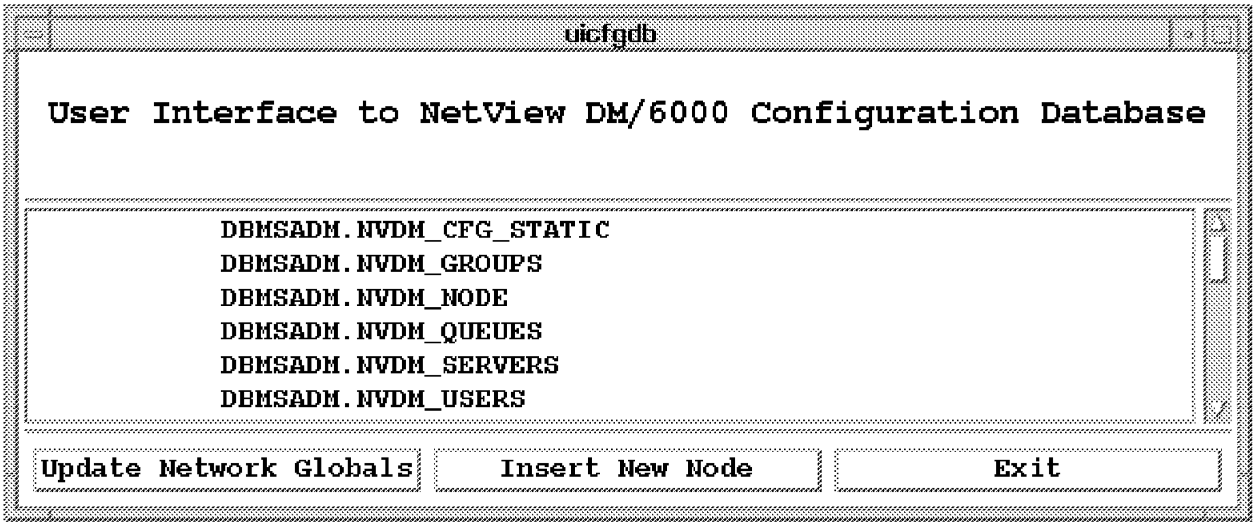


Figure 160. Main Window of the Graphical User Interface

If the database connection request fails, an error message is displayed and the application is exited. This may occur, for example, when the database manager is not running on the database server or the client is not correctly configured.

The main window of the graphical user interface lists all Software Distribution for AIX configuration tables including the system tables (with prefix SYSIBM). By double-clicking on a particular item from the list you can get a table view of the contained data. The buttons define model-specific actions such as updating the global distribution network parameters or inserting a new target into the database. These actions are described in the following sections.

15.1.1 Updating Network Global Parameters

Clicking on the button **Update Network Globals** leads to the table frame of `NVDM_CFG_STATIC` depicted in the following figure:

Note

In this chapter we are using the names of database objects in uppercase as they appear in the application frames. This is because the object names are taken from the DB2/6000 system tables where they are stored in uppercase. SQL does not distinguish between uppercase and lowercase.

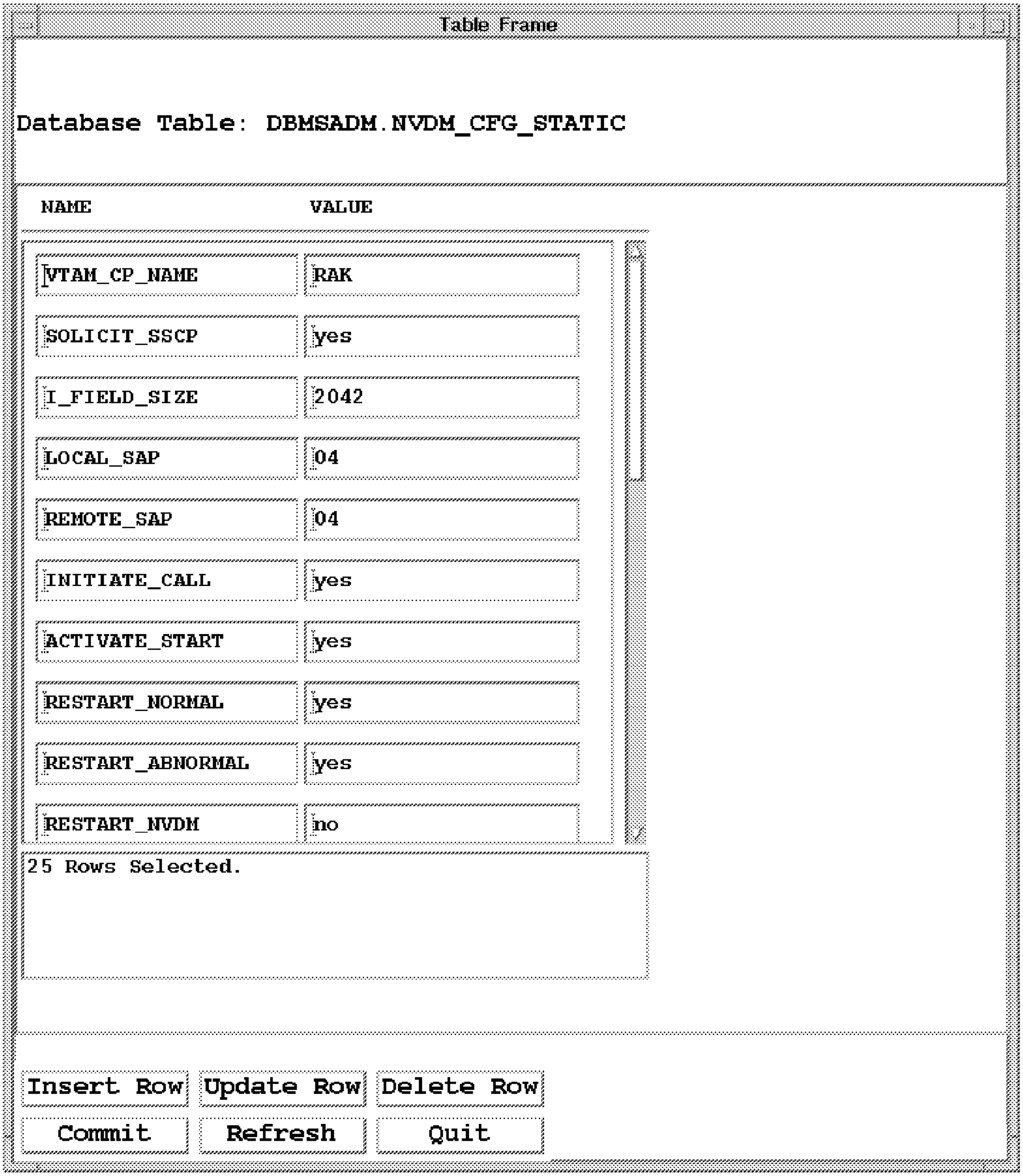


Figure 161. Updating Distribution Network Global Information (Table NVDM_CFG_STATIC)

The table frame allows the user to insert, update or delete rows from the table with the network global parameters. Since this table is not involved in any referential dependencies (see Figure 127 on page 228), you can update it separately without affecting the data integrity.

After the initial display of the window, the input focus is positioned on the first table row (if any). Generally the current row (the one with the input focus on one of its fields) is highlighted. The database messages and warnings are displayed below the table data. The six buttons at the bottom of the table frame define the following actions:

Inserting a new row:

After clicking on the button **Insert Row** a new empty row is generated at the bottom of the table and the input focus is moved to this row. You can then enter the data for the new table row.

Updating the current row:

Initially the table frame is in browse mode. That is, you cannot alter the field data having positioned the cursor on a particular field, even if you have update rights for the table. This is done to avoid an inadvertent altering of the row data.

In order to switch to update mode for the current row, you must click on the button **Update Row**. Thereafter you are able to update the fields of that row, until you move to another row. Then, the browse mode is restored again.

Deleting a row:

When clicking on the button **Delete Row**, the current row disappears from the table frame and the input focus is moved to the next available row (trying first to move to the next row below the deleted one).

Committing the work:

After executing the actions above, only the data viewed in the table frame is changed. In order to perform the appropriate database changes, you must click on the button **Commit**. The initiated action updates the contents of the processed database table in one transaction according to the data in the frame and refreshes the table frame.

Refreshing the table frame:

Click on the button **Refresh**. This rereads the database contents and corresponding changing of the data depicted in the table frame. This action is sensible in two cases: rolling back to previous database contents after some changes in the table frame or refreshing the table frame after changes made from other forms (for example, after a node insertion).

Quitting the table form:

Clicking on the **Quit** button leaves the table form without any other actions. That is, the database contents are not affected. In order to take over the changes made in the table frame, you must commit before quitting the frame.

The described actions above apply to all the database tables, since the table frame is created by a standard function provided by the database frames' interface (see 15.2, “Implementation Insights” on page 306).

In this respect our graphical user interface offers a way of browsing databases in the same manner as the Visualizer Query/6000. Of course, the tool cannot be expected to provide the full functionality of the product (see 15.2.4, “Features of the Graphical Interface Program” on page 314 for details).

15.1.2 A Guided Way for Inserting New Software Distribution for AIX Nodes

Our main goal when designing the graphical user interface to the Software Distribution for AIX configuration database was to provide a consistent and comfortable way for inserting data according to the referential dependences of the presented data model. DB2/6000 controls the data integrity of the database and the obeying of the defined table dependences.

For example, when trying to insert a new user for a non-existing node, the following DB2/6000 error message occurs and the `insert` fails:

```
DB21034E  The command was processed as an SQL statement and returned:
SQL0530N  The insert or update value of FOREIGN KEY "R_NODE" is not equal to
some value of the primary key of the parent table.  SQLSTATE=23503
```

This error occurs independently of the database access approach (by import files, interactive SQL, Visualizer Query/6000 or our graphical interface). Therefore, a particular sequence by the inserting of the configuration information is required. This sequence is determined by the referential dependencies between the configuration tables, starting from the parent tables to the dependent tables. As shown in Figure 127 on page 228, the table `NVDM_NODE` is the parent table of all and should be the start of inserting new configuration data. This appears reasonable from the intuitive point of view too, as the information in the remaining tables, `NVDM_SERVERS`, `NVDM_QUEUES`, `NVDM_GROUPS` and `NVDM_USERS` is always tied to a particular node (target).

Thus we provide a consistent and intuitive method of entering a new node and the information related to it. The user does not need to be aware of the referential dependencies of the data model. It is guided by the application through the tables that need information inserted into them in a data model determined sequence.

After clicking on the button **Insert New Node**, the following dialog box appears on the screen:

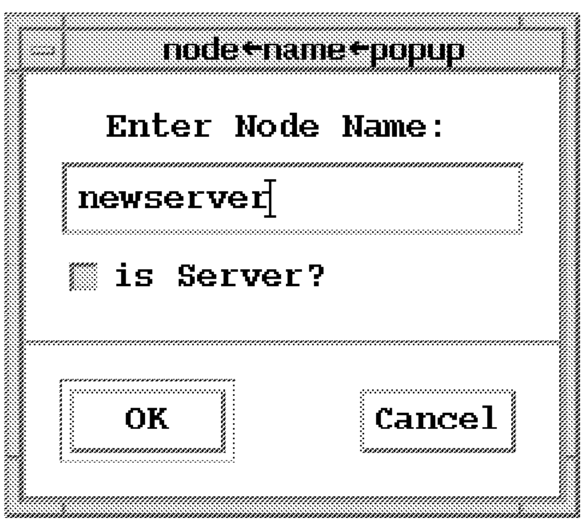


Figure 162. Entering a New NetView DM/6000 Node Name

In this dialog box you are required to enter the name of the new node and determine whether it will be a Software Distribution for AIX server. Depending on the latter information you pass a different number of insert table frames:

Server case:

1. NVDM_NODE insert frame
2. NVDM_SERVERS insert frame
3. NVDM_QUEUES insert frame
4. NVDM_GROUPS insert frame
5. NVDM_USERS insert frame

Client case:

1. NVDM_NODE insert frame
2. NVDM_USERS insert frame

In our example we are using the server case and we click on the toggle button **is Server?**.

After clicking on the **OK** button, the insert frame for the table NVDM_NODE appears on the screen. Figure 163 on page 301 and Figure 164 on page 302 depict this insert frame and the example data for the node newserver.

Note

The first field NODE_NAME cannot be changed. It always appears when a new insert is prepared and is based on the name you entered from the initial dialog box.

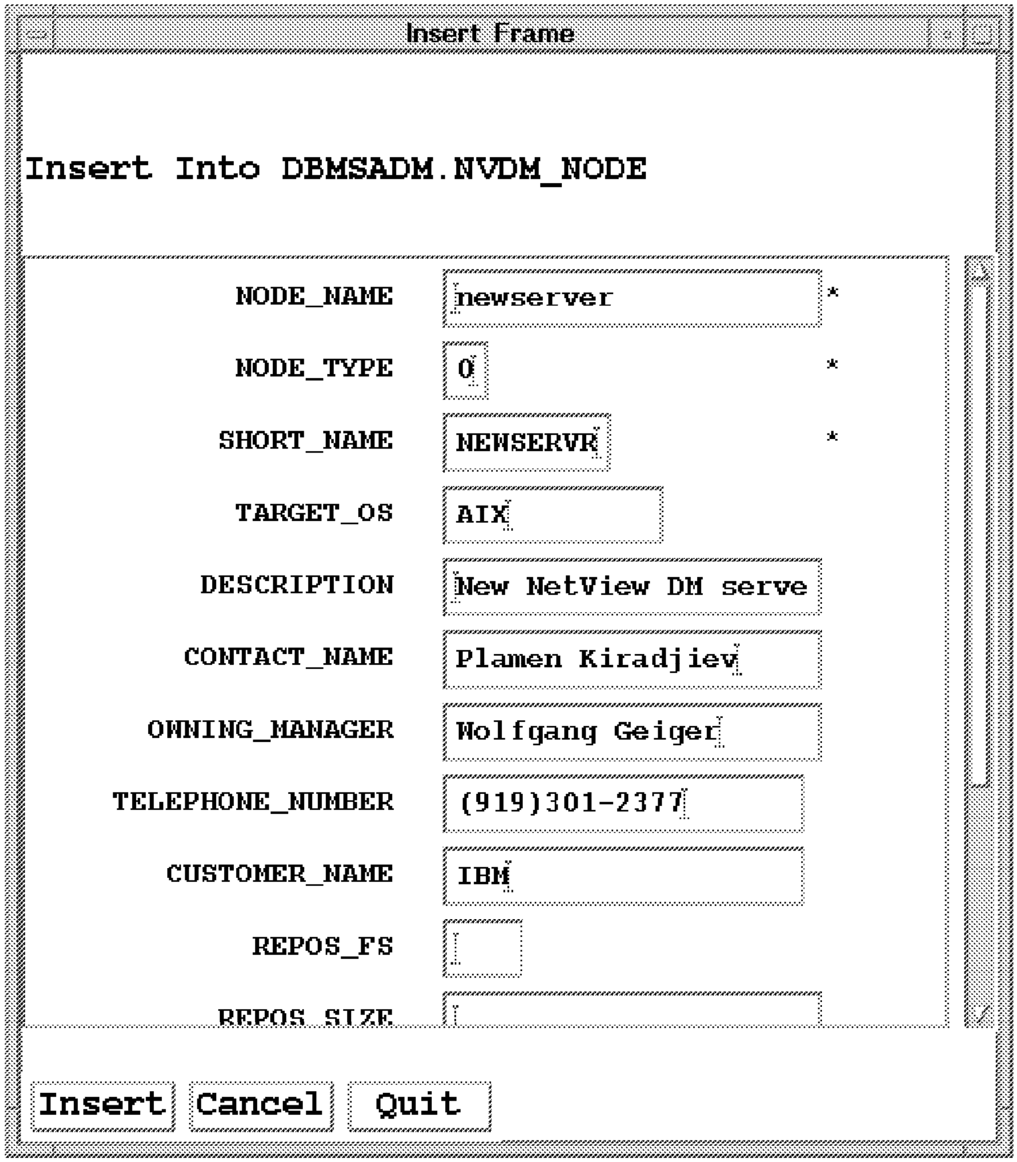


Figure 163. Inserting the Node-Specific Data into Table NVDM_NODE (Part 1)

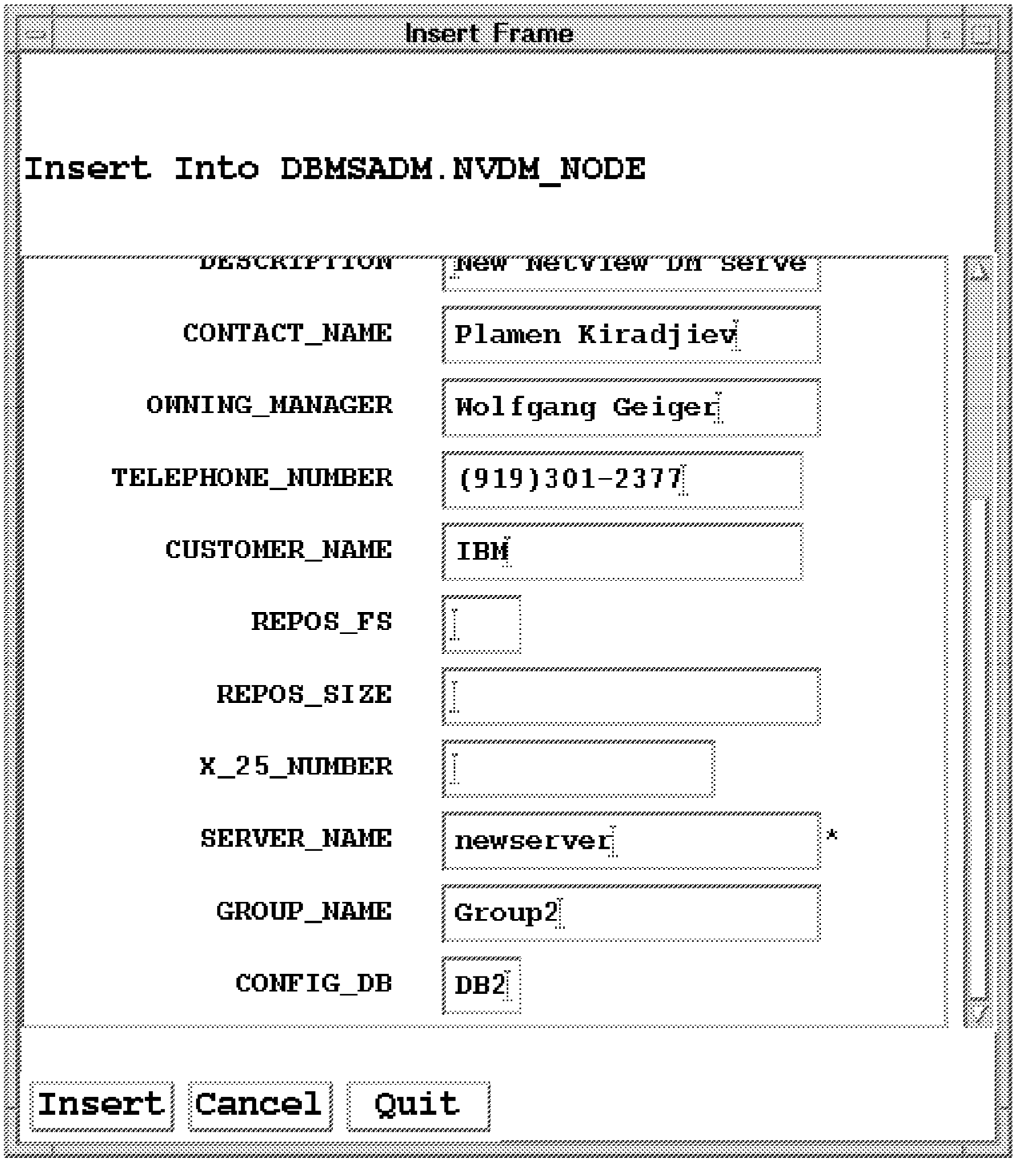


Figure 164. Inserting the Node-Specific Data into Table NVDM_NODE (Part 2)

From the insert frame you can perform the following actions:

Inserting the data filled in the frame:

Click on the **Insert** button. The database contents are changed and the frame is refreshed by clearing the fields except from the `NODE_NAME`.

Canceling the insertion:

Click on the **Cancel** button. The frame is refreshed by leaving the NODE_NAME field.

Proceed with inserting further information:

Click on **Quit**. You leave the insert frame for the present database table and pass to the next table in the insertion sequence (if any).

Note

If you want the last field information to be inserted into the processed table, you must click on the **Insert** button before quitting the insert frame.

Similar to the table frame, the insert frame is a standard window provided by the database frames' interface `dbFrames.h` (see 15.2, "Implementation Insights" on page 306). Thus, the above actions apply to all accessed tables from the configuration database. For all of these actions the node name is filled in previously from the application based on the name you entered in the dialog box. It is impossible to change this name from the insert frame.

After inserting data into `NVDM_NODE`, you can go to the next table insertion by clicking on **Quit**. Since `NODE_NAME` constitutes the primary key of `NVDM_NODE` on its own and you cannot alter it in the insert frame, you are not allowed to insert into the table `NVDM_NODE`. In such a way you are guided to enter all the information related to the particular node before switching to another.

The next insert frame in our example is the one for the table `NVDM_SERVERS`, since we specified `newserver` to be a NetView DM/6000 server. The following figure shows the initial frame that appears with the node name fixed again:

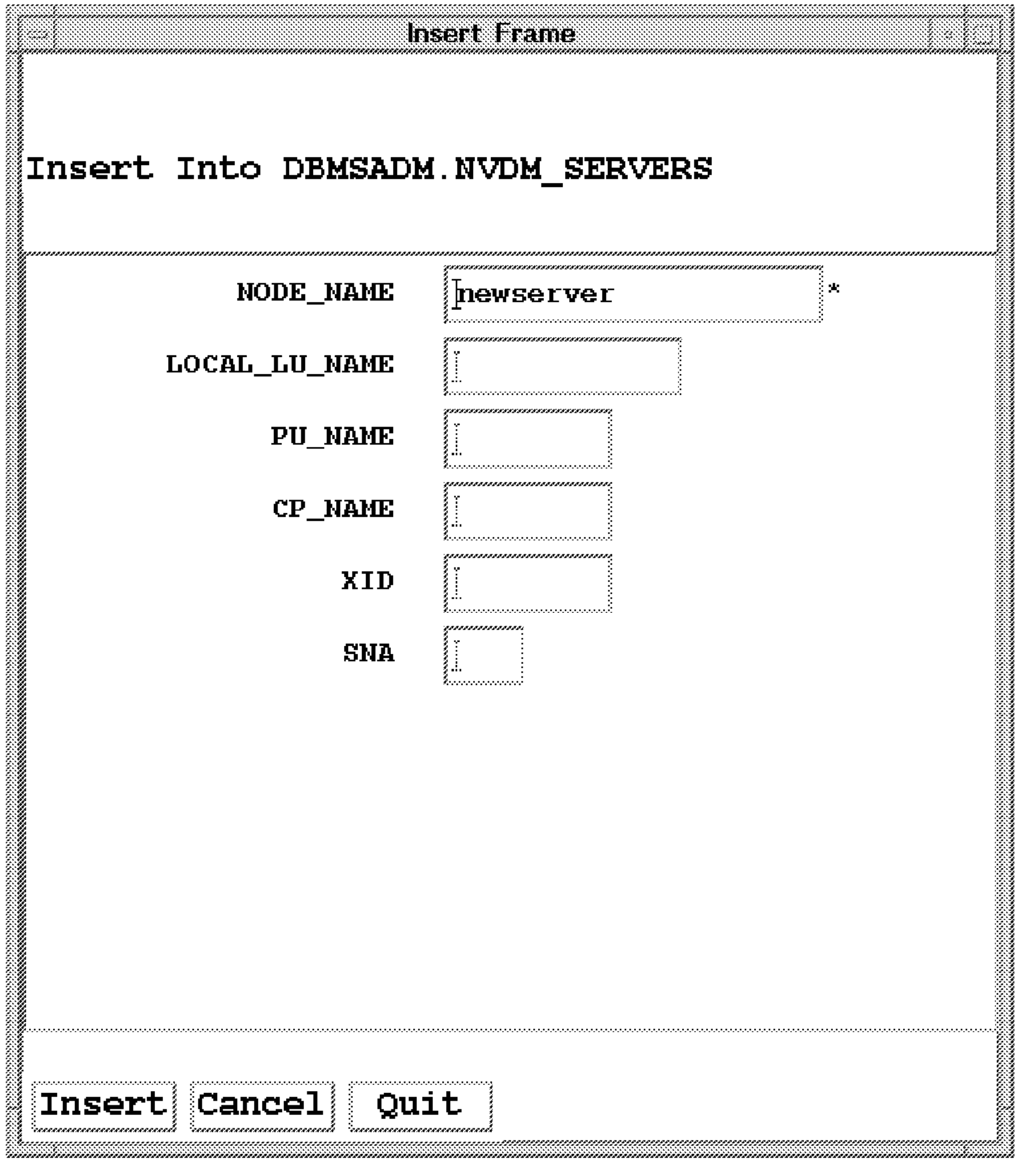


Figure 165. Inserting the Server-Specific Data into Table NVDM_SERVERS

In a similar way you then go to the server case through the database tables `NVDM_QUEUES` and `NVDM_GROUPS` and at last reach the insert frame of `NVDM_USERS` (see Figure 166 on page 305).

Note

For the latter three tables you are allowed to perform multiple inserts as their primary keys do not consist only of `NODE_NAME`.

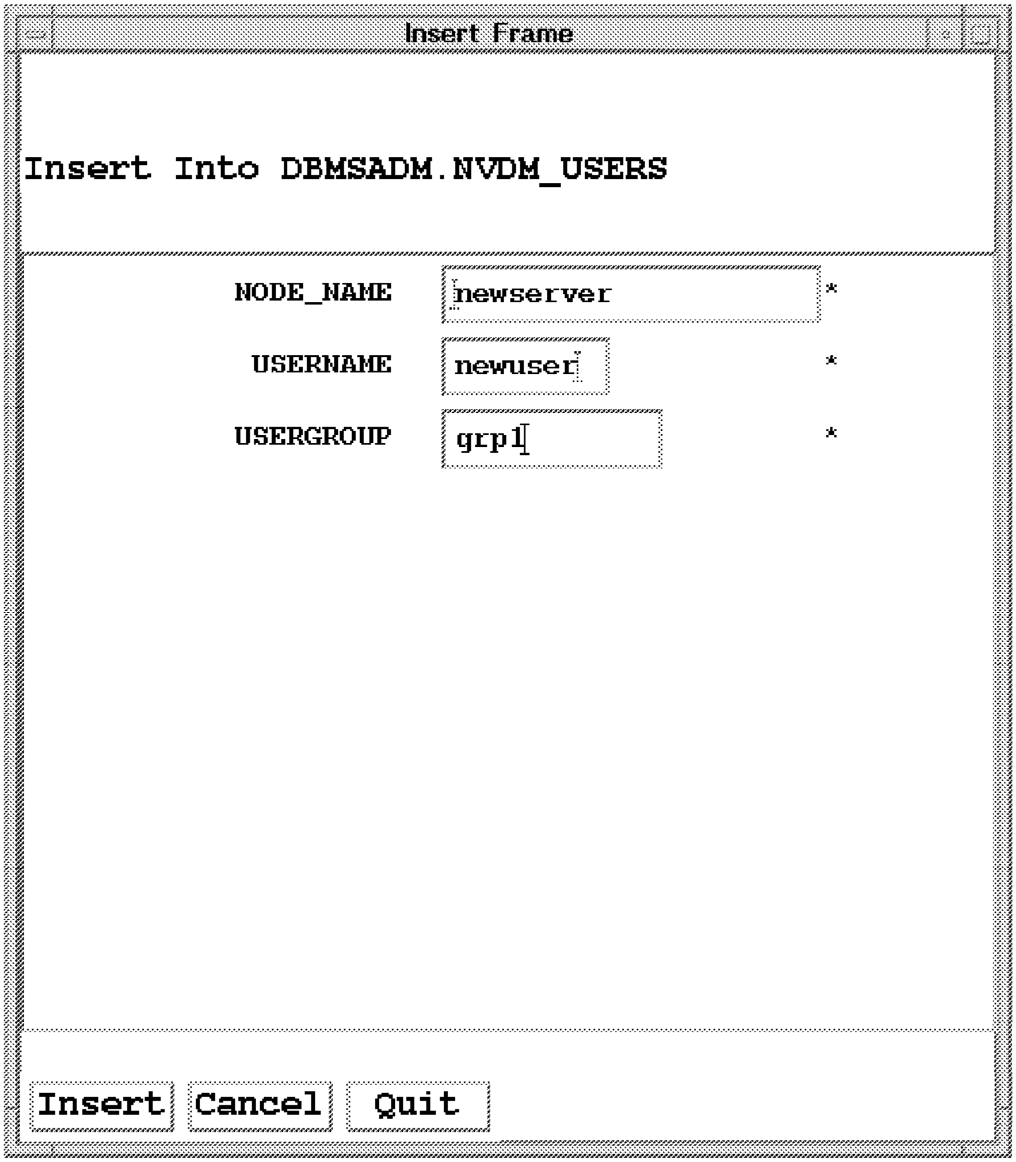


Figure 166. Inserting the User-Specific Data into Table NVDM_USERS

15.1.3 Conventional Database Table Browsing and Updating

After inserting the node data for `newserver`, check the result. Choose the table `NVDM_NODE` from the list in the main application window and double-click on it. In the opened table frame for `NVDM_NODE` we can see the inserted new node as shown in Figure 167 on page 306.

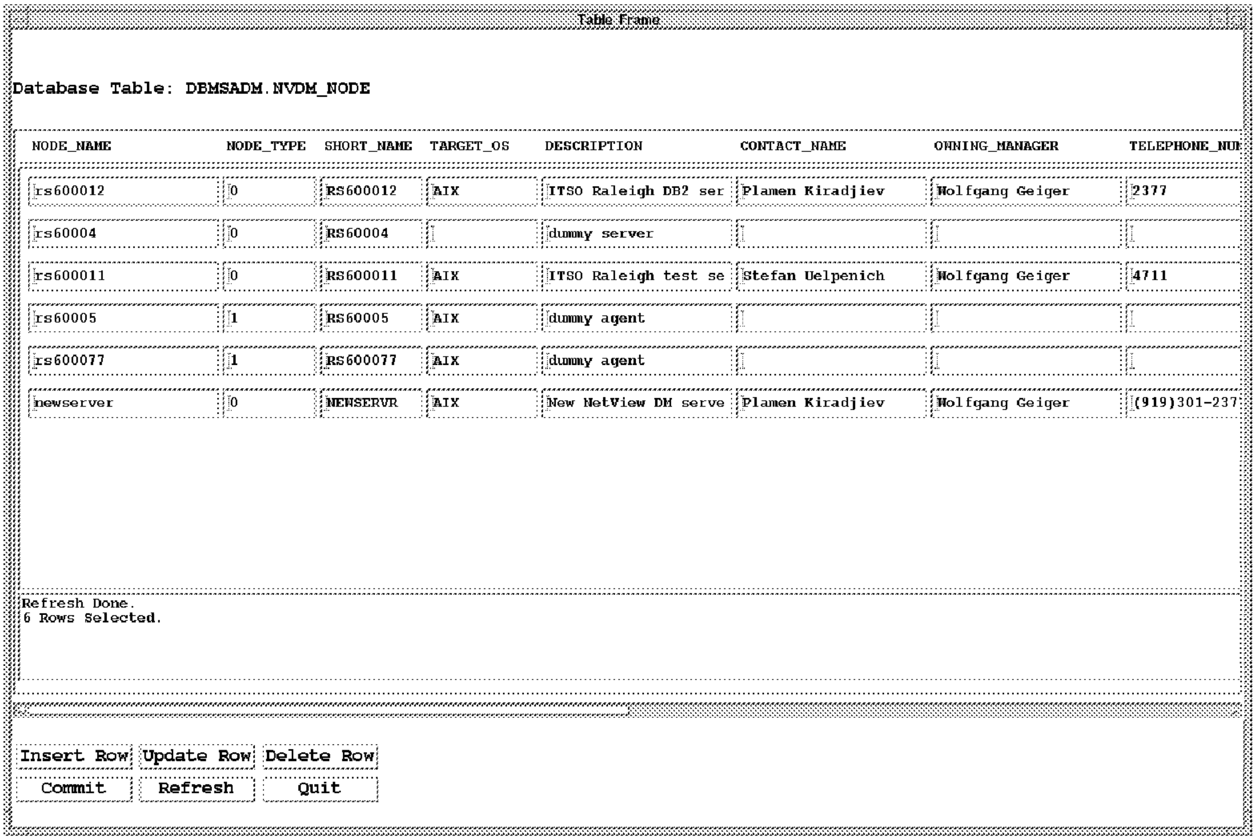


Figure 167. Table View of NVDM_NODE

In this way you are able to browse and update all the tables for which you are authorized. Refer to 15.1.1, "Updating Network Global Parameters" on page 296 for the possible actions on the table frame in general.

Note

Similar to Chapter 3, "Designing a Data Model for Configuration Data" on page 11 we provide a procedure `edit_db` that updates the database import files. First, it calls the graphical user interface and then it exports the Software Distribution for AIX configuration tables.

15.2 Implementation Insights

Now that we have presented the use of the graphical user interface, we now describe some of its implementation principles. The tools consist of three separated parts that cover the following tasks:

Database access methods:

The file `dbAccess.h` offers a database-independent interface to the stored data. Moreover, it defines the main database functions operating on a database system independent data structure. Although we implement the DB2/6000 scenario in our tool, it should be quite easier to implement an ODM implementation and exchange it transparently for the user.

Database frames:

The interface file `dbFrames.h` provides three standard frames that can be used for any table and database.

Model dependent part:

The main program `uicfgdb` specifies the model-dependent, NetView DM/6000 configuration database relevant actions.

When designing and implementing the graphical user interface, we separated the different tasks from each other. The database access and database frame parts of the tool are reusable and applicable to any database respectively database objects. Only the main program is tightly coupled with our Software Distribution for AIX configuration data model.

In the following we describe the main features and implementation details of the three program parts.

15.2.1 The Database Access Part

This part provides a system-independent access to a database. The core of the database access interface is the data structure `Table` that serves to describe a table independently of the database system. Figure 168 on page 308 shows the C include file dbAccess.h, which defines the functionality of the interface.

```
/*****************************************************************************
**
** File:    dbAccess.h
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: Database-Independent Interface
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
******************************************************************************/



/*----------------- Includes --------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>


/*----------------- Constants -------------------------------------------*/
#define NONE 0
#define SELECTED 1
#define INSERTED 2
#define UPDATED 3
#define DELETED 4


/*---------------- Type Definitions --------------------------------------*/
typedef int Mode     /* SELECTED, INSERTED or UPDATED */
typedef char *Message   /* SQL message string */
typedef struct Row{
   char **data   /* column values array */
   Mode mode   /* row mode */
   unsigned char changed  /* 1, if any column changed, else 0 */
   char **chData   /* changed values array
                    (NULL for unchanged columns) */
  }Row;
typedef struct ColAttributes{
   char **name    /* column names */
   int *length    /* column lengths */
   unsigned char *isNullable  /* is nullable array */
  }ColAttributes;
typedef struct Table{
   char *name   /* table name */
   ColAttributes colAttr  /* column attributes */
   int colCount   /* result column count */
   Row *rows   /* row array */
   int rowCount   /* selected row count */
   Message message  /* SQL message in error case */
  }Table;
```

Figure 168 (Part 1 of 2). Database Access Include File dbAccess.h

```
/*--------------- Procedures ---------------------------------------------*/

/* Connect to database */
Message dbConnect(char *dbName);

/* Disconnect from the database */
Message dbDisconnect();

/* Make a check point */
Message dbCheckPoint(void);

/* Select columns from table tableName with selection criteria selection */
Table *dbSelect(char *tableName, int colCount,
           char *columns[], char *selection);

/* Insert values into table tableName with columns sequence columns */
Message dbInsert(char *tableName, int colCount,
         char *columns[], char *values[]);

/* Update table tableName by setting setColumns to values for selection */
Message dbUpdate(char *tableName, int colCount, char *setColumns[],
         char *values[], char *selection);

/* Delete from table tableName with selection criteria selection */
Message dbDelete(char *tableName, char *selection);

/* Build selection string for a particular row */
char *selectThisRow(Row row, int colCount, char *columns[]);

/* Build the order-by part of selection statement */
char *dbOrderString(int colCount, char *columns[]);

/* Build an always false where part of selection statement */
char *dbAlwaysFalse(void);

/* free space allocated for table */
void freeTable(Table *table);
```

Figure 168 (Part 2 of 2). Database Access Include File dbAccess.h

The type `Table` contains the following information:

- The database table name (`name`)
- A data structure describing the column attributes (`colAttr`)
- The output column number (`colCount`)
- An array of the retrieved rows (`rows`)
- The number of retrieved rows (`rowCount`)
- A message or warning from the database system (`message`)

The column attributes (type `ColAttributes`) consists of three arrays containing the column names, their lengths and nullable flags in the column output order.

Each row (type Row) is made up of a string data array ordered according to the column output, a mode field, a change flag and an array of changed data. The mode field can be set to one of the following values:

- SELECTED: the row data comes from a selected operation on the database.
- INSERTED: the row data is to be inserted into the database.
- UPDATED: the row data is updated since the last selection.
- DELETED: the row is to be deleted from the database.
- NONE: no information about the row contents.

The array chData contains the changed values at the appropriate column positions. It contains null values at the column positions where no changes have been made since the last selection.

The table data structure serves to represent the database information that the database frames operate on. It is not related to a database-specific structure, like the SQLDA structure from DB2/6000 when using embedded SQL. This enables us to exchange the database implementation, for example with ODM, by obeying the interface definitions. In particular, the ODM implementation would be even easier because of the absence of such a variety of system meta information such as in DB2/6000 (contained in the system tables).

Our concrete implementation is based on DB2/6000 and uses the Call Level Interface (CLI). There are three types of procedures defined in our database access interface:

- General database procedures: for connecting to the database (dbConnect), disconnecting from the database (dbDisconnect) and making a checkpoint (dbCheckPoint)
- Data manipulation procedures: dbSelect, dbInsert, dbUpdate, dbDelete
- Help procedures: for building selection strings (selectThisRow, dbOrderString and dbAlwaysFalse) and for freeing the allocated space for the table data structure (freeTable)

The four manipulation procedures provide the main database access operations: data selection, insertion, updating and deletion. The select function dbSelect takes as arguments the table name (from part of the SQL select statement), the output column number and the column name array (corresponding to the select part of the SQL select statement); a selection string determining the selection predicates output ordering or grouping. It returns the data in a table data structure, which is then used by the database frames to display the table contents (see the next section).

The other manipulation procedures, dbInsert, dbUpdate, and dbDelete, provide in a similar way parameters corresponding to the appropriate SQL statements they represent. They return a NULL in the case of success, or the error message in the case of failure.

The help procedures serve on the one hand to abstract from the particular database syntax when building the selection strings and on the other hand to free the table data structure. In the former case they return a selection string for a particular row (selectThisRow), the false predicate (dbAlwaysFalse) respectively define an order for the row output (dbOrderString). In the latter case represented by `freeTable` the memory space allocated for a table data structure (returned by `dbSelect`) is freed.

All the database access procedures defined in the interface are designed according to the particular needs of our task to provide a graphical user interface to the Software Distribution for AIX configuration interface. But they are kept general in such a way that they apply to any database and database object. The abstraction of the concrete task and modularization of the program structure make the program maintenance easier and enable the enhancement of its functionality.

The DB2/6000 implementation is listed in Appendix C, “Source Code of the Graphical User Interface” on page 419. The detailed explanation of the C code is beyond the scope of this book. We believe that after the user gets acquainted with the data structures and functionality of the procedures provided by the interface and supported by the program comments, an experienced C-programmer could change, improve, and reimplement parts according to his/her own needs and tastes without affecting other parts of the program. Our intention, providing the graphical user interface to the Software Distribution for AIX configuration database, is to show a way for convenient and consistent database access, relieving the network administrator of concerns database specifics and data model details.

15.2.2 The Database Frames Part

The same modular principle is applied when designing the needed frames for the graphical representation of the configuration data. The C include file is listed in Figure 169 on page 312.

```
/******************************************************************************
**
** File:    dbFrames.h
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: Graphic Database Frames Interface
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
******************************************************************************/


/*---------------------- Includes -------------------------------------------*/
#include "dbAccess.h"
#include <Xm/PushB.h>



/*---------------------- Procedures -----------------------------------------*/

/* View Database Data in a Table Frame */
void tableFrame(Widget w, Table *table);

/* Insert Table Frame for a Sequence of Tables */
void insertFrame(Widget w, int tabCount, Table **table);

/* Message Box */
void messageBox(Widget top, String msg, int fatal);
```

Figure 169. Database Frames Include File dbFrames.h

In this interface we provide three general frames used by the main program:

- `tableFrame`: displaying a database table in a usual way by rows and columns
- `insertFrame`: providing the possibility of inserting data into a sequence of database tables
- `messageBox`: for the output of error messages and warnings

The frames rely on the database access interface and use only the data structures and procedures defined there. They are kept independent of the concrete configuration database implementation, so they can be used unchanged for the ODM case as soon as an ODM access implementation is provided according to the interface `dbAccess.h`.

All the frame procedures take as first argument the parent widget. This is needed because of the hierarchical structure of the Motif graphic elements (widgets, see AIXwindows Programming Guide). The table and insert frame rely on the table data structure when processing data from the database. Here you can recognize the benefit of a neutral data structure defining the interface between the database and graphical part of the program. It separates both tasks and enables enhancement and development of the modules independently from each other.

The insert frame provides a special feature needed by our tool for the convenient insertion of data into the configuration tables without considering the defined referential dependencies of the model: the capability of defining of a table insert sequence. In this manner you can specify a `Table` array from the application determining the order that the insert frames appear according to the referential rules.

The functionality of the frames was already described in the first part of this chapter.

The message box displays a message in a separate dialog window. Among the parent widget and the message string, the procedure `messageBox` takes a flag `fatal` that indicates whether the application should be exited after the particular error occurs.

The modular approach enables the extensibility of the graphical part of the tool similar to the database part. With some experience in Motif programming you can add new elements to the existing frames or even design new ones. The implementation (see Appendix C, "Source Code of the Graphical User Interface" on page 419) is structured in three parts defining the graphic elements, the callbacks and some help procedures, respectively.

15.2.3 The Main Program

The only application and data model dependent part of our graphical user interface is the main program, `uicfgdb.c` (see Appendix C, "Source Code of the Graphical User Interface" on page 419). It uses the database access procedures and frames for the particular task of accessing the Software Distribution for AIX configuration database.

The specific configuration data model is represented by a couple of constants for the used database table names and objects and by the two global arrays `serverDependentTabs` and `clientDependentTabs` determining the table insert sequences for a new server and client node, respectively (refer to Appendix C, "Source Code of the Graphical User Interface" on page 419).

The main program initializes the Motif environment and tries to make a connection to the Software Distribution for AIX configuration database NVDM_CFG. In the case of failure (for example, the configuration database is not available), an error message is displayed and the application is exited.

After the successful connection to NVDM_CFG, the main application window occurs on the screen. The possible actions from the main window are covered by the following callback procedures:

- `listSelectCB`: after double-clicking on a database table from the list, a table frame is generated for the chosen table.
- `updateGlobalsCB`: after clicking on the button **Update Network Globals**, a table frame occurs for NVDM_CFG_STATIC, containing the network global parameters.
- `insertNodeCB`: after clicking on the button **Insert New Node**, a dialog window is called requesting the new node name and the specification of whether or not it is a server (see Figure 162 on page 299).
- `exitAppCB`: the application is left.

The processing of the new node to be inserted depends on whether it is defined as a server or as a client. In the former case the server-dependent tables are taken (global array `serverDependentTabs`); in the latter the global array `clientDependentTabs` is taken. In both cases after taking over the inserted information, the callback `readNameCB` is called when clicking on the **OK** button (see Figure 162 on page 299). It prepares the needed column data by performing empty selections from the tables of the specified sequence (by applying the function `dbAlwaysFalse` from `dbAccess.h`). In this manner the array of table data structures is constructed and delivered to the function `insertFrames`.

Note

Of course, the same result can be obtained by retrieving information from the DB2/6000 system tables. We do not make use of the system tables for the following two reasons:

- Relying on the DB2/6000 specific meta information would be inconsistent with our modular approach to separate the different tasks and let the program part communicate through well-defined interfaces.
- A select operation with the false predicate is handled much faster by the optimizers of relational database systems, while the selection of a system table like SYSIBM.SYSCOLUMNS can be time consuming because of its size.

15.2.4 Features of the Graphical Interface Program

Now that we have presented the functionality and some implementation details of the graphical user interface, we now summarize the features of the tool. Providing this tool, our main intention is to demonstrate a comfortable, easy to use and consistent interface to the Software Distribution for AIX configuration database. You can directly employ this interface for the configuration data model discussed in this book as well as modify and improve it when needed by slight changes to the code.

15.2.4.1 Advantages

The graphical user interface to the Software Distribution for AIX configuration database has the following advantages:

- The graphical user interface provides a conventional user-friendly access to the tables of the Software Distribution for AIX configuration database as well as a model-driven, consistent way of inserting new targets.
- It relieves the Software Distribution for AIX administrator of the in-depth knowledge of the configuration data normally required model and of the defined referential dependences between tables.
- The graphical user interface is designed in order to guarantee its extensibility, so it is not difficult to improve its behavior and add a new functionality.
- Because of its modularity the changes affect only a limited scope of the program.
- As the database access and database frame procedures do not rely on a concrete model or application, they are reusable to a large extent. Moreover, they constitute most of the code for the tool.

15.2.4.2 Limitations

It is reasonable to expect from such a tool, implemented for the sake of completeness of our relational database approach in processing Software Distribution for AIX configuration data to show some limitations, which are as follows:

- Although keeping the generality of the interfaces, only the needed database operations and frames are implemented.
- Since our configuration data model uses only the SQL CHAR type, we do not complicate the table data processing with type conversions according to the particular column types.
- Although provided by the database access procedures by the selection string parameter, there is no graphic support for restricted table selections. When the number of selected rows exceeds the value of a constant MAXSELECT (currently set to 100), the remaining rows are not shown and an appropriate message occurs in the message part of the table frame.
- In the case of data model change, a recompilation is needed because of the required modifications of the globals in the main program `uicfgdb`. Relying on the makefile shipped with the package, the user can call `make` in the directory of the graphical user interface.
- The database connection is made implicitly and relies on client authentication in the DB2/6000 client/server environment.

Chapter 16. Cloning Systems Using Software Distribution for AIX 3.1

In this chapter we will examine the tools available in Software Distribution for AIX 3.1 for either installing or migrating workstations.

In order to demonstrate this feature we will show the steps needed to install the AIX 4.1 operating system on a pristine workstation.

16.1 Overview and Objective

With Software Distribution for AIX 3.1 the tools available for installing the AIX operating system on workstations have been significantly enhanced as compared to NetView DM/6000.

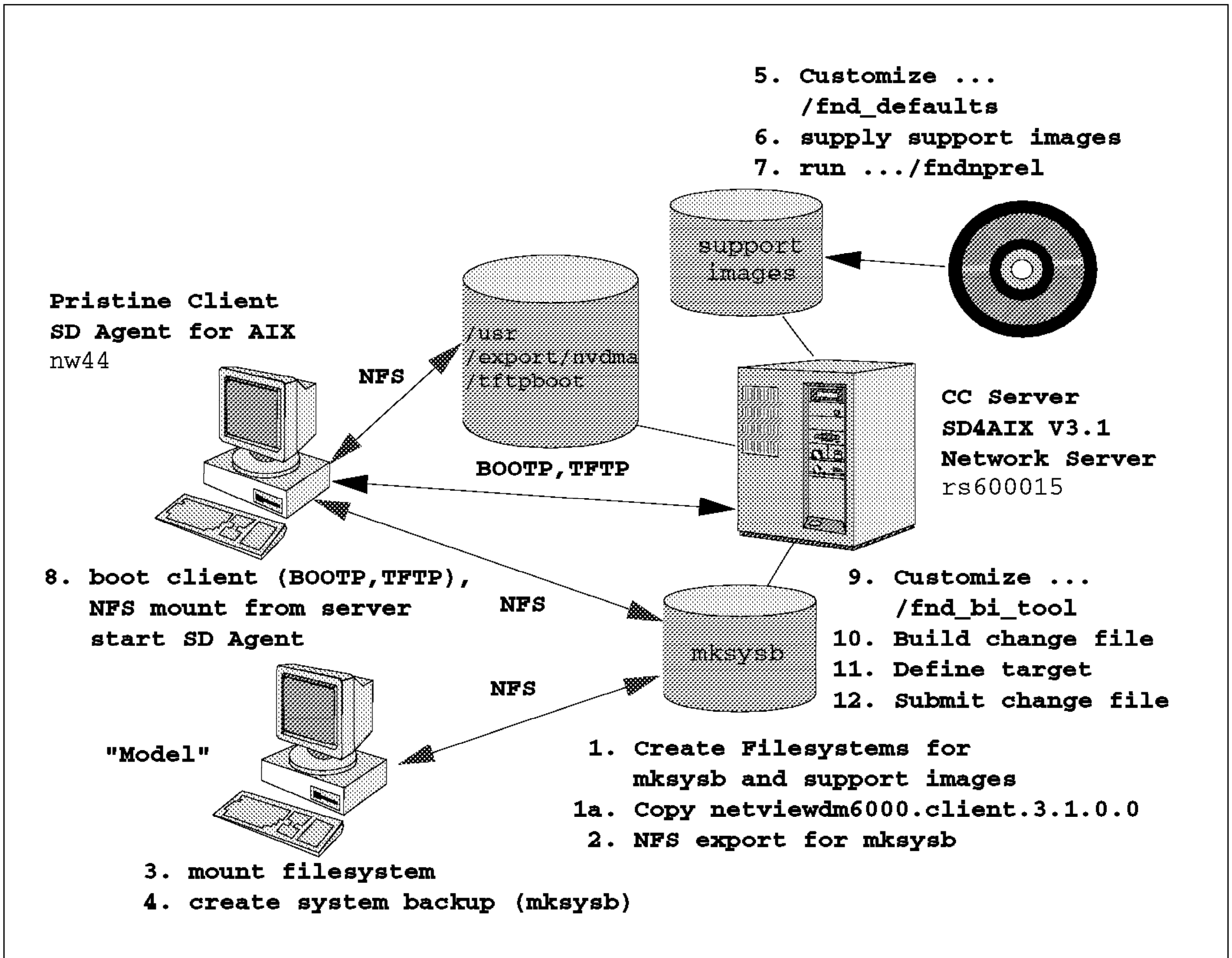


Figure 170. Steps in Pristine Installation Scenario

The major enhancements are features to install an AIX 4.1 system backup image (`mksysb`) on a pristine workstation and to update an installed workstation from AIX 3.2.5 or from a previous version of AIX 4.1.

Hence, the tools supplied with the product can be used to do the following:

- Clone an AIX 3.2.5 system by copying the complete `rootvg` of a model workstation to the workstation that needs to be installed. This feature was also available in NetView DM/6000 1.2 and is described in detail in the redbook *NetView DM/6000 Agents and Advanced Scenarios, GG24-4490*.
- Clone an AIX 3.2.5 system by installing a system backup image (`mksysb`) on the target workstation. We have shown an example of how to do that with NetView DM/6000 1.2 in Chapter 10, "Pristine Installation" on page 167.
- Clone an AIX 4.1 system by installing a system backup image on the target workstation. This is the feature that we will demonstrate in this chapter.
- Migrate an installed workstation from AIX 3.2.5 to AIX 4.1 using an AIX 4.1 system backup image.
- Migrate a workstation installed with AIX 4.1 to a newer version of AIX 4.1, for example, migrate from AIX 4.1.2 to 4.1.3. This is also achieved by installing an AIX 4.1.3 system backup image.

Although new features have been introduced to the pristine installation process with Software Distribution for AIX 3.1, the basic principle is the same as it was in NetView DM/6000. However, there have been significant changes in the scripts used to perform the preparation and execution of the pristine installation.

Additional Hints

You will find additional hints to avoid potential problems with the pristine installation in the file:

```
/4508code/README.first
```

Therefore we will not describe the entire installation process again, but concentrate on the new features and changes. If you need an introduction to the pristine installation process you can refer to the redbook *NetView DM/6000 Agents and Advanced Scenarios, GG24-4490* or to Chapter 10, "Pristine Installation" on page 167.

16.2 File and Directory Structure

In order to be able to clone workstations you have to install the Tools option of Software Distribution for AIX 3.1.

All files for cloning and migrating workstations are located in the `/usr/lpp/netviewdm/tool/AIX.install` directory. This directory contains two subdirectories, `4_1` which contains the files needed to install AIX 4.1 and `3_2_5` which contains the files needed to install AIX 3.2.5.

Each of these directories has two subdirectories, `scripts` which contains the shell scripts for preparing and performing the cloning and `profiles` which contains change file profiles that are used to build the change files needed to prepare and perform the cloning.

Further, there is a directory `/etc/aixfnd` that holds configuration and status information maintained by the cloning scripts. The file `fnddb` holds a simple pristine installation database which is actually a simple text file containing information about the clients that have been prepared, etc.

This file is accessed by the preparation script `fndnprel` as well as the cleanup script `fndcln`.

Also, a file `lock` is created in `/etc/aixfnd` whenever the preparation script `fndnprel` is running. This avoids multiple instances of `fndnprel` running at the same time.

16.3 Prerequisites

In order to be able to install an AIX 4.1 system backup image, the following software has to be installed on the network server:

- AIX Base Operating System Version 4.1
- TCP/IP Version 2.1
- Network File System (NFS)
- The `bos.sysmgt.sysbr` fileset

In our example we also use the CC server as the network server and have installed AIX 4.1.3.

> **Note**
>
> In our example we also use the CC server as the network server because this simplifies the scenario. In case you want to use another machine, this machine has to have the Software Distribution Agent installed. In order to prepare this system as the network server you will have to build a change file to transmit and execute the preparation script `fndnprel`. Refer to *Software Distribution 3.1 for AIX User's Guide,* SH19-4163 on how to do that.

Normally TCP/IP and NFS are automatically installed when you install your AIX 4.1 operating system. Whether the `bos.sysmgt.sysbr` file set is also installed can be checked by typing:

```
lslpp -h bos.sysmgt.sysbr
```

This should produce an output similar to the following:

```
  Fileset         Level     Action      Status      Date        Time
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.sysmgt.sysbr
                  4.1.3.0   COMMIT      COMPLETE    10/12/95    17:41:50

Path: /etc/objrepos
  bos.sysmgt.sysbr
                  4.1.3.0   COMMIT      COMPLETE    10/12/95    17:43:08
```

16.4 Installing AIX 4.1 on a Pristine Client

In order to install a pristine client with AIX 4.1, we will perform the following steps:

- Create a file system to hold the system backup image on the network server.
- Create a file system to hold the AIX 4.1 support images needed to install the client.
- Copy the Software Distribution for AIX client image to the directory where the support images reside.
- Create a system backup image (`mksysb`) of a model workstation and store it on the network server.
- Customize the `fnd_defaults` file.
- Create a list file containing information about the pristine client.
- Run the preparation script `fndnprel`.
- Customize the `bosinst.data` file.
- Define the pristine client as a target on the CC server.
- Boot the pristine client.
- Customize the `fnd_bi_tool` file.
- Build the change file to perform the cloning.
- Submit the change file to perform the cloning.
- Clean up the network server.

We will now describe each of the above steps in detail.

16.4.1 Creating File Systems

In order to perform the cloning we will need two file systems, one to hold the image to be installed on the client and another to hold the AIX 4.1 support images.

Unlike with the AIX 3.2.5 cloning procedure we need to supply certain AIX 4.1 images to support the installation. These images are copied by `fndnprel` from a source specified in the `fnd_defaults` file to the file system we create; the name of this file system is also specified in the `fnd_defaults` file.

The preparation script `fndnprel` uses the support file sets to install the `/usr` SPOT which is used by the pristine client after booting.

The default name of the file system for support images is `/inst.images`, so we create a file system with that name being 300 MB of size. We can use SMIT or the following command:

```
crfs -v jfs -g rootvg -a size=600000 -m /inst.images -A yes -p rw \
-f no -a frag=yes -a nbpi=4096 -a compress=no
```

We use a similar command to create a file system for storing the system backup image:

```
crfs -v jfs -g rootvg -a size=1000000 -m /inst.images -A yes -p rw \
-f no -a frag=yes -a nbpi=4096 -a compress=no
```

Note

A file system size of 500 MB is normally enough to store an AIX 4.1 `mksysb` image. Unlike in AIX 3.2 where system backup images were in `tar` format, AIX 4.1 uses the `backup` format which significantly reduces the size of the image.

The next step is mounting both file systems:

```
mount /inst.images
mount /mksysb
```

16.4.2 Copying the Client Image

In order to enable the pristine client to act as a software distribution agent right after it has been booted, the agent code is installed in the `/usr` SPOT.

This is done by `fndnprel`, which needs the install image of the client located in the directory where the support images reside.

We can either copy the install image from tape or, if we already have the file stored on our hard disk, by a simple file copy.

We assume that we have the image stored in the `/usr/sys/inst.images` directory. We can then copy the file by typing:

```
cd /usr/sys/inst.images
cp netviewdm6000.client.3.1.0.0 /inst.images
```

If we want to save disk space we can also create a symbolic link:

```
ln -s /usr/sys/inst.images/netviewdm6000.client.3.1.0.0 \
/inst.images/netviewdm6000.client.3.1.0.0
```

16.4.3 Creating the System Backup Image

Before creating the system backup image we have to pick a system that we want to clone.

When doing so you must consider that with AIX 4.1 you must be careful when selecting the model system. Since with AIX 4.1 you cannot install a `mksysb` image on a different hardware model, you must pick a system as your model that is similar to the system you want to install.

For example, it is not possible to install an image created on an RS/6000 model 40P on a C10 because these systems have different hardware architectures.

The reason for this is that in AIX 4.1, normally, only the device drivers actually needed are installed. Therefore it is very likely that device drivers are missing when trying to install the system backup on a different machine. For example, the 40P has an ISA bus, whereas the C10 uses a Micro-Channel.

However, you can normally install the backup image on a similar machine. We had, for example, no problems when installing an image created on a C10 on a C20 since these machines differ only in the CPU type. Also, you should normally have no problems if the target system has a different disk configuration, as long as the hard disks are of the same architecture and the target system disk space is sufficient to store the data contained in the image.

In order to store the image on the network server, we NFS-export the file system we have previously created to the model workstation. This can be done using SMIT. Remember to grant the model workstation root access and export the file system in read-write mode.

On the model we have to mount the file system. Assuming that our network server has the hostname `rs600015`, we type on the model:

```
mkdir /mksysb
mount rs600015:/mksysb /mksysb
```

Now we can create the system backup. The easiest way is using SMIT by typing `smitty mksysb`. Enter the name of the system backup image, for example, `/mksysb/aix41.image` and also select to create map files.

The backup will run for some time. As soon as it is finished you can `unmount` the NFS file system again and return to the CC server.

16.4.4 Customizing the Default File

The `fnd_defaults` file located in the `/usr/lpp/netviewdm/tool/AIX.install/4_1/scripts` directory contains information for the preparation script telling it where to find information.

In order to customize the file we type:

```
vi fnd_defaults
```

Jump to the end of the file where you will find the customizable parameters. For our scenario we enter the following values:

```
DEFAULT_source=/dev/cd0
DEFAULT_inst_images=/inst.images
DEFAULT_mksysb=/mksysb
MKSYSB_IMAGE=aix41.image
INSTALLP_LOG=/tmp/installp
```

We specify the device file of the CD-ROM drive as `DEFAULT_source` in order to copy the support file sets from CD-ROM.

In order to be able to watch the progress of the SPOT installation we specify `/tmp/installp` as the log file. This will enable us to see the output of the `installp` that is used by `fndnprel` to install the `/usr` SPOT.

16.4.5 Describing the Pristine Client

In order to describe the parameters needed to install client, we create a list file with the appropriate data. A sample file can be found in `fnd.cfg.sample` located in the same directory where `fnd_defaults` resides.

In order to fill this file you will need the following information:

- The TCP/IP address and hostname of the pristine client
- The MAC address of the pristine client
- The installation method

The TCP/IP hostname in our example is `nw44`, the TCP/IP address is `192.1.1.44`. The machine has a token-ring adapter with the MAC address `08005a81d33d`.

Note

You can use the `netstat -v | more` command to determine the MAC address of the client if it already has an operating system installed. If not, you just need to try a network boot and the address will be displayed on the console.

The installation method in our case is `mksysb_install`. The other possible value is `migrate` if you want to migrate an installed system.

The following figure shows the `list` file for our example:

```
CLIENT NAME:                         nw44
BOOT DEVICE:                         tok0
DESCRIPTION:                         Pristine Installation Test
NETWORK DEVICE HARDWARE ADDRESS:     08005A81D33D
SERVER:                              rs600015
GATEWAY ADDRESS:
SUBNETMASK:
INSTALL METHOD:                      mksysb_install
BACKUP IMAGE NAME:                   /mksysb/aix41.image
DEBUG MODE:                          no
```

Figure 171. Configuration File for Pristine Client

16.4.6 Running the Preparation Script

Now that we have supplied all necessary information and completed all prerequisite tasks we can run the preparation script.

Since we want to copy our support images from CD-ROM, we have to insert an AIX 4.1 installation CD in the CD drive of the network server before running the script.

To run the script and write the output to `logfile` we type:

```
cd /usr/lpp/netviewdm/tool/AIX.install/4_1/scripts
 ./fndnprel -f list | tee logfile
```

Note about CD-ROM

With some versions of Software Distribution for AIX 3.1 the preparation script will fail when using a CD-ROM to supply the support images. The reason for this is that the `fndnprel` script will use the `inutoc` command to create a `.toc` file in source directory. Since the CD-ROM is read-only this command will fail and the `fndnprel` will quit.

In order to fix this you can edit `fndnprel` and comment out the line where the `inutoc` command is invoked. In the version we worked with this was line 1219:

```
#${INUTOC} ${source} 2>/dev/null || handle_error
```

Also, when using a CD-ROM as the source for support images, the `fndnprel` script will not `unmount` the CD file system after it has completed. This will cause the script to fail the next time it is invoked.

In order to fix this you should `unmount` the file system manually before running `fndnprel` again. The file system name is `/tmp/PID`, for example, `/tmp/19802`. To `unmount` it we type:

```
unmount /tmp/19802
```

The `fndnprel` will perform the following tasks in order to prepare the pristine installation:

- Create `/etc/aixfnd/lock`.
- Copy the support images from the source to `/inst.images`.
- Create a file system `/export/nvdma` and export it to the pristine client.
- Create a directory `/export/nvdma/workstation`.
- Create a `/usr` SPOT.
- Install the SPOT with the support images.
- Install the Software Distribution Agent in `/export/nvdma/workstation` and customize it for use with the CC server.
- Insert the pristine client into `/etc/bootptab`.
- Create a network boot image in the `/tftpboot` directory.
- Make an entry in `/etc/aixfnd/fnddb`.

Note

We encountered an error when installing the SPOT with the support images. The installp command failed because it could not install some of the support file sets due to a missing prerequisite. The images that could not be installed, however, were not actually needed, but the fndnprel command failed.

To fix this, we commented out the line in fndnprel where the return code of the installp command is checked. In the version we used this was line 2981. We replaced rc=$? with rc=0 in order to ignore the return code.

The output of the installp can be examined by browsing the file /tmp/installp assuming that the parameter in fnd_defaults is set appropriately.

As soon as fndnprel has completed successfully we can perform the pristine installation.

Note

If the fndnprel script complains that there is already a preparation script running, you can check this by typing:

```
ps -ef | grep fndnprel
```

If the script is not running, remove the file /etc/aixfnd/lock by typing rm /etc/aixfnd/lock and then start fndnprel again.

The following figure shows logfile for a successful run of fndnprel:

```
        Configuring AIX Version 4.1 Network Server
        for the installation of a remote client using
        Software Distribution for AIX.


        Creating /usr/lpp/netviewdm/tmp directory.
        Creating /export/nvdma filesystem.
Based on the parameters chosen, the new /export/nvdma JFS file system
is limited to a maximum size of 134217728 (512 byte blocks)

New File System size is 24576
        Created /export/nvdma filesystem 12Mb large.
        Mounting /export/nvdma filesystem.
        Creating /export/nvdma/nw44 directory.
        Creating /tftpboot filesystem.
        Created /tftpboot filesystem 4Mb large.
        Mounting /tftpboot filesystem.

bosboot: Boot image is 5068 512 byte blocks.
New volume on /inst.images/netviewdm6000.client.3.1&
 Cluster 51200 bytes (100 blocks).
    Volume number 1
    Date of backup: Wed Oct 18 22:12:20 DFT 1995
    Files backed up by name
    User builder
    files restored: 14
        Creating the /usr-spot.

        Exporting filesystems to nw44 client.
        It may take some minutes...

    /usr/bin/fndnprel executed successfully.
```

Figure 172. Preparation Script Log

16.4.7 Customizing the bosinst.data File

The `bosinst.data` file is located in `/export/nvdma/workstation` and contains information about how to install the operating system.

It is important that the `CONSOLE` and `PROMPT` fields in this file are set correctly. The `CONSOLE` field contains the console device of the pristine client, for example, `/dev/tty0` or `/etc/lft0`.

The following figure shows the `bosinst.data` file for our example:

```
control_flow:
    CONSOLE = /dev/tty0
    INSTALL_METHOD = overwrite
    PROMPT = no
    EXISTING_SYSTEM_OVERWRITE = yes
    INSTALL_X_IF_ADAPTER = yes
    RUN_STARTUP = no
    RM_INST_ROOTS = no
    ERROR_EXIT =
    CUSTOMIZATION_FILE =
    TCB = no
    INSTALL_TYPE = eserver
    BUNDLES =

target_disk_data:
    LOCATION =
    SIZE_MB =
    HDISKNAME =

target_disk_data:
    LOCATION =
    SIZE_MB =
    HDISKNAME =

locale:
    BOSINST_LANG =
    CULTURAL_CONVENTION = en_US
    MESSAGES = en_US
    KEYBOARD = en_US
```

Figure 173. bosinst.data File

16.4.8 Defining the Pristine Client as a CC Client

In order for the pristine client to act as a CC client we have to define it as a target on our CC server. We can do this by either using the graphical user interface `nvdmgi` or by using the command line interface:

```
nvdm addtg nw44 -s NW44 -y AIX -b client
```

In the above example the hostname of our client is `nw44`.

16.4.9 Booting the Pristine Client

We have described how to boot a pristine client in Chapter 10, "Pristine Installation" on page 167 and will not describe this procedure again. However, the procedure described there is only valid for Micro-Channel systems. Therefore we will describe the procedure for ISA-bus systems, namely a model 43P now.

The following are steps that need to be performed:

1. Insert the diskette labeled "System Management Services".
2. Turn on the system.

3. Wait until the keyboard icon is displayed and press F4. The system management utilities will be loaded from the diskette.
4. Select **Utilities** from the menu and press Enter.
5. Select **Remote Initial Program Load Setup** and press Enter.
6. Select **IP Parameters** and press Enter.
7. Enter Client IP Address, Server IP Address and Netmask and then press Enter. Then press the Escape key.
8. Select **Ping** and press Enter. Start a test transmission. If the test is not successful check network connections, cables, routers, etc. If the test is successful press the Escape key.
9. Select **Select Boot Devices** and press Enter.
10. Select **Boot Other Devices** and press Enter.
11. Select the appropriate network adapter and press Enter. The system will boot from the network.

Note

If you want to migrate a system, you will not need to boot it manually. You can then use the script `/usr/lpp/netviewdm/tool/AIX.install/4_1/scripts/bootl.proc` to change the boot list on the client system. To do so change the network device in `bootl.proc`, for example, to `tok0`. Then catalog the procedure on the CC server by typing `nvdm cat BOOTLIST.CHANGE.PROC bootl.proc -o PROC -t` and execute the procedure on the client by typing:

```
nvdm exec BOOTLIST.CHANGE.PROC -w workstation
```

This will change the boot list in normal mode to boot from the network. Then use the command:

```
nvdm act -w workstation -f
```

This will reboot the client from the network.

16.4.10 Customizing the fnd_bi_tool File

The `fnd_bi_tool` script in the `/usr/lpp/netviewdm/tool/AIX.install/4_1/script` directory performs the actual cloning. It will be contained in the change file that is installed on the pristine client to do the installation.

Before the change file is built, we might need to customize this script:

- The variable `CURLEVEL` contains the current level of AIX that is to be installed. In the current version this field is set to `4.1.1`.
- In our scenario, however, we use AIX 4.1.3 so we replace the line `CURLEVEL=4.1.1` with `CURLEVEL=4.1.3`.

16.4.11 Building the Installation Change File

The change file to perform the pristine installation can be built from the change file profile /usr/lpp/netviewdm/tool/AIX.install/4_1/profiles/profile.install:

```
cd /usr/lpp/netviewdm/tool/AIX.install/4_1/profiles
nvdm bld profile.install
```

This will add the change file NVDM.AIX.INSTALL.REF.1 to the catalog at the CC server.

16.4.12 Submitting the Change Request

Before submitting the change request we have to make sure that the pristine client has successfully started the CC client code.

This can be checked by typing:

```
nvdm stattg nw44
```

As soon as the status is available we can submit the change request.

In order to actually start the installation we have to install the change file NVDM.AIX.INSTALL.REF.1 on the target nw44. This can be done using the graphical user interface nvdmgi.

The progress of the installation can be checked by watching the file /export/nvdma/**workstation**/work/request.out, for example:

```
tail -f /export/nvdma/nw44/work/request.out
```

16.4.13 Cleaning Up the Network Server

The preparation script fndnprel creates file systems, a /usr SPOT, a network boot image, etc. This will consume some of your disk space which you might want to free up again after you have installed the pristine client.

For that purpose you can use the script fndcln. It will remove the information created for a specific system. In our example we typed the following to remove the information for our pristine client:

```
fndcln -w nw44
```

This will remove the boot image in /tftpboot, the entry from /etc/bootptab, etc. It will also remove the /export/nvdma file system and the SPOT if there are no other clients in the database file /etc/aixfnd/fnddb.

However, it will not remove the directory containing the support images by default because they may be needed to perform another installation. If you also want to remove this directory, you must type:

```
fndcln -w nw44 all
```

Appendix A. The Configuration Script Listings

In this appendix we list the file that contains the configuration script `config_nvdm`, which is used to configure software distribution networks.

A.1 Script for NetView DM/6000 Version 1.2 Using ODM

```
#!/bin/ksh
#
#
# Configure NVDM node
# Main Configuration Script
# For NetView DM/6000 V1.2
# ------------------------------------------------------
# This script can be used to configure any RS/6000
# workstation in your software distribution network
# automatically
# ------------------------------------------------------
# Author : Stefan Uelpenich/IBM Germany
# RCS Revision :  $Revision: 1.1 $
# ------------------------------------------------------
#
# This script will cover:
#
# 1. For all nodes
#    - configuration of WORKSTATION NAME in nvdm.cfg
#    - configuration of SERVER in nvdm.cfg
#    - configuration of TCP/IP ports used by NVDM
#    - configuration of log file size & other things
#      in nvdm.cfg
#    - add NVDM Users to AIX Operating System
#
# 2. For servers/prep sites
#    - modification of server's own target
#    - add DLC Device for SNA adapter
#    - SNA initial node setup
#    - configuration of SNA CP profile
#    - configuration of SNA DLC profile
#    - configuration of SNA Link profile
#    - configuration of SNA Local LU profile
#    - configuration of SNA Mode profile
#    - configuration of SNA TPN Send profile
#    - configuration of SNA TPN Receive profile
#    - configuration of SNA LU6.2 Location profile
#    - configuration of SNA Side Info profile (Send)
#    - configuration of SNA Side Info profile (Receive)
#    - configuration of SNA/DS connection profiles
#    - configuration of SNA/DS Routing table
#    - configuration of local targets
#    - configuration of local target groups
#    - configuration of remote targets/focal points
#    - reload NVDM Configuration
#    - refresh SNA Server Configuration
#    - start SNA Server
#    - restart NVDM
#    - release NVDM SNA communications
```

```
#
#

#
# The command line parameter supplied with this command
# must be the IP hostname of the system to be configured.
# This hostname will be used as the argument when
# accessing the configuration database
#

if [ $# != 1 ]
then
  print "Syntax : $0 node_name"
  exit 1
fi

#
# extract hostname (without domain information)
#

HNAME=`echo $1 | cut -d'.' -f1`
print "NVDM CONFIG : Extracted hostname ... $HNAME"

#
# Variables
#

CONFIG=/usr/lpp/netviewdm/db/nvdm.cfg
NUM_QUEUE=0
PROTOCOL=""
REMOTE_SERVER=""
EXPORT_SNA=/tmp/sna.org
SNA_DS_DIR="/usr/lpp/netviewdm/db/snads_conn"
SNA_DS_ROUTE="/usr/lpp/netviewdm/db/routetab"
HISTORY_DIR="/usr/lpp/netviewdm/db/cm_status"
SAVE_DIR="/tmp/target_save"
USE_CP_XID=no
SW_INV="/usr/lpp/netviewdm/fndswinv"

#
#
# useful stuff
#
#

# print a line

line ()
{
 print "====================================\
```

Figure 174 (Part 2 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
===========================================”
}

#
# print debug information
#

debug_info ()
{
 line
 print "Software distribution network configuration script"
 print "\$Revision: 1.1 $"
 BEK=`hostname`
 print "IP Hostname = $DEB"
 print "Name resolution = "`host $DEB`
 line
}

#
# abort configuration script
# and print an error message
# $1 = text of error message
#

abort ()
{
 line
 banner "FAILURE!"
 line
 print "NVDM CONFIG ERROR :\
 Could not properly configure node."
 print "Cause : $1"
 line
 exit 1
}

#
#
# DATABASE ACCESS METHODS (ODM)
# these access methods may be replaced with
# access methods for any other database at
# a later time
#
#

#
# get list parameters from odm_class
# $1 = class name
# $2 = search field
# $3 = search field value
```

Figure 174 (Part 3 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
# $4 = attribute name
# The list of parameters is stored in the VALUE_LIST variable
# The number of parameters is stored in VALUE_NUM
#

get_attribute_list ()
{
 VALUE_LIST=`odmget -q $2=$3 $1 | grep "$4 =" | cut -d'=' -f2 |\
 sed "s/\"//g" | cut -c2-79`
 VALUE_NUM=`odmget -q $2=$3 $1 | grep "$1:" | wc -l`
}

#
# get single parameters
# $1 = class name
# $2 = search field
# $3 = search field value
# $4 = attribute name
#

get_attribute ()
{
 VALUE=`odmget -q $2=$3 $1 | grep "$4 =" | cut -d '=' -f2 | sed "s/\"//g" |\
 cut -c2-79`
}

#
# get single parameters (AND)
# $1 = class name
# $2 = search field1
# $3 = search field value1
# $4 = search field2
# $5 = search field value2
# $6 = attribute name
#

get_attribute_and ()
{
 VALUE=`odmget -q "$2=$3 AND $4=$5" $1 | grep "$6 =" | cut -d '=' -f2 |\
 sed "s/\"//g" | cut -c2-79`
}

#
#
# CONFIGURATION METHODS
#
#

#
# Set Attributes in nvdm.cfg file
```

Figure 174 (Part 4 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
# $1 parameter name (e.g. WORKSTATION NAME, SERVER)
# $2 parameter value
#

configure_nvdm_cfg ()
{
 mv $CONFIG /tmp/config
 print "NVDM CONFIG : Setting nvdm.cfg ($1) to $2"
 #
 # the TCP/IP port parameter is special
 # because it contains a / in its name
 # and also needs modification of
 # /etc/services
 #
 if [ "$1" = "TCP/IP PORT" ]
 then
   sed "s/TCP\/IP PORT:.*/TCP\/IP PORT:          $2/" \
/tmp/config >$CONFIG
   mv /etc/services /tmp/services
   sed "s/NetViewDM6000.*\/tcp/NetViewDM6000   $2\/tcp/" \
/tmp/services >/etc/services
   return
 fi
 #
 # adjust to right column
 #
 len=`echo $1 | wc -c`
 SUBST=$2
 while [ $len -lt 22 ]
 do
  SUBST=" "$SUBST
  len=`expr $len + 1`
 done
 #
 # replace parameter
 #
 sed "s/$1:.*/$1:$SUBST/" /tmp/config >$CONFIG
}

#
# configure SNA Control Point Profile
#
# SNA_NET contains SNA Network Name
# CP_NAME contains SNA Control Point Name
# CP_TYPE contains SNA Control Point Type
#

configure_sna_cp ()
{
 print "NVDM CONFIG : Configuring SNA Control Point Profile"
```

Figure 174 (Part 5 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 line
set -x
 chsnaobj -t 'control_pt' -e "$SNA_NET" -a "$CP_NAME" -A "$CP_NAME"\
 -N "$CP_TYPE" node_cp
set +x
 line
}

#
# configure SNA dlc
# for all SNA communications a DLC for the
# communications adapter is needed.
# if the DLC already exists, the mkdev command
# will print an error message - this will be
# redirected to /dev/null
#

configure_sna_dlc ()
{
 print "NVDM CONFIG : Adding DLC Device for $DEVICE"
 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) mkdev -c dlc -s dlc -t tokenring 1>/dev/null 2>&1 ;;
  "ent" ) mkdev -c dlc -s dlc -t ethernet 1>/dev/null 2>&1 ;;
  "x25" ) mkdev -c dlc -s dlc -t x25_qllc 1>/dev/null 2>&1 ;;
  "*"   ) print "NVDM CONFIG : Device type $CHECK unknown." ;;
 esac
}

#
# SNA initial node setup
#

sna_initial ()
{
 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="token_ring" ;;
  "ent" ) DEV_TYPE="ethernet" ;;
  "fdd" ) DEV_TYPE="fddi" ;;
  "x25" ) DEV_TYPE="x.25_call_SVC" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
    abort "No device type found for $DEVICE."
 fi

 print "NVDM CONFIG : Configuring SNA Initial Node Setup"
```

Figure 174 (Part 6 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
set -x
 mk_qcinit -y $DEV_TYPE -t $CP_TYPE -w $SNA_NET -d $CP_NAME
set +x

}

#
# configure SNA dlc profile
#

configure_sna_dlc_profile ()
{
 # determine type of DLC from datalink device name
 # get only first 3 characters from device name
 # e.g. if datalink device is x25s1, then x25 determines
 # the type to be X.25

 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="sna_dlc_token_ring" ;;
  "ent" ) DEV_TYPE="sna_dlc_ethernet" ;;
  "fdd" ) DEV_TYPE="sna_dlc_fddi" ;;
  "x25" ) DEV_TYPE="sna_dlc_x.25" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
    abort "No device type found for $DEVICE."
 fi

 #
 # create new DLC Profile
 # use Datalink Device Name as Profile Name
 #

 print "NVDM CONFIG : Configuring SNA DLC Profile"
 line
set -x
# change !!!
 if [ "$DEV_TYPE" = "sna_dlc_x.25" ]
 then
   mksnaobj -t "$DEV_TYPE" "$DEVICE"
   RC=$?
 else
   mksnaobj -t "$DEV_TYPE" -d "$DEVICE" -b $SOLICIT -w yes -m $IFIELD \
   -H $LSAP -c no -q 0 "$DEVICE"
   RC=$?
 fi
set +x
```

Figure 174 (Part 7 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
# change !!!
   if [ "$DEV_TYPE" = "sna_dlc_x.25" ]
   then
     chsnaobj -t "$DEV_TYPE" "$DEVICE"
   else
     chsnaobj -t "$DEV_TYPE" -d "$DEVICE" -b $SOLICIT -w yes -m $IFIELD \
 -H $LSAP -c no -q 0 "$DEVICE"
   fi
set +x
   line
 fi

}

#
# configure SNA Link Station Profile
#

configure_sna_link ()
{
 # determine type of DLC from datalink device name
 # get only first 3 characters from device name

 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="token_ring" ;;
  "ent" ) DEV_TYPE="ethernet" ;;
  "fdd" ) DEV_TYPE="fddi" ;;
  "x25" ) DEV_TYPE="x.25" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
   abort "No device type found for $DEVICE. Exiting"
 fi

 #
 # create new Link Station Profile
 # use Datalink Device Name as DLC Profile Name
 #
```

Figure 174 (Part 8 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 print "NVDM CONFIG : Configuring SNA Link Station Profile"
 line
set -x
# change !!!
 if [ "$DEV_TYPE" = "x.25" ]
 then
   mksnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -q "$X25_TYPE"\
   -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
   -s "$ADDR" "$PUNAME"
   RC=$?
 else
   mksnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -d "$ADDR" -l $XID\
   -s $RSAP -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
   -c "$USE_CP_XID" "$PUNAME"
   RC=$?
 fi
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   if [ "$DEV_TYPE" = "x.25" ]
   then
     chsnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -q "$X25_TYPE" \
     -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
     -s "$ADDR" "$PUNAME"
   else
     chsnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -d "$ADDR" -l $XID\
     -s $RSAP -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
     -c "$USE_CP_XID" "$PUNAME"
   fi
set +x
   line
 fi
}

#
# configure local LU profile for node
#

configure_sna_local_lu ()
{
 print "NVDM CONFIG : Configuring SNA Local LU Profile"
```

Figure 174 (Part 9 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 #
 # create new Local LU Profile
 # use Local LU Name as Profile Name
 #

 line
set -x
 mksnaobj -t local_lu -u lu6.2 -l "$LLUNAME" -L "$LLUNAME" "$LLUNAME"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_lu -u lu6.2 -l "$LLUNAME" -L "$LLUNAME" "$LLUNAME"
set +x
   line
 fi

}

#
# configure LU6.2 location profile
#

configure_sna_location ()
{
 print "NVDM CONFIG : Configuring SNA LU 6.2 Location Profile"

 #
 # create new LU 6.2 Location Profile
 # use Local LU Name as Profile Name
 #

 line
set -x
 mksnaobj -t partner_lu6.2_location -P "$SNA_NET.$PARTNER" \
 -O "$SNA_NET.$VTAMCP" -m link_station -l $LLUNAME \
 -s $PUNAME $PARTNER
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
```

Figure 174 (Part 10 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
 chsnaobj -t partner_lu6.2_location -P "$SNA_NET.$PARTNER" \
 -O "$SNA_NET.$VTAMCP" -m link_station -l $LLUNAME \
 -s $PUNAME $PARTNER
set +x
   line
 fi

}

#
# configure SNA Mode Profile
#

configure_sna_mode ()
{
 #
 # create new Mode Profile
 #

 print "NVDM CONFIG : Configuring SNA Mode Profile"
 line
set -x
 mksnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N "#CONNECT" -m "$MODE" "$MPROF"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N "#CONNECT" -m "$MODE" "$MPROF"
set +x
   line
 fi

}

#
# configure TPN send profile
#
```

Figure 174 (Part 11 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
configure_sna_send ()
{
 #
 # create TPN Profile (Send)
 #

 print "NVDM CONFIG : Configuring SNA TPN Profile (SEND)"
 line
set -x
 mksnaobj -t local_tp -n 21F0F0F7 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndts -s none "$SND"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_tp -n 21F0F0F7 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndts -s none "$SND"
set +x
   line
 fi

}

#
# configure TPN receive profile
#

configure_sna_receive ()
{
 #
 # create TPN Profile (Receive)
 #

 print "NVDM CONFIG : Configuring SNA TPN Profile (Receive)"
 line
set -x
 mksnaobj -t local_tp -n 21F0F0F8 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndtr -s none "$RCV"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
```

Figure 174 (Part 12 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_tp -n 21F0F0F8 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndtr -s none "$RCV"
set +x
   line
 fi

}

#
# Configure partner LU profile (Focal Point)
#

configure_sna_partner ()
{
 #
 # create LU 6.2 Partner Profile
 #

 print "NVDM CONFIG : Configuring SNA LU6.2 Partner LU"
 line
set -x
 mksnaobj -t partner_lu6.2 -p no -P "$SNA_NET"."$PARTNER" \
 -O none -A "$PARTNER" "$PARTNER"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
 chsnaobj -t partner_lu6.2 -p no -P "$SNA_NET"."$PARTNER" \
 -O none -A "$PARTNER" "$PARTNER"
set +x
   line
 fi

}

#
# configure Side Info Profile (Send)
```

Figure 174 (Part 13 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
#

configure_side_snd ()
{
 #
 # create Side Info Profile (Send)
 #

 print "NVDM CONFIG : Configuring SNA Side Info Profile (Send)"
 line
set -x
 mksnaobj -t side_info -L "$CP_NAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F7 -h yes "$SIDS"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t side_info -L "$CP_NAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F7 -h yes "$SIDS"
set +x
   line
 fi

}

#
# configure Side Info Profile (Receive)
#

configure_side_rcv ()
{
 #
 # create Side Info Profile (Receive)
 #

 print "NVDM CONFIG : Configuring SNA Side Info Profile (Receive)"
 line
set -x
 mksnaobj -t side_info -L "$LLUNAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F8 -h yes "$SIDR"
 RC=$?
set +x
 line
```

Figure 174 (Part 14 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t side_info -L "$LLUNAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F8 -h yes "$SIDR"
set +x
   line
 fi

}

#
# get queues defined for a server
# since this class can contain more
# than one entry for a server, we have
# to store the result in a list
#
# $1 = server name
#

get_queues ()
{
 #
 # first, determine number of entries for
 # that server
 #

 #
 # Fill in Fields
 #

 get_attribute_list nvdm_queues node_name $1 protocol
 NUM_QUEUE=$VALUE_NUM
 if [ $NUM_QUEUE = 0 ]
 then
   return
 fi

 PROTOCOL=$VALUE_LIST
 get_attribute_list nvdm_queues node_name $1 remote_server
 REMOTE_SERVER=$VALUE_LIST

}

#
```

Figure 174 (Part 15 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
# Configure SNA/DS connection configuration file (APPC)
#

configure_sna_ds_appc ()
{
 print "NVDM CONFIG : Configuring SNA/DS connection\
 configuration file $SNA_DS_DIR/$PARTNER"
 echo "PROTOCOL:                      APPC
SEND TP SYMBOLIC DESTINATION:     $SIDS
RECEIVE TP SYMBOLIC DESTINATION:  $SIDR
NEXT DSU:                         $SNA_NET.$PARTNER
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" > $SNA_DS_DIR/$PARTNER
}

#
# Configure SNA/DS connection configuration file (TCP/IP)
# $1 = TCP/IP Hostname of remote system
#

configure_sna_ds_tcpip ()
{
 #
 # get short name of remote server
 #

 get_attribute nvdm_node node_name $1 short_name
 A=$VALUE
 print "NVDM CONFIG : Configuring SNA/DS connection configuration file."
 print "NVDM CONFIG : (TCP/IP) for remote Server $A."

 echo "PROTOCOL:                          TCP/IP
REMOTE SERVER NAME:               $1
TCP/IP TIME-OUT:                  300
NEXT DSU:                         $A.$A
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" >$SNA_DS_DIR/$A
}


#
# delete local targets from NVDM Server configuration
# $1 = Server IP Hostname
#

nvdm_delete_targets()
```

Figure 174 (Part 16 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
{
 #
 # get list of existing targets
 #

 TLIST=`nvdm lstg '*' | grep "Target:" | cut -d':' -f2`

 #
 # get list of all defined targets for this server
 #

 get_attribute_list nvdm_node server_name $1 node_name
 YLIST=$VALUE_LIST
 XLIST=""
 for i in $YLIST
 do
  XLIST=$XLIST" "`echo $i | cut -d'.' -f1`
 done

 #
 # delete all targets which are not defined for this server
 #

 for i in $TLIST
 do
  match=0
  for x in $XLIST
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    nvdm_save_history $i
    print "NVDM CONFIG : Deleting Target $i from Server $1 configuration."
    nvdm deltg $i -f
  fi
 done
}

#
# Delete all existing groups before adding groups from
# configuration database
# $1 = IP Hostname of server to be configured
#

nvdm_delete_groups ()
{
```

Figure 174 (Part 17 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 #
 # determine existing groups
 #
 GP=`nvdm lsgp '*' | grep -E "Push|Pull" | cut -d' ' -f1`
 #
 # determine list of defined groups
 #
 get_attribute_list nvdm_groups node_name $1 group_name
 XGP=$VALUE_LIST

 for i in $GP
 do
  match=0
  for x in $XGP
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    print "NVDM CONFIG : Deleting group $i from $1 configuration."
    nvdm delgp $i -f
  fi
 done
}

#
# configure Targets for an NVDM/6000 Server
# $1 = Server IP Hostname
#

nvdm_configure_targets ()
{
 #
 # First, determine all Nodes which have this Server
 # defined as their NVDM/6000 server
 #

 # access database

 get_attribute_list nvdm_node server_name $1 node_name
 ATLIST=$VALUE_LIST
 TLIST=""
 for i in $ATLIST
 do
  TLIST=$TLIST" "`echo $i | cut -d'.' -f1`
 done
```

Figure 174 (Part 18 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 count=0
 for i in $TLIST
 do
   count=`expr $count + 1`
   print "NVDM CONFIG : Defining Target $i on server $1"

   nvdm lstg $i 1>/dev/null 2>&1

   #
   # if return code = 0 then target exists already
   #

   if [ $? -ne 0 ]
   then
     COMMAND="nvdm addtg $i"
   else
     COMMAND="nvdm updtg $i"
     print "NVDM CONFIG : Target already exists. Updating..."
   fi

   #
   # get required target attributes
   #

   huhn=`echo $ATLIST | cut -d' ' -f$count`

   for a in short_name target_os description contact_name\
 owning_manager telephone_number customer_name
   do
     get_attribute nvdm_node node_name $huhn $a
     v=$VALUE
     if [ "$v" != "" ]
     then
       case $a in
         short_name)     COMMAND=$COMMAND" -s '$v'" ;;
         target_os)      COMMAND=$COMMAND" -y '$v'" ;;
         description)    COMMAND=$COMMAND" -d '$v'" ;;
         contact_name)   COMMAND=$COMMAND" -q '$v'" ;;
         owning_manager) COMMAND=$COMMAND" -o '$v'" ;;
         telephone_number) COMMAND=$COMMAND" -t '$v'" ;;
         customer_name)    COMMAND=$COMMAND" -r '$v'" ;;
       esac
     fi
   done
   echo $COMMAND
   eval $COMMAND

#
# add users for target
#
```

Figure 174 (Part 19 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
get_attribute_list nvdm_users node_name $huhn username
if [ $VALUE_NUM != 0 ]
then
  print "NVDM CONFIG : Adding Target Users..."
  for x in $VALUE_LIST
  do
    print "NVDM CONFIG : Adding $x User"
    nvdm updtg $i -u $x
  done
fi

done

}

#
# configure groups defined for NVDM/6000 server
#

nvdm_configure_groups ()
{
 print "NVDM CONFIG : Configuring Target Groups for $1"
 get_attribute_list nvdm_groups node_name $1 group_name
 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No groups defined"
   return
 fi
 GROUP_LIST=$VALUE_LIST
 for i in $GROUP_LIST
 do
   print "NVDM CONFIG : Adding group $i"
   get_attribute nvdm_groups group_name $i short_name
   SHORT=$VALUE
   get_attribute nvdm_groups group_name $i description
   DESC=$VALUE
   #
   # get all targets being defined for this group
   #

   get_attribute_list nvdm_node group_name $i node_name

   for a in $VALUE_LIST
   do
     TNGP=`echo $a | cut -d'.' -f1`
     eval nvdm addgp $i $TNGP -s "'$SHORT'" -d "'$DESC'"
   done
 done
}
```

Figure 174 (Part 20 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
#
# add user at OS level (AIX)
# $1 = IP Hostname
# $2 = Type: either "server" or "target"
#            use "target", when you want to add a user to AIX
#            add a target workstation; the user will always be
#            assigned group FNDADMN
#            use "server", when you want to add a user to AIX
#            add a server workstation; the user will be assigned
#            the appropriate usergroup defined in the database
#

add_users_aix ()
{
print "NVDM CONFIG : --> Adding AIX users for NVDM..."
get_attribute_list nvdm_users node_name $1 username
if [ $VALUE_NUM != 0 ]
then
  for i in $VALUE_LIST
  do
    #
    # First, add NVDM user to operating system...
    # check if user exists
    #
    lsuser $i 2>/dev/null 1>&2
    #
    # if not (RC 2 from lsuser command)
    #
    if [ $? = 2 ]
    then
      print "NVDM CONFIG : Adding user $i to AIX OS."
      mkuser $i
    fi
    #
    # check if user has NVDM group
    #
    get_attribute_and nvdm_users node_name $1 username $i usergroup
    GRP=$VALUE
    #
    # if we configure a target, set group to FNDADMN
    #
    if [ "$2" = "target" ]
    then
      GRP=FNDADMN
    fi
    DEFGRP=`lsuser -a groups $i | cut -d'=' -f2`
    # if user is not in NVDM group, add him
    if [ "`echo $DEFGRP | grep $GRP`" = "" ]
    then
```

Figure 174 (Part 21 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
      chuser groups="$DEFGRP,$GRP" $i
    fi
  done
fi
}

#
# configure SNA/DS routing table
# $1 = IP Hostname
#

configure_routetab ()
{
 #
 # first, determine what network protocols we have
 #
 a=0
 b=0
 print "NVDM CONFIG : Configuring SNA/DS routing table."
 get_attribute_and nvdm_queues node_name $1 protocol TCP/IP remote_server
 if [ "$VALUE" != "" ]
 then
   print "NVDM CONFIG : System has TCP/IP connection to remote server."
   a=1
 fi

 get_attribute_and nvdm_queues node_name $1 protocol APPC remote_server
 if [ "$VALUE" != "" ]
 then
   print "NVDM CONFIG : System has APPC connection to remote server."
   b=1
 fi

 if [ $a -eq 0 -a $b -eq 0 ]
 then
   print "NVDM CONFIG : There are no connections defined."
   return
 fi

 if [ $a -eq 1 -a $b -eq 1 ]
 then
   RPROT="BOTH"
 fi

 if [ $a -eq 1 -a $b -eq 0 ]
 then
   RPROT="TCP/IP"
 fi

 if [ $a -eq 0 -a $b -eq 1 ]
```

Figure 174 (Part 22 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
 then
   RPROT="APPC"
 fi

 print "NVDM CONFIG : Writing routing table to $SNA_DS_ROUTE"
 echo "NETWORK PROTOCOL: $RPROT

#
# SNA connections
#
" >$SNA_DS_ROUTE

#
# get all SNA Routes
#

 get_attribute_and nvdm_queues node_name $1 protocol APPC remote_server
 SNA_R=$VALUE
 if [ "$SNA_R" != "" ]
 then
   for i in $SNA_R
   do
     # check if intermediate node is used
     get_attribute_and nvdm_queues node_name $1 remote_server $i inter_node
     if [ "$VALUE" != "" ]
     then
       echo "$SNA_NET.$i   ANY     ANY     ANY     ANY     $VALUE     5" >>$SNA_DS_ROUTE
     else
       echo "$SNA_NET.$i   ANY     ANY     ANY     ANY     $i     5" >>$SNA_DS_ROUTE
     fi
   done
 fi

 echo "
#
# TCP/IP connections
#
" >>$SNA_DS_ROUTE
 get_attribute_and nvdm_queues node_name $1 protocol TCP/IP remote_server
 TCP_R=$VALUE
 if [ "$TCP_R" != "" ]
 then
   for i in $TCP_R
   do
     # in the routing table we need the short name, not the
     # TCP/IP hostname as specified in remote_server ; therefore
     # we have to get the shortname first
     # check if intermediate node is used
     get_attribute nvdm_node node_name $i short_name
     sn=$VALUE
```

Figure 174 (Part 23 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
     get_attribute_and nvdm_queues node_name $1 remote_server $i inter_node
     if [ "$VALUE" != "" ]
     then
       echo "$sn.*                                        $VALUE" >>$SNA_DS_ROUTE
     else
       echo "$sn.*                                        $sn" >>$SNA_DS_ROUTE
     fi
   done
 fi
}

#
# configure Remote Targets
# $1 = IP Hostname
#

nvdm_remote_targets ()
{
 #
 # First, get all remote targets defined for this server
 # Remote Targets are determined by searching the nvdm_queues
 # class because any connection to a remote system requires a
 # queue

 get_attribute_list nvdm_queues node_name $1 remote_server

 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No remote targets defined"
   return
 fi

 for i in $VALUE_LIST
 do
   print "NVDM CONFIG : Defining remote target for $i"

   #
   # determine if system to be configured is a Remote Target or
   # a Focal Point
   #
   get_attribute_and nvdm_queues node_name $1 remote_server $i focal_point

   if [ "$VALUE" = "yes" ]
   then
     print "NVDM CONFIG : $i will be configured as focal point."
     # for the MVS focal point short name will be the same as node name
     # network id will be the SNA Network Name

set -x
     eval nvdm addtg $i -m report_to -s $i -n $SNA_NET -d "'NVDM_MVS'"
```
Figure 174 (Part 24 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
set +x
   else
     # get short name for remote server from class nvdm_node
     get_attribute nvdm_node node_name $i short_name
     if [ "$VALUE" = "" ]
     then
       abort "No Short Name defined for $i in class nvdm_node. Exiting..."
     fi
     RSHORT=$VALUE
     #
     # This remote server is assumed to be connected via TCP/IP
     # so, we set the network name to be the same as the short name
     #
     nvdm addtg $i -m remote -s $RSHORT -n $RSHORT
   fi
 done
}

restart_nvdm ()
{
 print "NVDM CONFIG : --> In order for the changes to become active"
 print "NVDM CONFIG :     NetView DM/6000 will be restarted on this node"

 #
 # determine if nvdm is running
 #

 nvdm stat 1>/dev/null 2>&1

 if [ $? = 121 ]
 then
   print "NVDM CONFIG : NVDM is not running. It will be started now."
   nvdm start
   nvdm start
 else
   print "NVDM CONFIG : Stopping NVDM."
   nvdm stop -x 1>/dev/null 2>&1
   s=1
   print "NVDM CONFIG : Restarting NVDM."
   while [ $s = 1 ]
   do
     print "NVDM CONFIG : Restarting NVDM."
     nvdm start
     nvdm stat
     if [ $? != 121 ]
     then
       s=0
     fi
   done
 fi
```

Figure 174 (Part 25 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
}

#
# configure SNA/DS connection profiles
#
# $1 = IP Hostname of system to be configured
#

configure_sna_ds_conn ()
{

#
# perform SNA/DS configuration (connection profiles)
#

#
# remove demo profile CONNSNA,CONNTCP if existent
#
cd $SNA_DS_DIR
rm *

get_queues $1

if [ $NUM_QUEUE != 0 ]
then
  a=1
  for i in $PROTOCOL
  do
    print "NVDM CONFIG : Configuring $i connection"
    if [ "$i" != "APPC" -a  "$i" != "TCP/IP" ]
    then
      abort "Protocol is neither APPC nor TCP/IP. Exiting..."
    fi

    # determine if connection is made through an intermediate node

    INODE=`echo $REMOTE_SERVER | cut -d' ' -f"$a"`
    get_attribute_and nvdm_queues node_name $1 remote_server $INODE inter_node
    if [ "$VALUE" != "" ]
    then
      print "NVDM CONFIG : Remote connection to $INODE is made"
      print "              through intermediate node $VALUE."
      print "              No SNA/DS connection file is created."
    else
      if [ "$i" = "APPC" ]
      then
        configure_sna_ds_appc
      else
        REMSERV=`echo $REMOTE_SERVER | cut -d' ' -f "$a"`
        configure_sna_ds_tcpip $REMSERV
```

Figure 174 (Part 26 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
      fi
    fi
    a=`expr $a + 1`
  done
fi
}



#
# get all static SNA attributes (SNA Net Name, etc.)
# $1 = IP Hostname of node to be configured
#

get_sna_attributes ()
{

  #
  # get static SNA parameters
  #

  for i in SNA_NET_NAME DATALINK_DEVICE REM_LINK_ADDR MODE_PROF_NAME\
 MODE_NAME TPN_PROF_NAME_SND TPN_PROF_NAME_RCV PARTNER_LU_NAME\
 SIDE_INFO_PROF_SND SIDE_INFO_PROF_RCV SOLICIT_SSCP I_FIELD_SIZE\
 LOCAL_SAP REMOTE_SAP INITIATE_CALL ACTIVATE_START RESTART_NORMAL\
 RESTART_ABNORMAL VTAM_CP_NAME
  do
    get_attribute nvdm_cfg_static NAME $i VALUE
    case $i in
      SNA_NET_NAME)       text="SNA Network Name"
                          SNA_NET=$VALUE ;;
      DATALINK_DEVICE)    text="SNA Datalink Device"
                          DEVICE=$VALUE ;;
      REM_LINK_ADDR)      text="SNA Remote Link Address"
                          ADDR=$VALUE ;;
      MODE_PROF_NAME)     text="SNA NVDM Mode Profile Name"
                          MPROF=$VALUE ;;
      MODE_NAME)          text="SNA NVDM Mode Name"
                          MODE=$VALUE ;;
      TPN_PROF_NAME_SND)  text="SNA TPN Profile Name (Send)"
                          SND=$VALUE ;;
      TPN_PROF_NAME_RCV)  text="SNA TPN Profile Name (Receive)"
                          RCV=$VALUE ;;
      PARTNER_LU_NAME)    text="SNA Partner LU Name (MVS Host)"
                          PARTNER=$VALUE ;;
      SIDE_INFO_PROF_SND) text="SNA Side Info Profile Name (Send)"
                          SIDS=$VALUE ;;
      SIDE_INFO_PROF_RCV) text="SNA Side Info Profile Name (Receive)"
                          SIDR=$VALUE ;;
      SOLICIT_SSCP)       text="Solicit SSCP Field (yes|no)"
                          SOLICIT=$VALUE ;;
```

Figure 174 (Part 27 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
      I_FIELD_SIZE)       text="I-Field Size"
                          IFIELD=$VALUE ;;
      LOCAL_SAP)          text="SNA Local SAP No."
                          LSAP=$VALUE ;;
      REMOTE_SAP)         text="Remote SAP No."
                          RSAP=$VALUE ;;
      INITIATE_CALL)      text="SNA Initiate Call Field (yes|no)"
                          ICALL=$VALUE ;;
      ACTIVATE_START)     text="SNA Activate on start (yes|no)"
                          ACTSTART=$VALUE ;;
      RESTART_NORMAL) text="SNA Restart on normal termination (yes|no)"
                          RNORM=$VALUE ;;
      RESTART_ABNORMAL) text="SNA Restart on abnormal termination (yes|no)"
                          RABNORM=$VALUE ;;
      VTAM_CP_NAME) text="SNA VTAM CP Name (for LU6.2 Location Profile)"
                          VTAMCP=$VALUE ;;

    esac
    if [ "$VALUE" = "" ]
    then
      abort "Could not determine $text. Exiting..."
    else
      print "NVDM CONFIG : Setting $text to $VALUE"
    fi
  done

  get_attribute nvdm_servers node_name $1 pu_name
  PUNAME=$VALUE
  if [ "$PUNAME" = "" ]
  then
    abort "Could not determine PU NAME for $1 configuration
. Exiting..."
  fi

  print "NVDM CONFIG : Setting PU NAME for $1 to $PUNAME "

  get_attribute nvdm_servers node_name $1 local_lu_name
  LLUNAME=$VALUE
  if [ "$LLUNAME" = "" ]
  then
    abort "Could not determine Local LU Name for $1 configu
ration. Exiting..."
  fi

  print "NVDM CONFIG : Setting Local LU Name for $1 to $LLUNAME "

  get_attribute nvdm_servers node_name $1 cp_name
  CP_NAME=$VALUE
  if [ "$CP_NAME" = "" ]
  then
```

Figure 174 (Part 28 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
    abort "Could not determine Control Point Name for $1.\
 Exiting..."
  fi
  CP_TYPE=appn_end_node

  print "NVDM CONFIG : Setting Control Point Name for $1\
to $CP_NAME"

  get_attribute nvdm_servers node_name $1 xid
  XID=$VALUE
  if [ "$XID" = "" ]
  then
    print "NVDM CONFIG : Could not determine XID for $1 configu
ration."
    print "NVDM CONFIG : Setting USE_CP_XID to yes"
    USE_CP_XID="yes"
    # set XID to dummy value
    XID=07100000
  else
    print "NVDM CONFIG : Setting XID for $1 to $XID "
    print "NVDM CONFIG : Setting USE_CP_XID to no"
    USE_CP_XID="no"
  fi
}

#
# Save NVDM target history by creating software inventory
# file and copying it to corresponding node
# requires /.rhosts file on target
# $1 = target name
#

nvdm_save_history ()
{
 print "NVDM CONFIG : Saving target history for $1"
 nvdm inv
 SLIST="`nvdm lscm -w $1 '*' | grep 'Global file name:' | cut -d':' -f2`"
 >/tmp/inv
 if [ "$SLIST" != "" ]
 then
   for o in $SLIST
   do
     print "NVDM CONFIG : Adding $o to software inventory file."
     print "PRODUCT: "$o >>/tmp/inv
     print "DESCRIPTION: Target has been moved!" >>/tmp/inv
   done
   print "NVDM CONFIG : Copying inventory file $SW_INV to $1."
   echo "GLOBAL NAME:                   HISTORY.REF.1
CHANGE FILE TYPE:              GEN
COMPRESSION TYPE:              LZW
```

Figure 174 (Part 29 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
REBOOT REQUIRED:               NO
PACK FILES:                    NO
SECURE PACKAGE:                NO
OBJECT:
SOURCE NAME:               /tmp/inv
TARGET NAME:               /usr/lpp/netviewdm/fndswinv
TYPE:                      FILE
ACTION:                    COPY
INCLUDE SUBDIRS:           NO" >/tmp/hist.pro
   nvdm delcm HISTORY.REF.1 -w '*'
   nvdm uncat HISTORY.REF.1 -d -f
   nvdm bld /tmp/hist.pro -f
   nvdm inst HISTORY.REF.1 -w $1 -f -i
   print "CONFIG NVDM : Sleeping for 5 secs."
   sleep 5
 fi
}

#
# add file system for repository
# $1 = node name
#

add_fs_repos ()
{
 # get repository path
 REPOS=`grep "REPOSITORY" /usr/lpp/netviewdm/db/nvdm.cfg \
 | cut -d':' -f2`
 get_attribute nvdm_node node_name $1 repos_fs
 if [ "$VALUE" = "yes" ]
 then
   get_attribute nvdm_node node_name $1 repos_size
   if [ "$VALUE" = "" ]
   then
     SIZE=20000
   else
     SIZE=$VALUE
   fi
   print "NVDM CONFIG : Creating file system $REPOS."
   print "NVDM CONFIG : Size = $SIZE blocks."
   # first, save old files
   tar -cvf/tmp/save.tar $REPOS/.
   crfs -v jfs -g rootvg -a size=$SIZE -m $REPOS -A yes -p rw -t no
   mount $REPOS
   # restore files
   tar -xvf/tmp/save.tar $REPOS/.
 fi
}
```

Figure 174 (Part 30 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
#
# check if TCP/IP ports for NetView DM/6000 are
# existing. If not, add them to /etc/services file
#
check_ports ()
{
 #
 # first, make a backup copy of /etc/services
 #
 cp /etc/services /etc/services.nvdm
 #
 # check for port NetViewDM-rcv
 #
 print "CONFIG NVDM : Checking NetViewDM-rcv port..."
 R=`grep NetViewDM-rcv /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM-rcv   731/tcp" >>/etc/services
 fi
 #
 # check for port NetViewDM-snd
 #
 print "CONFIG NVDM : Checking NetViewDM-snd port..."
 R=`grep NetViewDM-snd /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM-snd   730/tcp" >>/etc/services
 fi
 #
 # check for port NetViewDM6000
 #
 print "CONFIG NVDM : Checking NetViewDM6000 port..."
 R=`grep NetViewDM6000 /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM6000   729/tcp" >>/etc/services
 fi
}

#
# export existing SNA profiles
# in case they need to be restored if
# NVDM configuration fails
#
# $1 = name of export file
#
export_sna ()
```

Figure 174 (Part 31 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
{
 print "NVDM CONFIG : Exporting existing SNA profiles to $1 ..."
 exportsna -A -f $1 -r -UT -C
}

#
# update NVDM/6000 server definition
#

nvdm_update_server ()
{
#-------------change--------------
# if we configure a server and want to change the
# WORKSTATION NAME we must stop the server before
# reconfiguring this field.
# To do so we must be sure that the hostname and
# the WORKSTATION NAME match
#
if [ "$NODE_TYPE" = "0" -o "$NODE_TYPE" = "2" ]
then
  # get current hostname
  OHN=`hostname`
  print "NVDM CONFIG : Current hostname of server is $OHN."
  # get WORKSTATION NAME currently configured
  HN=`grep "WORKSTATION NAME:" $CONFIG | cut -d':' -f2`
  print "NVDM CONFIG : Current WORKSTATION NAME of server is $HN."
  print "NVDM CONFIG : Stopping Server..."
  # make sure that both match
  hostname $HN
  nvdm stop -x
  print "NVDM CONFIG : Sleeping 20 seconds..."
  sleep 20
  # set back hostname
  print "NVDM CONFIG : Setting hostname to $OHN."
  hostname $OHN
  # also, we must be sure that there is an initial target record for
  # the servername configured
  ls /usr/lpp/netviewdm/db/target_config/$1 >/dev/null 2>&1
  if [ $? -ne 0 ]
  then
    echo "DESCRIPTION:     INITIAL TARGET CONFIGURATION RECORD
TARGET TYPE:     PUSH
TARGET OS:       AIX
RBAPI TRACE:     NONE
LOG LEVEL:       N
SHORT NAME:      SERVER
CM WINDOW START:           0 : 0
CM WINDOW STOP:            23:59
DISTRIBUTION WINDOW START: 0 : 0
DISTRIBUTION WINDOW STOP:  23:59
```

Figure 174 (Part 32 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
NUMBER OF PARMS: 0
NUMBER OF USERS: 1
USER: root" >/usr/lpp/netviewdm/db/target_config/$1
  fi
fi
#-----------end-of-change---------
}

#
#
# ***** MAIN *****
#
#

debug_info
print "NVDM CONFIG : --> Trying to configure node $1"

#
# determine node type
#

get_attribute nvdm_node node_name $1 node_type

if [ "$VALUE" = "" ]
then
  abort "No Database match found for $1."
fi
NODE_TYPE=$VALUE
print "NVDM CONFIG : Node type is $NODE_TYPE (0 = Server, 1 = Agent, 2 = Prep)"

#
#                                                           #
# steps necessary for all nodes (server, agent, prep site) #
#                                                           #
#

print "NVDM CONFIG : --> NVDM Base Node Configuration"

#
# add file system for repository
#

#add_fs_repos $1

#
# update hostname/NVDM server name
#
nvdm_update_server
```

Figure 174 (Part 33 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
#
# configure WORKSTATION NAME
#

configure_nvdm_cfg "WORKSTATION NAME" $HNAME

#
# configure SERVER
#

get_attribute nvdm_node node_name $1 server_name
SERVER=$VALUE
HSERVER=`echo $SERVER | cut -d'.' -f1`
configure_nvdm_cfg "SERVER" $HSERVER

#
# configure NVDM LOG SIZE
#
get_attribute nvdm_cfg_static NAME NVDM_LOG_SIZE VALUE
if [ "$VALUE" != "" ]
then
 configure_nvdm_cfg "LOG FILE SIZE" $VALUE
fi

#
# check for NetView DM ports

check_ports

#
# configure TCP/IP port to be used by NetView DM
#
get_attribute nvdm_cfg_static NAME TCPIP_PORT VALUE
if [ "$VALUE" != "" ]
then
  configure_nvdm_cfg "TCP/IP PORT" $VALUE
fi

#-------------change--------------
get_attribute nvdm_cfg_static NAME REPOS_DIR VALUE
if [ "$VALUE" != "" ]
then
  VALUE=`echo "$VALUE"`
  configure_nvdm_cfg "REPOSITORY" $VALUE
fi
#-----------end-of-change---------

#
# add users at target
#
```

Figure 174 (Part 34 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
if [ "$NODE_TYPE" = "2" ]
then
    add_users_aix $1 target
fi

#-------change-----------

# reset /usr/lpp/netviewdm/uicfg/root.cli file
print "NVDM CONFIG : Resetting root.cli ... ($HSERVER)"
./uicfg $HSERVER

#-------end of change----

if [ "$NODE_TYPE" = "1" ]
then
  print "NVDM CONFIG : Starting NVDM Agent (fndcmps)...."
  /usr/lpp/netviewdm/bin/fndcmps &
  line
  banner SUCCESS!
  line
  print "NVDM CONFIG : !!! Configuration of Agent completed successfully !!!"
  line
  exit 0
fi

#
#
# Server configuration
#
#

#-------------change--------------
# restart server, in case it is not already running
print "NVDM CONFIG : Restarting Server..."
restart_nvdm
#-----------end-of-change---------

# add all target users also to server

get_attribute_list nvdm_node server_name $1 node_name
if [ "$VALUE_LIST" != "" ]
then
  for i in $VALUE_LIST
  do
      add_users_aix $i server
  done
fi
```

Figure 174 (Part 35 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
#
#
# SNA Configuration for Servers
#
#

#
# determine if node needs SNA configuration
#

#
# get all static SNA attributes
#

get_sna_attributes $1

get_attribute nvdm_servers node_name $1 sna


if [ "$VALUE" = "yes" ]
then

  print "NVDM CONFIG : --> Configuring SNA "

  #
  # Configure SNA
  #

  export_sna $EXPORT_SNA
  configure_sna_dlc
  sna_initial
  configure_sna_cp
  configure_sna_dlc_profile
  configure_sna_link
  configure_sna_local_lu
  configure_sna_mode
  configure_sna_send
  configure_sna_receive
  configure_sna_partner
  configure_sna_location
  configure_side_snd
  configure_side_rcv

  #
  # After SNA has been configured, update configuration database
  # to make changes become active
  #

  print "NVDM CONFIG : Updating SNA Server..."
  verifysna -R
```

Figure 174 (Part 36 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
fi # end of SNA configuration

#
# perform SNA/DS configuration (connection profiles)
#

configure_sna_ds_conn $1

#
# configure SNA/DS Routing Table
#

configure_routetab $1

#
# Delete all existing targets in case of node reconfiguration
#

nvdm_delete_targets $1

#
# Configure all local targets for NVDM/6000 server
#

nvdm_configure_targets $1

#
# Delete existing groups
#

nvdm_delete_groups $1

#
# Configure all groups
#

nvdm_configure_groups $1

#
# Configure all remote targets/focal point for NVDM/6000 Server
#

nvdm_remote_targets $1

#
# Reload nvdm configuration
# This will only refresh SNA/DS configuration 'in flight'

nvdm rld
```

Figure 174 (Part 37 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

```
#
# Start SNA Server subsystem
#

startsrc -s sna

#
# determine if NVDM has to be restarted
#

get_attribute nvdm_cfg_static NAME RESTART_NVDM VALUE
if [ "$VALUE" = "yes" ]
then
  restart_nvdm
fi

#
# release all SNA communications
#

print "NVDM CONFIG : Releasing NVDM SNA communications."
nvdm relc

line
banner SUCCESS!
line
print "NVDM CONFIG : !!! Configuration of Server completed successfully !!!"
line
```

Figure 174 (Part 38 of 38). config_nvdm Shell Script for NetView DM/6000 V1.2 with ODM Database

A.2 Configuration Script for NetView DM for AIX Version 3.1 Using DB2/6000

```
#!/bin/ksh
#
#
# Configure NVDM node
# Main Configuration Script
# For NetView DM/6000 V3.1
# ----------------------------------------------------
# This script can be used to configure any RS/6000
# workstation in your software distribution network
# automatically
# ----------------------------------------------------
# Author : Stefan Uelpenich/IBM Germany
# RCS Revision :  $Revision: 1.1 $
# ----------------------------------------------------
#
# This script will cover:
#
# 1. For all nodes
#    - configuration of WORKSTATION NAME in nvdm.cfg
#    - configuration of SERVER in nvdm.cfg
#    - configuration of TCP/IP ports used by NVDM
#    - configuration of log file size & other things
#      in nvdm.cfg
#    - add NVDM Users to AIX Operating System / NetView DM
#
# 2. For servers/prep sites
#    - modification of server's own target
#    - add DLC Device for SNA adapter
#    - SNA initial node setup
#    - configuration of SNA CP profile
#    - configuration of SNA DLC profile
#    - configuration of SNA Link profile
#    - configuration of SNA Local LU profile
#    - configuration of SNA Mode profile
#    - configuration of SNA TPN Send profile
#    - configuration of SNA TPN Receive profile
#    - configuration of SNA LU6.2 Location profile
#    - configuration of SNA Side Info profile (Send)
#    - configuration of SNA Side Info profile (Receive)
#    - configuration of SNA/DS connection profiles
#    - configuration of SNA/DS Routing table
#    - configuration of local targets
#    - configuration of local target groups
#    - configuration of remote targets/focal points
#    - reload NVDM Configuration
#    - refresh SNA Server Configuration
#    - start SNA Server
#    - restart NVDM
#    - release NVDM SNA communications
```

Figure 175 (Part 1 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
#
#

#
# The command line parameter supplied with this command
# must be the IP hostname of the system to be configured.
# This hostname will be used as the argument when
# accessing the configuration database
#

if [ $# != 1 ]
then
  print "Syntax : $0 node_name"
  exit 1
fi

#
# extract hostname (without domain information)
#

HNAME=`echo $1 | cut -d'.' -f1`
print "NVDM CONFIG : Extracted hostname ... $HNAME"

#
# Variables
#

CONFIG=/usr/lpp/netviewdm/db/nvdm.cfg
NUM_QUEUE=0
PROTOCOL=""
REMOTE_SERVER=""
EXPORT_SNA=/tmp/sna.org
SNA_DS_DIR="/usr/lpp/netviewdm/db/snadscon"
SNA_DS_ROUTE="/usr/lpp/netviewdm/db/routetab"
HISTORY_DIR="/usr/lpp/netviewdm/db/cm_status"
SAVE_DIR="/tmp/target_save"
USE_CP_XID=no
SW_INV="/usr/lpp/netviewdm/fndswinv"

#
#
# useful stuff
#
#

# print a line

line ()
{
 print "====================================\
```

Figure 175 (Part 2 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
============================================"
}

#
# print debug information
#

debug_info ()
{
 line
 print "Software distribution network configuration script"
 print "\$Revision: 1.1 $"
 BEK=`hostname`
 print "IP Hostname = $DEB"
 print "Name resolution = "`host $DEB`
 line
}

#
# abort configuration script
# and print an error message
# $1 = text of error message
#

abort ()
{
 line
 banner "FAILURE!"
 line
 print "NVDM CONFIG ERROR :\
 Could not properly configure node."
 print "Cause : $1"
 line
 exit 1
}




#
#
# DATABASE ACCESS METHODS (DB2)
#
#



# database owner name
#---------------------
DBOWNER=dbmsadm
```

Figure 175 (Part 3 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
#
# connect to the configuration database
#--------------------------------------
print "DB2/6000 : Connect to configuration database"
db2 connect



#
# get data output from SQL (extract SQL header and trailer)
# $1: select clause
# $2: tables (from clause)
# $3: conditions (where clause)
#-----------------------------------------------------------
get_data()
{
  WHERE="$3"
  if [ "$WHERE" = "" ]
  then
    WHERE="1=1"
  fi
  SELECT="$1"
  db2 select "$SELECT" from $2 where "$WHERE" | awk '
        BEGIN   {
                  inlist = 0
                }
        /^SQL[0-9][0-9][0-9][0-9][N,C]/ {
                  cmd = sprintf("exec 1>&2;echo DB2/6000 : %s",$0)
                  system(cmd)
                }
        /^-+/   {
                  inlist = 1
                  next
                }
        /^$/    {
                  if (inlist == 1) inlist++
                  next
                }
        inlist == 1 {
                  gsub(/ *$/,"")
                  print
                }
'

}
```

Figure 175 (Part 4 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
#
# get list of selected column values from a DB2 table
# $1 = table name
# $2 = search column name
# $3 = search column value
# $4 = output column name
# The list of selected column values is stored in the VALUE_LIST variable
# The number of selected values is stored in VALUE_NUM
#---------------------------------------------------------------------------
get_attribute_list ()
{
  VALUE_LIST=`get_data "$4" $DBOWNER.$1 "$2 = '$3'"`
  VALUE_NUM=`echo "$VALUE_LIST" | wc -w | sed 's/ //g'`
}

#
# get single select value
# $1 = table name
# $2 = search column name (must be the primary key of the table)
# $3 = search column value
# $4 = output column name
#----------------------------------------------------------------
get_attribute ()
{
  VALUE=`get_data "$4" $DBOWNER.$1 "$2 = '$3'"`
}




#
# get single select value (AND)
# $1 = table name
# $2 = search field1
# $3 = search field value1
# $4 = search field2
# $5 = search field value2
# $6 = output column name
# field1 and field2 must constitute the primary key of the table
#----------------------------------------------------------------
get_attribute_and ()
{
  VALUE=`get_data "$6" $DBOWNER.$1 "$2 = '$3' and $4 = '$5'"`
}
```

Figure 175 (Part 5 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
#
# CONFIGURATION METHODS
#



#
# add user at OS level (AIX)
# $1 = IP Hostname
# $2 = Type: either "server" or "target"
#            use "target", when you want to add a user to AIX
#            add a target workstation; the user will always be
#            assigned group FNDADMN
#            use "server", when you want to add a user to AIX
#            add a server workstation; the user will be assigned
#            the appropriate usergroup defined in the database
#

add_users_aix ()
{
print "NVDM CONFIG : --> Adding AIX users for NVDM..."
get_attribute_list nvdm_users node_name $1 username
if [ $VALUE_NUM != 0 ]
then
  for i in $VALUE_LIST
  do
    #
    # First, add NVDM user to operating system...
    # check if user exists
    #
    lsuser $i 2>/dev/null 1>&2
    #
    # if not (RC 2 from lsuser command)
    #
    if [ $? = 2 ]
    then
      print "NVDM CONFIG : Adding user $i to AIX OS."
      mkuser $i
    fi
    #
    # only continue, if we are about to configure a server
    #
    if [ "$2" = "server" ]
    then
      get_attribute_and nvdm_users node_name $1 username $i usergroup
      GRP=$VALUE
      print "NVDM CONFIG : Authorization profile $GRP assigned to $i."
      nvdm lsusr $i 2>/dev/null
      #
      # if RC != 0 then user does not exist yet
```

Figure 175 (Part 6 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
      #
      if [ $? -ne 0 ]
      then
        nvdm addusr $i $GRP -t $1
      else
        nvdm updusr $i $GRP -t $1
      fi
    fi
  done
fi
}

#
# delete all users currently defined on that server
# the root user profile cannot be deleted
#
nvdm_delete_users ()
{
 #
 # determine all users that are defined on this server
 #
 USRLIST=`nvdm lsusr '*' | grep "User:" | cut -d':' -f2`
 for i in $USRLIST
 do
   if [ "$i" != "root" ]
   then
     print "NVDM CONFIG : Deleting existing user profile : $i"
     nvdm delusr $i -f
   fi
 done
}


#
# Set Attributes in nvdm.cfg file
# $1 parameter name (e.g. WORKSTATION NAME, SERVER)
# $2 parameter value
#

configure_nvdm_cfg ()
{
 mv $CONFIG /tmp/config
 print "NVDM CONFIG : Setting nvdm.cfg ($1) to $2"
 #
 # the TCP/IP port parameter is special
 # because it contains a / in its name
 # and also needs modification of
 # /etc/services
 #
 if [ "$1" = "TCP/IP PORT" ]
```

Figure 175 (Part 7 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 then
   sed "s/TCP\/IP PORT:.*/TCP\/IP PORT:          $2/" \
/tmp/config >$CONFIG
   mv /etc/services /tmp/services
   sed "s/NetViewDM6000.*\/tcp/NetViewDM6000   $2\/tcp/" \
/tmp/services >/etc/services
   return
 fi
 #
 # adjust to right column
 #
 len=`echo $1 | wc -c`
 SUBST=$2
 while [ $len -lt 22 ]
 do
  SUBST=" "$SUBST
  len=`expr $len + 1`
 done
 #
 # replace parameter
 #
 sed "s/$1:.*/$1:$SUBST/" /tmp/config >$CONFIG
}


#
# get all static SNA attributes (SNA Net Name, etc.)
# $1 = IP Hostname of node to be configured
#

get_sna_attributes ()
{

  #
  # get static SNA parameters
  #

  for i in SNA_NET_NAME DATALINK_DEVICE REM_LINK_ADDR MODE_PROF_NAME\
 MODE_NAME TPN_PROF_NAME_SND TPN_PROF_NAME_RCV PARTNER_LU_NAME\
 SIDE_INFO_PROF_SND SIDE_INFO_PROF_RCV SOLICIT_SSCP I_FIELD_SIZE\
 LOCAL_SAP REMOTE_SAP INITIATE_CALL ACTIVATE_START RESTART_NORMAL\
 RESTART_ABNORMAL VTAM_CP_NAME
  do
    get_attribute nvdm_cfg_static NAME $i VALUE
    case $i in
      SNA_NET_NAME)       text="SNA Network Name"
                          SNA_NET=$VALUE ;;
      DATALINK_DEVICE)    text="SNA Datalink Device"
                          DEVICE=$VALUE ;;
      REM_LINK_ADDR)      text="SNA Remote Link Address"
```

Figure 175 (Part 8 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
                            ADDR=$VALUE ;;
      MODE_PROF_NAME)       text="SNA NVDM Mode Profile Name"
                            MPROF=$VALUE ;;
      MODE_NAME)            text="SNA NVDM Mode Name"
                            MODE=$VALUE ;;
      TPN_PROF_NAME_SND)    text="SNA TPN Profile Name (Send)"
                            SND=$VALUE ;;
      TPN_PROF_NAME_RCV)    text="SNA TPN Profile Name (Receive)"
                            RCV=$VALUE ;;
      PARTNER_LU_NAME)      text="SNA Partner LU Name (MVS Host)"
                            PARTNER=$VALUE ;;
      SIDE_INFO_PROF_SND)   text="SNA Side Info Profile Name (Send)"
                            SIDS=$VALUE ;;
      SIDE_INFO_PROF_RCV)   text="SNA Side Info Profile Name (Receive)"
                            SIDR=$VALUE ;;
      SOLICIT_SSCP)         text="Solicit SSCP Field (yes|no)"
                            SOLICIT=$VALUE ;;
      I_FIELD_SIZE)         text="I-Field Size"
                            IFIELD=$VALUE ;;
      LOCAL_SAP)            text="SNA Local SAP No."
                            LSAP=$VALUE ;;
      REMOTE_SAP)           text="Remote SAP No."
                            RSAP=$VALUE ;;
      INITIATE_CALL)        text="SNA Initiate Call Field (yes|no)"
                            ICALL=$VALUE ;;
      ACTIVATE_START)       text="SNA Activate on start (yes|no)"
                            ACTSTART=$VALUE ;;
      RESTART_NORMAL) text="SNA Restart on normal termination (yes|no)"
                            RNORM=$VALUE ;;
      RESTART_ABNORMAL) text="SNA Restart on abnormal termination (yes|no)"
                            RABNORM=$VALUE ;;
      VTAM_CP_NAME) text="SNA VTAM CP Name (for LU6.2 Location Profile)"
                            VTAMCP=$VALUE ;;

    esac
    if [ "$VALUE" = "" ]
    then
      abort "Could not determine $text. Exiting..."
    else
      print "NVDM CONFIG : Setting $text to $VALUE"
    fi
  done

  get_attribute nvdm_servers node_name $1 pu_name
  PUNAME=$VALUE
  if [ "$PUNAME" = "" ]
  then
    abort "Could not determine PU NAME for $1 configuration
. Exiting..."
  fi
```

Figure 175 (Part 9 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
  print "NVDM CONFIG : Setting PU NAME for $1 to $PUNAME "

  get_attribute nvdm_servers node_name $1 local_lu_name
  LLUNAME=$VALUE
  if [ "$LLUNAME" = "" ]
  then
    abort "Could not determine Local LU Name for $1 configu
ration. Exiting..."
  fi

  print "NVDM CONFIG : Setting Local LU Name for $1 to $LLUNAME "

  get_attribute nvdm_servers node_name $1 cp_name
  CP_NAME=$VALUE
  if [ "$CP_NAME" = "" ]
  then
    abort "Could not determine Control Point Name for $1.\
 Exiting..."
  fi
  CP_TYPE=appn_end_node

  print "NVDM CONFIG : Setting Control Point Name for $1\
to $CP_NAME"

  get_attribute nvdm_servers node_name $1 xid
  XID=$VALUE
  if [ "$XID" = "" ]
  then
    print "NVDM CONFIG : Could not determine XID for $1 configu
ration."
    print "NVDM CONFIG : Setting USE_CP_XID to yes"
    USE_CP_XID="yes"
    # set XID to dummy value
    XID=07100000
  else
    print "NVDM CONFIG : Setting XID for $1 to $XID "
    print "NVDM CONFIG : Setting USE_CP_XID to no"
    USE_CP_XID="no"
  fi
}


#
# export existing SNA profiles
# in case they need to be restored if
# NVDM configuration fails
#
# $1 = name of export file
#
```

Figure 175 (Part 10 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
export_sna ()
{
 print "NVDM CONFIG : Exporting existing SNA profiles to $1 ..."
 exportsna -A -f $1 -r -UT -C
}

#
# configure SNA dlc
# for all SNA communications a DLC for the
# communications adapter is needed.
# if the DLC already exists, the mkdev command
# will print an error message - this will be
# redirected to /dev/null
#

configure_sna_dlc ()
{
 print "NVDM CONFIG : Adding DLC Device for $DEVICE"
 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) mkdev -c dlc -s dlc -t tokenring 1>/dev/null 2>&1 ;;
  "ent" ) mkdev -c dlc -s dlc -t ethernet 1>/dev/null 2>&1 ;;
  "x25" ) mkdev -c dlc -s dlc -t x25_qllc 1>/dev/null 2>&1 ;;
  "*"   ) print "NVDM CONFIG : Device type $CHECK unknown." ;;
 esac
}

#
# SNA initial node setup
#

sna_initial ()
{
 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="token_ring" ;;
  "ent" ) DEV_TYPE="ethernet" ;;
  "fdd" ) DEV_TYPE="fddi" ;;
  "x25" ) DEV_TYPE="x.25_call_SVC" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
    abort "No device type found for $DEVICE."
 fi

 print "NVDM CONFIG : Configuring SNA Initial Node Setup"
set -x
 mk_qcinit -y $DEV_TYPE -t $CP_TYPE -w $SNA_NET -d $CP_NAME
```
Figure 175 (Part 11 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
set +x

}

#
# configure SNA Control Point Profile
#
# SNA_NET contains SNA Network Name
# CP_NAME contains SNA Control Point Name
# CP_TYPE contains SNA Control Point Type
#

configure_sna_cp ()
{
 print "NVDM CONFIG : Configuring SNA Control Point Profile"
 line
set -x
 chsnaobj -t 'control_pt' -e "$SNA_NET" -a "$CP_NAME" -A "$CP_NAME"\
 -N "$CP_TYPE" node_cp
set +x
 line
}

#
# configure SNA dlc profile
#

configure_sna_dlc_profile ()
{
 # determine type of DLC from datalink device name
 # get only first 3 characters from device name
 # e.g. if datalink device is x25s1, then x25 determines
 # the type to be X.25

 CHECK=`echo $DEVICE | cut -c1-3`
 case "$CHECK" in
  "tok" ) DEV_TYPE="sna_dlc_token_ring" ;;
  "ent" ) DEV_TYPE="sna_dlc_ethernet" ;;
  "fdd" ) DEV_TYPE="sna_dlc_fddi" ;;
  "x25" ) DEV_TYPE="sna_dlc_x.25" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
    abort "No device type found for $DEVICE."
 fi

 #
 # create new DLC Profile
```

Figure 175 (Part 12 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 # use Datalink Device Name as Profile Name
 #

 print "NVDM CONFIG : Configuring SNA DLC Profile"
 line
set -x
# change !!!
 if [ "$DEV_TYPE" = "sna_dlc_x.25" ]
 then
   mksnaobj -t "$DEV_TYPE" "$DEVICE"
   RC=$?
 else
   mksnaobj -t "$DEV_TYPE" -d "$DEVICE" -b $SOLICIT -w yes -m $IFIELD \
   -H $LSAP -c no -q 0 "$DEVICE"
   RC=$?
 fi
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
# change !!!
   if [ "$DEV_TYPE" = "sna_dlc_x.25" ]
   then
     chsnaobj -t "$DEV_TYPE" "$DEVICE"
   else
     chsnaobj -t "$DEV_TYPE" -d "$DEVICE" -b $SOLICIT -w yes -m $IFIELD \
 -H $LSAP -c no -q 0 "$DEVICE"
   fi
set +x
   line
 fi

}

#
# configure SNA Link Station Profile
#

configure_sna_link ()
{
 # determine type of DLC from datalink device name
 # get only first 3 characters from device name

 CHECK=`echo $DEVICE | cut -c1-3`
```

Figure 175 (Part 13 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 case "$CHECK" in
  "tok" ) DEV_TYPE="token_ring" ;;
  "ent" ) DEV_TYPE="ethernet" ;;
  "fdd" ) DEV_TYPE="fddi" ;;
  "x25" ) DEV_TYPE="x.25" ;;
  "*"   ) DEV_TYPE="none"
 esac

 if [ "$DEV_TYPE" = "none" ]
 then
   abort "No device type found for $DEVICE. Exiting"
 fi

 #
 # create new Link Station Profile
 # use Datalink Device Name as DLC Profile Name
 #

 print "NVDM CONFIG : Configuring SNA Link Station Profile"
 line
set -x
# change !!!
 if [ "$DEV_TYPE" = "x.25" ]
 then
   mksnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -q "$X25_TYPE"\
   -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
   -s "$ADDR" "$PUNAME"
   RC=$?
 else
   mksnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -d "$ADDR" -l $XID\
   -s $RSAP -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
   -c "$USE_CP_XID" "$PUNAME"
   RC=$?
 fi
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   if [ "$DEV_TYPE" = "x.25" ]
   then
     chsnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -q "$X25_TYPE" \
     -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
     -s "$ADDR" "$PUNAME"
   else
```

Figure 175 (Part 14 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
     chsnaobj -t link_station -w "$DEV_TYPE" -y "$DEVICE" -d "$ADDR" -l $XID\
     -s $RSAP -a $SOLICIT -O $ICALL -F $ACTSTART -h $RNORM -z $RABNORM \
     -c "$USE_CP_XID" "$PUNAME"
   fi
set +x
   line
 fi
}


#
# configure local LU profile for node
#

configure_sna_local_lu ()
{
 print "NVDM CONFIG : Configuring SNA Local LU Profile"

 #
 # create new Local LU Profile
 # use Local LU Name as Profile Name
 #

 line
set -x
 mksnaobj -t local_lu -u lu6.2 -l "$LLUNAME" -L "$LLUNAME" "$LLUNAME"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_lu -u lu6.2 -l "$LLUNAME" -L "$LLUNAME" "$LLUNAME"
set +x
   line
 fi

}


#
# configure SNA Mode Profile
#

configure_sna_mode ()
```
Figure 175 (Part 15 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
{
 #
 # create new Mode Profile
 #

 print "NVDM CONFIG : Configuring SNA Mode Profile"
 line
set -x
 mksnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N "#CONNECT" -m "$MODE" "$MPROF"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t mode -x 1 -w 0 -l 0 -a 0 -N "#CONNECT" -m "$MODE" "$MPROF"
set +x
   line
 fi

}

#
# configure TPN send profile
#

configure_sna_send ()
{
 #
 # create TPN Profile (Send)
 #

 print "NVDM CONFIG : Configuring SNA TPN Profile (SEND)"
 line
set -x
 mksnaobj -t local_tp -n 21F0F0F7 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndts -s none "$SND"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."
```

Figure 175 (Part 16 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
   line
set -x
   chsnaobj -t local_tp -n 21F0F0F7 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndts -s none "$SND"
set +x
   line
 fi

}

#
# configure TPN receive profile
#

configure_sna_receive ()
{
 #
 # create TPN Profile (Receive)
 #

 print "NVDM CONFIG : Configuring SNA TPN Profile (Receive)"
 line
set -x
 mksnaobj -t local_tp -n 21F0F0F8 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndtr -s none "$RCV"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t local_tp -n 21F0F0F8 -h yes -c basic \
 -d 0 -P yes -w /usr/lpp/netviewdm/bin/fndtr -s none "$RCV"
set +x
   line
 fi

}


#
# Configure partner LU profile (Focal Point)
#
```

Figure 175 (Part 17 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
configure_sna_partner ()
{
 #
 # create LU 6.2 Partner Profile
 #

 print "NVDM CONFIG : Configuring SNA LU6.2 Partner LU"
 line
set -x
 mksnaobj -t partner_lu6.2 -p no -P "$SNA_NET"."$PARTNER" \
 -O none -A "$PARTNER" "$PARTNER"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
 chsnaobj -t partner_lu6.2 -p no -P "$SNA_NET"."$PARTNER" \
 -O none -A "$PARTNER" "$PARTNER"
set +x
   line
 fi

}

#
# configure LU6.2 location profile
#

configure_sna_location ()
{
 print "NVDM CONFIG : Configuring SNA LU 6.2 Location Profile"

 #
 # create new LU 6.2 Location Profile
 # use Local LU Name as Profile Name
 #

 line
set -x
 mksnaobj -t partner_lu6.2_location -P "$SNA_NET.$PARTNER" \
 -O "$SNA_NET.$VTAMCP" -m link_station -l $LLUNAME \
 -s $PUNAME $PARTNER
 RC=$?
set +x
```

Figure 175 (Part 18 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
 chsnaobj -t partner_lu6.2_location -P "$SNA_NET.$PARTNER" \
 -O "$SNA_NET.$VTAMCP" -m link_station -l $LLUNAME \
 -s $PUNAME $PARTNER
set +x
   line
 fi

}


#
# configure Side Info Profile (Send)
#

configure_side_snd ()
{
 #
 # create Side Info Profile (Send)
 #

 print "NVDM CONFIG : Configuring SNA Side Info Profile (Send)"
 line
set -x
 mksnaobj -t side_info -L "$CP_NAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F7 -h yes "$SIDS"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t side_info -L "$CP_NAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F7 -h yes "$SIDS"
set +x
   line
 fi
```

Figure 175 (Part 19 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
}

#
# configure Side Info Profile (Receive)
#

configure_side_rcv ()
{
 #
 # create Side Info Profile (Receive)
 #

 print "NVDM CONFIG : Configuring SNA Side Info Profile (Receive)"
 line
set -x
 mksnaobj -t side_info -L "$LLUNAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F8 -h yes "$SIDR"
 RC=$?
set +x
 line

 if [ $RC = 255 ]
 then
   print "NVDM CONFIG RECOVER : Profile already existed.\
 Changing existing one ..."

   line
set -x
   chsnaobj -t side_info -L "$LLUNAME" -P "$SNA_NET"."$PARTNER" -m "$MODE"\
 -d 21F0F0F8 -h yes "$SIDR"
set +x
   line
 fi

}



#
# get queues defined for a server
# since this class can contain more
# than one entry for a server, we have
# to store the result in a list
#
# $1 = server name
#

get_queues ()
{
 #
```

Figure 175 (Part 20 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 # first, determine number of entries for
 # that server
 #

 #
 # Fill in Fields
 #

 get_attribute_list nvdm_queues node_name $1 protocol
 NUM_QUEUE=$VALUE_NUM
 if [ $NUM_QUEUE = 0 ]
 then
   return
 fi

 PROTOCOL=$VALUE_LIST
 get_attribute_list nvdm_queues node_name $1 remote_server
 REMOTE_SERVER=$VALUE_LIST

}

#
# configure SNA/DS connection profiles
#
# $1 = IP Hostname of system to be configured
#

configure_sna_ds_conn ()
{

#
# perform SNA/DS configuration (connection profiles)
#

#
# remove demo profile CONNSNA,CONNTCP if existent
#
cd $SNA_DS_DIR
rm * 2>/dev/null

get_queues $1

if [ $NUM_QUEUE != 0 ]
then
  a=1
  for i in $PROTOCOL
  do
    print "NVDM CONFIG : Configuring $i connection"
    if [ "$i" != "APPC" -a  "$i" != "TCP/IP" ]
    then
```

Figure 175 (Part 21 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
      abort "Protocol is neither APPC nor TCP/IP. Exiting..."
    fi

    # determine if connection is made through an intermediate node

    INODE=`echo $REMOTE_SERVER | cut -d' ' -f"$a"`
    get_attribute_and nvdm_queues node_name $1 remote_server $INODE inter_node
    if [ "$VALUE" != "" ]
    then
      print "NVDM CONFIG : Remote connection to $INODE is made"
      print "              through intermediate node $VALUE."
      print "              No SNA/DS connection file is created."
    else
      if [ "$i" = "APPC" ]
      then
        configure_sna_ds_appc
      else
        REMSERV=`echo $REMOTE_SERVER | cut -d' ' -f "$a"`
        configure_sna_ds_tcpip $REMSERV
      fi
    fi
    a=`expr $a + 1`
  done
fi
}

#
# Configure SNA/DS connection configuration file (APPC)
#

configure_sna_ds_appc ()
{
 print "NVDM CONFIG : Configuring SNA/DS connection\
 configuration file $SNA_DS_DIR/$PARTNER"

 if [ "$PARTNER" = "" ]
 then
   print "NVDM CONFIG ERROR : APPC Partner LU not defined."
   print "NVDM CONFIG ERROR : Cannot create SNA/DS connection profile."
   return 1
 fi

 echo "PROTOCOL:                      APPC
TYPE:                             SNA
SEND TP SYMBOLIC DESTINATION:     $SIDS
RECEIVE TP SYMBOLIC DESTINATION:  $SIDR
NEXT DSU:                         $SNA_NET.$PARTNER
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
```

Figure 175 (Part 22 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
RECEIVE MU_ID TIME-OUT:           120" > $SNA_DS_DIR/$PARTNER
}


#
# Configure SNA/DS connection configuration file (TCP/IP)
# $1 = TCP/IP Hostname of remote system
#

configure_sna_ds_tcpip ()
{
 #
 # get short name of remote server
 #

 get_attribute nvdm_node node_name $1 short_name
 A=$VALUE
 print "NVDM CONFIG : Configuring SNA/DS connection configuration file."
 print "NVDM CONFIG : (TCP/IP) for remote Server $1."

 if [ "$A" = "" ]
 then
   print "NVDM CONFIG ERROR :  Could not determine short name for $1."
   print "NVDM CONFIG ERROR :  Please update nvdm_node class."
   return
 fi

 echo "PROTOCOL:                          TCP/IP
TYPE:                             SNA
REMOTE SERVER NAME:               $1
TCP/IP TIME-OUT:                  300
NEXT DSU:                         $A.$A
TRANSMISSION TIME-OUT:            60
RETRY LIMIT:                      3
SEND MU_ID TIME-OUT:              60
RECEIVE MU_ID TIME-OUT:           120" >$SNA_DS_DIR/$A
}


#
# configure SNA/DS routing table
# $1 = IP Hostname
#

configure_routetab ()
{
 #
 # first, determine what network protocols we have
 #
 a=0
```

Figure 175 (Part 23 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
  b=0
  print "NVDM CONFIG : Configuring SNA/DS routing table."
  get_attribute_and nvdm_queues node_name $1 protocol TCP/IP remote_server
  if [ "$VALUE" != "" ]
  then
    print "NVDM CONFIG : System has TCP/IP connection to remote server."
    a=1
  fi

  get_attribute_and nvdm_queues node_name $1 protocol APPC remote_server
  if [ "$VALUE" != "" ]
  then
    print "NVDM CONFIG : System has APPC connection to remote server."
    b=1
  fi

  if [ $a -eq 0 -a $b -eq 0 ]
  then
    print "NVDM CONFIG : There are no connections defined."
    return
  fi

  if [ $a -eq 1 -a $b -eq 1 ]
  then
    RPROT="BOTH"
  fi

  if [ $a -eq 1 -a $b -eq 0 ]
  then
    RPROT="TCP/IP"
  fi

  if [ $a -eq 0 -a $b -eq 1 ]
  then
    RPROT="APPC"
  fi

  print "NVDM CONFIG : Writing routing table to $SNA_DS_ROUTE"
  echo "NETWORK PROTOCOL: $RPROT

#
# SNA connections
#
" >$SNA_DS_ROUTE

#
# get all SNA Routes
#

  get_attribute_and nvdm_queues node_name $1 protocol APPC remote_server
```

Figure 175 (Part 24 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 SNA_R=$VALUE
 if [ "$SNA_R" != "" ]
 then
   for i in $SNA_R
   do
     # check if intermediate node is used
     get_attribute_and nvdm_queues node_name $1 remote_server $i inter_node
     if [ "$VALUE" != "" ]
     then
       echo "$SNA_NET.$i   ANY     ANY     ANY     ANY     $VALUE     5" >>$SNA_DS_ROUTE
     else
       echo "$SNA_NET.$i   ANY     ANY     ANY     ANY     $i     5" >>$SNA_DS_ROUTE
     fi
   done
 fi

 echo "
#
# TCP/IP connections
#
" >>$SNA_DS_ROUTE
 get_attribute_and nvdm_queues node_name $1 protocol TCP/IP remote_server
 TCP_R=$VALUE
 if [ "$TCP_R" != "" ]
 then
   for i in $TCP_R
   do
     # in the routing table we need the short name, not the
     # TCP/IP hostname as specified in remote_server ; therefore
     # we have to get the shortname first
     # check if intermediate node is used
     get_attribute nvdm_node node_name $i short_name
     sn=$VALUE
     get_attribute_and nvdm_queues node_name $1 remote_server $i inter_node
     if [ "$VALUE" != "" ]
     then
       echo "$sn.*                                        $VALUE" >>$SNA_DS_ROUTE
     else
       echo "$sn.*                                        $sn" >>$SNA_DS_ROUTE
     fi
   done
 fi
}


#
# delete local targets from NVDM Server configuration
# $1 = Server IP Hostname
#
```

Figure 175 (Part 25 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
nvdm_delete_targets()
{
 #
 # get list of existing targets
 #

 TLIST=`nvdm lstg '*' | grep "Target:" | cut -d':' -f2`

 #
 # get list of all defined targets for this server
 #

 get_attribute_list nvdm_node server_name $1 node_name
 YLIST=$VALUE_LIST
 XLIST=""
 for i in $YLIST
 do
  XLIST=$XLIST" "`echo $i | cut -d'.' -f1`
 done

 #
 # delete all targets which are not defined for this server
 #

 for i in $TLIST
 do
  match=0
  for x in $XLIST
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    nvdm_save_history $i
    print "NVDM CONFIG : Deleting Target $i from Server $1 configuration."
    #
    # before a target can be deleted, we have to
    # discard all pending requests
    #
    PEND=`nvdm lsrq -w $i | grep "Request ID:" | cut -d':' -f2 | \
          awk '{ print $3 }'`
    if [ "$PEND" != "" ]
    then
      print "NVDM CONFIG : Requests IDs $PEND for $i will be deleted."
      print "NVDM CONFIG : Information about pending requests for"
      print "NVDM CONFIG : Target $i will be written to $i.req"
      echo "The following requests were purged:" >$i.req
```

Figure 175 (Part 26 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
      for x in $PEND
      do
        nvdm lsrq -l $x >>$i.req
      done
    fi
    nvdm hldq $i
    nvdm prgq $i -f
    for x in $PEND
    do
      echo "y" >/tmp/yes
      nvdm delrq $x -f
      nvdm eraserq $x </tmp/yes
      sleep 2
      nvdm delrq $x -f
      nvdm eraserq $x </tmp/yes
    done
    nvdm deltg $i -f
  fi
 done
}

#
# Save NVDM target history by creating software inventory
# file and copying it to corresponding node
# requires /.rhosts file on target
# $1 = target name
#
nvdm_save_history ()
{
 print "NVDM CONFIG : Saving target history for $1"
 #nvdm inv
 SLIST="`nvdm lscm -w $1 '*' | grep 'Global file name:' | cut -d':' -f2`"
 >/tmp/inv
 if [ "$SLIST" != "" ]
 then
   for o in $SLIST
   do
     print "NVDM CONFIG : Adding $o to software inventory file."
     print "GLOBAL NAME: "$o >>/tmp/inv
     print "DESCRIPTION: Target has been moved!" >>/tmp/inv
   done
   print "NVDM CONFIG : Copying inventory file $SW_INV to $1."
   echo "GLOBAL NAME:                  HISTORY.REF.1
CHANGE FILE TYPE:              GEN
COMPRESSION TYPE:              LZW
REBOOT REQUIRED:               NO
PACK FILES:                    NO
SECURE PACKAGE:                NO
OBJECT:
SOURCE NAME:               /tmp/inv
```

Figure 175 (Part 27 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
TARGET NAME:              /usr/lpp/netviewdm/fndswinv
TYPE:                     FILE
ACTION:                   COPY
INCLUDE SUBDIRS:          NO" >/tmp/hist.pro
   nvdm delcm HISTORY.REF.1 -w '*'
   nvdm uncat HISTORY.REF.1 -d -f
   nvdm bld /tmp/hist.pro -f
   nvdm inst HISTORY.REF.1 -w $1 -f -i
#
# we will sleep here for 15 secs to allow
# the CF to be sent to the target before
# it is deleted. You might need to adjust
# this value, especially if you are, for example,
# in a WAN environment
#
 fi
 print "NVDM CONFIG : Sleeping for 15 secs."
 sleep 15
}

#
# configure Targets for an NVDM/6000 Server
# $1 = Server IP Hostname
#

nvdm_configure_targets ()
{
 #
 # First, determine all Nodes which have this Server
 # defined as their NVDM/6000 server
 #

 # access database

 get_attribute_list nvdm_node server_name $1 node_name
 ATLIST=$VALUE_LIST
 TLIST=""
 for i in $ATLIST
 do
  TLIST=$TLIST" "`echo $i | cut -d'.' -f1`
 done

 count=0
 for i in $TLIST
 do
   count=`expr $count + 1`
   print "NVDM CONFIG : Defining Target $i on server $1"
   A=`nvdm lstg $i 2>&1 | grep FNDCL129E`
   #
   # if FNDCL129E not found then target exists already
```

Figure 175 (Part 28 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
   #

   if [ "$A" != "" ]
   then
     COMMAND="nvdm addtg $i"
   else
     COMMAND="nvdm updtg $i"
     print "NVDM CONFIG : Target already exists. Updating..."
   fi

   #
   # get required target attributes
   #

   huhn=`echo $ATLIST | cut -d' ' -f$count`

   for a in short_name target_os description contact_name\
  owning_manager telephone_number customer_name
   do
     get_attribute nvdm_node node_name $huhn $a
     v=$VALUE
     if [ "$v" != "" ]
     then
       case $a in
         short_name)     COMMAND=$COMMAND" -s '$v'" ;;
         target_os)      COMMAND=$COMMAND" -y '$v'" ;;
         description)    COMMAND=$COMMAND" -d '$v'" ;;
         contact_name)   COMMAND=$COMMAND" -q '$v'" ;;
         owning_manager) COMMAND=$COMMAND" -o '$v'" ;;
         telephone_number) COMMAND=$COMMAND" -t '$v'" ;;
         customer_name)    COMMAND=$COMMAND" -r '$v'" ;;
       esac
     fi
   done

   if [ "$i" != "$1" ]
   then
     COMMAND=$COMMAND" -b client"
   fi
   echo $COMMAND
   eval $COMMAND
  done
}

#
# Delete all existing groups before adding groups from
# configuration database
# $1 = IP Hostname of server to be configured
#
```

Figure 175 (Part 29 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
nvdm_delete_groups ()
{
 #
 # determine existing groups
 #
 GP=`nvdm lsgp '*' | grep "Group:" | cut -c24-`
 #
 # determine list of defined groups
 #
 get_attribute_list nvdm_groups node_name $1 group_name
 XGP=$VALUE_LIST

 for i in $GP
 do
  match=0
  for x in $XGP
  do
    if [ "$i" = "$x" ]
    then
      match=1
    fi
  done
  if [ match -eq 0 ]
  then
    print "NVDM CONFIG : Deleting group $i from $1 configuration."
    nvdm delgp $i -f
  fi
 done
}

#
# configure groups defined for NVDM/6000 server
#

nvdm_configure_groups ()
{
 print "NVDM CONFIG : Configuring Target Groups for $1"
 get_attribute_list nvdm_groups node_name $1 group_name
 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No groups defined"
   return
 fi
 GROUP_LIST=$VALUE_LIST
 for i in $GROUP_LIST
 do
   print "NVDM CONFIG : Adding group $i"
   get_attribute nvdm_groups group_name $i short_name
   SHORT=$VALUE
   get_attribute nvdm_groups group_name $i description
```

Figure 175 (Part 30 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
   DESC=$VALUE
   #
   # get all targets being defined for this group
   #

   get_attribute_list nvdm_node group_name $i node_name

   for a in $VALUE_LIST
   do
     TNGP=`echo $a | cut -d'.' -f1`
     eval nvdm addgp $i $TNGP -s "'$SHORT'" -d "'$DESC'"
   done
 done
}

#
# configure Remote Targets
# $1 = IP Hostname
#

nvdm_remote_targets ()
{
 #
 # First, get all remote targets defined for this server
 # Remote Targets are determined by searching the nvdm_queues
 # class because any connection to a remote system requires a
 # queue

 get_attribute_list nvdm_queues node_name $1 remote_server

 if [ $VALUE_NUM = 0 ]
 then
   print "NVDM CONFIG : No remote targets defined"
   return
 fi

 for i in $VALUE_LIST
 do
   print "NVDM CONFIG : Defining remote target for $i"

   #
   # determine if system to be configured is a Remote Target or
   # a Focal Point
   #
   get_attribute_and nvdm_queues node_name $1 remote_server $i focal_point

   if [ "$VALUE" = "yes" ]
   then
     print "NVDM CONFIG : $i will be configured as focal point."
     # for the MVS focal point short name will be the same as node name
```

Figure 175 (Part 31 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
     # network id will be the SNA Network Name

set -x
     eval nvdm addtg $i -m focal -b server -s $i -n $SNA_NET \
 -d "'NVDM_MVS'" -tp appc:
set +x
   else
     # get short name for remote server from class nvdm_node
     get_attribute nvdm_node node_name $i short_name
     if [ "$VALUE" = "" ]
     then
       abort "No Short Name defined for $i in class nvdm_node. Exiting..."
     fi
     RSHORT=$VALUE
     #
     # This remote server is assumed to be connected via TCP/IP
     # so, we set the network name to be the same as the short name
     #
     nvdm addtg $i -s $RSHORT -n $RSHORT -b server
   fi
 done
}

restart_nvdm ()
{
 print "NVDM CONFIG : --> In order for the changes to become active"
 print "NVDM CONFIG :     NetView DM/6000 will be restarted on this node"

 #
 # determine if nvdm is running
 #

 nvdm stat 1>/dev/null 2>&1

 if [ $? = 218 ]
 then
   print "NVDM CONFIG : NVDM is not running. It will be started now."
   nvdm start
   nvdm start
 else
   print "NVDM CONFIG : Stopping NVDM."
   nvdm stop -x 1>/dev/null 2>&1
   s=1
   print "NVDM CONFIG : Restarting NVDM."
   while [ $s = 1 ]
   do
     print "NVDM CONFIG : Restarting NVDM."
     nvdm start
     nvdm stat
     if [ $? != 218 ]
```

Figure 175 (Part 32 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
     then
       s=0
     fi
   done
 fi
}

#
# update NVDM/6000 server definition
#

nvdm_update_server ()
{
#-------------change--------------
# if we configure a server and want to change the
# WORKSTATION NAME we must stop the server before
# reconfiguring this field.
# To do so we must be sure that the hostname and
# the WORKSTATION NAME match
#
if [ "$NODE_TYPE" = "0" -o "$NODE_TYPE" = "2" ]
then
  # get current hostname
  OHN=`hostname`
  print "NVDM CONFIG : Current hostname of server is $OHN."
  # get WORKSTATION NAME currently configured
  HN=`grep "WORKSTATION NAME:" $CONFIG | cut -d':' -f2`
  print "NVDM CONFIG : Current WORKSTATION NAME of server is $HN."
  print "NVDM CONFIG : Stopping Server..."
  # make sure that both match
  hostname $HN
  nvdm stop -x
  print "NVDM CONFIG : Sleeping 20 seconds..."
  sleep 20
  # set back hostname
  print "NVDM CONFIG : Setting hostname to $OHN."
  hostname $OHN
  # also, we must be sure that there is an initial target record for
  # the servername configured
  # we rename the initial target record to the new hostname
  nvdm rentg $HN $OHN -f
fi
#-----------end-of-change---------
}
#
# check if TCP/IP ports for NetView DM/6000 are
# existing. If not, add them to /etc/services file
#
check_ports ()
{
```

Figure 175 (Part 33 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
 #
 # first, make a backup copy of /etc/services
 #
 cp /etc/services /etc/services.nvdm
 #
 # check for port NetViewDM-rcv
 #
 print "CONFIG NVDM : Checking NetViewDM-rcv port..."
 R=`grep NetViewDM-rcv /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM-rcv   731/tcp" >>/etc/services
 fi
 #
 # check for port NetViewDM-snd
 #
 print "CONFIG NVDM : Checking NetViewDM-snd port..."
 R=`grep NetViewDM-snd /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM-snd   730/tcp" >>/etc/services
 fi
 #
 # check for port NetViewDM6000
 #
 print "CONFIG NVDM : Checking NetViewDM6000 port..."
 R=`grep NetViewDM6000 /etc/services`
 if [ "$R" = "" ]
 then
   print "CONFIG NVDM : Port did not exist. Adding it to /etc/services..."
   echo "NetViewDM6000   729/tcp" >>/etc/services
 fi
}

#
#
# ***** MAIN *****
#
#

debug_info
print "NVDM CONFIG : --> Trying to configure node $1"

#
# determine node type
#

get_attribute nvdm_node node_name $1 node_type
```

Figure 175 (Part 34 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
if [ "$VALUE" = "" ]
then
  abort "No Database match found for $1."
fi
NODE_TYPE=$VALUE
print "NVDM CONFIG : Node type is $NODE_TYPE (0 = Server, 1 = Agent, 2 = Prep)"

#
#                                                             #
# steps neccessary for all nodes (server, agent, prep site) #
#                                                             #
#

print "NVDM CONFIG : --> NVDM Base Node Configuration"

#
# add file system for repository
#

#add_fs_repos $1

#
# update hostname/NVDM server name
#
nvdm_update_server

#
# configure WORKSTATION NAME
#

configure_nvdm_cfg "WORKSTATION NAME" $HNAME

#
# configure SERVER
#

get_attribute nvdm_node node_name $1 server_name
SERVER=$VALUE
HSERVER=`echo $SERVER | cut -d'.' -f1`
configure_nvdm_cfg "SERVER" $HSERVER

#
# configure NVDM LOG SIZE
#
get_attribute nvdm_cfg_static NAME NVDM_LOG_SIZE VALUE
if [ "$VALUE" != "" ]
then
 configure_nvdm_cfg "LOG FILE SIZE" $VALUE
fi
```

Figure 175 (Part 35 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
#
# check for NetView DM ports

check_ports

#
# configure TCP/IP port to be used by NetView DM
#
get_attribute nvdm_cfg_static NAME TCPIP_PORT VALUE
if [ "$VALUE" != "" ]
then
  configure_nvdm_cfg "TCP/IP PORT" $VALUE
fi

#-------------change--------------
get_attribute nvdm_cfg_static NAME REPOS_DIR VALUE
if [ "$VALUE" != "" ]
then
  VALUE=`echo "$VALUE"`
  configure_nvdm_cfg "REPOSITORY" $VALUE
fi
#-----------end-of-change---------

#
# add users at target
#

if [ "$NODE_TYPE" = "2" ]
then
    add_users_aix $1 target
fi

#-------change-----------

# reset /usr/lpp/netviewdm/uicfg/root.cli file
print "NVDM CONFIG : Resetting root.cli ... ($HSERVER)"
./uicfg $HSERVER

#-------end of change----

if [ "$NODE_TYPE" = "1" ]
then
  print "NVDM CONFIG : Starting NVDM Agent (fndcmps)...."
  /usr/lpp/netviewdm/bin/fndcmps &
  line
  banner SUCCESS!
  line
  print "NVDM CONFIG : !!! Configuration of Agent completed successfully !!!"
  line
```

Figure 175 (Part 36 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
  exit 0
fi

#
#
# Server configuration
#
#

#-------------change-------------
# restart server, in case it is not already running
print "NVDM CONFIG : Restarting Server..."
restart_nvdm
#-----------end-of-change---------

#
#
# SNA Configuration for Servers
#
#

#
# determine if node needs SNA configuration
#

#
# get all static SNA attributes
#

get_sna_attributes $1

get_attribute nvdm_servers node_name $1 sna


if [ "$VALUE" = "yes" ]
then

  print "NVDM CONFIG : --> Configuring SNA "

  #
  # Configure SNA
  #

  export_sna $EXPORT_SNA
  configure_sna_dlc
  sna_initial
  configure_sna_cp
  configure_sna_dlc_profile
  configure_sna_link
  configure_sna_local_lu
```

Figure 175 (Part 37 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
  configure_sna_mode
  configure_sna_send
  configure_sna_receive
  configure_sna_partner
  configure_sna_location
  configure_side_snd
  configure_side_rcv

  #
  # After SNA has been configured, update configuration database
  # to make changes become active
  #

  print "NVDM CONFIG : Updating SNA Server..."
  verifysna -R

fi # end of SNA configuration

#
# perform SNA/DS configuration (connection profiles)
#

configure_sna_ds_conn $1

#
# configure SNA/DS Routing Table
#

configure_routetab $1

#
# Delete all existing targets in case of node reconfiguration
#

nvdm_delete_targets $1

#
# Configure all local targets for NVDM/6000 server
#

nvdm_configure_targets $1

#
# add all target users also to server
#

get_attribute_list nvdm_node server_name $1 node_name
if [ "$VALUE_LIST" != "" ]
then
  for i in $VALUE_LIST
```

Figure 175 (Part 38 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
  do
      add_users_aix $i server
  done
fi

#
# Delete existing groups
#

nvdm_delete_groups $1

#
# Configure all groups
#

nvdm_configure_groups $1

#
# Configure all remote targets/focal point for NVDM/6000 Server
#

nvdm_remote_targets $1

#
# Reload nvdm configuration
# This will only refresh SNA/DS configuration 'in flight'

nvdm rld

#
# Start SNA Server subsystem
#

startsrc -s sna

#
# determine if NVDM has to be restarted
#

get_attribute nvdm_cfg_static NAME RESTART_NVDM VALUE
if [ "$VALUE" = "yes" ]
then
  restart_nvdm
fi


sleep 10

#
# release all SNA communications
```

Figure 175 (Part 39 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

```
#

print "NVDM CONFIG : Releasing NVDM SNA communications."
nvdm relc

line
banner SUCCESS!
line
print "NVDM CONFIG : !!! Configuration of Server completed successfully !!!"
line
```

Figure 175 (Part 40 of 40). config_nvdm Shell Script Software Distribution for AIX V3.1 with DB2/6000 Database

Appendix B. Script Reference Information

This appendix contains some reference information about the configuration scripts.

B.1 Shell Variables

The following table contains global shell variables used in the sample configuration script `config_nvdm`.

The table is included to help you when modifying the configuration script.

Table 3 (Page 1 of 3). Global Shell Variables

Variable Name	**Purpose**	**Example Value**
ACTSTART	Activate on start	yes, no
ADDR	Remote Link Address for SNA connections	400010002000, 49221123456
CHECK	Used to determine datalink device type for SNA	Token-ring, X.25, Ethernet
COMMAND	Used to contruct NVDM command strings	nvdm lstg '*'
CONFIG	Name of NetView DM base configuration file	/usr/lpp/netviewdm/db/nvdm.cfg
CP_NAME	Local CP Name of server	
CP_TYPE	Control Point type	appn_end_node, appn_network_node
DBOWNER	DB2/6000 instance owner	dbmsadm
DESC	Description for a target	
DEVICE	Datalink device for SNA connections	tok0, x25s0
EXPORT_SNA	File name for exporting SNA configuration	/tmp/sna
GP	List containing existing groups for a server	
GRP	NVDM group defined fo AIX user	FNDUSER, FNDBLD, FNDADMN
HN	Workstation name of node to be configured	
HNAME	TCP/IP hostname of node to be configured	rs600012
HSERVER	Name of NVDM server (without domain information)	
IFIELD	I-Field Size	2048
INODE	Used to check if connection is made through an intermediate node	

Table 3 (Page 2 of 3). Global Shell Variables		
Variable Name	**Purpose**	**Example Value**
LLUNAME	Local LU Name of server	
LSAP	Local SAP	04
MODE	Name of NVDM mode	NVDMNORM
MPROF	Name of NVDM mode profile	NVDMNORM
NODE_TYPE	Type of node to be configured	0, 1, 2
NUM_QUEUE	Number of remote connections	0, 1, 2, ...
OHN	Current host name of node to be configured	
PARTNER	Partner LU Name	
PEND	Requests for a target that is to be deleted	
PROTOCOL	Protocol to be used for remote connections	APPC, TCP/IP
PUNAME	Local PU Name of server	
RABNORM	Restart on abnormal deactivation	yes, no
RC	Used to store return code from commands	
RCV	TPN Profile Name (Receive)	
REMOTE_SERVER	List containing remote server names	RA39TCF1 RA60004
REMSERV	Name of remote server	
RNORM	Restart on normal deactivation	yes, no
RPROT	Protocols used for remote connections	TCP/IP, APPC, BOTH
RSAP	Remote SAP	08
RSHORT	Short name of remote server	
SELECT		
SERVER	Name of NVDM server (full domain name)	
SHORT	Short name for a target	
SIDR	Side Info Profile Name (Receive)	
SIDS	Side Info Profile Name (Send)	
SLIST	List of installed Change Files (to save target history)	
SNA_DS_DIR	Name of SNA/DS connection configuration file directory	/usr/lpp/netviewdm/db/snads_conn
SNA_DS_ROUTE	Name of SNA/DS routing table	/usr/lpp/netviewdm/db/routetab
SNA_R	List of remote servers connected using APPC	
SND	TPN Profile Name (Send)	
SOLICIT	Solicit SSCP	yes, no

Table 3 (Page 3 of 3). Global Shell Variables		
Variable Name	**Purpose**	**Example Value**
SUBST	Used to substitute fields in base configuration file	
SW_INV	Name of software inventory file	/usr/lpp/netviewdm/fndswinv
TCP_R	List of remote servers connected using TCP/IP	
TLIST	List containing existing targets for a server	
TNGP	List of targets belonging to a target group	
USE_CP_XID	Determines whether to use Control Point XID	yes, no
USRLIST	List of defined NVDM users	
VALUE	Used to pass data from database query	A, B, C
VALUE_LIST	Used to pass data from database query	A B C
VALUE_NUM	Number of elements in VALUE_LIST	0, 1, 2, ...
VTAMCP	VTAM Control Point Name	
WHERE		
XGP	List containing defined groups for a server in database	
XID	XID of server	
XLIST	List containing defined targets for a server from database	

B.2 Files Contained in Sample Configuration Code

There are four versions of the procedure for automatic configuration of NetView DM/6000 constituted by the combinations of the particular NetView DM/6000 Version (1.2 or 3.1) and the different configuration data model we presented (ODM or DB2/6000).

These four versions are located in four different directories of our code package:

/4508code/version_1

Configuration Procedure for NetView DM/6000 Version 1.2 with ODM configuration database

/4508code/version_2

Configuration Procedure for NetView DM/6000 Version 3.1 with ODM configuration database

/4508code/db_version_1

Configuration Procedure for NetView DM/6000 Version 1.2 with DB2/6000 configuration database

/4508code/db_version_2

Configuration Procedure for NetView DM/6000 Version 3.1 with DB2/6000 configuration database

See the file /4508code/README for details about the structure of the package.

In the following table we present the contents of the directories relevant to the scenarios treated in Chapter 5, "Testing the Automatic Configuration Script" on page 83 and Chapter 13, "Testing the Automatic Configuration Procedure with Software Distribution for AIX V3.1 with DB2/6000" on page 259. Taking into account your current configuration, you can combine the appropriate blocks forming the configuration script for the different NetView DM/6000 versions (in /4508code/blocks respectively /4508code/bookie_version_2) with the desired configuration database support.

Table 4 (Page 1 of 4). Filelist for Sample Code

Directory	File Name	Description
/4508code/version_1	config_nvdm	Main Configuration Script for NetView DM/6000 V1.2
/4508code/version_1	config_db.cre	ODM Creation File 1
/4508code/version_1	config_db2.cre	ODM Creation File 2
/4508code/version_1	config_db2_remote.cre	ODM Creation File for Different Database Support
/4508code/version_1	nvdm_cfg_static.odmadd	Class Definition File
/4508code/version_1	nvdm_groups.odmadd	Class Definition File
/4508code/version_1	nvdm_node.odmadd	Class Definition File
/4508code/version_1	nvdm_node2.odmadd	Class Definition File
/4508code/version_1	nvdm_queues.odmadd	Class Definition File
/4508code/version_1	nvdm_servers.odmadd	Class Definition File
/4508code/version_1	nvdm_users.odmadd	Class Definition File
/4508code/version_1	uicfg	Program to modify root.cli
/4508code/version_1	uicfg.c	C source code of uicfg
/4508code/version_1	build_db	Shell script to create ODM
/4508code/version_1	build_net_db	Shell script to create ODM for use with configure_network
/4508code/version_1	build_net_db2	Shell script to create ODM for use with configure_network_univ
/4508code/version_1	configure_network	Shell script to configure NVDM network
/4508code/version_1	configure_network_univ	Shell script to configure NVDM network with different database support
/4508code/version_1	node_list	File containing nodes to be configured by configure_network
/4508code/version_1	rebuild_db	Shell script to rebuild ODM after change
/4508code/version_1	edit_db	Shell script to edit ODM using odme

Table 4 (Page 2 of 4). Filelist for Sample Code		
Directory	**File Name**	**Description**
/4508code/blocks	ODM	ODM Access Procedures
/4508code/blocks	configure_nvdm_cfg	Shell Procedure
/4508code/blocks	configure_sna_cp	Shell Procedure
/4508code/blocks	configure_sna_dlc	Shell Procedure
/4508code/blocks	sna_initial	Shell Procedure
/4508code/blocks	configure_sna_dlc_profile	Shell Procedure
/4508code/blocks	configure_sna_link	Shell Procedure
/4508code/blocks	configure_sna_local_lu	Shell Procedure
/4508code/blocks	configure_sna_location	Shell Procedure
/4508code/blocks	configure_sna_mode	Shell Procedure
/4508code/blocks	configure_sna_send	Shell Procedure
/4508code/blocks	configure_sna_receive	Shell Procedure
/4508code/blocks	configure_sna_partner	Shell Procedure
/4508code/blocks	configure_side_snd	Shell Procedure
/4508code/blocks	configure_side_rcv	Shell Procedure
/4508code/blocks	get_queues	Shell Procedure
/4508code/blocks	configure_sna_ds_appc	Shell Procedure
/4508code/blocks	configure_sna_ds_tcpip	Shell Procedure
/4508code/blocks	nvdm_delete_targets	Shell Procedure
/4508code/blocks	nvdm_delete_groups	Shell Procedure
/4508code/blocks	nvdm_configure_targets	Shell Procedure
/4508code/blocks	nvdm_configure_groups	Shell Procedure
/4508code/blocks	add_users_aix	Shell Procedure
/4508code/blocks	configure_routetab	Shell Procedure
/4508code/blocks	nvdm_remote_targets	Shell Procedure
/4508code/blocks	restart_nvdm	Shell Procedure
/4508code/blocks	configure_sna_ds_conn	Shell Procedure
/4508code/blocks	get_sna_attributes	Shell Procedure
/4508code/blocks	nvdm_save_history	Shell Procedure
/4508code/blocks	add_fs_repos	Shell Procedure
/4508code/blocks	check_ports	Shell Procedure
/4508code/blocks	export_sna	Shell Procedure
/4508code/blocks	nvdm_update_server	Shell Procedure
/4508code/blocks	header	Header of configuration script `config_nvdm`. Contains global variables as well as useful procedures. Should be included in your own script.
/4508code/db_version_2	DB2	Shell Procedure with Database Access Methods

Table 4 (Page 3 of 4). Filelist for Sample Code		
Directory	**File Name**	**Description**
/4508code/db_version_2	NVDM_CFG_STATIC.del	Import Data File in DEL-Format
/4508code/db_version_2	NVDM_GROUPS.del	Import Data File in DEL-Format
/4508code/db_version_2	NVDM_NODE.del	Import Data File in DEL-Format
/4508code/db_version_2	NVDM_QUEUES.del	Import Data File in DEL-Format
/4508code/db_version_2	NVDM_SERVERS.del	Import Data File in DEL-Format
/4508code/db_version_2	NVDM_USERS.del	Import Data File in DEL-Format
/4508code/db_version_2	build_db	Shell Procedure for Building the NetView DM/6000 Configuration Database
/4508code/db_version_2	build_db2	Shell Procedure for Building the Database for Remote Configuration
/4508code/db_version_2	build_db_odm	Shell Procedure for Creating Database from ODM
/4508code/db_version_2	config_nvdm	Main Configuration Script for NetView DM/6000 with DB2/6000 Support
/4508code/db_version_2	configure_network_univ	Remote Configuration Script for Nodes with Different Database Support
/4508code/db_version_2	db22odm	Shell Procedure for DB2/6000-to-ODM Conversion
/4508code/db_version_2	db_authorize.sql	SQL Statements File for Authorizing Database Users
/4508code/db_version_2	db_create	Shell Procedure for Creating/Recreating the Database
/4508code/db_version_2	db_import.sql	SQL Statements File for Importing Configuration Data
/4508code/db_version_2	db_model.sql	SQL Statements File for Database Table Definitions
/4508code/db_version_2	db_model2.sql	SQL Statements File for Database Table Definitions (Remote Configuration)
/4508code/db_version_2	node_list	Similar to /4508code/version_1
/4508code/db_version_2	odm2db2	Shell Procedure for ODM-to-DB2/6000 Conversion
/4508code/db_version_2	rebuild_db	Shell Procedure for Exporting the Configuration Database
/4508code/db_version_2	uicfg	The Same as in /4508code/version_1
/4508code/db_version_2	uicfg.c	The Same as in /4508code/version_1
/4508code/uidb2	dbAccess.h	Database Access Include File
/4508code/uidb2	dbAccessDB2.c	Database Access Implementation File (DB2/6000)

Directory	File Name	Description
/4508code/uidb2	dbFrames.h	Database Frames Include File
/4508code/uidb2	dbFrames.c	Database Frames Implementation File
/4508code/uidb2	makefile	Makefile for Compiling and Linking the Graphical Interface Program
/4508code/uidb2	uicfgdb.c	Main Program File of the Graphical User Interface to the NetView DM/6000 Conf
/4508code/mvs	nvdm_mvs	Sample Script for NVDM/MVS

Table 4 (Page 4 of 4). Filelist for Sample Code

Appendix C. Source Code of the Graphical User Interface

The following are code listings of the modules constituting the graphical user interface to the NetView DM/6000 configuration database, presented in Chapter 15, “Modifying Configuration Data Using a Graphical User Interface” on page 293.

C.1 Makefile

```
# Makefile  (c)'95 PLamen Kiradjiev
#
#

# compiler and other variables
CC=cc
MOTIFLIBS=-lXm -lXt -lX11
DB2LIBS=-ldb2 -lc -lm -ls
LOADMAP=-bloadmap:map
OBJ=uicfgdb.o dbAccessDB2.o dbFrames.o
RM=rm -f

# Debug information
#CC_FLAGS=-v -g
# Normal work flags
CC_FLAGS=-O2

uicfgdb: $(OBJ)
	$(CC) -o $@  $(DB2LIBS) $(LOADMAP) $(MOTIFLIBS) $(OBJ)

uicfgdb.o: uicfgdb.c dbAccess.h dbFrames.h
	$(CC) -c $< $(CC_FLAGS)

dbFrames.o: dbFrames.c dbFrames.h dbAccess.h
	$(CC) -c $< $(CC_FLAGS)

dbAccessDB2.o: dbAccessDB2.c dbAccess.h
	$(CC) -c $< $(CC_FLAGS) -I $(DBOWNERHOME)/sqllib/include

clean:
	$(RM) uicfgdb $(OBJ) map
```

Figure 176. Makefile for the Graphical User Interface uicfgdb

C.2 Database Access

Generic Database Access Interface:

```
/*************************************************************************
**
** File:    dbAccess.h
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: Database-Independent Interface
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
*************************************************************************/



/*----------------- Includes --------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>


/*----------------- Constants -------------------------------------------*/
#define NONE 0
#define SELECTED 1
#define INSERTED 2
#define UPDATED 3
#define DELETED 4


/*---------------- Type Definitions -------------------------------------*/
typedef int Mode   /* SELECTED, INSERTED or UPDATED */
typedef char *Message   /* SQL message string */
typedef struct Row{
   char **data   /* column values array */
   Mode mode   /* row mode */
   unsigned char changed  /* 1, if any column changed, else 0 */
   char **chData   /* changed values array
                     (NULL for unchanged columns) */
  }Row;
typedef struct ColAttributes{
   char **name    /* column names */
   int *length    /* column lengths */
   unsigned char *isNullable  /* is nullable array */
  }ColAttributes;
typedef struct Table{
   char *name   /* table name */
   ColAttributes colAttr  /* column attributes */
   int colCount   /* result column count */
   Row *rows   /* row array */
   int rowCount   /* selected row count */
   Message message  /* SQL message in error case */
  }Table;
```

Figure 177 (Part 1 of 2). Generic Database Interface dbAccess.h

```
/*--------------- Procedures ---------------------------------------------*/

/* Connect to database */
Message dbConnect(char *dbName);

/* Disconnect from the database */
Message dbDisconnect();

/* Make a check point */
Message dbCheckPoint(void);

/* Select columns from table tableName with selection criteria selection */
Table *dbSelect(char *tableName, int colCount,
           char *columns[], char *selection);

/* Insert values into table tableName with columns sequence columns */
Message dbInsert(char *tableName, int colCount,
         char *columns[], char *values[]);

/* Update table tableName by setting setColumns to values for selection */
Message dbUpdate(char *tableName, int colCount, char *setColumns[],
         char *values[], char *selection);

/* Delete from table tableName with selection criteria selection */
Message dbDelete(char *tableName, char *selection);

/* Build selection string for a particular row */
char *selectThisRow(Row row, int colCount, char *columns[]);

/* Build the order-by part of selection statement */
char *dbOrderString(int colCount, char *columns[]);

/* Build an always false where part of selection statement */
char *dbAlwaysFalse(void);

/* free space allocated for table */
void freeTable(Table *table);
```

Figure 177 (Part 2 of 2). Generic Database Interface dbAccess.h

DB2/6000 Implementation:

```
/*****************************************************************************
**
** File:    dbAccessDB2.c
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: DB2 Access Procedures
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
******************************************************************************/


/*------------------ Includes ---------------------------------------------*/
#include "dbAccess.h"
#include "sqlcli1.h"
#include <string.h>


/*------------------ Constants -------------------------------------------*/
#define MAX_STMT_LEN 2000
#define MAXCOLS 100
#define MAXSELECT 100


/*------------------ Macros ----------------------------------------------*/
#define  max(a,b) (a > b ? a : b)


/*------------------ Globals ---------------------------------------------*/
static SQLHENV henv;
static SQLHDBC hdbc;



/*------------------ Forward Definitions ---------------------------------*/
Message error(SQLHENV henv, SQLHDBC hdbc, SQLHSTMT hstmt);
Table *allocTable(char *name, ColAttributes *colAttrPtr, int colCount,
             Row rows[], int rowCount, Message msg);


/*--------------- Procedures ---------------------------------------------*/


Message dbConnect(char *dbName)
{
/* allocate an environment handle */
  if (SQLAllocEnv(&henv) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* allocate a connection handle */
```

Figure 178 (Part 1 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
  if (SQLAllocConnect(henv, &hdbc) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* connect to database (without providing user name and password ) */
  if (SQLConnect(hdbc, dbName, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS)
   != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

}


Message dbDisconnect()
{
  RETCODE rc;

/* commit */
  if (SQLTransact(henv, hdbc, SQL_COMMIT) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* disconnect from database */
  if ((rc=SQLDisconnect(hdbc)) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* free connection handle */
  if (SQLFreeConnect(hdbc) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* free environment handle */
  if (SQLFreeEnv(henv) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

}




Message dbCheckPoint(void)
{
  Message msg = (Message) calloc(1, SQL_MAX_MESSAGE_LENGTH+1);

  *msg = '\0';
  if (SQLTransact(henv, hdbc, SQL_COMMIT) != SQL_SUCCESS) {
    dbDisconnect();
    sprintf(msg, "Fatal Error.\nCommit failed. Exiting application...\n");
  }
  return msg;
}
```

Figure 178 (Part 2 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
Table *dbSelect(char *tableName, int colCount,
               char *columns[], char *selection)
{
  SQLRETURN rc;
  SQLHSTMT hstmt;
  int isNullable;
  int n=0, i=0, j, outColCount;
  SQLINTEGER collen[MAXCOLS], outlen[MAXCOLS];
  SQLCHAR sqlstmt[MAX_STMT_LEN + 1] = "select";
  SQLCHAR colname[32];
  SQLCHAR *data[MAXCOLS];
  ColAttributes colAttr;
  Row *rowArray = NULL;
  Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);

  *msg = '\0';
/* build SQL statement string */
  if (colCount == 0)
    sprintf(sqlstmt, "select * from %s", tableName);
  else {
    for (i=0 i<colCount i++)
      sprintf(sqlstmt, "%s%s%s", sqlstmt, (i==0)?" ":",", columns[i]);
    sprintf(sqlstmt, "%s from %s", sqlstmt, tableName);
  }
  if (selection != NULL)
    sprintf(sqlstmt, "%s %s", sqlstmt, selection);

/* allocate SQL statement handle */
  if (SQLAllocStmt(hdbc, &hstmt) != SQL_SUCCESS)
    return allocTable(tableName, NULL, 0, NULL, 0, error(henv, hdbc, SQL_NULL_HSTMT));

/* execute SQL statement */
  rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
  if (rc != SQL_SUCCESS)
   return allocTable(tableName, NULL, 0, NULL, 0, error(henv, hdbc, hstmt));

/* determine the result column count */
    SQLColAttributes(hstmt, 1, SQL_COLUMN_COUNT, NULL, 0,
              NULL, &outColCount);

/* allocate column attribute arrays */
    colAttr.name = (char **) calloc(outColCount, sizeof(char *));
    colAttr.length = (int *) calloc(outColCount, sizeof(int));
    colAttr.isNullable = (unsigned char *) calloc(outColCount, sizeof(unsigned char));

/* for each output column... */
  for (i=0 i<outColCount i++) {

/* get column attributes */
```

Figure 178 (Part 3 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
    SQLColAttributes(hstmt, i+1, SQL_COLUMN_NAME, colname, sizeof(colname),
             NULL, NULL);
    SQLColAttributes(hstmt, i+1, SQL_COLUMN_NULLABLE, NULL, 0,
             NULL, &isNullable);
    SQLColAttributes(hstmt, i+1, SQL_COLUMN_DISPLAY_SIZE, NULL, 0,
             NULL, &collen[i]);

/* fill values into column attributes pointer */
    colAttr.name[i] = (char *) calloc(strlen(colname) + 1, sizeof(char));
    strcpy(colAttr.name[i],colname);
    colAttr.length[i] = collen[i];
    colAttr.isNullable[i] = (isNullable)?1:0;

/* bind column */
    data[i] = (SQLCHAR *) calloc(collen[i] + 1, sizeof(SQLCHAR));
    SQLBindCol(hstmt, i+1, SQL_C_CHAR, data[i], collen[i]+1, &outlen[i]);
  }

/* fetch data */
  while (SQLFetch(hstmt) != SQL_NO_DATA_FOUND) {

  if (n >= MAXSELECT)
    sprintf(msg, "Maximum of %d Viewable Rows Reached.\nThe Rest Has Been Cut.", n);

/* reallocate row pointer */
    rowArray = (Row *)realloc(rowArray, (n+1)*sizeof(Row));
    rowArray[n].data = (char **) calloc(outColCount, sizeof(char *));
    rowArray[n].mode = SELECTED;
    rowArray[n].changed = 0;
    rowArray[n].chData = (char **) calloc(outColCount, sizeof(char *));

    for (i=0  i<outColCount i++) {

/* set output of null values */
      if (outlen[i] == SQL_NULL_DATA)
   *data[i] = '\0';

    for (j=strlen(data[i])-1; j>=0; j--)
      if (data[i][j] == ' ')
   data[i][j] = '\0';
      else
   break;
    rowArray[n].data[i] = (char *) calloc(collen[i]+1, sizeof(char));
    strcpy(rowArray[n].data[i],data[i]);
    }
    n++;
  }

  SQLFreeStmt (hstmt, SQL_DROP )        /* free statement handle */
```

Figure 178 (Part 4 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
/* set message string */
  if (strlen(msg) == 0)
    sprintf(msg, "%d Rows Selected.",n);

  for(i=0;i<outColCount;i++)
    free(data[i]);

  return allocTable(tableName, &colAttr, outColCount, rowArray, n, msg);
}



Message dbInsert(char *tableName, int colCount,
         char *columns[], char *values[])
{
  int i;
  SQLHSTMT      hstmt;
  SQLCHAR  sqlstmt[MAX_STMT_LEN + 1] = "insert into",
        sqlstmtExt[MAX_STMT_LEN + 1] = ") values ";
  SQLRETURN rc;

/* build SQL insert statement */
  sprintf(sqlstmt, "%s %s ", sqlstmt, tableName);
  for (i=0 i<colCount i++) {
    sprintf(sqlstmt, "%s%s%s", sqlstmt, (i!=0)?",":"(", columns[i]);
    if (strlen(values[i]) == 0)
      sprintf(sqlstmtExt, "%s%sNULL", sqlstmtExt, (i!=0)?",":"(");
    else
      sprintf(sqlstmtExt, "%s%s'%s'", sqlstmtExt, (i!=0)?",":"(", values[i]);
  }
  sprintf(sqlstmt, "%s %s)", sqlstmt, sqlstmtExt);

/* allocate a statement handle */
  if (SQLAllocStmt(hdbc, &hstmt) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* execute statement */
  rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
  switch (rc) {
    case SQL_SUCCESS: {
        Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);
   *msg = '\0';
   return msg;
    }
    case SQL_NO_DATA_FOUND: {
   Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);
        sprintf(msg, "\nSQL0100W  No row was found for FETCH, UPDATE or DELETE; ");
        sprintf(msg, "%sor the result of a query is an empty table.  SQLSTATE=02000\n",msg);
        return msg;
    }
```

Figure 178 (Part 5 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
    default:
        return error(henv, hdbc, hstmt);
  }
}



Message dbUpdate(char *tableName, int colCount, char *setColumns[],
         char *values[], char *selection)
{
  int i;
  SQLHSTMT      hstmt;
  SQLCHAR       sqlstmt[MAX_STMT_LEN + 1] = "update";
  SQLRETURN     rc;

/* build SQL insert statement */
  sprintf(sqlstmt, "%s %s set", sqlstmt, tableName);
  for (i=0 i<colCount i++)
    if (strlen(values[i]) == 0)
      sprintf(sqlstmt, "%s%s%s=NULL", sqlstmt, (i!=0)?",":" ", setColumns[i]);
    else
      sprintf(sqlstmt, "%s%s%s='%s'", sqlstmt, (i!=0)?",":" ", setColumns[i], values[i]);
  if (selection != NULL)
    sprintf(sqlstmt, "%s %s", sqlstmt, selection);

/* allocate a statement handle */
  if (SQLAllocStmt(hdbc, &hstmt) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* execute statement */
  rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
  switch (rc) {
    case SQL_SUCCESS: {
        Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);
        *msg = '\0';
        return msg;
    }
    case SQL_NO_DATA_FOUND: {
   Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);
        sprintf(msg, "\nSQL0100W  No row was found for FETCH, UPDATE or DELETE; ");
        sprintf(msg, "%sor the result of a query is an empty table.  SQLSTATE=02000\n",msg);
        return (Message) msg;
    }
    default:
        return error(henv, hdbc, hstmt);
  }
}
```

Figure 178 (Part 6 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
Message dbDelete(char *tableName, char *selection)
{
  SQLHSTMT      hstmt;
  SQLCHAR       sqlstmt[MAX_STMT_LEN + 1] = "";
  SQLRETURN     rc;

/* build SQL insert statement */
  sprintf(sqlstmt, "delete from %s", tableName);
  if (selection != NULL)
    sprintf(sqlstmt, "%s %s", sqlstmt, selection);

/* allocate a statement handle */
  if (SQLAllocStmt(hdbc, &hstmt) != SQL_SUCCESS)
    return error(henv, hdbc, SQL_NULL_HSTMT);

/* execute statement */
  rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
  switch (rc) {
    case SQL_SUCCESS: {
        Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);
        *msg = '\0';
        return msg;
    }
    case SQL_NO_DATA_FOUND: {
   Message msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);
        sprintf(msg, "\nSQL0100W  No row was found for FETCH, UPDATE or DELETE; ");
        sprintf(msg, "%sor the result of a query is an empty table.  SQLSTATE=02000\n",msg);
        return (Message) msg;
    }
    default:
        return error(henv, hdbc, hstmt);
  }
}



void freeTable(Table *table)
{
  int i,j;

  for (i=0  i<table->rowCount i++) {
    for (j=0; j<table->colCount; j++)
      free(table->rows[i].data[j]);
    free(table->rows[i].data);
  }
  for (j=0; j<table->colCount; j++)
    free(table->colAttr.name[j]);
  free(table->colAttr.name);
  free(table->colAttr.length);
  free(table->colAttr.isNullable);
```

Figure 178 (Part 7 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
  free(table->message);
  free(table->rows);
  free(table);
}




char *selectThisRow(Row row, int colCount, char *columns[])
{
  char *selection;
  int i;

  if (colCount == 0) return NULL;
  selection = (char *) calloc(MAX_STMT_LEN + 1, sizeof(char));
  for (i=0  i<colCount  i++) {
    if (i == 0)
      if (strlen(row.data[i]) == 0)
   sprintf(selection, "where %s is null", columns[i]);
      else
        sprintf(selection, "where %s='%s'", columns[i], row.data[i]);
    else
      if (strlen(row.data[i]) == 0)
   sprintf(selection, "%s and %s is null", selection, columns[i]);
      else
        sprintf(selection, "%s and %s='%s'", selection, columns[i], row.data[i]);
  }
  return selection;
}



char *dbOrderString(int colCount, char *columns[])
{
  char *selection;
  int i;

  if (colCount == 0) return NULL;
  selection = (char *) calloc(MAX_STMT_LEN + 1, sizeof(char));
  for (i = 0; i<colCount; i++)
    sprintf(selection, "%s%s%s", selection, (i==0)?"order by ":",", columns[i]);
  return selection;
}



char *dbAlwaysFalse(void)
{
  return "where 1<>1";
```

Figure 178 (Part 8 of 9). DB2/6000 Implementation File dbAccessDB2.c

```
}



/*--------------------- Help Procedures --------------------------------*/


/* Return  SQL error code */
Message error(SQLHENV henv, SQLHDBC hdbc, SQLHSTMT hstmt)
{
  SQLCHAR buffer[SQL_MAX_MESSAGE_LENGTH + 1] = "";
  SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1] = "";
  SQLINTEGER sqlcode = 0;
  SQLSMALLINT length = 0;
  char *msg = (Message) calloc(1,SQL_MAX_MESSAGE_LENGTH+1);

  *msg = '\0';
  while (SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
         SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS)
    sprintf(msg, "%s\n%s\n", msg, buffer);
  return (Message) msg;
}


/* build a table structure */
Table *allocTable(char *name, ColAttributes *colAttrPtr, int colCount,
                 Row rows[], int rowCount, Message msg)
{
  Table *table = (Table *) calloc(1, sizeof(Table));

  table->name = name;
  table->colAttr = *colAttrPtr;
  table->colCount = colCount;
  table->rows = rows;
  table->rowCount = rowCount;
  table->message = msg;
  return table;
}
```

Figure 178 (Part 9 of 9). DB2/6000 Implementation File dbAccessDB2.c

C.3 Database Frames

Database Frames Interface:

```
/*****************************************************************************
**
** File:    dbFrames.h
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: Graphic Database Frames Interface
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
*****************************************************************************/


/*---------------------- Includes -------------------------------------------*/
#include "dbAccess.h"
#include <Xm/PushB.h>



/*---------------------- Procedures ----------------------------------------*/

/* View Database Data in a Table Frame */
void tableFrame(Widget w, Table *table);

/* Insert Table Frame for a Sequence of Tables */
void insertFrame(Widget w, int tabCount, Table **table);

/* Message Box */
void messageBox(Widget top, String msg, int fatal);
```

Figure 179. Database Frames Include File dbFrames.h

Database Frames Implementation:

```
/***************************************************************************
**
** File:    dbFrames.c
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: Graphic Database Frames
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
***************************************************************************/


/*---------------------- Includes -------------------------------------------*/
#include "dbFrames.h"
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/Label.h>
#include <Xm/Text.h>
#include <Xm/ScrolledW.h>
#include <Xm/Separator.h>



/*--------------------- Constants ------------------------------------------*/
#define MAXIDLEN 40
#define DEFAULTFIELDLEN 20
#define DEFAULTMARGIN 3
#define DEFAULTSPACING 0
#define DEFTXTMARGIN 5
#define HIGHLIGHT 2
#define SWMARGIN 0
#define SWSPACING 4
#define SCROLLBARWIDTH 5
#define OKSTRING "Ok!"



/*-------------------- Macros ------------------------------------------*/
#define min(a,b) (a < b ? a : b)
#define max(a,b) (a > b ? a : b)




/*---------------------- Type Definitions ------------------------------*/
typedef struct TableInfo{
    int currentRow;
    int currentCol;
    Table *table;
```

Figure 180 (Part 1 of 20). Database Frames Implementation File dbFrames.c

```
    Widget *rows;
    Widget **field;
    Widget tMessage;
  }TableInfo;

typedef struct InsertInfo{
    int tabCount;
    Table **tables;
    Widget *fields;
    Widget shell;
  }InsertInfo;
typedef struct Index{
    int row;
    int col;
  }Index;



/*----------------------- Globals -------------------------------------*/
static XmFontList fontList14, fontList18;
static Pixel fg, bg;



/*---------------------- Forward Definitions ------------------------------*/
void quitCB(Widget w, XtPointer client_data, XtPointer call_data);
void entryFieldCB(Widget w, XtPointer client_data, XtPointer call_data);
void exitFieldCB(Widget w, XtPointer client_data, XtPointer call_data);
void valueChangedCB(Widget w, XtPointer client_data, XtPointer call_data);
void updateRowCB(Widget w, XtPointer client_data, XtPointer call_data);
void insertRowCB(Widget w, XtPointer client_data, XtPointer call_data);
void deleteRowCB(Widget w, XtPointer client_data, XtPointer call_data);
void commitCB(Widget w, XtPointer client_data, XtPointer call_data);
void refreshCB(Widget w, XtPointer client_data, XtPointer call_data);
void okMsgCB(Widget w, XtPointer client_data, XtPointer call_data);
void insertCB(Widget w, XtPointer client_data, XtPointer call_data);
void cancelInsertCB(Widget w, XtPointer client_data, XtPointer call_data);
void quitInsertCB(Widget w, XtPointer client_data, XtPointer call_data);
void showRows(Widget parentW, TableInfo *tabinfo);
void buildCols(Widget parentW, InsertInfo *ininfo);
void allocateNewRow(TableInfo *tabinfo);
void clearTable(TableInfo *tabinfo, Boolean unmanage);



/*---------------------- Procedures ----------------------------------------*/


void tableFrame(Widget w, Table *table)
{
```

Figure 180 (Part 2 of 20). Database Frames Implementation File dbFrames.c

```
  Display *display;
  XFontStruct *font;
  Widget tableShell, tableForm, tLabel, tScrollW, tButtonRC;
  Widget tSep, tTableRC, tColumnRC, tScrollRowsW, tRowsRC, tScrollMsg;
  Widget *tColumnNames = (Widget *) calloc(table->colCount, sizeof(Widget));
  Widget tButtons[6];
  XmString t;
  Dimension sw;
  int i, j, len;
  char *colname;
  char title[MAXIDLEN + 20] = "Database Table: ";
  char *tBtnLbls[] = {"Insert Row", "Update Row", "Delete Row",
              "Commit", "Refresh", "Quit"};
  TableInfo *tabinfo = (TableInfo *) calloc(1, sizeof(TableInfo));

  display = XtDisplay(w);
  tableShell = XtVaAppCreateShell(NULL, "tableShell",
    topLevelShellWidgetClass,display,
    XtNtitle, "Table Frame",
    NULL);

/* set used font lists */
  font = XLoadQueryFont(display, "-*-courier-bold-r-*--14-*");
  fontList14 = XmFontListCreate(font, " ");
  font = XLoadQueryFont(display, "-*-courier-bold-r-*--18-*");
  fontList18 = XmFontListCreate(font, " ");

  tableForm = XtVaCreateWidget("tableForm",
    xmFormWidgetClass, tableShell,
    NULL);

  strcat(title,table->name);
  t = XmStringCreateLocalized(title);
  tLabel = XtVaCreateManagedWidget("tLabel",
    xmLabelWidgetClass, tableForm,
    XmNtopAttachment, XmATTACH_FORM,
    XmNtopOffset, 50,
    XmNlabelString, t,
        XmNfontList, fontList18,
        NULL);
  XmStringFree(t);

  tScrollW = XtVaCreateWidget("tScrollW",
        xmScrolledWindowWidgetClass, tableForm,
        XmNtopAttachment, XmATTACH_WIDGET,
        XmNtopWidget, tLabel,
        XmNtopOffset, 30,
        XmNleftAttachment, XmATTACH_FORM,
        XmNrightAttachment, XmATTACH_FORM,
        XmNwidth, 700,
```

Figure 180 (Part 3 of 20). Database Frames Implementation File dbFrames.c

```
        XmNheight, 600,
        XmNscrollingPolicy, XmAUTOMATIC,
        NULL);

    tTableRC = XtVaCreateWidget("tTableRC",
      xmRowColumnWidgetClass, tScrollW,
      NULL);

    tColumnRC = XtVaCreateWidget("tColumnRC",
        xmRowColumnWidgetClass, tTableRC,
      XmNorientation, XmHORIZONTAL,
      XmNmarginWidth, DEFAULTMARGIN + DEFTXTMARGIN + SWMARGIN + SWSPACING,
      XmNspacing, DEFAULTSPACING + 2*DEFTXTMARGIN +2*HIGHLIGHT,
      XmNnumColumns, 1,
      NULL);

    for (i=0  i<table->colCount i++) {
      len = min(DEFAULTFIELDLEN, table->colAttr.length[i]);
      len = max(len, strlen(table->colAttr.name[i]));
      if (i == table->colCount-1)
        len = len + SCROLLBARWIDTH;
      colname = (char *) calloc(len + 1, sizeof(char));
      strcpy(colname, table->colAttr.name[i]);
      for(j=strlen(colname);j<len;j++)
        colname[j] = ' ';
      colname[len] = '\0';
      tColumnNames[i] = XtVaCreateManagedWidget(colname,
        xmLabelWidgetClass, tColumnRC,
        XmNfontList, fontList14,
        NULL);
      free(colname);
    }

    tSep = XtVaCreateManagedWidget("tSep",
      xmSeparatorWidgetClass, tTableRC, NULL);

    tScrollRowsW = XtVaCreateManagedWidget("tScrollRowsW",
        xmScrolledWindowWidgetClass, tTableRC,
        XmNscrollingPolicy, XmAUTOMATIC,
      XmNscrolledWindowMarginWidth, SWMARGIN,
      XmNspacing, SWSPACING,
        XmNheight, 430,
        NULL);

    tRowsRC= XtVaCreateWidget("tRowsRC",
        xmRowColumnWidgetClass, tScrollRowsW,
        XmNentryAlignment, XmALIGNMENT_BEGINNING,
        XmNnumColumns, 1,
        NULL);
```

Figure 180 (Part 4 of 20). Database Frames Implementation File dbFrames.c

```
tabinfo->table = table;

showRows(tRowsRC, tabinfo);

tScrollMsg = XtVaCreateManagedWidget("tScrollMsg",
      xmScrolledWindowWidgetClass, tTableRC,
      XmNscrollingPolicy, XmAUTOMATIC,
      XmNheight, 90,
      NULL);

if (strlen(table->message) == 0)
  t = XmStringCreateLocalized(OKSTRING);
else
  t = XmStringCreateLtoR(table->message, " ");
tabinfo->tMessage = XtVaCreateManagedWidget("Message:",
      xmLabelWidgetClass, tScrollMsg,
      XmNlabelString, t,
  XmNalignment, XmALIGNMENT_BEGINNING,
      XmNfontList, fontList14,
      NULL);
XmStringFree(t);

tButtonRC = XtVaCreateWidget("tButtonRC",
      xmRowColumnWidgetClass, tableForm,
      XmNtopAttachment, XmATTACH_WIDGET,
  XmNtopWidget, tScrollW,
  XmNtopOffset, 20,
      XmNorientation, XmHORIZONTAL,
      XmNnumColumns, 2,
      XmNentryAlignment, XmALIGNMENT_CENTER,
      XmNpacking, XmPACK_COLUMN,
      NULL);

for (i=0  i<6  i++)
  tButtons[i] = XtVaCreateManagedWidget(tBtnLbls[i],
      xmPushButtonWidgetClass, tButtonRC,
      XmNfontList, fontList18,
  XmNuserData, tabinfo,
      NULL);

XtAddCallback(tButtons[0], XmNactivateCallback, insertRowCB, tScrollRowsW);
XtAddCallback(tButtons[1], XmNactivateCallback, updateRowCB, NULL);
XtAddCallback(tButtons[2], XmNactivateCallback, deleteRowCB, tScrollRowsW);
XtAddCallback(tButtons[3], XmNactivateCallback, commitCB, tScrollRowsW);
XtAddCallback(tButtons[4], XmNactivateCallback, refreshCB, tScrollRowsW);
XtAddCallback(tButtons[5], XmNactivateCallback, quitCB, tableShell);

XtManageChild(tRowsRC);
XtManageChild(tColumnRC);
XtManageChild(tTableRC);
```

Figure 180 (Part 5 of 20). Database Frames Implementation File dbFrames.c

```
  XtManageChild(tScrollW);
  XtManageChild(tButtonRC);
  XtManageChild(tableForm);
  XtRealizeWidget(tableShell);

  if (tabinfo->table->rowCount > 0) {
      XtVaGetValues(tabinfo->field[0][0],
    XmNforeground, &fg,
    XmNbackground, &bg, NULL);
      XmProcessTraversal(tabinfo->field[0][0], XmTRAVERSE_CURRENT);
  }
}




void insertFrame(Widget w, int tabCount, Table **tables)
{
  Display *display;
  XFontStruct *font;
  Widget insertShell, insertForm, iLabel, iBody, iButtonRC;
  Widget iButtons[3];
  char *iBtnLbls[] = {"Insert", "Cancel", "Quit"};
  char title[MAXIDLEN + 20] = "Insert Into ";
  XmString t;
  int i;
  InsertInfo *ininfo = (InsertInfo *) calloc(1, sizeof(InsertInfo));

  if (tabCount == 0) return;
  display = XtDisplay(w);
  insertShell = XtVaAppCreateShell(NULL, "insertShell",
        topLevelShellWidgetClass,display,
        XtNtitle, "Insert Frame",
        NULL);

/* set used font lists */
  font = XLoadQueryFont(display, "-*-courier-bold-r-*--14-*");
  fontList14 = XmFontListCreate(font, " ");
  font = XLoadQueryFont(display, "-*-courier-bold-r-*--18-*");
  fontList18 = XmFontListCreate(font, " ");

  insertForm = XtVaCreateWidget("insertForm",
        xmFormWidgetClass, insertShell,
        NULL);

  strcat(title,(*tables)->name);
  t = XmStringCreateLocalized(title);
  iLabel = XtVaCreateManagedWidget("iLabel",
        xmLabelWidgetClass, insertForm,
        XmNtopAttachment, XmATTACH_FORM,
```

Figure 180 (Part 6 of 20). Database Frames Implementation File dbFrames.c

```
       XmNtopOffset, 50,
       XmNlabelString, t,
       XmNfontList, fontList18,
       NULL);
  XmStringFree(t);

  iBody = XtVaCreateManagedWidget("iBody",
       xmScrolledWindowWidgetClass, insertForm,
       XmNtopAttachment, XmATTACH_WIDGET,
       XmNtopWidget, iLabel,
       XmNtopOffset, 30,
       XmNleftAttachment, XmATTACH_FORM,
       XmNrightAttachment, XmATTACH_FORM,
       XmNwidth, 500,
       XmNheight, 400,
       XmNscrollingPolicy, XmAUTOMATIC,
       NULL);

  ininfo->tabCount = tabCount;
  ininfo->tables = tables;
  buildCols(iBody, ininfo);

  iButtonRC = XtVaCreateWidget("iButtonRC",
       xmRowColumnWidgetClass, insertForm,
       XmNtopAttachment, XmATTACH_WIDGET,
       XmNtopWidget, iBody,
       XmNtopOffset, 20,
       XmNorientation, XmHORIZONTAL,
       XmNentryAlignment, XmALIGNMENT_CENTER,
       XmNpacking, XmPACK_COLUMN,
       NULL);

  for (i=0  i<3  i++)
    iButtons[i] = XtVaCreateManagedWidget(iBtnLbls[i],
       xmPushButtonWidgetClass, iButtonRC,
       XmNfontList, fontList18,
    XmNuserData, ininfo,
       NULL);

  ininfo->shell = insertShell;

  XtAddCallback(iButtons[0], XmNactivateCallback, insertCB, NULL);
  XtAddCallback(iButtons[1], XmNactivateCallback, cancelInsertCB, NULL);
  XtAddCallback(iButtons[2], XmNactivateCallback, quitInsertCB, w);

  XtManageChild(iButtonRC);
  XtManageChild(insertForm);
  XtRealizeWidget(insertShell);
}
```

Figure 180 (Part 7 of 20). Database Frames Implementation File dbFrames.c

```
void messageBox(Widget top, String msg, int fatal)
{
  Widget message;
  XmString t;
  Arg args[5];
  int n = 0;

/* create selection box */
  t = XmStringCreateLtoR(msg, " ");

  XtSetArg(args[n], XmNmessageString,t); n++;
  XtSetArg(args[n], XmNlabelFontList,fontList14); n++;
  XtSetArg(args[n], XmNbuttonFontList,fontList14); n++;
  XtSetArg(args[n], XmNautoUnmanage,False); n++;
  message = (Widget)MessageDialog(top, "message", args, n);
  XmStringFree(t);

  XtUnmanageChild(XmMessageBoxGetChild(message, XmDIALOG_CANCEL_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(message, XmDIALOG_HELP_BUTTON));

  XtAddCallback(message, XmNokCallback, okMsgCB, (XtPointer) fatal);

  XtManageChild(message);
  XtPopup(XtParent(message), XtGrabNone);
}




/*----------------------- Callbacks -------------------------------------*/


/* select row after entering field of it */
void entryFieldCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i;
  XmString t;
  Index *index = (Index *) client_data;
  TableInfo *tabinfo;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  for (i=0  i< tabinfo->table->colCount i++)
      XtVaSetValues(tabinfo->field[index->row][i],
    XmNforeground, bg, XmNbackground, fg, NULL);
  if (tabinfo->currentRow != index->row) {
    for (i=0  i< tabinfo->table->colCount i++)
      XtVaSetValues(tabinfo->field[index->row][i], XmNeditable, False, NULL);
```

Figure 180 (Part 8 of 20). Database Frames Implementation File dbFrames.c

```
    t = XmStringCreateLocalized(OKSTRING);
    XtVaSetValues(tabinfo->tMessage, XmNlabelString, t, NULL);
    XmStringFree(t);
  }
  tabinfo->currentRow = index->row;
  tabinfo->currentCol = index->col;
}



/* deselect row after exiting a field of it */
void exitFieldCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i;
  TableInfo *tabinfo;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  for (i=0  i< tabinfo->table->colCount i++)
    XtVaSetValues(tabinfo->field[tabinfo->currentRow][i],
    XmNforeground, fg, XmNbackground, bg, NULL);
}



/* quit table frame */
void quitCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  Widget top = (Widget) client_data;
  TableInfo *tabinfo;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  XtDestroyWidget(top);
  clearTable(tabinfo,0);
}



/* change value in table data structure */
void valueChangedCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i;
  TableInfo *tabinfo;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  switch(tabinfo->table->rows[tabinfo->currentRow].mode) {
    case SELECTED:
    case UPDATED: {
      char *val;

      tabinfo->table->rows[tabinfo->currentRow].mode = UPDATED;
```

Figure 180 (Part 9 of 20). Database Frames Implementation File dbFrames.c

```
      tabinfo->table->rows[tabinfo->currentRow].changed = 1;
      if (tabinfo->table->rows[tabinfo->currentRow].chData[tabinfo->currentCol] == NULL)
    tabinfo->table->rows[tabinfo->currentRow].chData[tabinfo->currentCol] =
      (char *) calloc(tabinfo->table->colAttr.length[tabinfo->currentCol]+1,
                 sizeof(char));
      val = XmTextGetString(w);
      for (i=tabinfo->table->colAttr.length[tabinfo->currentCol]-1; i>=0;i--) {
    if (val[i] == ' ') val[i] = '\0';
    else break;
      }
      strcpy(tabinfo->table->rows[tabinfo->currentRow].chData[tabinfo->currentCol],
         val);
      free(val);
      break;
    }
    case INSERTED: {
      char *val;

      val = XmTextGetString(w);
      for (i=tabinfo->table->colAttr.length[tabinfo->currentCol]-1; i>=0;i--) {
    if (val[i] == ' ') val[i] = '\0';
    else break;
      }
      strcpy(tabinfo->table->rows[tabinfo->currentRow].data[tabinfo->currentCol],
         val);
      free(val);
      break;
    }
    default: break;
  }
}



/* switch to update mode for the current row */
void updateRowCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i;
  XmString t;
  TableInfo *tabinfo;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  for (i=0  i< tabinfo->table->colCount i++)
    XtVaSetValues(tabinfo->field[tabinfo->currentRow][i], XmNeditable, True, NULL);
  XmProcessTraversal(tabinfo->field[tabinfo->currentRow][tabinfo->currentCol], XmTRAVERSE_CURRENT);
  t = XmStringCreateLocalized("Row Update.");
  XtVaSetValues(tabinfo->tMessage, XmNlabelString, t, NULL);
  XmStringFree(t);
}
```

Figure 180 (Part 10 of 20). Database Frames Implementation File dbFrames.c

```
/* insert row */
void insertRowCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int n, i, len;
  XmString t;
  Widget tRowsRC, tScrollRowsW = (Widget) client_data;
  TableInfo *tabinfo;
  Index *index;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  allocateNewRow(tabinfo);
  n = tabinfo->table->rowCount-1;
  tabinfo->currentRow = n; tabinfo->currentCol = 0;
  tabinfo->table->rows[n].mode = INSERTED;
  tabinfo->table->rows[n].changed = 0;
  XtVaGetValues(tScrollRowsW, XmNworkWindow, &tRowsRC, NULL);
  tabinfo->rows[n] = XtVaCreateWidget("iRow",
        xmRowColumnWidgetClass, tRowsRC,
        XmNorientation, XmHORIZONTAL,
        XmNmarginWidth, DEFAULTMARGIN,
        XmNspacing, DEFAULTSPACING,
        NULL);
  for (i=0  i< tabinfo->table->colCount i++) {
    len = min(DEFAULTFIELDLEN, tabinfo->table->colAttr.length[i]);
    len = max(len, strlen(tabinfo->table->colAttr.name[i]));
    tabinfo->field[n][i] = XtVaCreateManagedWidget("ijField",
        xmTextWidgetClass, tabinfo->rows[n],
        XmNcolumns, len,
        XmNmarginWidth, DEFTXTMARGIN,
        XmNfontList, fontList14,
        XmNmaxLength, tabinfo->table->colAttr.length[i],
    XmNuserData, tabinfo,
        NULL);
    XtAddCallback(tabinfo->field[n][i], XmNlosingFocusCallback, exitFieldCB, NULL);
    index = (Index *)calloc(1, sizeof(Index));
    index->row = n; index->col = i;
    XtAddCallback(tabinfo->field[n][i], XmNfocusCallback, entryFieldCB, index);
    XtAddCallback(tabinfo->field[n][i], XmNvalueChangedCallback, valueChangedCB, NULL);
  }
  XtManageChild(tabinfo->rows[n]);

  XmScrollVisible(tScrollRowsW, tabinfo->field[n][0], 10, 10);
  XmProcessTraversal(tabinfo->field[n][0], XmTRAVERSE_CURRENT);
  t = XmStringCreateLocalized("Insert Row.");
  XtVaSetValues(tabinfo->tMessage, XmNlabelString, t, NULL);
  XmStringFree(t);
}
```

Figure 180 (Part 11 of 20). Database Frames Implementation File dbFrames.c

```
/* delete row */
void deleteRowCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  Widget tScrollRowsW = (Widget) client_data;
  int row, i, j;
  TableInfo *tabinfo;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  row = tabinfo->currentRow;
  XtUnmanageChild(XtParent(tabinfo->field[tabinfo->currentRow][tabinfo->currentCol]));

  if (tabinfo->table->rows[tabinfo->currentRow].mode == INSERTED)
    tabinfo->table->rows[tabinfo->currentRow].mode = NONE;
  else
    tabinfo->table->rows[tabinfo->currentRow].mode = DELETED;
  for (i=row+1;i < tabinfo->table->rowCount; i++)
    if ((tabinfo->table->rows[i].mode != DELETED) &&
    (tabinfo->table->rows[i].mode != NONE)) {
      XmScrollVisible(tScrollRowsW, tabinfo->field[i][tabinfo->currentCol], 10, 10);
      XmProcessTraversal(tabinfo->field[i][tabinfo->currentCol], XmTRAVERSE_CURRENT);
      return;
    }
  for (i=row-1;i >= 0; i--)
    if ((tabinfo->table->rows[i].mode != DELETED) &&
        (tabinfo->table->rows[i].mode != NONE)) {
      XmScrollVisible(tScrollRowsW, tabinfo->field[i][tabinfo->currentCol], 10, 10);
      XmProcessTraversal(tabinfo->field[i][tabinfo->currentCol], XmTRAVERSE_CURRENT);
      return;
    }
}




/* commit changes */
void commitCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i, j, n;
  Widget parentW, tScrollRowsW = (Widget) client_data;
  Message msg = NULL;
  XmString t;
  char **setColumns = NULL, **values = NULL;
  TableInfo *tabinfo;
  char *tabname;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  tabname = tabinfo->table->name;
```

Figure 180 (Part 12 of 20). Database Frames Implementation File dbFrames.c

```
if (tabinfo->table->rowCount == 0) return;
parentW = XtParent(tabinfo->rows[tabinfo->currentRow]);
XtUnmanageChild(tScrollRowsW);
for (i=0 i<tabinfo->table->rowCount i++) {
  switch (tabinfo->table->rows[i].mode) {
    case DELETED: {
  char *selection = selectThisRow(tabinfo->table->rows[i],
                        tabinfo->table->colCount,
                        tabinfo->table->colAttr.name);
  msg =  dbDelete(tabinfo->table->name, selection);
  free(selection);
  break;
    }
    case INSERTED:  {
  msg =  dbInsert(tabinfo->table->name,
             tabinfo->table->colCount,
             tabinfo->table->colAttr.name,
             tabinfo->table->rows[i].data);
  break;
    }
    case UPDATED: {
      char *selection = selectThisRow(tabinfo->table->rows[i],
                                      tabinfo->table->colCount,
                                      tabinfo->table->colAttr.name);
  n = 0;
  for (j=0;j<tabinfo->table->colCount;j++)
    if (tabinfo->table->rows[i].chData[j] != NULL) {
      setColumns = (char **) realloc(setColumns, (n+1)*sizeof(char *));
      values = (char **) realloc(values, (n+1)*sizeof(char *));
      setColumns[n] = tabinfo->table->colAttr.name[j];
      values[n] = tabinfo->table->rows[i].chData[j];
      n++;
    }
  msg =  dbUpdate(tabinfo->table->name,
             n,
             setColumns,
             values,
             selection);
  free(selection);
  free(setColumns);
  free(values);
  break;
    }
    default: break;
  }
  if (strlen(msg) != 0) break;
  free(msg);
  msg = NULL;
}
if (msg == NULL) {
```

Figure 180 (Part 13 of 20). Database Frames Implementation File dbFrames.c

```
    msg = dbCheckPoint();
    if (strlen(msg) != 0)
      messageBox(w, msg, 1);
    clearTable(tabinfo,1);
    tabinfo->table = dbSelect(tabname, 0, NULL, NULL);
    showRows(parentW, tabinfo);
    msg = (Message) calloc(1, strlen(tabinfo->table->message) + 20);
    sprintf(msg, "Commit Done.\n%s", tabinfo->table->message);
    t = XmStringCreateLtoR(msg, " ");
    XtVaSetValues(tabinfo->tMessage, XmNlabelString, t, NULL);
    free(msg);
    XmStringFree(t);
    XtManageChild(tScrollRowsW);
    if (tabinfo->table->rowCount > 0) {
      tabinfo->currentRow = 0; tabinfo->currentCol = 0;
      XmScrollVisible(tScrollRowsW, tabinfo->field[0][0], 10, 10);
      XmProcessTraversal(tabinfo->field[0][0], XmTRAVERSE_CURRENT);
    }
  }
  else {
    t = XmStringCreateLtoR(msg, " ");
    XtVaSetValues(tabinfo->tMessage, XmNlabelString, t, NULL);
    free(msg);
    XmStringFree(t);
    XtManageChild(tScrollRowsW);
    XmScrollVisible(tScrollRowsW,
    tabinfo->field[tabinfo->currentRow][tabinfo->currentCol], 10, 10);
    XmProcessTraversal(tabinfo->field[tabinfo->currentRow][tabinfo->currentCol], XmTRAVERSE_CURRENT);
  }
}



/* refresh frame with current database contents */
void refreshCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  XmString t;
  Widget parentW, tScrollRowsW = (Widget) client_data;
  Message msg;
  TableInfo *tabinfo;
  char *tabname;

  XtVaGetValues(w, XmNuserData, &tabinfo, NULL);
  tabname = tabinfo->table->name;
  if (tabinfo->table->rowCount == 0) return;
  parentW = XtParent(tabinfo->rows[tabinfo->currentRow]);
  XtUnmanageChild(tScrollRowsW);
  clearTable(tabinfo,1);
  tabinfo->table = dbSelect(tabname, 0, NULL, NULL);
  showRows(parentW, tabinfo);
```

Figure 180 (Part 14 of 20). Database Frames Implementation File dbFrames.c

```
  msg = (Message) calloc(1, strlen(tabinfo->table->message) + 20);
  sprintf(msg, "Refresh Done.\n%s", tabinfo->table->message);
  t = XmStringCreateLtoR(msg, " ");
  XtVaSetValues(tabinfo->tMessage, XmNlabelString, t, NULL);
  free(msg);
  XmStringFree(t);
  XtManageChild(tScrollRowsW);
  if (tabinfo->table->rowCount > 0) {
    tabinfo->currentRow = 0; tabinfo->currentCol = 0;
    XmScrollVisible(tScrollRowsW, tabinfo->field[0][0], 10, 10);
    XmProcessTraversal(tabinfo->field[0][0], XmTRAVERSE_CURRENT);
  }
}




/* quit message box */
void okMsgCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int fatal = (int)client_data;

  XtDestroyWidget(w);
  if (fatal)
    exit(1);
}




/* insert new row from insertFrame */
void insertCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i, j;
  Message msg;
  Boolean notFixed;
  char *val;
  InsertInfo *ininfo;

  XtVaGetValues(w, XmNuserData, &ininfo, NULL);
  for (i=0 i<(*ininfo->tables)->colCount i++) {
    XtVaGetValues(ininfo->fields[i], XmNeditable, &notFixed, NULL);
    if (notFixed) {
      val = (char *) calloc((*ininfo->tables)->colAttr.length[i]+1, sizeof(char));
      val = XmTextGetString(ininfo->fields[i]);
      for (j=(*ininfo->tables)->colAttr.length[i]-1; j>=0;j--) {
        if (val[j] == ' ') val[j] = '\0';
        else break;
      }
      (*ininfo->tables)->rows[0].data[i] =
```

Figure 180 (Part 15 of 20). Database Frames Implementation File dbFrames.c

```
    (char *) calloc((*ininfo->tables)->colAttr.length[i] + 1, sizeof(char));
      strcpy((*ininfo->tables)->rows[0].data[i], val);
      free(val);
    }
  }

  msg = dbInsert((*ininfo->tables)->name, (*ininfo->tables)->colCount,
           (*ininfo->tables)->colAttr.name, (*ininfo->tables)->rows[0].data);
  if (strlen(msg) != 0) {
    messageBox(w, msg, 0);
    return;
  }
  msg = dbCheckPoint();
  if (strlen(msg) != 0)
    messageBox(w, msg, 1);
  for (i=0 i<(*ininfo->tables)->colCount i++) {
    XtVaGetValues(ininfo->fields[i], XmNeditable, &notFixed, NULL);
    if (notFixed) {
      free((*ininfo->tables)->rows[0].data[i]);
      XmTextSetString(ininfo->fields[i], NULL);
    }
  }
}



/* cancel insertion from insertFrame */
void cancelInsertCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  int i;
  Boolean notFixed;
  InsertInfo *ininfo;

  XtVaGetValues(w, XmNuserData, &ininfo, NULL);
  for (i=0 i<(*ininfo->tables)->colCount i++) {
    XtVaGetValues(ininfo->fields[i], XmNeditable, &notFixed, NULL);
    if (notFixed)
      XmTextSetString(ininfo->fields[i], NULL);
  }
}



/* go to next table insert */
void quitInsertCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  Widget parent = (Widget) client_data;
  Table *tab;
  Widget top;
  InsertInfo *ininfo;
```

Figure 180 (Part 16 of 20). Database Frames Implementation File dbFrames.c

```
  XtVaGetValues(w, XmNuserData, &ininfo, NULL);
  top = ininfo->shell;
  XtDestroyWidget(top);
  free(ininfo->fields);
  tab = *ininfo->tables;
  freeTable(tab);
  if (ininfo->tabCount <= 1) {
    free(ininfo->tables);
  }
  else
    insertFrame(parent, ininfo->tabCount-1, &ininfo->tables[1]);
}





/*-------------------- Help Procedures -----------------------------------*/


/* show table rows */
void showRows(Widget parentW, TableInfo *tabinfo)
{
  Widget *tRows = (Widget *) calloc(tabinfo->table->rowCount, sizeof(Widget));
  Widget **tField = (Widget **) calloc(tabinfo->table->rowCount, sizeof(Widget *));
  int i, j, len;
  Index *index;

  for(i=0;i<tabinfo->table->rowCount; i++)
    tField[i] = (Widget *) calloc(tabinfo->table->colCount, sizeof(Widget));

  for (i=0 i<tabinfo->table->rowCount i++) {
    tRows[i] = XtVaCreateWidget("iRow",
        xmRowColumnWidgetClass, parentW,
        XmNorientation, XmHORIZONTAL,
        XmNmarginWidth, DEFAULTMARGIN,
        XmNspacing, DEFAULTSPACING,
        NULL);

    for(j=0; j<tabinfo->table->colCount;j++) {
      len = min(DEFAULTFIELDLEN, tabinfo->table->colAttr.length[j]);
      len = max(len, strlen(tabinfo->table->colAttr.name[j]));
      tField[i][j] = XtVaCreateManagedWidget("ijField",
        xmTextWidgetClass, tRows[i],
        XmNvalue, tabinfo->table->rows[i].data[j],
        XmNeditable, False,
        XmNcolumns, len,
        XmNmarginWidth, DEFTXTMARGIN,
```

Figure 180 (Part 17 of 20). Database Frames Implementation File dbFrames.c

```
    XmNhighlightThickness, HIGHLIGHT,
        XmNfontList, fontList14,
        XmNmaxLength, tabinfo->table->colAttr.length[j],
        XmNuserData, tabinfo,
        NULL);
      XtAddCallback(tField[i][j], XmNlosingFocusCallback, exitFieldCB, NULL);
      index = (Index *)calloc(1, sizeof(Index));
      index->row = i; index->col = j;
      XtAddCallback(tField[i][j], XmNfocusCallback, entryFieldCB, index);
      XtAddCallback(tField[i][j], XmNvalueChangedCallback, valueChangedCB, NULL)
;
    }

    if ((tabinfo->table->rows[i].mode != DELETED) &&
        (tabinfo->table->rows[i].mode != NONE))
      XtManageChild(tRows[i]);
  }
  tabinfo->field = tField;
  tabinfo->rows = tRows;
}


/* build columns for insert frame */
void buildCols(Widget parentW, InsertInfo *ininfo)
{
  int i;
  Widget *colNames = (Widget *) calloc((*ininfo->tables)->colCount, sizeof(Widget));
  Widget *colValues = (Widget *) calloc((*ininfo->tables)->colCount, sizeof(Widget));
  Widget asterisk, iFieldForm, iFieldRC;
  char ast[2];

  iFieldRC = XtVaCreateWidget("iFieldRC", xmRowColumnWidgetClass, parentW, NULL);
  for (i=0 i<(*ininfo->tables)->colCount i++) {
    iFieldForm = XtVaCreateWidget("iFieldForm", xmFormWidgetClass, iFieldRC, NULL);
    colNames[i] = XtVaCreateManagedWidget((*ininfo->tables)->colAttr.name[i],
        xmLabelWidgetClass, iFieldForm,
        XmNfontList, fontList14,
    XmNtopAttachment, XmATTACH_FORM,
    XmNbottomAttachment, XmATTACH_FORM,
    XmNleftAttachment, XmATTACH_FORM,
    XmNrightAttachment, XmATTACH_POSITION,
    XmNrightPosition, 45,
    XmNalignment, XmALIGNMENT_END,
        NULL);
    colValues[i] = XtVaCreateManagedWidget("iValue",
        xmTextWidgetClass, iFieldForm,
        XmNvalue, (*ininfo->tables)->rows[0].data[i],
        XmNcolumns, min(DEFAULTFIELDLEN, (*ininfo->tables)->colAttr.length[i]),
        XmNfontList, fontList14,
        XmNmaxLength, (*ininfo->tables)->colAttr.length[i],
```

Figure 180 (Part 18 of 20). Database Frames Implementation File dbFrames.c

```
        XmNtopAttachment, XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNleftAttachment, XmATTACH_POSITION,
        XmNleftPosition, 50,
        XmNalignment, XmALIGNMENT_BEGINNING,
        NULL);
    if (strlen((*ininfo->tables)->rows[0].data[i]) > 0)
      XtVaSetValues(colValues[i], XmNeditable, False, NULL);
    if ((*ininfo->tables)->colAttr.isNullable[i])
      strcpy(ast, " ");
    else
      strcpy(ast, "*");
    asterisk = XtVaCreateManagedWidget(ast,
        xmLabelWidgetClass, iFieldForm,
        XmNfontList, fontList14,
        XmNtopAttachment, XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNleftAttachment, XmATTACH_POSITION,
        XmNleftPosition, 97,
        XmNrightAttachment, XmATTACH_FORM,
        XmNalignment, XmALIGNMENT_BEGINNING,
        NULL);
    XtManageChild(iFieldForm);
  }
  XtManageChild(iFieldRC);
  ininfo->fields = colValues;
}



/* allocate space for an additional row to the global tabinfo */
void allocateNewRow(TableInfo *tabinfo)
{
  int i,n;

  n = tabinfo->table->rowCount++;
  tabinfo->table->rows = (Row *)realloc(tabinfo->table->rows, (n+1)*sizeof(Row));
  tabinfo->table->rows[n].data = (char **) calloc(tabinfo->table->colCount, sizeof(char *));
  tabinfo->table->rows[n].chData = (char **) calloc(tabinfo->table->colCount, sizeof(char *));
  tabinfo->field = (Widget **) realloc(tabinfo->field,(n+1)*sizeof(Widget *));
  tabinfo->rows = (Widget *) realloc(tabinfo->rows, (n+1)*sizeof(Widget));
  tabinfo->field[n] = (Widget *) calloc(tabinfo->table->colCount, sizeof(Widget));
  for (i=0  i< tabinfo->table->colCount i++)
    tabinfo->table->rows[n].data[i] =
        (char *) calloc(tabinfo->table->colAttr.length[i]+1, sizeof(char));
}



/* deallocate space of tabinfo */
```

Figure 180 (Part 19 of 20). Database Frames Implementation File dbFrames.c

```
void clearTable(TableInfo *tabinfo, Boolean unmanage)
{
  int i;

  for (i=0  i<tabinfo->table->rowCount i++) {
    if (unmanage)
      XtUnmanageChild(tabinfo->rows[i]);
    free(tabinfo->field[i]);
  }
  free(tabinfo->rows);
  freeTable(tabinfo->table);
}
```

Figure 180 (Part 20 of 20). Database Frames Implementation File dbFrames.c

C.4 Main Program

```
/***************************************************************************
**
** File:    uicfgdb.c
** System:  User Interface to NetView DM/6000 Configuration Database
** Purpose: Main Procedure
** Author:  Plamen Kiradjiev
** Date:    10/09/1995
**
***************************************************************************/


/*------------------------ Includes ---------------------------------------*/
#include "dbFrames.h"
#include <Xm/Label.h>
#include <Xm/Text.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <Xm/ToggleB.h>
#include <Xm/Separator.h>
#include <Xm/Form.h>
#include <Xm/MessageB.h>
#include <Xm/SelectioB.h>



/*----------------------- Constants --------------------------------------*/
#define DBNAME "NVDM_CFG"
#define DBOWNER "DBMSADM"
#define MAXNAMELEN 40
/* tables */
#define NODETAB "NVDM_NODE"
#define SERVERTAB "NVDM_SERVERS"
#define QUEUETAB "NVDM_QUEUES"
#define GROUPTAB "NVDM_GROUPS"
#define USERTAB "NVDM_USERS"
#define STATICTAB "NVDM_CFG_STATIC"



/*--------------- information relevant for inserting ----------------------*/
#define NODENAME "NODE_NAME"
#define STABCOUNT 5
#define CTABCOUNT 2

char *serverDependentTabs[] = {NODETAB, SERVERTAB, QUEUETAB, GROUPTAB, USERTAB};
char *clientDependentTabs[] = {NODETAB, USERTAB};
```

Figure 181 (Part 1 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
/*---------------------- Type Definitions ---------------------------*/
typedef struct NodeInfo{
   char *name;
   Boolean isServer;
  }NodeInfo;


/*----------------------- Help Functions ---------------------------------*/
/* determine full table name */
char *fullname(char *tab)
{
  char *fn = (char *) calloc(MAXNAMELEN+1, sizeof(char));

  sprintf(fn, "%s.%s", DBOWNER, tab);
  return fn;
}



/*----------------------- Globals -------------------------------------*/
static XmFontList fontList14, fontList18;




/*------------------------ Main -----------------------------------------*/
main(int argc, char *argv[])
{
  Message msg;
  XtAppContext app;
  Widget toplevel;
  Display *display;
  XFontStruct *font;
  extern void mainFrame(Widget top);
  extern void messageBox(Widget top, String msg, int fatal);

/* setting default language environment */
  XtSetLanguageProc(NULL, NULL, NULL);

/* initializing Xt */
  toplevel = XtVaAppInitialize(&app, "UICFGDB", NULL, 0,
   &argc, argv, NULL, NULL);

/* set used font lists */
  display = XtDisplay(toplevel);
  font = XLoadQueryFont(display, "-*-courier-bold-r-*--14-*");
  fontList14 = XmFontListCreate(font, " ");
  font = XLoadQueryFont(display, "-*-courier-bold-r-*--18-*");
  fontList18 = XmFontListCreate(font, " ");
```
Figure 181 (Part 2 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
/* connect to configuration database */
  msg=dbConnect(DBNAME);
  if (strlen(msg) > 0)
    messageBox(toplevel, msg, 1);
  else
/* call application main window */
    mainFrame(toplevel);

/* enter main application loop */
  XtAppMainLoop(app);

}



/* ------------------------ Frames ------------------------------------------*/


/* Application main window */
void mainFrame(Widget top)
{
  Widget mainRowCol, mainLabel, mainSep, mainButtonRC;
  Widget mainGlobalsBtn, mainInsertBtn, exitBtn;
  Widget mainList;
  XmString xstr;
  char *creator_name[] = {"CREATOR", "NAME"};
  int i=0;
  Arg args[5];
  Table *table;
  char tname[MAXNAMELEN+1];
  extern void listSelectCB(Widget w, XtPointer client_data, XtPointer call_data);
  extern void exitAppCB(Widget w, XtPointer client_data, XtPointer call_data);
  extern void insertNodeCB(Widget w, XtPointer client_data, XtPointer call_data);
  extern void updateGlobalsCB(Widget w, XtPointer client_data, XtPointer call_data);

  mainRowCol = XtVaCreateWidget("mainRowCol",
   xmRowColumnWidgetClass, top,
   XmNentryAlignment, XmALIGNMENT_CENTER,
   NULL);

  xstr = XmStringCreateLocalized(
     "User Interface to NetView DM/6000 Configuration Database");
  mainLabel = XtVaCreateManagedWidget("mainLabel",
   xmLabelWidgetClass, mainRowCol,
   XmNlabelString, xstr,
   XmNfontList, fontList18,
   XmNmarginTop, 20,
   XmNmarginBottom, 30,
   NULL);
```

Figure 181 (Part 3 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
  XmStringFree(xstr);

  mainSep = XtVaCreateManagedWidget("mainSep",
   xmSeparatorWidgetClass, mainRowCol, NULL);

  XtSetArg(args[i], XmNvisibleItemCount, 6); i++;
  XtSetArg(args[i], XmNfontList, fontList14); i++;
  XtSetArg(args[i], XmNlistMarginWidth, 100); i++;
  mainList = (Widget)XmCreateScrolledList(mainRowCol, "mainList", args, i)

  table = dbSelect("sysibm.systables", 2, creator_name,
           dbOrderString(2, creator_name));
  if (table->colCount == 0)
    messageBox(top, table->message, 1);
  for (i=0  i<table->rowCount  i++) {
    sprintf(tname, "%s.%s", table->rows[i].data[0],
table->rows[i].data[1]);
    xstr = XmStringCreateLtoR(tname, " ");
    XmListAddItemUnselected (mainList, xstr, i+1);
    XmStringFree(xstr);
  }

  XtManageChild(mainList);

  mainSep = XtVaCreateManagedWidget("mainSep",
   xmSeparatorWidgetClass, mainRowCol, NULL);

  mainButtonRC = XtVaCreateWidget("mainButtonRC",
   xmRowColumnWidgetClass, mainRowCol,
   XmNentryAlignment, XmALIGNMENT_CENTER,
   XmNorientation, XmHORIZONTAL,
   XmNpacking, XmPACK_COLUMN,
   NULL);

  mainGlobalsBtn = XtVaCreateManagedWidget("Update Network Globals",
   xmPushButtonWidgetClass, mainButtonRC,
   XmNfontList, fontList14,
   NULL);

  mainInsertBtn = XtVaCreateManagedWidget("Insert New Node",
   xmPushButtonWidgetClass, mainButtonRC,
   XmNfontList, fontList14,
   NULL);

  exitBtn = XtVaCreateManagedWidget("Exit",
   xmPushButtonWidgetClass, mainButtonRC,
   XmNfontList, fontList14,
   NULL);

  XtAddCallback(mainList, XmNdefaultActionCallback, listSelectCB, NULL);
  XtAddCallback(exitBtn, XmNactivateCallback, exitAppCB, NULL);
```

Figure 181 (Part 4 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
  XtAddCallback(mainInsertBtn, XmNactivateCallback, insertNodeCB, NULL);
  XtAddCallback(mainGlobalsBtn, XmNactivateCallback, updateGlobalsCB, NULL);

/* display widgets */
  XtManageChild(mainButtonRC);
  XtManageChild(mainRowCol);
  XtRealizeWidget(top);

}




/* question dialog for node name */
void nodeNameQuestion(Widget w)
{
  Table *table;
  NodeInfo *node=(NodeInfo *)calloc(1, sizeof(NodeInfo));
  char *node_name[] = {NODENAME};
  Widget qDialog, qServerToggle, qText, qLabel, qRC;
  XmString t;
  Arg args[5];
  int n=0;
  extern void readNameCB(Widget w, XtPointer client_data, XtPointer call_data);
  extern void serverToggleCB(Widget w, XtPointer client_data, XtPointer call_data);
  extern void nameChangeCB(Widget w, XtPointer client_data, XtPointer call_data);

  table = dbSelect(fullname(NODETAB), 1, node_name, dbAlwaysFalse());
  if ((table->colCount == 0) && (strlen(table->message) > 0))
    messageBox(w, table->message, 0);

  node->name = (char *) calloc(table->colAttr.length[0] + 1, sizeof(char));
  XtSetArg(args[n], XmNbuttonFontList, fontList14); n++;
  XtSetArg(args[n], XmNautoUnmanage, False); n++;
  qDialog = XmCreatePromptDialog(w, "node_name", args, n);

  XtAddCallback(qDialog, XmNokCallback, readNameCB, node);
  XtAddCallback(qDialog, XmNcancelCallback, XtDestroyWidget, NULL);

  XtUnmanageChild(XmSelectionBoxGetChild(qDialog, XmDIALOG_HELP_BUTTON));
  XtUnmanageChild(XmSelectionBoxGetChild(qDialog, XmDIALOG_SELECTION_LABEL));
  XtUnmanageChild(XmSelectionBoxGetChild(qDialog, XmDIALOG_TEXT));

  qRC =  XtVaCreateWidget("qRC",
   xmRowColumnWidgetClass, qDialog, NULL);

  t = XmStringCreateLocalized("Enter Node Name:");
  qLabel = XtVaCreateManagedWidget("qLabel",
   xmLabelWidgetClass, qRC,
   XmNlabelString, t,
```

Figure 181 (Part 5 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
    XmNfontList, fontList14,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
  XmStringFree(t);

  qText = XtVaCreateManagedWidget("qText",
    xmTextWidgetClass, qRC,
    XmNfontList, fontList14,
         XmNmaxLength, table->colAttr.length[0],
    NULL);

  t = XmStringCreateLocalized("is Server?");
  qServerToggle = XtVaCreateManagedWidget("qServerToggle",
    xmToggleButtonWidgetClass, qRC,
    XmNalignment, XmALIGNMENT_BEGINNING,
    XmNfontList, fontList14,
    XmNlabelString, t,
    NULL);
  XmStringFree(t);

  XtAddCallback(qText, XmNvalueChangedCallback, nameChangeCB, node);
  XtAddCallback(qServerToggle, XmNvalueChangedCallback, serverToggleCB, node);

  XtManageChild(qRC);
  XtManageChild(qDialog);

  XmProcessTraversal(qText, XmTRAVERSE_CURRENT);

  XtPopup(XtParent(qDialog), XtGrabNone);
}





/* ----------------------- Callbacks ----------------------------------------*/


/* exit application */
void exitAppCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  Message msg;

/* disconnect from the configuration database */
  msg = dbDisconnect();
  if (strlen(msg) > 0)
    printf("%s\n",msg);

  exit(0);
```
Figure 181 (Part 6 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
}



/* call question dialog frame for node name */
void insertNodeCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  nodeNameQuestion(w);
}



/* action after altering the server toggle button */
void nameChangeCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  NodeInfo *node = (NodeInfo *) client_data;

  node->name = strcpy(node->name, XmTextGetString(w));
}



/* action after altering the server toggle button */
void serverToggleCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  NodeInfo *node = (NodeInfo *) client_data;

  node->isServer = XmToggleButtonGetState(w);
}



/* read node name from node name question dialog */
void readNameCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  NodeInfo *node = (NodeInfo *) client_data;
  int i, j, fixIndex;
  int cnt = (node->isServer)?STABCOUNT:CTABCOUNT;
  Table **tables = (Table **) calloc(cnt,sizeof(Table *));
  char **tabnames;
  char msg[60];
  Widget parent = XtParent(w);

  if (strlen(node->name) == 0) return;
  XtUnmanageChild(w);
  if (node->isServer)
    tabnames = serverDependentTabs;
  else
    tabnames = clientDependentTabs;
```

Figure 181 (Part 7 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
  for (i = 0; i < cnt; i++) {
    tables[i] = dbSelect(fullname(tabnames[i]), 0, NULL, dbAlwaysFalse());
    if (tables[i]->colCount == 0) {
      messageBox(parent, tables[i]->message, 0);
      free(tables);
      return;
    }
    tables[i]->rows = (Row *) calloc(1, sizeof(Row));
    tables[i]->rows[0].data = (char **) calloc(tables[i]->colCount, sizeof(char *));
    fixIndex = -1;
    for (j=0  j<tables[i]->colCount  j++)
      if (!strcmp(NODENAME, tables[i]->colAttr.name[j])) {
   fixIndex = j;
        break;
      }
    if (fixIndex < 0) {
      sprintf(msg, "No column with name %s found in table %s" ,
          NODENAME, tables[i]->name);
      messageBox(parent, msg, 0);
      free(tables);
      return;
    }
    tables[i]->rows[0].data[fixIndex] =
   (char *) calloc(tables[i]->colAttr.length[fixIndex]+1, sizeof(char));
    strcpy(tables[i]->rows[0].data[fixIndex], node->name);
  }
  insertFrame(parent, cnt, &tables[0]);
}



/* update global network attributes */
void updateGlobalsCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  Table *table;

  table = dbSelect(fullname(STATICTAB), 0, NULL, NULL);
  if (table->colCount == 0)
    messageBox(w, table->message, 0);
  else
    tableFrame(w, table);
}


/* select table from list */
void listSelectCB(Widget w, XtPointer client_data, XtPointer call_data)
{
  XmString *strlist;
  char *tname = (char *) calloc(MAXNAMELEN + 1, sizeof(char));
  XmListCallbackStruct *cbs = (XmListCallbackStruct *) call_data;
```

Figure 181 (Part 8 of 9). Graphical User Interface Main Program (uicfgdb.c)

```
  Table *table;

  XmStringGetLtoR(cbs->item, " ", &tname);
  table = dbSelect(tname, 0, NULL, NULL);
  if (table->colCount == 0)
    messageBox(w, table->message, 0);
  else
    tableFrame(w, table);
}
```

Figure 181 (Part 9 of 9). Graphical User Interface Main Program (uicfgdb.c)

Index

Special Characters

A

B

D

E

F

G

H

I

N

T

U

V

X

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
Software Distribution for AIX:
A Solution for Installation and Configuration of
Pristine AIX Environments
February 1996

Publication No. SG24-4508-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

____

____

What other topics would you like to see in this Bulletin?

____

____

What other Technical Bulletins would you like to see published?

____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

______________________________ ______________________________
Name Address

______________________________ ______________________________
Company or Organization

______________________________ ______________________________
Phone No.

ITSO Technical Bulletin Evaluation **RED000**
SG24-4508-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 985, Building 657
P.O. BOX 12195
RESEARCH TRIANGLE PARK NC
USA 27709-2195

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.

SG24-4508-00