Bull

AIX 4.3 Network Information Services (NIS and NIS+) Guide

AIX

ORDER REFERENCE
86 A2 22KX 00

Bull

AIX 4.3 Network Information Services (NIS and NIS+) Guide

AIX

Software

November 1999

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 22KX 00

Trademarks and Acknowledgements

Year 2000

The product documented in this manual is Year 2000 Ready.

About This Book

This book is for AIX system administrators who maintain Network Information System or NIS+ servers and clients. Familiarity with the Base Operating System and the material covered in *AIX 4.3 System Management Guide: Communications and Networks*, *AIX 4.3 System Management Guide: Operating System and Devices*, and *AIX 4.3 System User's Guide: Communications and Networks* is necessary.

Who Should Use This Book

This book is intended for system administrators who perform system management tasks that involve NIS or NIS+.

How to Use This Book

Chapters in this book provide overview, installation and configuration, maintenance, troubleshooting, and reference information for NIS and for NIS+. Procedural information is located near the relevant conceptual information for easier access.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, keywords, files, directories, and other items whose names are predefined by the system.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

AIX Support for the X/Open UNIX95 Specification

The AIX operating system is designed to support the X/Open UNIX95 Specification for portability of UNIX–based operating systems. Many new interfaces, and some current ones, have been added or enhanced to meet this specification. AIX 4.3 is even more open and portable for applications.

At the same time, compatibility with previous AIX releases is preserved. This is accomplished by the creation of a new environment variable that can be used to set the system environment on a per–system, per–user, or per–process basis.

To determine the proper way to develop a UNIX95–portable application, you may need to refer to the X/Open UNIX95 Specification, which can be obtained on a CD–ROM by ordering the printed copy of *AIX Commands Reference*, order number 86 A2 38JX to 86 A2 43JX, or by ordering *Go Solo: How to Implement and Go Solo with the Single Unix Specification*, a book which includes the X/Open UNIX95 Specification Specification on a CD–ROM.

Related Publications

The following book contains information about or related to communications:

AIX and Related Products Documentation Overview, order number 86 A2 71WE.

AIX 4.3 System Management Guide: Communications and Networks,
order number 86 A2 31JX.

AIX 4.3 System Management Guide: Operating System and Devices,
order number 86 A2 99HX.

AIX 4.3 System User's Guide: Communications and Networks,
order number 86 A2 98HX.

AIX General Programming Concepts: Writing and Debugging Programs,
order number 86 A2 34JX.

AIX Commands Reference, order number 86 A2 38JX to 86 A2 43JX.

AIX 4.3 Installation Guide, order number 86 A2 43GX.

Albitz, Paul and Liu, Cricket, [1992], *DNS and BIND in a Nutshell*, O'Reilly & Associates, Inc., Sebastopol, CA, ISBN 0–56592–010–4.

Stern, Hal, [1991], *Managing NFS and NIS*, O'Reilly & Associates, Inc., Sebastopol, CA, ISBN 0–93–717575–7.

Tanenbaum, Andrew S., *Computer Networks*, Prentice Hall, Englewood Cliffs, NJ, ISBN 0–13–165183–8.

Ordering Publications

You can order publications from your sales representative or from your point of sale.

To order additional copies of this book, use order number 86 A2 22KX.

Use *AIX and Related Products Documentation Overview* for information on related publications and how to obtain them.

Contents

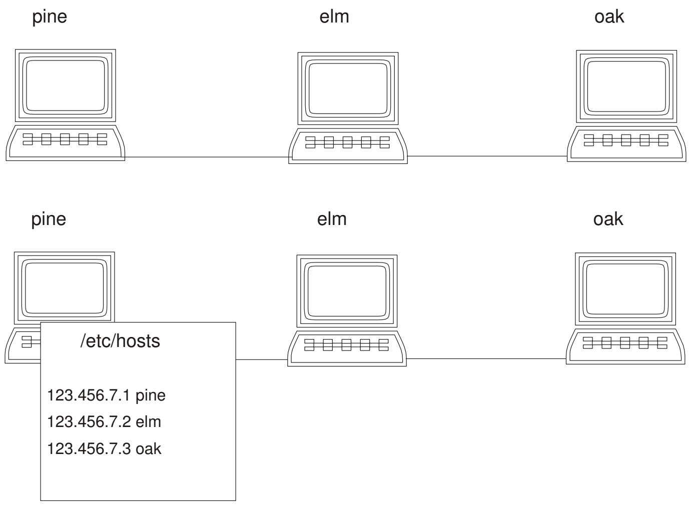
Chapter 1. Introduction to Name Services

This chapter provides a brief overview of name services (also called *network information services*). It introduces the DNS, NIS, and NIS+ name services then concludes with comparisons and interoperability issues.

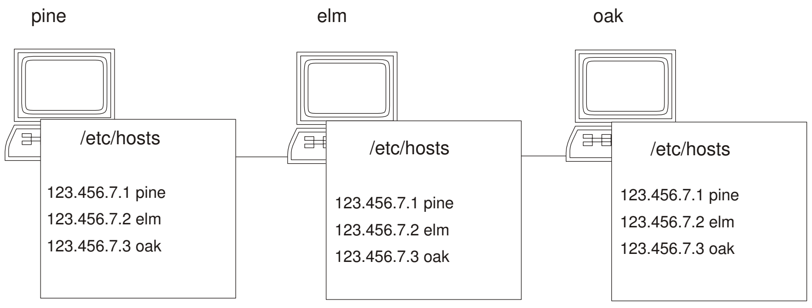
Name Services Overview

Name services store information that allows users, workstations, and applications to communicate across the network. Without a name service, each workstation would have to maintain its own copy of such information as machine addresses, user names, passwords, and network access permissions. With name services, the information may be stored in centrally located files or database tables, which makes it easier to administer large networks.

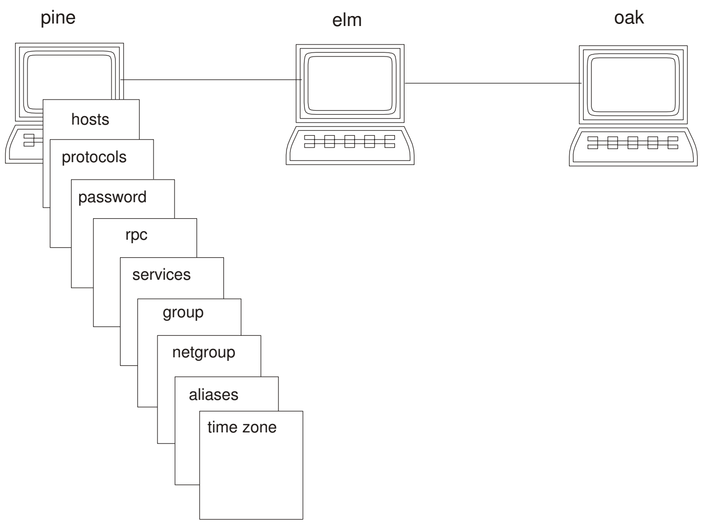
For example, take a simple network of three workstations, **pine**, **elm**, and **oak**. Before **pine** can send a message to either **elm** or **oak**, it must know their network addresses. For this reason, it keeps a file, /**etc**/**hosts**, that stores the network address of every workstation in the network, including itself.



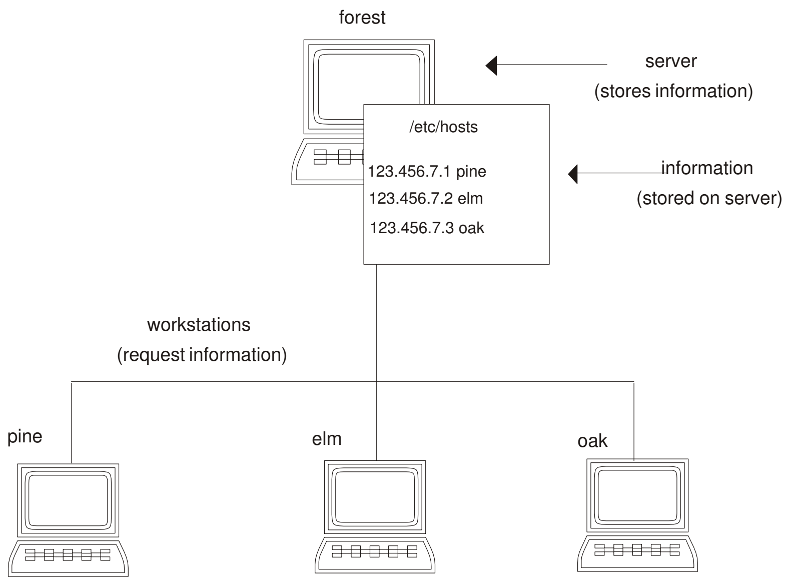
Likewise, for **elm** and **oak** to communicate with **pine** or with each other, they must keep similar files.



Addresses are not the only network information that workstations need to store. They also need to store security information, mail information, information about their Ethernet interfaces, network services, groups of users allowed to use the network, services offered on the network, and so on. As networks offer more services, the list grows. As a result, each workstation may need to keep an entire set of files similar to /**etc**/**hosts**.



As this information changes, administrators must keep it current on every workstation in the network. In a small network, this is simply tedious, but on a medium or large network, the job becomes not only time–consuming but nearly unmanageable. A network information service addresses this problem. It stores network information on servers and provides it to any workstation that asks for it.



The workstations are known as *clients* of the server. Whenever information about the network changes, instead of updating each client's local file, an administrator updates only the information stored by the network information service. This reduces errors, inconsistencies between clients, and the sheer size of the task.

Although the chief purpose of a network information service is to centralize information, another is to simplify network names. A network information service enables workstations to be identified by common names instead of numerical addresses. (This is why these services are sometimes called *name services*.) This makes communication simpler because users do not have to remember and try to enter cumbersome physical addresses, such as `129.44.3.1`. Instead, they can use descriptive names, such as `Sales`, `Lab1`, or `Arnold`.

Names are also more flexible than physical addresses. While physical addresses and networks tend to remain stable, the organizations that use them tend to change structure and nomenclature. A network information service uses software to map these changes to an unchanged physical network.

Domain Name System (DNS) Overview

The Domain Name System (DNS) is the name service provided by the Internet for TCP/IP networks. It was developed so workstations on the network could be identified with common names instead of Internet addresses. DNS performs naming between hosts within your local administrative domain and across domain boundaries.

The collection of networked workstations that use DNS are referred to as the DNS *namespace*. The DNS namespace can be divided into a hierarchy of *domains*. A DNS domain is simply a group of workstations. Each domain is supported by two or more *name servers* (a principal server and one or more secondary servers). Each server implements DNS by running a daemon called **named**.

On the client side, DNS is implemented through a resolver. The resolver's function is to resolve user queries; to do that, it queries a name server, which then returns either the requested information or a referral to another server.

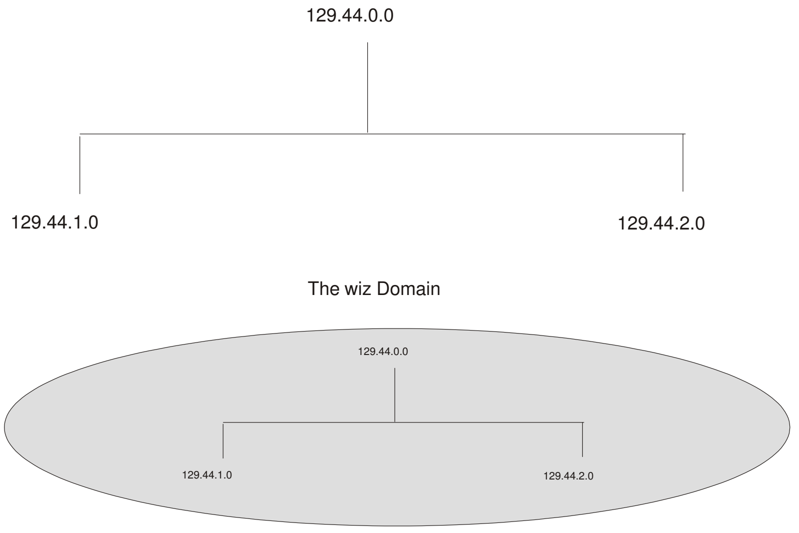
Network Information Service (NIS) Overview

The Network Information Service (NIS) was developed independently of DNS and has a slightly different focus. Whereas DNS focuses on simplifications that use workstation names instead of addresses, NIS focuses on simplifying network administration by providing centralized control over a variety of network information. NIS stores information not only about workstation names and addresses, but also about users, the network itself, and network services. This collection of network information is referred to as the *NIS namespace*.

NIS Architecture

NIS uses a client–server arrangement similar to DNS. Replicated NIS servers provide services to NIS clients. The principal servers are called *master* servers and, for reliability, they have backup or *replica* servers (also referred to as *slave* servers). Both server types use the NIS information retrieval software and both store NIS maps. The method used by NIS+ is described briefly in the NIS+ Overview, on page 1-7.

NIS, like DNS, uses domains to arrange the workstations, users, and networks in its namespace. However, it does not use a domain hierarchy; an NIS namespace is flat. Thus, a hierarchical physical network is arranged by NIS into one domain. See NIS+ Namespace and Structure, for information on NIS+ hierarchy.



An NIS domain cannot be connected directly to the Internet. Organizations that want to use NIS and be connected to the Internet use NIS to manage all local information and DNS for host name resolution. NIS provides special client routines for this purpose (*DNS forwarding*). When a client needs access to any type of information except IP addresses, the request goes to the client's NIS server. When a client needs name resolution, the request goes to the DNS server. From the DNS server, the client has access to the Internet in the usual way.

NIS Maps

NIS stores information in a set of files called *maps*. NIS maps were designed to replace traditional UNIX /**etc** files, as well as other configuration files, so they store much more than names and addresses. As a result, the NIS namespace has a large set of maps, as shown in the NIS Maps table below.

NIS maps are essentially two–column tables. One column is the key and the other column is information about the key. NIS finds information for a client by searching through the keys. Thus, some information is stored in several maps because each map uses a different key.

For example, the names and addresses of workstations are stored in two maps: **hosts.byname** and **hosts.byaddr**. When a server has a workstation's name and needs to find its address, it looks in the **hosts.byname** map. When it has the address and needs to find the name, it looks in the **hosts.byaddr** map.

Table 1. NIS Maps

NIS Map	Description
bootparams	Lists the names of the diskless clients and the location of the files they need during booting.
ethers.byaddr	Lists the Ethernet addresses of workstations and their corresponding names.
ethers.byname	Lists the names of workstations and their corresponding Ethernet addresses.
group.bygid	Provides membership information about groups, using the group ID as the key.
group.byname	Provides membership information about groups, using the group name as the key.
hosts.byaddr	Lists the names and addresses of workstations, using the address as the key.
hosts.byname	Lists the names and addresses of workstations, using the name as the key.
mail.aliases	Lists the mail aliases in the namespace and all the workstations that belong to them.
mail.byaddr	Lists the mail aliases in the namespace, using the address as the key.
netgroup	Contains netgroup information, using the group name as the key.
netgroup.byhost	Contains information about the netgroups in the namespace, using workstation names as the key.
netgroup.byuser	Contains netgroup information, using the user as the key.
netid.byname	Contains the Secure remote procedure call (RPC) netname of workstations and users, along with their user IDs and group IDs.
netmasks.byaddr	Contains network masks used with Internet Protocol (IP) subnetting, using the address as the key.
networks.byaddr	Contains the names and addresses of the networks in the namespace, and their Internet addresses.
networks.byname	Contains the names and addresses of the networks in the namespace, using the names as the key.
passwd.byname	Contains password information, with the username as the key.
passwd.byuid	Contains password information, with the user ID as the key.
protocols.byname	Lists the network protocols used.
protocols.bynumber	Lists the network protocols used but uses their number as the key.
publickey.byname	Contains public and secret keys for Secure RPC.
rpc.bynumber	Lists the known program name and number of RPCs.
services.byname	Lists the available Internet services.
ypservers	Lists the NIS servers in the namespace, along with their IP addresses.

Network Information Service contains detail about the development and use of NIS.

Network Information Service+ (NIS+) Overview

NIS+ expands the network name service provided by NIS. NIS+ enables you to store information about workstation addresses, security information, mail information, Ethernet interfaces, and network services in central locations where all workstations on a network can access it. This configuration of network information is referred to as the NIS+ *namespace*.

The NIS+ namespace is hierarchical and is similar in structure to a traditional UNIX directory file system. The hierarchical structure allows an NIS+ namespace to be configured to conform to the logical hierarchy of an organization. The layout of information in the namespace is unrelated to its physical arrangement. Therefore, an NIS+ namespace can be divided into multiple domains, each of which can be administered autonomously. Clients may have access to information in other domains as well as their own if they have the appropriate permissions.

NIS+ uses a client–server model to store and have access to the information contained in an NIS+ namespace. Each domain is supported by a set of servers. The principal server is called the *master* server and the backup servers *replicas*. The network information is stored in standard NIS+ tables in an internal NIS+ database. Both master and replica servers run NIS+ server software and both maintain copies of NIS+ tables. Changes made to the NIS+ data on the master server are incrementally propagated automatically to the replicas.

NIS+ includes a sophisticated security system to protect the structure of the namespace and its information. It uses authentication and authorization to verify whether a client's request for information should be fulfilled. Authentication determines whether the information requester is a valid user on the network. Authorization determines whether a particular user is allowed to have or modify the information requested.

Differences Between NIS and NIS+

NIS+ differs from NIS in several ways. It has many new features and the terminology for similar concepts is different. The following table gives an overview of the major differences between NIS and NIS+. The sections that follow the table describe key differences more fully.

Table 2. Differences Between NIS and NIS+

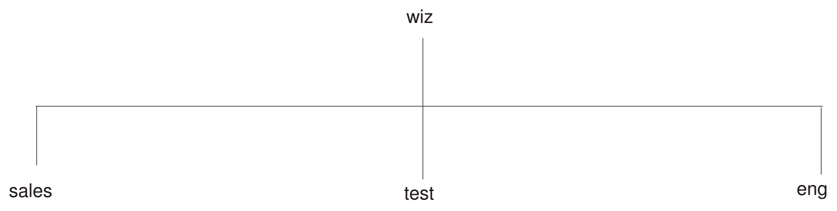
NIS	NIS+
Machine name and user's name can be the same	Machine name and user names must be unique. Furthermore, you cannot have a dot (.) in your machine or user name.
Domains are flat—no hierarchy.	Domains are hierarchical—data stored in different levels in the namespace.
Names and commands are case sensitive.	Names and commands are not case sensitive.
Data is stored in two–column maps.	Data is stored in multicolumn tables.
Uses no authentication.	Uses DES authentication.
An NIS record has a *maximum size* of 1024 bytes. This limitation applies to all NIS map files. For example, a list of users in a group can contain a *maximum* of 1024 characters in single–byte character set file format.	Has no limit.
Provides single choice of network information source.	Client chooses information source: NIS, NIS+, DNS, or local /**etc** files.
Updates are delayed for batch propagation.	Incremental updates are propagated immediately.

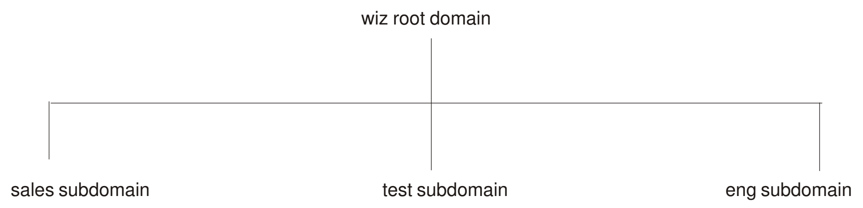
Domain Structure

NIS+ is designed to *replace* NIS, not enhance it. NIS was intended to address the administration requirements of smaller client–server computing networks. Typically, NIS works best in environments with no more than a few hundred clients, a few multipurpose servers, only a few remote sites, and trusted users (since lack of security cannot be a crucial concern).

The size and complexity of modern client–server networks require new, autonomous administration practices. NIS+ was designed to meet the requirements of networks that typically range from 100–10,000 multivendor clients supported by 10–100 specialized servers located in sites throughout the world. Such networks are often connected to several unguarded public networks. In addition, the information they store can change rapidly.

Because more distributed networks require scalability and decentralized administration, the NIS+ namespace was designed with hierarchical domains, like those of DNS. (See figure.) NIS+ domains *may* be flat, but you can also construct hierarchical NIS+ domains. Such hierarchies consist of a root domain with an infinite number of subdomains as shown in the following diagram.





Hierarchical design makes NIS+ useful for a range of network sizes, from small to very large. It also allows the NIS+ service to adapt to the growth of an organization. For example, if a corporation splits itself into two divisions, its NIS+ namespace can be divided into two domains that can be administered autonomously. Just as the Internet delegates downward the administration of domains, NIS+ domains can be administered more or less independently.

Although NIS+ uses a domain hierarchy similar to that of DNS, an NIS+ domain is much more than a DNS domain. A DNS domain only stores name and address information about its clients. An NIS+ domain, on the other hand, is a collection of information about the workstations, users, and network services in a portion of an organization.

Although this division into domains makes administration more autonomous and growth easier to manage, it does not make information harder to access. Clients have the same access to information in other domains as they would have had under one umbrella domain. A domain can even be administered from within another domain.

The NIS+ domain structure is described in detail in NIS+ Namespace and Structure, on page 4-1.

DNS, NIS, and NIS+ Interoperability

NIS+ provides interoperability features designed for upgrading from NIS and for continuing the interaction with DNS originally provided by the NIS service.

To help convert from NIS, NIS+ provides an NIS–compatibility mode and the **nispopulate** command. The NIS–compatibility mode enables an NIS+ server running AIX Version 4.3.3 software to answer requests from NIS clients while continuing to answer requests from NIS+ clients. The **nispopulate** command helps administrators keep NIS maps and NIS+ tables synchronized.

NIS–compatibility mode requires slightly different setup procedures than those used for a standard NIS+ server. Also, NIS–compatibility mode has security implications for tables in the NIS+ namespace.

NIS client machines interact with the NIS+ namespace differently from NIS+ client machines when NIS+ servers are running in NIS–compatibility mode. The differences are:

- NIS client machines cannot follow NIS+ table paths or links, nor can they read operations in other domains.
- NIS client machines can have their unsatisfied host requests forwarded to DNS (called *DNS forwarding*) if you run **rpc.nisd** with the **–Y –B** options, but the NIS+ server will not forward these requests for an NIS+ client. DNS request forwarding for NIS+ client machines is controlled by the **resolv.conf** file configuration in conjunction with the **/etc/irs.conf** file.
- Authorized NIS+ administrators can use the **passwd** command to change users' or administrators' passwords. NIS+ client users can use the **password** command to change their own passwords.
- Even if all the servers on a local subnet no longer respond, the NIS+ client machines can still have their name service calls answered if they can contact any of the replicas of that domain. NIS client machines do not have access to information on the network outside their subnet unless the server names have been set with **ypset** or, for NIS clients only, with **ypinit**.

- NIS client machines cannot be sure that the data they are receiving comes from an authorized NIS server, while authorized NIS+ clients are certain that the data is coming from an authorized NIS+ server.
- Under NIS, if the server is no longer responding, the NIS **ypmatch** call continues to retry this call until the server starts responding and answers the request. The NIS+ API (application programing interface) returns an error message to the application when this situation occurs.

Note: In the AIX Version 4.3.3 and later releases, the NIS–compatibility mode supports DNS forwarding.

Although an NIS+ domain cannot be connected to the Internet directly, the NIS+ client machines can be connected to the Internet using the **/etc/irs.conf** and **/etc/netsvc.conf** configuration files and the **NSORDER** system environment variable.

Server Configuration

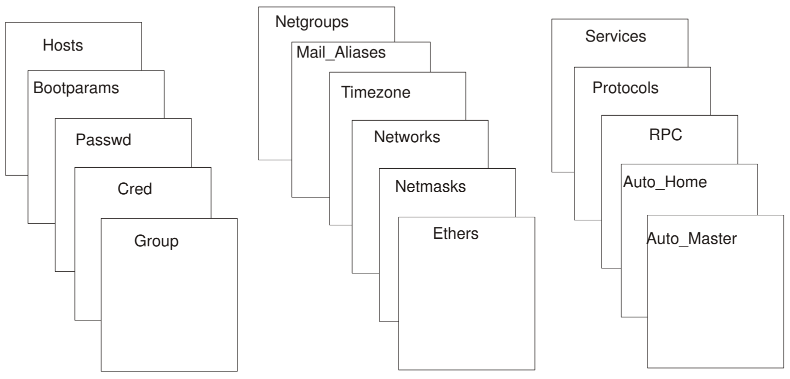
The NIS+ client–server arrangement is similar to those of NIS and DNS in that each domain is supported by a set of servers. The main server is called the *master* server, and the backup servers are called *replicas*. Both master and replica servers run NIS+ server software and both maintain copies of NIS+ tables.

However, NIS+ uses a different update model from the one used by NIS. At the time NIS was developed, it was assumed that most of the information NIS would store would be static. NIS updates are handled manually, and its maps have to be remade and propagated in full every time any information in the map changes.

NIS+, however, accepts incremental updates to the replicas. Changes must still be made to the master database on the master server, but once made, they are automatically propagated to the replica servers. You do not have to ”make” any maps or wait for propagation. Propagation now takes only a matter of minutes.

Information Management

NIS+ stores information in **tables** instead of maps or zone files. NIS+ provides predefined, or **system**, tables, as shown in the figure, each of which stores a different type of information. For instance, the **hosts** table stores information about workstation addresses, while the **passwd** table stores information about users of the network. The master server stores the original tables, and the replicas store copies.



NIS+ tables are not ASCII files, but are tables in the NIS+ relational database. You can view and edit their contents only by using the NIS+ commands.

An NIS+ table can be searched by any searchable column, not just by the first column (sometimes referred to as the *key*). This eliminates the need for duplicate maps, such as the **hosts.byname** and **hosts.byaddr** maps used by NIS. (To know whether a particular column is searchable, run the **niscat –o** command on a table. The command returns a list of the table’s columns and their attributes, one of which is whether a column is searchable.)

Also, the information in NIS+ tables has access controls at three levels: the table level, the entry (row) level, and the column level. NIS+ tables—and the information stored in them—are described in NIS+ Tables and Information, on page 4-18.

NIS maps are located on the server in /**var**/**yp**/*domainname*, whereas NIS+ directories are located in /**var**/**nis**/**data**. The NIS+ tables are contained in the database. The tables' information is loaded into memory as requests are made to the database. Keeping data in memory in the order requested minimizes calls to the disk, thereby improving request–response time.

Another improvement is that NIS+ uses a different updating model from the one used by NIS. At the time NIS was developed, the type of information it stored would change infrequently, NIS was developed with an update model that focused on stability. Its updates are handled manually and, in large organizations, can take more than a day to propagate to all the replicas. Part of the reason for this is the need to remake and propagate an entire map every time any information in the map changes.

NIS+, however, accepts incremental updates. Changes must still be made on the master server, but once made, they are automatically propagated to the replica servers and made available to the entire namespace. You do not have to make any maps or wait for propagation.

Security

The security features of NIS+ protect the information in the namespace and the structure of the namespace itself from unauthorized access. NIS+ security is provided by two means: authentication and authorization. *Authentication* is the process by which an NIS+ server identifies the NIS+ *principal* (a client user or client workstation) that sent a particular request. *Authorization* is the process by which a server identifies the access rights granted to that principal, whether a client machine or client user.

In other words, before users can access anything in the namespace, they must be identified as NIS+ clients and they must have the proper permission to access that information. Furthermore, requests for access to the namespace are only honored if they are made either through NIS+ client library routines or NIS+ administration commands. The NIS+ tables and structures cannot be edited directly.

NIS+ Security Overview

NIS+ protects the structure of the namespace, and the information it stores, by the complementary processes of authorization and authentication.

Authorization	Every component in the namespace specifies the type of operation it will accept and from whom.
Authentication	NIS+ attempts to authenticate every request for access to the namespace. Requests come from NIS+ principals. A NIS+ principal can be a process, machine, root, or a user. Valid NIS+ principals possess a NIS+ credential. NIS+ authenticates the originator of the request (principal) by checking the principal's credential.

If the principal possesses an authentic (valid) credential, and if the principal's request is one that the principal is authorized to perform, NIS+ carries out the request. If either the credential is missing or not valid, or the request is not one the principal is authorized to perform, NIS+ denies the request for access. The entire NIS+ security system is described in Security, on page 7-1.

NIS–Compatibility Mode

Although NIS+ is provided with AIX Version 4.3.3, NIS+ tables can be accessed by workstations running NIS. To access NIS+ service on machines running NIS, you must run the NIS+ servers in NIS–compatibility mode.

NIS–compatibility mode enables an NIS+ server to answer requests from NIS clients while continuing to answer requests from NIS+ clients. NIS+ does this by providing two service interfaces. One responds to NIS+ client requests, while the other responds to NIS client requests.

This mode does not require any additional setup or changes to NIS clients. In fact, NIS clients are not even aware that the server that is responding is not an NIS server — except for some differences including: the NIS+ server running in NIS–compatibility mode does not support the **ypupdate** and **ypxfr** protocols and thus it cannot be used as a replica or master NIS server. For more information on NIS–compatibility mode, see Using NIS–Compatibility Mode, on page 3-23.

Note: In AIX Version 4.3.3 and later releases, the NIS–compatibility mode **supports** DNS–forwarding.

Two more differences need to be pointed out. One is that instructions for setting up a server in NIS–compatibility mode are slightly different than those used to set up a standard NIS+ server. For details, see Using NIS–Compatibility Mode, on page 3-23. The other is that NIS–compatibility mode has security implications for tables in the NIS+ namespace. Since the NIS client software does not have the capability to provide the credentials that NIS+ servers expect from NIS+ clients, all their requests end up classified as **unauthenticated**. Therefore, to allow NIS clients to access information in NIS+ tables, those tables must provide access rights to unauthenticated requests. This is handled automatically by the utilities used to set up a server in NIS–compatibility mode, as described in Setting Up NIS+ Servers, on page 5-43. However, to understand more about the authentication process and NIS–compatibility mode, see Security.

Using NIS+ APIs

The NIS+ application programmer’s interface (API) functions can be called by an application to access and modify NIS+ objects. The NIS+ API functions that fall into the following categories:

- Transaction log functions (**nis_admin**)
- Database access functions (**nis_db**)
- Error message display functions (**nis_error**)
- Group manipulation functions (**nis_groups**)
- Local name functions (**nis_local_names**)
- Miscellaneous functions (**nis_misc**)
- Object manipulation functions (**nis_names**)
- Application subroutine functions (**nis_subr**)
- Table access functions (**nis_tables**)

Using NIS+ Commands

NIS+ provides a full set of commands for administering a namespace. The following table summarizes them. For a complete description of syntax and options, see their command descriptions.

Table 3. NIS+ Namespace Administration Commands

Command	Description
nisaddcred	Creates credentials for NIS+ principals and stores them in the cred table.
nisaddent	Adds information from /**etc** files or NIS maps into NIS+ tables.
niscat	Displays the contents of NIS+ tables.
nischgrp	Changes the group owner of an NIS+ object.
nischmod	Changes an object's access rights.
nischown	Changes the owner of an NIS+ object.
nischttl	Changes an NIS+ object's time–to–live value.
nisdefaults	Lists an NIS+ object's default values: domain name, group name, workstation name, NIS+ principal name, access rights, directory search path, and time–to–live.
nisgrep	Searches for entries in an NIS+ table.
nisgrpadm	Creates or destroys an NIS+ group, or displays a list of its members. Also adds members to a group, removes them, or tests them for membership in the group.
nisinit	Initializes an NIS+ client or server.
nisln	Creates a symbolic link between two NIS+ objects.
nisls	Lists the contents of an NIS+ directory.
nismatch	Searches for entries in an NIS+ table.
nismkdir	Creates an NIS+ directory and specifies its master and replica servers.
nismkuser	Creates an NIS+ user.
nispasswd	Not supported in AIX. Use the **passwd** command.
nisrm	Removes NIS+ objects (except directories) from the namespace.
nisrmdir	Removes NIS+ directories and replicas from the namespace.
nisrmuser	Removes an NIS+ user.
nissetup	Creates **org_dir** and **groups_dir** directories and a complete set of (unpopulated) NIS+ tables for an NIS+ domain.
nisshowcache	Lists the contents of the NIS+ shared cache maintained by the NIS+ cache manager.
nistbladm	Creates or deletes NIS+ tables, and adds, modifies or deletes entries in an NIS+ table.
nisupdkeys	Updates the public keys stored in an NIS+ object.
passwd	Changes password information stored in the NIS+ **passwd** table.

Chapter 2. Network Information Service

This chapter provides information on the Network Information Service (NIS), which is installed as part of the Network File System (NFS). This chapter includes the following sections:

- Components of NIS
- NIS Domain
- NIS Maps
- Maintaining Consistent System Information with NIS
- Installing and Configuring NIS, on page 2-6
- Maintaining NIS, on page 2-18
- Using NIS Automounter, on page 2-25
- NIS Reference, on page 2-31

See NIS and NIS+ Troubleshooting, on page 8-1 for information on diagnosing and resolving NIS–related problems.

Network Information Service (NIS) is a distributed database that allows you to maintain consistent configuration files throughout your network. NIS is the current name for the service originally known as *Yellow Pages* (YP). NIS and YP are functionally identical.

NIS is a part of the Network File System (NFS) software package that includes commands and daemons for NFS, NIS, and other services. Although NFS and NIS are installed together as one package, each is independent and each is configured and administered individually. For information on how NFS works with AIX see the *AIX 4.3 System Management Guide: Communications and Networks*. You should obtain a copy of the book *Managing NFS and NIS* (see Related Publications) for a thorough treatment of the advanced features of NIS.

Note: If the file /**var**/**yp**/**securenets** exists, the server only provides NIS services to the hosts within the Internet Protocol (IP) range specified.

Components of NIS

The NIS environment is composed of *clients* and *servers* logically grouped together in a *domain*. Each domain has a particular set of characteristics. These characteristics are defined in *maps,* or databases, that specify certain system information such as user names, passwords, and host names. Each of these components is discussed in detail below.

Servers

An NIS *server* is a host that provides configuration information to other hosts on the network. Servers retain a set of maps and run the **ypserv** daemon, which processes requests from clients for information contained in maps. There are two types of servers: a *master* server and a *slave* server.

Master Servers

A *master* server is the single host in a particular domain that maintains the authoritative maps. The master server runs **ypupdated** daemon, which prompts slave servers to update their copies of the maps (all other hosts in the domain must obtain their map information from the master server, either directly or indirectly). The master server also runs the **yppasswdd** daemon, which processes requests to change users' passwords. Recommended characteristics of the master server include:

- Accessible by the system administrator. If something goes wrong, or if updates need to be made, it is easy to reach the master server.
- Stable. The master server usually stays active for long periods of time. It needs to be stable so systems that depend on it can rely on uninterrupted service.
- Accessible from the network. Although networks can be complex with the presence of many gateways or bridges, the master server should be accessible from most systems on the network.

For a small number of hosts, each host can access the master server directly. However, for a larger number of hosts in a domain, the master server can become overloaded. To balance the NIS processing load and provide services when the master server is unavailable, additional hosts can be designated as slave servers.

Slave Servers

NIS *slave* servers act as intermediaries between clients and the master server by keeping exact replicas of the master server's maps. All changes to the maps are made on the master server. Then, the changes are propagated from the master server to the slave servers. Once a slave server is added to the domain, it is able to answer the same queries that the master is able to answer. In this way, slave servers can help with extra load on the master server without violating the authority of the master server.

Slave servers also act as a backup in case the master server or the network fails. A client requesting information waits until a server responds. This waiting time varies depending on the reason the server is unreachable. Adding slave servers increases the availability of information even if the master server is unavailable.

Normally, there should be at least one slave server for each domain. The number of slave servers in a domain should be balanced to achieve the desired level of availability and response time without adding the expense of copying data to too many systems.

Clients

NIS *clients* make up the majority of hosts in a NIS domain. Clients run the **ypbind** daemon, which enables client processes to obtain information from a server. Clients do not maintain maps themselves, but rather query servers for system and user account information. (Clients do not make a distinction between querying the master server or a slave server.) To access system information contained in an map, a client makes a Remote Procedure Call (RPC) to a server. The server searches its local database and returns the requested information to the client. (See *AIX Communications Programming Concepts* for detailed information about RPCs.)

NIS clients locate the server by broadcasting on the networks that are directly connected to the client machine. Since these broadcast messages are not forwarded by network gateways, if there is no NIS server that can be reached without using a network gateway, the client must specify a server when starting the **ypbind** daemon.

Note that every request for system information requires a server contact, and the speed of your network can affect the response time. Although a local retrieval is usually faster than a network retrieval, the benefits of NIS outweigh the compromise in access time.

NIS Domain

An NIS domain is a collection of systems that are logically grouped together. A group of hosts that share the same set of NIS maps belong to the same domain. The hosts are usually grouped together in the domain for a common reason; for example, when working in the same group at a particular location. Each NIS host is assigned to a domain when the system starts. The domain name must be set on all hosts that intend to use NIS.

There is one master server per NIS domain, and the systems in the domain are typically on the same network. However, access to data served by NIS is independent of the relative locations of an NIS client and server. All systems within the NIS domain use the master server to retrieve system information, and the number of systems in a domain must be limited for the sake of efficiency. As the number of systems grows, the response time from the master server increases because of the increased workload. By design, you cannot add another master server to a domain because there would be two authoritative sources for the maps. To reduce master server load, you can add slave servers to the domain, or define more than one domain. Each new domain, of course, has its own master server.

NIS Maps

NIS *maps* are databases that specify certain system information such as user names, passwords, and host names, in a database format called *DBM* (Database Management). Each map is constructed from a standard text file by associating an index *key* with a *value*. For example, the information in the master server's /**etc**/**hosts** file is used to create a map that uses each host name as a key, and the IP address as the value. The key and value pairs (also known as *records*) that are created from the entries in the /**etc**/**hosts** file comprise the *hosts.byname* map.

Attention: An NIS record has a *maximum size* of 1024 bytes. This limitation applies to all NIS map files. For example, a list of users in a group can contain a *maximum* of 1024 characters in single–byte character set file format. NIS cannot operate correctly with map files that exceed this maximum.

The most commonly used maps have *nicknames* that some commands can translate into map names. For instance, when you enter:

```
ypcat hosts
```

the output you receive is actually the contents of the *hosts.byname* map, because there is no map called `hosts` in the NIS database. (The **ypcat –x** command produces a list of available nicknames.)

By default, the maps listed in the following table are created if their corresponding files are available on the master server:

Maps Created by Default (if file is available on master server)		
Map	*Nickname*	*File*
passwd.byname	passwd	/**etc**/**passwd**
passwd.byuid		
group.byname	group	/**etc**/**group**
group.bygid		
hosts.byaddr	hosts	/**etc**/**hosts**
hosts.byname		
ethers.byaddr	ethers	/**etc**/**ethers**
ethers.byname		

networks.byaddr	networks	**/etc/networks**
networks.byname		
rpc.bynumber		**/etc/rpc**
services.byname	services	**/etc/services**
protocols.byname	protocols	**/etc/protocols**
protocols.bynumber		
netgroup		**/etc/netgroup**
netgroup.byhost		
netgroup.byuser		
bootparams		**/etc/bootparams**
mail.aliases	aliases	**/etc/aliases**
mail.byaddr		
publickey.byname		**/etc/publickey**
netid.byname		**/etc/passwd**
		/etc/group
		/etc/hosts
		/etc/netid
netmasks.byaddr		**/etc/netmasks**
ypservers		

ypservers: a Special Map

Notice that no file corresponds to the **ypservers** map. **ypservers** is a special map that contains the names of the NIS servers, both slave and master, in the domain. Clients use the **ypservers** map to find the nearest available server. The master server refers to it to determine the names of the slave servers that need to obtain updated copies of the NIS maps. Information about specifying the input to the **ypservers** map is discussed in Configuring the NIS Master Server, on page 2-6 and in Adding a New NIS Slave Server, on page 2-19.

Netgroups: Network–Wide Groups of Machines and Users

The **/etc/netgroup** file is not a standard Transmission Control Protocol/Internet Protocol (TCP/IP) file. Rather, it is strictly an NIS file that resides on the master server. NIS uses the **/etc/netgroup** file to generate the **netgroup.byuser** and **netgroup.byhost** maps. NIS provides these maps for authentication purposes during login, remote login, remote mount, and remote shell processes.

Specifically, the programs that consult these maps are:

login command
: Consults the maps for user classifications if it encounters netgroup names in the **/etc/passwd** file

rlogin command
: Consults the maps for machine classifications if it encounters **netgroup** names in the **/etc/exports** file

rlogin command and **rsh** command
: Consult the **netgroup** map for both machine and user classifications if they encounter **netgroup** names in the **/etc/hosts.equiv** or **/.rhosts** files.

mountd daemon
: Consults the maps for machine classifications if it encounters **netgroup** names in the **/etc/exports** file

For detailed information on the format of the /**etc**/**netgroup** file, see netgroup File for NIS in the *AIX Files Reference*.

makedbm and Makefile: Creating Maps

NIS maps are created by the **makedbm** command, converting text files into DBM format files. To simplify maintaining your maps, NIS provides a *makefile* for use with the **make** command. The default makefile (/**var**/**yp**/**Makefile**) contains all the instructions necessary to create all the default maps. You can add stanzas to /**var**/**yp**/**Makefile** to create additional maps. However, the default makefile is sufficient to address the basic needs of most NIS installations.

When the **makedbm** command generates an NIS map, it creates two files: *map.key***.pag** and *map.key***.dir**. For example, the *host.byname* map consists of the **hosts.byname.pag** and **hosts.byname.dir** files. The file with the **.pag** extension contains the key and value pairs, while the file with the **.dir** extension is the index for the **.pag** file. All the maps for a domain are stored on the servers in a subdirectory of the /**var**/**yp** directory. The subdirectory has the same name as the domain. For example, maps for the `literature` domain are located in the /**var**/**yp**/**literature** subdirectory.

Maintaining Consistent System Information with NIS

NIS maintains consistent system information throughout the domain by designating one system, the master server, as the sole source of information. All the other hosts, whether they are slave servers or clients, obtain their system information from the master server.

Clients obtain their information on an as–needed basis. When a client needs a piece of system information, such as an entry from the /**etc**/**passwd** file, it sends a request to a server. If the information exists, the server responds with the information. Since the client obtains the information only as needed, the system information at the client remains consistent with the server.

Slave servers, on the other hand, obtain a complete copy of the maps periodically from the master server. To ensure that the system information is consistent at the slave servers, and therefore throughout the whole domain, make all updates to the maps at the master server. Then, propagate the new maps to the slave servers. To propagate a map means to copy it from the master server to all the slave servers. Propagation eliminates the need to update each map individually. In addition, propagation ensures that all copies of the database are exactly the same; therefore, any server can respond to a client's request.

After you update a map on the master server, there are three ways to propagate the new map:

- Maps propagate automatically every few minutes if the master server is running the **ypupdated** daemon.
- If you stop and restart NIS on the master server, all the maps propagate to the slave servers.
- If you enter the **yppush** command at the master server, the changes propagate. The **yppush** command notifies all slave servers that a map must be transferred. The **ypserv** daemon on each slave server runs the **ypxfr** command to get the updated map. A slave server that is out of service when you enter the **yppush** command retains the earlier version of the map when it returns to the network. To prevent such situations, use the **cron** daemon to set each slave server to request updated maps from the master server at regular intervals.

Installing and Configuring NIS

For information on installing the Network Information Service (NIS) and the Network File System (NFS), see the *AIX Installation Guide*.

Configuring NIS

For each NIS domain you want to configure on your network, do the following:

1. Decide which hosts on your network you want to include in this domain. Choose a domain name for the domain and make a note of it for use later in the configuration process.
2. Choose a host that has the characteristics described in Master Servers, on page 2-2. Then follow the instructions in Configuring the NIS Master Server, on page 2-6.
3. Decide which hosts, if any, will act as slave servers. Then, for each slave server, follow the instructions in Configuring an NIS Slave Server, on page 2-8.
4. Decide which hosts will be clients in this domain. Then, for each client, follow the instructions in Configuring an NIS Client, on page 2-10.

Note:

1. If you want non–root users to be able to log into a server, you must configure the server as a NIS client as well.
2. If the file **/var/yp/securenets** exists, the server only provides NIS services to the hosts within the Internet Protocol (IP) range specified.

Setting the NIS Domain Name

To set the NIS domain name of a host (whether client or server), use the Web-based System Manager fast path, **wsm network**, or use one of the following procedures.

- Using the System Management Interface Tool (SMIT):
  a. Enter the fast path: `smit chypdom`
  b. Enter the domain name in the `Domain name of this host` field.
  c. Specify **both** in the `CHANGE domain name take effect...` field.
  d. Accept your changes and exit SMIT. The NIS domain name is now set.
- Using the command line, enter: `chypdom –B` *`newdomainname`*

Each of these methods perform two actions. First, they run the **domainname** command, setting the NIS domain name. Second, they modify the **/etc/rc.nfs** file so that the NIS domain name is set when the system restarts.

Configuring the NIS Master Server

Attention: An NIS record has a *maximum size* of 1024 bytes. This limitation applies to all NIS map files. For example, a list of users in a group can contain a *maximum* of 1024 characters in single–byte character set file format. Before doing the following procedure, ensure that no configuration file is beyond this limit. NIS cannot operate correctly with map files that exceed this maximum.

To configure an NIS master server, do the following tasks on the master server host:

1. Follow the instructions in Preparing a Host for NIS Configuration, on page 2-10.
2. Set the domain name by following the instructions in Setting the NIS Domain Name, on page 2-6.

3. Decide what information you want to manage using NIS. By default, you manage all the information contained in the files listed in NIS Maps, on page 2-3. You may want to customize how you manage users, groups, and host names, especially if you have already configured a domain name server. To do so, follow the instructions in Customizing NIS Map Input, on page 2-11.

You will now create the directory for this domain, build the NIS maps, and start the NIS daemons. Use the Web-based System Manager fast path, **wsm network**, or use one of the following procedures.

- Using SMIT, enter: `smit mkmaster`.
  - Specify in the `HOSTS that will be slave servers` field the names of the hosts, if any, that you want to act as slave servers.
  - Specify **yes** in the fields `Can existing MAPS for the domain be overwritten?` and `EXIT on errors, when creating master server?` because you will want to know if an error occurs.
  - If you want to configure your NIS domain for secure Remote Procedure Call (RPC) networking, specify **yes** in the `START the yppasswdd daemon?` and `START the ypupdated daemon?` fields. You should also configure secure NFS by following the instructions in *AIX 4.3 System Management Guide: Communications and Networks*.
  - Specify **yes** in the `START the ypbind daemon?` field to configure the master server to use the NIS databases.
  - Specify **both** in the `START the master server...` field.
  - Accept your changes and exit SMIT.

  The system takes a few minutes to perform several tasks. First, it runs the **ypinit** command. If the **ypinit** command exits successfully, the system uncomments the entries in the **/etc/rc.nfs** file for the daemons to which you specified **yes** above. Finally, the system starts these daemons.

  The **ypinit** command is a shell script that performs two tasks. First, it creates the directory **/var/yp/***domainname*, where *domainname* is the domain name you defined above. Second, it runs the **make** command on the **/var/yp/Makefile**, which creates all the NIS maps specified in the **/var/yp/Makefile**.

- Using the command line:

  a. Enter the **ypinit –m** command. This command prompts you for various information, including the names of any slave servers, and takes a few minutes to complete.

  b. Start the **ypserv** and **ypbind** daemons (and the **yppasswdd** and **ypupdated** daemons if you want) by following the instructions in Starting and Stopping the NIS Daemons, on page 2-13.

  c. Edit the **/etc/rc.nfs** file and uncomment the lines that use the **startsrc** commands to start these daemons (delete the pound signs at the beginning of each line). For example, if the original lines look like the following:

```
#if [ -x /usr/etc/ypserv -a -d /etc/yp/`domainname` ]; then
#       startsrc -s ypserv
#fi
```

  Remove the pound signs so the file looks like:

```
if [ -x /usr/etc/ypserv -a -d /etc/yp/`domainname` ]; then
       startsrc -s ypserv
fi
```

Further Considerations When Using the yppasswd Daemon

If you chose to use a password file other than **/etc/passwd** to build the **passwd** map (see Customizing NIS Map Input, on page 2-11), you must specify to the **yppasswdd** daemon the path to that file. By default, the **yppasswdd** daemon changes passwords for entries in the **/etc/passwd** file. To change the default password file to another file, do the following:

1. Edit the **/etc/rc.nfs** file, and locate the following stanza:

```
#Uncomment the following lines to start up the NIS
#yppasswd daemon.
DIR=/etc
if [ -x /usr/etc/rpc.yppasswdd -a -f $DIR/passwd ]; then
        start rpc.yppasswdd /usr/lib/netsvc/yp/rpc.yppasswdd
        /etc/passwd ~m
fi
```

2. Change the `DIR` statement so that it specifies the path to your alternate passwd file. For example, if you use the **/var/yp/passwd** file, the `DIR` statement should look like:

```
DIR=/var/yp
```

3. Save the file and exit the editor.
4. Enter the following three commands:

```
stopsrc -s yppasswdd
chssys -s yppasswdd -a '/var/yp/passwd -m passwd'
startsrc -s yppasswdd
```

The **yppasswdd** daemon will now use your alternate password file.

Configuring an NIS Slave Server

After configuring the master server, you must decide which hosts are to act as slave servers. Slave servers keep exact replicas of the master server's maps and share the processing burden by answering queries when the master server is busy or unavailable. The following procedure must be done for each slave server.

Prerequisites

The NIS master server is configured.

Procedure

To configure an NIS slave server, do the following tasks on the slave server host:

Note:

1. If you are configuring a slave server that is not on the same IP network, you must configure the new server as an NIS client first. Use the **ypset** command to explicitly point the new server to the NIS master. For example, `ypset 129.23.22.1`, where `129.23.22.1` is the IP address of the master server.
2. When using subnets, a slave server must be configured on each subnet that has NIS clients for the given NIS domain. This allows clients to bind at startup and provides a fallback if the master goes down for any reason.

1. Follow the instructions in Preparing a Host for NIS Configuration, on page 2-10.
2. Set the domain name by following the instructions in Setting the NIS Domain Name, on page 2-6.

You will now create the directory for this domain, start the NIS daemons, and obtain copies of the NIS maps from the master server. Use the Web-based System Manager fast path, **wsm network**, or use one of the following procedures.

- Using SMIT:
  a. Enter the fast path: `smit mkslave`.

b. Specify the hostname of the master server for this domain in the `HOSTNAME of the master server` field.

c. Specify **yes** in the fields `Can existing MAPS for the domain be overwritten?` and `Quit if errors are encountered?` because you will want to know if an error occurs.

d. Specify **both** in the `START the slave server...` field.

e. Accept your changes and exit SMIT.

The system takes a few minutes to perform several tasks. First, it runs the **ypinit** command. If the **ypinit** command exits successfully, the system uncomments the entries in the /**etc**/**rc.nfs** file for the **ypserv** and **ypbind** daemons. Finally, the system starts these daemons.

The **ypinit** command is a shell script that performs two tasks. First, it creates the directory /**var**/**yp**/*domainname*, where *domainname* is the domain name you defined above. Second, it runs the **ypxfr** command to obtain the NIS maps from the master server.

Note: If this NIS slave server is not on same IP network as the NIS master server (that is, a gateway router is positioned between the slave server and the master server), you must explicitly identify the NIS master server by using the **ypset** command. For example, enter the command:

```
ypset 129.23.22.1
```

where `129.23.22.1` is the IP address of the NIS master server.

- Using the command line:

a. Start the **ypbind** daemon by following the instructions in Starting and Stopping the NIS Daemons, on page 2-13 to bind to the master server.

b. Enter the **ypinit –s** *mastername* command, where *mastername* is the host name of the master server. This command prompts you for various information and takes a few minutes to complete.

c. Start the **ypserv** and **ypbind** daemons by following the instructions in Starting and Stopping the NIS Daemons, on page 2-13.

Note: If this NIS slave server is not on same IP network as the NIS master server (that is, a gateway router is positioned between the slave server and the master server), you must explicitly identify the NIS master server by using the **ypset** command. For example, enter the command:

```
ypset 129.23.22.1
```

where `129.23.22.1` is the IP address of the NIS master server.

d. Edit the /**etc**/**rc.nfs** file and uncomment the lines that use the **startsrc** commands to start these daemons. Delete the pound signs in the following example:

```
#if [ –x /usr/etc/ypserv –a –d /etc/yp/`domainname` ]; then
#       startsrc –s ypserv
#fi
```

so it looks like:

```
if [ –x /usr/etc/ypserv –a –d /etc/yp/`domainname` ]; then
       startsrc –s ypserv
fi
```

Note: If NIS users need to log into an NIS slave server, the slave server must also be configured as a client, and should have the following line as the last line in its /**etc**/**passwd** file:

```
+::::::
```

Configuring an NIS Client

NIS clients make up the majority of hosts in an NIS domain. Clients do not maintain maps, but rather query servers for information. (Clients do not distinguish between master and slave servers.) If you are configuring a slave server that is not on the same IP network as the master server, you must configure the new server as an NIS client first.

Prerequisites

The NIS master server is configured.

Procedure

To configure an NIS client, do the following tasks on the client host:

1. Follow the instructions in Preparing a Host for NIS Configuration, on page 2-10.
2. Set the domain name by following the instructions in Setting the NIS Domain Name, on page 2-6.

You then start the client using NIS. Use the Web-based System Manager fast path, **wsm network**, or use one of the following procedures.

- Using SMIT:
  a. Enter the fast path: `smit mkclient`.
  b. Specify **both** in the **START the NIS client...** field.
  c. Accept your changes and exit SMIT.

     The system performs two tasks. First, it starts the **ypbind** daemon. Second, it uncomments the entry in the **/etc/rc.nfs** file for the **ypbind** daemon.
  d. Follow the instructions in Setting Up NIS Client Files to Use NIS Services, on page 2-14.
- Using the command line:
  a. Start the **ypbind** daemon by following the instructions in Starting and Stopping the NIS Daemons, on page 2-13.
  b. Edit the **/etc/rc.nfs** file and uncomment the lines that use the **startsrc** command to start this daemon. Specifically, delete the pound signs in the following example:

```
#if [ -x /usr/etc/ypbind ]; then
#        startsrc -s ypbind
#fi
```

so it looks like:

```
if [ -x /usr/etc/ypbind ]; then
       startsrc -s ypbind
fi
```

Preparing a Host for NIS Configuration

Before you configure NIS on a master server, slave server, or client, do the following:

1. Verify that the PATH variable in the **/.profile** file includes the **/usr/sbin** directory where the NIS commands reside.
2. Verify that Transmission Control Protocol/Internet Protocol (TCP/IP) is running by entering the command:

```
lssrc -s inetd
```

A message similar to the following displays:

```
Subsystem          Group             PID     Status
 inetd             tcpip             4923    active
```

If the status does not indicate *active*, follow the instructions in Configuring the inetd Daemon for starting the **inetd** daemon.

3. Verify that the **portmap** daemon is running by entering the command:

   ```
   lssrc -s portmap
   ```

   A message similar to the following displays:

   ```
   Subsystem         Group            PID     Status
    portmap          portmap          14003   active
   ```

   If the status does not indicate *active*, enter the command:

   ```
   startsrc -s portmap
   ```

You are now ready to configure NIS on this host. If you are configuring a master server, continue with the following section, Customizing NIS Map Input. If you are configuring a client or slave server, continue with Starting and Stopping NIS Daemons, on page 2-13.

Customizing NIS Map Input

The most common customizations made to NIS involve users, groups, and host names. However, you can customize any of the information managed by NIS. Although this discussion focuses on users, groups, and host names, you can use the same techniques to customize input to other maps.

Note: Perform all of these instructions on the master server host.

Users and Groups

Attention: An NIS record has a *maximum size* of 1024 bytes. This limitation applies to all NIS map files. For example, a list of users in a group can contain a *maximum* of 1024 characters in single–byte character set file format. Before doing the following procedure, ensure that no configuration file is beyond this limit. NIS cannot operate correctly with map files that exceed this maximum.

By default, NIS uses the **/etc/passwd** and **/etc/group** files on the master server as the input for the **passwd** and **group** maps. All users and groups on the master server are thus included automatically in the maps. The simplest configuration is to add every user and group in this entire domain to the **/etc/passwd** and **/etc/group** files.

Note: It is possible to manage users and groups without using NIS; however, managing users and groups is the primary benefit of NIS. For more secure methods of user and group management, see NIS+ Security Mechanisms, on page 7-3.

Either for security, accounting, or performance reasons, you may not want certain users to log into the master server. If so, you can build the **passwd** and **group** maps from other files, such as **/var/yp/passwd** and **/var/yp/group**, that are for NIS users and groups only. With this, **/etc/passwd** and **/etc/group** can contain only the minimum necessary entries. (Using a separate password file also affects the **yppasswdd** daemon. See Configuring the NIS Master Server, on page 2-6 for more information.) To configure the master server in this way, do the following:

1. Create the new file to be used instead of the **/etc/group** file (for example, assume that you name the file **/var/yp/group**) by entering the following command:

   ```
   cp /etc/group /var/yp/group
   ```

   You can use a copy of any machine's **/etc/group** file, not just the **/etc/group** file on the master server. Then, using an editor, remove from the **/etc/group** file all the non–local entries, and add the NIS escape sequence (`+:`) as the last line in the file.

2. Create the new file to be used instead of the **/etc/passwd** file (for example, assume that you name the file **/var/yp/passwd**). Again, you can use a copy of any machine's **/etc/passwd** file, not just the **/etc/passwd** file on the master server. Also, you can use

the password information from another NIS domain by entering `ypcat passwd > passwd` at the command line of a client in the other domain. Then, copy the **passwd** file into the **/var/yp** directory of the master server in this domain.

You can either preserve the current passwords or reset the passwords.

– If you want to preserve existing passwords, use the **mrgpwd** command to merge the **/etc/passwd** file with the **/etc/security/passwd** file, where the encrypted passwords are stored. This step is actually two commands, as shown below:

```
cd /var/yp
/usr/sbin/mrgpwd > passwd
```

mrgpwd takes its input from the **/etc/passwd** and **/etc/security/passwd** files only.

– If you want to reset all the passwords, enter the following command:

```
cp /etc/passwd /var/yp/passwd
```

Then, using an editor, remove the ! (exclamation point) from the password field in each entry in the **/var/yp/passwd** file. Finally, using an editor, remove from the **/etc/passwd** file all the non–local entries, and add the NIS escape sequence (`+::0:0:::`) as the last line in the file.

Note: User IDs (UIDs) created in this way initially contain no passwords.

3. Change the **/var/yp/Makefile** file to reflect the new locations of the input files. You can do so using one of two methods:

– Locate only the **/etc/passwd** and **/etc/group** files in **/var/yp**. Using an editor, open the **/var/yp/Makefile** file and create a new variable called `PWDIR=/var/yp`. In the **passwd.time** and **group.time** stanzas, replace every occurrence of the `DIR` variable with `PWDIR`.

– Locate all the **/etc** files in **/var/yp**. Edit the **Makefile** file to modify the default `DIR` variable. Change `DIR=/etc` (the default configuration) to `DIR=/var/yp`. In contrast to the first method, you do not have to edit any of the **Makefile** stanzas.

Note: The Web-based System Manager fast path, **wsm users**, and the SMIT fast paths **smit mkuser** and **smit mkgroup** can be used to create users and groups only in the **/etc/passwd** and **/etc/group** files.

Host Names

By default, NIS only uses the **/etc/hosts** file to build the **hosts** map. If you have configured a domain name server in your network, you can configure NIS to include domain name system (DNS) information as well as **/etc/hosts** information in the **hosts** map. (Including DNS information in the **hosts** map eliminates re–entering all this information in the **/etc/hosts** file.) To do so, use an editor to change the **/var/yp/Makefile** file as follows:

1. Locate the **hosts.time** stanza in the **/var/yp/Makefile** file.
2. Change the two lines containing the word **MAKEDBM**:

```
...
| $(MAKEDBM) – $(YPDBDIR)/$(DOM)/hosts.byname; \
...
| $(MAKEDBM) – $(YPDBDIR)/$(DOM)/hosts.byaddr; \
...
```

so that they look like:

```
...
| $(MAKEDBM) –b – $(YPDBDIR)/$(DOM)/hosts.byname; \
...
| $(MAKEDBM) –b – $(YPDBDIR)/$(DOM)/hosts.byaddr; \
...
```

In other words, add the **–b** flag, with a space before and after, to both lines.

The **ypserv** and **ypxfrd** daemons use the file **/var/yp/securenets**, if it exists, and only respond to the IP addresses listed in the *netmask netaddr* pairs within that file.

Starting and Stopping NIS Daemons

Use Web-based System Manager, SMIT, or the following procedure to start or stop NIS daemons.

Prerequisites

1. NFS must be installed on your system.
2. The **portmap** daemon must be active. To check this, use the following command:

   ```
   lssrc -s portmap
   ```

   The result looks similar to the following:

   ```
   Subsystem          Group          PID           Status
    portmap           portmap        4388          active
   ```

Procedure

The five NIS daemons are controlled by the System Resource Controller (SRC). As illustrated in the following table, four of the daemons have the SRC group name **yp**:

SRC Controlled NIS Daemons		
Daemon Name	**Subsystem Name**	**Group Name**
keyserv	keyserv	keyserv
ypbind	ypbind	yp
yppasswdd	yppasswdd	yp
ypserv	ypserv	yp
ypupdated	ypupdated	yp

To start or stop NIS daemons, use the Web-based System Manager fast path, **wsm network**, or use one of the following procedures.

- Using SMIT:
  - Enter the SMIT **smit ypstartstop** fast path.

    Select a menu option, depending on whether you want to start or stop the **ypserv**, **ypbind**, **yppasswdd**, or **ypupdated** daemon. Once you make your selection, the daemon you specified will be started or stopped.
  - To start the **keyserv** daemon, use the SMIT **smit mkkeyserv** fast path.
  - To stop the **keyserv** daemon, use the SMIT **smit rmkeyserv** fast path.
- Using the command line to start or stop the NIS daemons, run the **startsrc** or **stopsrc** command. You can start or stop the daemons individually or as a group. For example:
  - To start all the daemons (not including the **keyserv** daemon), enter:

    ```
    startsrc -g yp
    ```
  - To stop all the daemons, enter:

    ```
    stopsrc -g yp
    ```
  - To start a single daemon, enter:

    ```
    startsrc -s daemon_name
    ```
  - To stop a single daemon, enter:

    ```
    stopsrc -s daemon_name
    ```

Setting Up NIS Client Files to Use NIS Services

In this procedure, you specify which NIS maps that this client will use by adding a special NIS marker to various system files. In general, the system configuration files on an NIS client should have a minimum number of entries because the client should rely primarily on a server for its information. However, you may want to configure a few local entries that you do not want defined throughout the entire domain.

Actually, NIS handles client configuration files in two ways. Some configuration files are completely ignored once the **ypbind** daemon starts, and other files are appended to. If NIS ignores a particular file, the client will only know what its server's map contains. If NIS appends map information to a file, the client can use local information that no other host knows as well as NIS map information.

Files that NIS Ignores

Once the **ypbind** daemon is running, the client relies solely on the following NIS maps instead of their corresponding files:

Maps Used by the ypbind Daemon Instead of Files

Map	*Nickname*	*File*
hosts.byaddr	hosts	**/etc/hosts**
hosts.byname		
ethers.byaddr	ethers	**/etc/ethers**
ethers.byname		
networks.byaddr	networks	**/etc/networks**
networks.byname		
rpc.bynumber		**/etc/rpc**
services.byname	services	**/etc/services**
protocols.byname	protocols	**/etc/protocols**
protocols.bynumber		
netgroup		**/etc/netgroup**
netgroup.byhost		
netgroup.byuser		
publickey.byname		**/etc/publickey**
netid.byname		**/etc/passwd**
		/etc/group
		/etc/hosts
		/etc/netid
netmasks.byaddr		**/etc/netmasks**

You do not need to perform any configuration on the above files in order to use their corresponding NIS maps; the **ypbind** daemon does this automatically. However, the **/etc/hosts** file should have entries for the local loopback name and client's host name. Use either an editor or the **smit hosts** fast path to verify that the **/etc/hosts** file has these entries. For example, the client's **/etc/hosts** file should look similar to this example:

```
127.1          localhost   # local loopback name
200.10.2.101   zepher      # client's host name
```

The **/etc/hosts** file is accessed at boot time before NIS is available. After the system is running and the **ypbind** daemon is started, NIS ignores the **/etc/hosts** file.

Files Where NIS Appends Map Information

Each of the following subheadings explains how to configure a client's configuration files to use a particular NIS map. You may choose to use all the available maps, or only a few. Many NIS installations use all the available maps, especially the **passwd** and **group** maps.

passwd.byname and passwd.byuid map

These two maps together are referred to by the nickname **passwd**. Using either an editor or the **smit lsuser** fast path, verify that the /**etc**/**passwd** file contains entries for the root user and the other primary users on the machine (in other words, the entries supplied in the default /**etc**/**passwd** file). Then, using an editor, add the NIS escape entry, + (plus sign), to enable the NIS password service. For example, the client's /**etc**/**passwd** file should look similar to the following:

```
root:!.k:0:1:/:/usr/bin/csh
nobody:*:-2:-2::/:
daemon:*:1:1::/:
sys:*:2:2::/:/usr/bin/csh
bin:*:3:3::/usr/bin:
uucp:*:4:4::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/usr/bin/csh
+::0:0:::
```

The NIS entry (the last line) instructs library routines to use the NIS password service after examining the local entries. So, when a program examines the /**etc**/**passwd** file, it first finds the local entries, and then it requests that NIS provide the password information.

In addition to using the entire **passwd** map, you can explicitly include (with a plus entry) and exclude (with a minus entry) NIS password information about specific users and groups.

The following are the types of + (plus) and – (minus) entries that you may define:

- A + (plus) by itself means to include the entire contents of the NIS **passwd** map.
- A + (plus) with a name means to include that name from the NIS map.
- A + (plus) followed by a @ and a netgroup (that is `+@netgroup_name`) means to insert the entries for all the members of the netgroup `netgroup_name` at that point.
- The – (minus) entries mean exclude the user or netgroup specified.

If the + (plus) entry contains data in one of the colon–separated fields (except for the user ID, or UID, and group ID, or GID, fields) of the password entry, that data overrides what is in the NIS map. Also, earlier entries in the file take precedence over later entries with the same user name or user ID. The following are some examples:

To remove the NIS password entry for a user, enter:

```
-user
```

To remove the NIS password for users in a netgroup, enter:

```
-@netgroup
```

The line that subtracts the netgroup or user must appear before any other /**etc**/**passwd** file entry that includes the netgroup or user. For example, to remove password entries for user `cliffc` and users in the `bad-users` netgroup, the password file entry must contain the user name, UID, and GID:

```
-cliffc:*:218:201::
-@bad-users
+::0:0:::
```

If user `cliffc` is a member of the `good-users` netgroup, the following example does *not* remove user `cliffc` from the /**etc**/**passwd** file:

```
+@good-users
-cliffc:*:218:201::
+::0:0:::
```

Once the routines that read the password's file find a match for `cliffc`, they stop parsing the file. Therefore, the `-cliffc` entry will never be found, because the `good-users` netgroup includes user `cliffc`.

group.byname and group.bygid maps

These two maps together are referred to by the nickname **group**. Using either an editor or the **smit lsgroup** fast path, verify that the /**etc**/**group** file contains entries for the system and other primary groups on the machine (in other words, the entries supplied in the default /**etc**/**group** file). Then, using an editor, add the NIS escape entry (+, plus sign) to enable the NIS group service. For example, the client's /**etc**/**group** file should look similar to the following:

```
system:!:0:root
staff:!:1:root
bin:!:2:root,bin
sys:!:3:root,su,bin,sys
adm:!:4:root,su,bin,adm
uucp:!:5:root,uucp
mail:!:6:root,su
security:!:7:root
cron:!:8:root
printq:!:9:root
audit:!:10:root
+:
```

mail.aliases and mail.byaddr maps

These two maps together are referred to by the nickname **aliases**. To enable use of the NIS aliases mapping:

1. Uncomment `O AliasFile` in the **sendmail.cf** file and specify the map name for NIS aliases.
2. Recompile the **sendmail.cf** file with the command **sendmail –bz**.
3. Recompile the alias database with the command **sendmail –bi**.

netgroup.byhost and netgroup.byuser maps

As noted in Files that NIS Ignores, on page 2-14, NIS uses these two maps automatically. However, you can configure two other system files to reference these maps, specifically, the /**etc**/**hosts.equiv** file and the /**.rhosts** file. Doing so can help you control remote logins more effectively.

For example, you can edit the /**etc**/**hosts.equiv** file and add a single line, with only the + (plus) character on it. This allows anyone to log into the machine because all further entries are retrieved from NIS rather than the local file. Or, for more control over logins, add a list of trusted hosts to the /**etc**/**hosts.equiv** file. For example:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

The names to the right of the `@` (at sign) should be netgroup names defined in the netgroup map.

You can also add a list of trusted hosts to the /**.rhosts** file. For example:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

Because this file controls remote root access to the local machine, unrestricted access is not recommended. You cannot use aliases for host names in the /**.rhosts**, **hosts.equiv**, or **netgroup** files, because they all enable local machines to access remote machines. You can, however, use aliases for host names in the /**etc**/**hosts** file.

Note: If none of the escape sequences are added to the **/etc/hosts.equiv** or /**.rhosts** files, NIS is not used when a program examines these files.

Maintaining NIS

The Network Information Service (NIS) environment requires adjustments from time to time. In large or complex networks, the NIS environment may change many times a day. This section discusses how to perform various maintenance tasks.

Prerequisite

All the tasks discussed in this section assume that NIS is installed and configured on your network. See Configuring NIS, on page 2-6.

NIS Security

For detailed information on NIS security, see the Security chapter.

/var/yp/securenets

Both the **ypserv** and the **ypxfrd** use the **/var/yp/securenets** file and, if present, only respond to Internet Protocol (IP) addresses in the range given. This file is read only when the daemons (both **ypserv** and **ypxfrd**) start. To get a change in **/var/yp/securenets** to take effect, you must kill and restart the daemon (see Starting and Stopping the NIS Daemons, on page 2-13). The format of the file is:

```
netmask netaddr
e.g.

255.255.0.0 128.30.0.0
255.255.255.0 128.311.10.0
```

The second line the netmask address is 255.255.255.0 and the network address is 128.311.10.0. This setup will only allow the **ypserv** daemon to respond to those IP addresses within the subnet 128.311.10.0 range.

Changing an NIS Map

Changing the NIS maps to reflect updated system information can be a common occurrence in your daily maintenance tasks. System information, such as a new user account or a changed password, can require constant updating. Whenever you need to modify an NIS map, you should do so on the master server, and then propagate the changes to the slave servers. Modifying maps on slave servers can break the NIS service algorithm, which can result in unreliable map data. The only exception to this rule is when users change their password with the **yppasswd** command. (See Changing NIS Passwords, on page 2-19 for more information.)

To change an NIS map, use the Web-based System Manager fast path, **wsm network**, on the master server. Alternatively, you can use one of the following procedures on the master server.

- Using the System Management Interface Tool (SMIT):
  a. Edit the text file that is used as the input file for the map.
  b. Enter the fast path:
     ```
     smit mkmaps
     ```
     Specify the map you changed in the `MAPS that are to be built` field.
  c. Exit SMIT.

     The map is now changed, and the master server has requested that all the slave servers update their maps.
- Using the command line:
  a. Edit the text file that is used as the input file for the map.

b. Update the map by entering the following two commands:

```
cd /var/yp
make map
```

where *map* specifies the map to be updated, for example **hosts** or **services**.

c. Propagate the new map by following the instructions in Propagating an NIS Map, on page 2-22.

Note:

1. If you have modified several text files and want to confirm that all NIS maps are updated, issue the make command without parameters to automatically evaluate every input file on the NIS master server. If the file has been modified since the latest NIS map for that file was built, the NIS map is automatically rebuilt.
2. If the file **/var/yp/securenets** exists, the server only provides NIS services to hosts within the IP range specified.

Changing NIS Passwords

Users can change their password using any of three methods. If the user is logged onto the master server, and you use the **/etc/passwd** file on the master server to build the **passwd** map, the user can:

- Enter the Web-based System Manager fast path, `wsm users`
- Enter the SMIT fast path, `smit passwd`
- Enter the command, `passwd`

All of these methods change the user's entry in the **/etc/passwd** file. Then, you will have to rebuild the **passwd** map manually (see Changing an NIS Map, on page 2-18). If the **yppasswdd** daemon is running on the master server, users can change their password from any host in the domain by entering the command:

```
yppasswd
```

This command changes the user's password in the **passwd** map itself as well as the **/etc/passwd** file, and thus requires no intervention by the system administrator. (For more information on the **yppasswdd** daemon, see Configuring the NIS Master Server, on page 2-6).

Adding a New NIS Slave Server

If your network configuration grows or changes, you may want to add additional slave servers to support the new configuration. Adding a new slave server involves modifying the **ypservers** map. The procedure for modifying the **ypservers** map differs from other maps because no text file is used as input for this map. Instead, the **makedbm** utility is used to create the modified **ypservers** maps. To add a new slave server to your network, do the following on the master server:

1. Change to the **/var/yp** directory by entering:

   ```
   cd /var/yp
   ```

Note: If the file **/var/yp/securenets** exists, the server only provides NIS services to hosts within the IP range specified.

2. Enter the command:

   ```
   (makedbm -u domain/ypservers ; echo new_server) |
   makedbm - tmpservers
   ```

   where *domain* specifies the name of this NIS domain, and *new_server* specifies the name of the slaver server host that you are adding to the **ypservers** map. This command lists the contents of the current **ypservers** map, and appends the name of the new slave server, and then creates a new map called **tmpservers**.

3. Verify that the new map contains the names of all the slave servers by entering the command:

   ```
   makedbm –u tmpservers
   ```

4. Replace the old **ypservers** map files with the new ones by entering the following two commands:

   ```
   mv tmpservers.pag domain/ypservers.pag
   mv tmpservers.dir domain/ypservers.dir
   ```

   where *domain* specifies the name of this NIS domain.

5. Finally, follow the instructions in Configuring an NIS Slave Server, on page 2-8.

Adding a New NIS User

To add a new NIS user, use the Web-based System Manager fast path, **wsm user**, on the master server. Alternatively, you can use one of the following procedures on the master server.

- Using SMIT:

  a. Enter the fast path:

     ```
     smit mkuser
     ```

     Enter the new NIS user's name in the `User NAME` field. This will add an entry to the **/etc/passwd** file for the new user.

  b. Exit SMIT.

  c. Enter the fast path:

     ```
     smit mkmaps
     ```

     Specify **passwd** in the `MAPS that are to be built` field.

  d. Exit SMIT.

     The new NIS user is now added, and the master server has requested that all the slave servers update their maps.

- Using the command line:

  a. Using an editor, add a line for the new NIS user to the **/etc/passwd** file (or whatever file you use as input to the **passwd** map). The new line should look something like:

     ```
     mel::1295:325:Mel Smith:/u/mel:/bin/ksh
     ```

  b. Update the passwd map by entering the following two commands:

     ```
     cd /var/yp
     make passwd
     ```

  c. Propagate the new **passwd** map by following the instructions in Propagating an NIS Map, on page 2-22.

Creating Nonstandard NIS Maps

As discussed in NIS Maps, on page 2-3, maps are databases that have a special format. The default, or standard, maps are built out of standard system text files. However, NIS maps are very flexible. You can build into a map any information that has a key and a value. This article discusses in detail an example of how you can build such a map.

This example shows how to create a nonstandard map called **udir.nam** that lists the home directories of users in this domain. You will use the **/etc/passwd** file as input file for the **udir.nam** map. (If you use a different file to create the **passwd** map, use that file instead of **/etc/passwd**.) The keys for the map are the user names, and their home directories are the corresponding values. To create this new map, enter the following two commands on the master server:

```
cd /var/yp
awk '{FS=":" ; OFS="\t" ; print $1,$6}' /etc/passwd | \
     makedbm - domain/udir.nam
```

where *domain* specifies this NIS domain, and ' is the single forward quote. The **awk** command extracts from the **/etc/passwd** file the first and sixth fields. Then, it passes this information to the **makedbm** command that builds the **udir.nam** map. You can now propagate and use this map just like any other map. For example, enter the command:

```
ypcat udir.nam
```

You should see a list of all the home directories specified in the **/etc/passwd** file.

The above example used the **awk** command to filter out the unwanted fields from the **/etc/passwd** file. However, you can use any system utility or programming language you want to create the map input. In fact, you can create the map input manually if you want. For example, to create the **udir.nam** map, you could have entered the command:

```
makedbm - domain/udir.nam
john /u/john
mary /u/mary
sam /u/sam
<Ctrl^D>
```

Or, suppose you used your favorite system utility or programming language to create the file **temp.input**. Then, you could have entered the command:

```
makedbm - domain/udir.nam < temp.input
```

Whatever method you use to create the map input file, you will probably want to update the map periodically. To do so, you can add stanzas to the **/var/yp/Makefile** file so that your nonstandard map can be updated as any other map (see Changing an NIS Map, on page 2-18). For detailed information on customizing the **/var/yp/Makefile**, see the make Command Overview in *AIX General Programming Concepts: Writing and Debugging Programs*.

Note: When you add a new map after the initial set of maps have been pushed to a slave server, you must make the new maps with the **NOPUSH** option set to 1. For example, use the command:

```
make NOPUSH=1 newmap
```

to make *newmap* map. If you do not use the **NOPUSH** option, the **make** command will hang. (This only applies if the new map does not already exist on the slave server.) Next, use the **ypxfr** command on each slave server for each new map you created (see Propagating an NIS Map, on page 2-22).

Propagating an NIS Map

Prerequisite

The NIS slave servers must be authorized to copy files remotely (using the **rcp** remote copy command) from the NIS master server.

Procedures

To propagate NIS maps from the master server to slave servers, use the Web-based System Manager fast path, **wsm network**, or use one of the following procedures.

- Use the ypxfr Command at Slave Server
  - To propagate an NIS map from a slave server using SMIT, enter the fast path:

    ```
    smit ypxfr
    ```

    and specify a map name in the `Name of the MAP to be transferred` field. This will retrieve the specified map from the master server.
  - To propagate an NIS map from a slave server using the command line, enter the command:

    ```
    ypxfr mapname
    ```
- Use the yppush Command at Master Server
  - To propagate an NIS map from the master server using SMIT, enter the fast path:

    ```
    smit yppush
    ```

    and specify a map names in the `Maps to be transferred to slave` field.
  - To propagate an NIS map from the master server using the command line, enter the command:

    ```
    yppush mapname
    ```

All of these methods use the **yppush** command, which uses the **ypserver** database to generate a list of slave servers in your domain. It then sends a *transfer database* request to the **ypserv** daemon on each of the slave servers. The **ypserv** daemon on the slave server executes a copy of the **ypxfr –C** command and then passes a summary of the information it needs to identify the database and call back the initiating **yppush** command.

Automating Map Propagation

You may need to propagate some maps more frequently than others. For example, the **passwd** map can change many times a day and must be propagated more frequently than the **protocols** or **services** maps, which may not change for months at a time. Rather than propagate each map manually, you may want to automate the propagation of maps with the **cron** daemon. To do so, follow the instructions below. (You will need to perform these instructions on each slave server.)

1. Group the maps together according to how often you want to propagate them. The system provides three example shell scripts that organize the maps into three groups:

   ```
   /usr/sbin/ypxfr_1perhour
   /usr/sbin/ypxfr_2perday
   /usr/sbin/ypxfr_1perday
   ```

   For example, the **/usr/sbin/ypxfr_1perhour** shell script contains **ypxfr** commands for several maps that change frequently and should, therefore, be propagated frequently. The other two shell scripts are for maps that change less frequently. Use an editor to modify these shell scripts to meet the needs of your network.

   Next, configure the **cron** daemon to execute these scripts at the appropriate times. The following steps lead you through this process.

2. As the root user, enter the command:

   ```
   crontab -l > cron.tmp
   ```

   This command records the current **crontab** settings in the file **cron.tmp**.

3. Edit the **cron.tmp** file and add entries for each of the scripts listed above. For example, you might add the entries:

   ```
   00 * * * * /usr/sbin/ypxfr_1perhour      # run every hour
   00 00 * * * /usr/sbin/ypxfr_2perday      # run at midnight
   00 12 * * * /usr/sbin/ypxfr_2perday      # run at noon
   00 1 * * * /usr/sbin/ypxfr_1perday       # run at 1 A.M.
   ```

   To minimize the performance impact on the master server, alter the exact time of execution of the shell scripts on each server. For example, if the first slave server has the above entries, the second slave server might run its shell scripts five minutes later by having the entries:

   ```
   5 * * * * /usr/sbin/ypxfr_1perhour      # run every hour
   5 00 * * * /usr/sbin/ypxfr_2perday      # run at 12:05 A.M.
   5 12 * * * /usr/sbin/ypxfr_2perday      # run at 12:05 P.M.
   5 1 * * * /usr/sbin/ypxfr_1perday       # run at 1:05 A.M.
   ```

   Save the file and exit the editor.

4. Enter the command:

   ```
   crontab cron.tmp
   ```

   This command defines the **crontab** settings to what the **cron.tmp** file contains. (For more information on defining **crontab** settings, see the **crontab** command.)

Logging Map Propagation

Transfers, and transfer attempts, are logged in the **/var/yp/ypxfr.log** file on slave servers. If the file exists, logging results are appended to it. To start logging, enter the command:

```
touch /var/yp/ypxfr.log
```

To stop logging, enter the command:

```
mv /var/yp/ypxfr.log /var/yp/ypxfr.log.old
```

or, enter the command:

```
rm /var/yp/ypxfr.log
```

Moving the Master Server to a Different Host

Once you have NIS configured on your network, your security, performance, or other needs may change. To accommodate the change, you may want to move the master server configuration to a different host, perhaps one that is more secure or offers greater performance. To do so, use the following procedure.

1. Copy all the map input files (not the map files themselves) from the old master server host to the new master server host. In the case of the **ypservers** map, there is no input file to copy. So, to create a temporary **ypservers** map input file, do the following on the old master server host:

   a. Change to the **/var/yp** directory by entering:

      ```
      cd /var/yp
      ```

   b. Enter the command:

      ```
      makedbm -u domain/ypservers > tmpservers
      ```

      where `domain` specifies the name of this NIS domain. This command sends the contents of the current **ypservers** map to a file called **tmpservers**.

c. Edit the **tmpservers** file, and change the name of the master server from the old master server host to the new master server host. For example, if the **tmpservers** file contains the line:

```
YP_MASTER_NAME old_master
```

change this line to:

```
YP_MASTER_NAME new_master
```

Save the file and exit the editor.

d. Copy the **tmpservers** file to the **/var/yp** directory on the new master server host.

2. Configure the new master server host by following the instructions in Configuring the NIS Master Server, on page 2-6.

3. On the new master server host, enter:

   ```
   cd /var/yp
   ```

   Then, enter the commands:

   ```
   makedbm - tmpservers < tmpservers
   mv tmpservers.pag domain/ypservers.pag
   mv tmpservers.dir domain/ypservers.dir
   ```

4. On each of the slave servers, enter the fast path:

   ```
   smit ypxfr
   ```

   Specify **ypservers** in the **Name of the MAP to be transferred** field, and specify the hostname of the new master server host in the **HOSTNAME of the master server** field.

5. Exit SMIT. This will retrieve the updated **ypservers** map from the new master server host.

6. Propagate the rest of the newly rebuilt maps by following the instructions in Propagating an NIS Map, on page 2-22.

Using NIS Automount

Using automount Maps

The use of **automount** maps involves map formats, replicated file systems, comments in maps, directory patterns, multiple mounts, included maps, maps for the **automount** command, and the **auto_master** (or, when necessary for compatibility, **auto.master**) map file.

Map Entry Format

A simple map entry (mapping) follows:

```
key [-mount-options] location ...
```

where

`key`
: Is the full path name of the directory to mount when used in a direct map. In an indirect map, `key` represents a simple name.

`mount-options`
: Is a list separated by commas.

`location`
: Specifies a file system from which the directory can be mounted. In the example case, *`location`* takes the form:

  `hostname``:`*`pathname`*

where *hostname* is the server name where the file system will mount and *pathname* is the directory path to mount.

Replicated File Systems

Multiple *location* fields can be specified for replicated NFS file systems, in which case **automount** and the kernel will each try to use that information to increase availability. If the read–only flag is set in the map entry, **automount** mounts a list of locations that the kernel may use, sorted by several criteria. If a server does not respond, the kernel will switch to an alternative server. Sort ordering is used by **automount** to determine how the next server is chosen. If the read–only flag is not set, **automount** will mount the best single location, chosen by the same sort ordering, and new servers are chosen only when an unmount has been possible and a remount is done. Servers on the same local subnet are given the strongest preference, and servers on the local net are given the second strongest preference. Among servers equally far away, response times determine the order (if no weighting factors are used).

If the list includes server locations using both Versions 2 and 3 of the NFS protocol, **automount** chooses only a subset of the server locations on the list to ensure all entries are the same protocol level. It gives preference to Version 3 servers and uses Version 2 servers if Version 3 servers are unavailable.

If each location in the list shares the same path name, a single location can be used with a list of host names, separated by commas.

```
hostname,hostname...:pathname
```

Weighting Factor

Requests for a server can be weighted by appending the weighting factor (as an integer within parentheses) to the server name. A weighting factor of zero is the highest priority. Servers without a weighting factor are assumed to have a weighting factor of zero. Progressively higher values decrease a server's chance of being selected. In the following example,

```
man –ro austin,books,mystery(1),person(4):/usr/man
```

hosts `austin` and `books` have the highest priority; host `person`, the lowest.

Note: In the selection process, server proximity takes higher priority than weighting. In the previous example, if server `person` were on the same network as the user and the other servers were on different network segments, then `person` would be selected and the weighting value would be ignored.

When each server has a different export point, you can still apply weighting. For example:

```
man –ro austin:/usr/man books,mystery(1):/usr/share/man
person(3):/export/man
```

Mapping can be continued across input lines by escaping the newline character with a backslash (\). Comments begin with a pound sign (#) and end at the subsequent newline character.

Map Key Substitution

The ampersand (&) character is expanded to the value of the key field for the entry in which it occurs. In the following example,

```
baby
       austin:/toddler/&
```

the ampersand expands to `baby`.

Wildcard Key

When used in the key field, the asterisk (*) is recognized as a *catch–all* entry. The asterisk will match any key not previously matched. For instance, if the following entry appears in the indirect map for /**books**:

```
*
       &:/austin/books/&
```

the entry would allow automatic mounts in /**books** of any remote file system whose location was specified as

hostname:/austin/books/*hostname*

Multiple Mounts

A multiple mount entry takes the form:

```
key [–mount–options] [ [/mountpoint] [–mount–options] location ...  ] ...
```

The initial `/[mountpoint]` is optional for the first mount and mandatory for all subsequent mounts. The optional `mountpoint` is taken as a pathname relative to the directory named by **key**. If `mountpoint` is omitted in the first occurrence, a mountpoint of / (root) is implied. Given an entry in the indirect map for /**src**:

```
beta –ro\

     /          svr1,svr2:/export/src/beta \
     /1.0       svr1,svr2:/export/src/beta/1.0 \
     /1.0/man   svr1,svr2:/export/src/beta/1.0/man
```

All offsets must exist on the server under `beta.automount`, which will automatically mount `/src/beta`, `/src/beta/1.0`, and `/src/beta/1.0/man`, as needed, from either `svr1` or `svr2` (whichever is nearer and responds more quickly).

The initial / (forward slash) within `/[mountpoint]` is required. The optional mount point is taken as a path name relative to the destination of the symbolic link for key. If the `mountpoint` option is omitted in the first occurrence, a mount point of `/` (or the root directory) is assumed. An example of a direct map entry follows:

Other File System Types

The automounter assumes NFS mounts as a default file system type. Other file system types can be described using the **fstype** mount option. Other mount options specific to this file system type can be combined with the **fstype** option. The location field must contain information specific to the file system type. If the location field begins with a slash (/), a colon (:) must be prepended. For instance, to mount a CD file system:

```
cdrom –fstype=cdrfs,ro      :/dev/cd0
```

or to perform an **autofs** mount:

```
src   –fstype=autofs      auto_src
```

Indirect Maps

An indirect map allows you to specify mappings for the subdirectories you wish to mount under the directory indicated on the command line. In an indirect map, each **key** consists of a simple name that refers to one or more file systems that are to be mounted as needed.

Direct Maps

Entries in a direct map are associated directly with **autofs** mount points. Each **key** is the full pathname of an **autofs** mount point. The direct map, as a whole, is not associated with any single directory.

Special Maps

There are two special maps available: **–hosts** and **null**. The **–hosts** map is used with the /**net** directory and assumes that the map key is the host name of an NFS server. The **automountd** daemon dynamically constructs a map entry from the server's list of exported file systems. For instance, a reference to /**net**/**sales**/**usr** would initiate an automatic mount of all exported file systems from **sales** that are mountable by the client. References to a directory under /**net**/**sales** would refer to the corresponding directory relative to the **sales** root.

The **–null** map, when indicated on the command line, cancels a previous map for the directory indicated. This is most useful in the /**etc**/**auto_master** map for canceling entries that would otherwise be inherited from the **+auto_master** include entry. To be effective, the **–null** entries must be inserted before the included map entry.

Executable Maps

Note: Local maps that have the executable bit set in their file permissions will be executed by the automounter and provided with a key to be looked up as an argument.

The executable map is expected to return the content of an automounter map entry in its stdout or return no output if the entry cannot be determined.

Configuration and the Master (auto_master or auto.master) Map

When initiated without arguments, **automount** consults the master map for a list of **autofs** mount points and their maps. It mounts any **autofs** mounts that are not already mounted and unmounts **autofs** mounts that have been removed from the master map or direct map. The **automount** command may be initiated to establish mounts by specifying the map information on the command line. This information can be lost if the **automount** command is invoked a second time. For the sake of administration and debugging, the **automount** command will write a reference map file that reflects the map information specified on the command line. The file is generated on *each* invocation of the command and is called /**etc**/**autofs_cmdline**.

Note: The master map can be called **auto_master** or and **auto.master**. If **auto_master** isn't found, NIS looks for **auto.master**.

Included Maps

The contents of another map can be included within another map by entering:

```
+mapname
```

The `mapname` can be a file name or the name of a Network Information Service (NIS) map. Or, the `mapname` can be one of the special maps described in the following section. If the key being searched for is not located in an included map, the search continues with the next entry.

Managing NIS automount Maps

Prerequisites

NIS must be configured on your network.

Procedure

1. Edit the **/etc/auto.master** file. The **automount** daemon, by default, reads the NIS **/etc/auto.master** map to find which directories to watch for mounts. The **auto.master** map has the following format:

   ```
   DirectoryPath      AutomountMapName
   ```

   The *AutomountMapName* field specifies a file containing the **automount** map for the directory specified by the *DirectoryPath* field. For example, the contents of the **/etc/auto.master** file on the NIS server might be as follows:

   ```
   /home/home      /etc/auto.home
   /usr/lpp        /etc/auto.direct
   ```

   The above **auto.master** file entries will direct the **automount** daemon to use the `/etc/auto.home` **automount** map for the `/home/home` directory and the `/etc/auto.direct` **automount** map for the `/usr/lpp` directory.

2. Create the *AutomountMapName* files. The *AutomountMapName* files have the following format:

   ```
   Subdirectory   MountOptions   ServerName:ServerDirectory
   ```

   The *`Subdirectory`* field specifies a subdirectory of the *DirectoryPath* field directory of the **auto.master** file. For example, the contents of the `/etc/auto.home` file on the NIS client might be as follows:

   ```
   john            -rw,hard,intr    host1:/home/john
   bill            -rw,hard,intr    host3:/home/bill
   sally           -rw,hard,intr    host5:/home/sally
   fred            -rw,hard,intr    host9:/home/fred
   jane            -rw,hard,intr    host1:/home/jane
   ```

   The contents of the `/etc/auto.direct` file on the NIS client might be as follows:

   ```
   X11             -ro,hard,intr    lppserver:/usr/lpp/X11
   bsmEn_US        -ro,hard,intr    lppserver:/usr/lpp/bsmEn_US
   gnuemacs        -ro,hard,intr    lppserver:/usr/lpp/gnuemacs
   info            -ro,hard,intr    lppserver:/usr/lpp/info
   ```

3. Update the **/var/yp/Makefile** file. The following must be added to the **Makefile** file:

   a. Add **auto.master** to the `all:` listing.

   b. Add an entry for `$(DIR)/auto.master:` at the appropriate point in the file.

c. Add the following stanza to the **Makefile** file:

```
auto.master.time: $(DIR)/auto.master
        -@if [ -f $(DIR)/auto.master ] ; then \
                $(MAKEDBM) $(DIR)/auto.master
$(YPDBDIR)/$(DOM)/auto.master; \
                touch auto.master.time ; \
                echo ”updated auto.master” ; \
                if [ ! $(NOPUSH) ] ; then \
                        $(YPPUSH) auto.master ; \
                        echo ”pushed auto.master” ; \
                else \
                        : ; \
                fi \
        else \
                echo ”couldn’t find $(DIR)/auto.master” ; \
        fi
```

d. Add an entry for `auto.master: auto.master.time` at the appropriate point in the **Makefile** file.

In general, the same format that is used for the **netmasks** entry in the **Makefile** file can be used for the **auto.master** entry.

4. Build the **auto.master** map by executing the following command:

```
make auto.master
```

If errors are generated, check for improper configuration of NIS, errors in the **Makefile** file, or errors in the syntax of the **/etc/auto.master** file.

5. Start the **automount** daemon by executing the following command:

```
/usr/sbin/automount
```

This starts the **automount** daemon, which reads the **auto.master** NIS map.

In the preceding examples, when these procedures are completed, a user on the client can issue the `cd /home/home/bill` command and have the `/home/bill` directory mounted from the `host3` system onto the `/home/home/bill` directory.

Maintaining All of the automount Maps with NIS

In the first example, the `/etc/auto.home` and `/etc/auto.direct` were local files on the client that contained all of the **automount** map needed. The contents of the **automount** maps can also be maintained by NIS. The files would still exist on the client, but the contents would be different. For example, the `/etc/auto.home` file would contain the following:

```
+auto.home
```

And the **/etc/auto.direct** file would contain the following:

```
+auto.direct
```

This directs the **automount** daemon to consult the NIS maps `auto.home` and `auto.direct` when it reads the local files. The NIS server would contain two new NIS maps. The maps would be `auto.home` and `auto.direct`. They would be added to the **/var/yp/Makefile** in the same way that the **auto.master** NIS map was added. This makes them available for use by the NIS clients running the **automount** daemon.

This facility can also be used to define local portions of the **automount** maps and then refer to the NIS maps for the rest of the **automount** map. For example, the `/etc/auto.home` file could contain the following:

```
sandy          -rw,hard,intr             host10:/home/sandy
james          -rw,hard,intr             host2:/home/james
bill           -rw,hard,intr             host20:/home/bill
+auto.home
```

This **automount** map has three local entries and then it contains the NIS map `auto.home`. This way, local definitions can be maintained while taking advantage of the NIS map for the `/home/home` directory. The entry `bill` in the local map would appear in the `auto.home` NIS map. The local map entry will override the NIS map entry.

NIS Reference

See List of NIS Programming References in *AIX Communications Programming Concepts* for information about Network Information Service (NIS) subroutines and files.

Daemons

keyserv	Stores public and private keys.
ypbind	Enables client processes to bind, or connect, to NIS.
yppasswdd	Receives and executes requests from the **yppasswd** command.
ypserv	Looks up information in local NIS maps.
ypupdated	Updates information in NIS maps.

Commands

chkey	Changes the user's encryption key.
chmaster	Re–executes the **ypinit** daemon and restarts the NIS daemons.
chslave	Retrieves maps from a master server and restarts the **ypserv** daemon.
chypdom	Changes the current domain name of the system.
domainname	Displays or sets the name of the current domain.
keyenvoy	Provides an intermediary between user processes and the key server.
keylogin	Decrypts and stores the user's secret key.
lsmaster	Displays the characteristics of the configuration of an NIS master server.
lsnfsexp	Displays the characteristics of directories that are exported with NFS.
lsnfsmnt	Displays the characteristics of mounted NFS file systems.
makedbm	Makes an NIS map.
mkclient	Starts the **ypbind** daemons and uncomments the appropriate entries in the /**etc**/**rc.nfs** file.
mkkeyserv	Starts the **keyserv** daemon and uncomments the appropriate entries in the /**etc**/**rc.nfs** file.
mkmaster	Starts the NIS daemons on the master server.
mkslave	Retrieves maps from the NIS master server and starts the **ypserv** daemon.
newkey	Creates a new key in the **publickey** file.
revnetgroup	Reverses the listing of users and hosts in network group files in NIS maps.
rmkeyserv	Stops the **keyserv** daemon and comments the entry for the **keyserv** daemon in the /**etc**/**rc.nfs** file.
rmyp	Removes the configuration for NIS.
ypcat	Prints out the NIS map specified by the *MapName* parameter.
ypinit	Sets up NIS maps on an NIS server.

ypmatch	Displays the value of a given key within an NIS map.
yppasswd	Changes your network password in NIS.
yppoll	Displays the order number (ID number) of the NIS map currently in use on the server.
yppush	Prompts the NIS slave servers to copy updated NIS maps.
ypset	Directs a client machine to a specific server.
ypwhich	Identifies either the NIS server or the server that is the master for a given map.
ypxfr	Transfers an NIS map from an NIS server to a local host.

Chapter 3. Moving from NIS to NIS+

This chapter introduces the issues involved in converting from NIS to NIS+. It describes the changes required for moving to NIS+ and outlines a suggested transition process.

Other sections in this chapter cover the following key phases of transition:

Changes Required to Move to NIS+

NIS and NIS+ have several differences that have an impact on a transition. For example, NIS uses a flat, non–hierarchical namespace for each domain. NIS+ introduces a more complex domain hierarchy, similar to that of DNS. Before converting to NIS+, you must plan your networking needs and design the NIS+ namespace.

Also, NIS+ provides security, which limits access not only to the information in the namespace but also to the structural components of the namespace. Users, groups, access rights, and other security issues should be analyzed.

As implied by the previous paragraphs, NIS+ is not simply an upgrade to NIS but an entirely new product. Therefore, the transition from NIS to NIS+ is largely directed by the differences between the products, and understanding these differences is critical to a successful transition to NIS+. Differences are described in Differences Between NIS and NIS+.

Suggested Transition Phases

The following outline is a suggested NIS–to–NIS+ transition:

1. Review basic transition principles.
2. Become familiar with NIS+.
3. Design your final NIS+ namespace.
4. Select security measures.
5. Decide how to use NIS–compatibility mode.
6. Complete prerequisites to transition.
7. Implement the transition.

Transition Principles

Before you begin the transition, you should review the following basic principles:

- Consider the alternatives to making the transition immediately. You can defer the upgrade to NIS+ until after your site has completed its transition to the AIX Version 4.3.3 release. This would allow you to focus your resources on one transition effort at a time.
- Keep things simple. You can take several steps to simplify the transition. While these steps will diminish the effectiveness of NIS+, they will consume fewer servers and less administrative time. Once the transition is complete, you can change the NIS+ setup to better suit your needs. Here are some suggestions:
  - Do not change domain names unless they violate name restrictions. See Naming Conventions.
  - Do not use any hierarchies; keep a flat NIS+ namespace.
  - Use the NIS–compatibility features.
  - Use default tables and directory structures.
  - Do not establish credentials for clients.
- Minimize impact on client users. There are two major user–related considerations. First, users should not notice any change in service. Second, the transition phase itself should cause minimal disruption to client users. To ensure the second consideration, be sure the administrators responsible for each domain migrate their client machines to NIS+, rather than ask the users to implement the migration. This ensures that proper procedures are implemented, that procedures are consistent across client machines, and that irregularities can be dealt with immediately by the administrator.
- Avoid known mistakes.
  - Do not change the name services currently provided by NIS or the way NIS functions.
  - Do not change the structure of DNS.
  - Do not change the IP network topology.
  - Do not upgrade applications that use NIS to NIS+; leave the migration to NIS+ APIs for the future.
  - Do not consider additional uses for NIS+ during the implementation phase; add them later.

Become Familiar with NIS+

Familiarize yourself with NIS+, particularly with the concepts summarized in this section.

One of the best ways to become familiar with NIS+ is to build a prototype namespace. There is no substitute for hands–on experience with the product; administrators need the opportunity to practice in a forgiving test environment.

Note: Do not use your prototype domain as the basis for your actual running NIS+ namespace. Deleting your prototype when you have learned all you can from it will avoid namespace configuration problems. Start anew to create the real namespace after following all the planning steps.

When you create the test domains, make small, manageable domains. For guidance, you can use Prerequisites for Installing and Configuring NIS+, on page 5-2, which describes how to plan and create a simple test domain and subdomain (with or without NIS–compatibility mode) using the NIS+ setup scripts.

Note: The NIS+ scripts described in NIS+ Setup Scripts, on page 5-7 are the recommended method of setting up an NIS+ namespace. The recommended procedure is to first set up your basic NIS+ namespace using the scripts, and then customize that namespace for your particular needs using the NIS+ command set.

Design Your Final NIS+ Namespace

Design the final NIS+ namespace, following the guidelines in Designing the NIS+ Namespace, on page 3-5. While designing the namespace, do not worry about limitations imposed by the transition from NIS. You can add those later, once you know what your final NIS+ goal is.

Plan Security Measures

NIS+ security measures provide a great benefit to users and administrators, but they require additional knowledge and setup steps on the part of both users and administrators. They also require several planning decisions. Planning NIS+ Security Measures, on page 3-17, describes the implications of NIS+ security and the decisions you need to make for using it in your NIS+ namespace.

Decide How to Use NIS–Compatibility Mode

The use of parallel NIS and NIS+ namespaces is virtually unavoidable during a transition. Because of the additional resources required for parallel namespaces, try to develop a transition sequence that reduces the amount of time your site uses dual services or the extent of dual services within the namespace (for example, convert as many domains as possible to NIS+ only).

Using NIS–Compatibility Mode, on page 3-23, explains the transition issues associated with the NIS–compatibility mode and suggests a way to make the transition from NIS, through NIS compatibility, to NIS+ alone.

Complete Prerequisites to Transition

In addition to the planning decisions mentioned above, you must complete several miscellaneous prerequisites, as described in Prerequisites to Transition, on page 3-28.

Implement the Transition

The section on Implementing the Transition, on page 3-33 provides suggested steps to implement the transition you have planned in the previous steps.

Designing the NIS+ Namespace

This section provides general guidelines and recommendations for designing the final NIS+ namespace your site will have.

- Identifying the Goals of Your Administrative Model, on page 3-5
- Designing the Namespace Structure, on page 3-5
- Selecting the Namespace Servers, on page 3-10
- Determine Table Configurations, on page 3-12
- Resolving User/Host Name Conflicts, on page 3-16

When designing the namespace, do not worry about limitations imposed by the transition from NIS. You can modify your NIS+ domain later, once you know what your final NIS+ configuration will look like.

Identifying the Goals of Your Administrative Model

Select the model of information administration, such as the domain structure, that your site will use. Without a clear idea of how information at your site will be created, stored, used, and administered, it is difficult to make the design decisions suggested in this section. You could end up with a design that is more expensive to operate than necessary. You also run the risk of designing a namespace that does not suit your needs. Changing the namespace design after it has been set up is costly.

Designing the Namespace Structure

Designing the NIS+ namespace is one of the most important tasks you can perform, since changing the domain structure after NIS+ has been set up is a time–consuming, complex job. It is complex because information, security, and administration policies are woven into the domain structure of the namespace. Rearranging domains would require rearranging information, reestablishing security, and recreating administration policies.

When designing the structure of an NIS+ namespace, consider the following factors, which are discussed in the following sections:

- Domain Hierarchy
- Domain Names
- The Electronic Mail Environment

Domain Hierarchy

The main benefit of an NIS+ domain hierarchy is that it allows the namespace to be divided into more easily managed components. Each component can have its own security, information management, and administration policies. It is advisable to have a hierarchy if the number of clients you have exceeds 500, if you want to set up different security policies for a set of users, or if you have geographically distributed sites.

Unless there is a need for a domain hierarchy, not having a hierarchy will simplify your transition to NIS+ because the NIS+ servers for each subdomain are not part of the subdomain that they serve, with the exception of the root domain. The NIS+ servers are in the parent domain of the subdomain they serve. This relationship of server to subdomain creates problems for applications that expect the servers to be able to get their name service data from the subdomain. For example, if a subdomain NIS+ server is also an NFS server, then the server would not get its netgroups information from the subdomain, but instead retrieve the information from its domain, which is the domain above the subdomain; this can be confusing. Another example of when a hierarchy could cause problems would be where the NIS+ server was also used by users to log in remotely and to execute certain commands that they could not execute from their own workstations. If you have only a single root domain, you will not have these problems because NIS+ root servers live in the domain that they serve.

When all users are in the same NIS domain, they are directly visible to each other without using fully qualified names. Creating an NIS+ hierarchy, however, puts users in separate domains, which means that the users in one domain are not directly visible to users in another domain unless you use fully qualified names or paths.

For example, if there are two subdomains, `sales.wiz.com` and `factory.wiz.com`, created out of the earlier `wiz.com` domain, then for user `juan` in the `sales.wiz.com` domain to be able to send mail to user `myoko` in `factory.wiz.com`, he would have to specify her name as `myoko@hostname.factory.wiz.com` (or `myoko@hostname.factory`) instead of just `myoko`, as was sufficient when they were in the same domain. Remote logins also require fully qualified names between domains.

You could use the table path to set up connections between tables in one domain and another domain, but to do so would negate the advantages of having a domain hierarchy. You would also be reducing the reliability of the NIS+ service because now clients would have to depend upon the availability of not only their own home domains, but also of other domains to which their tables are pathed. Using table paths may also slow request response time.

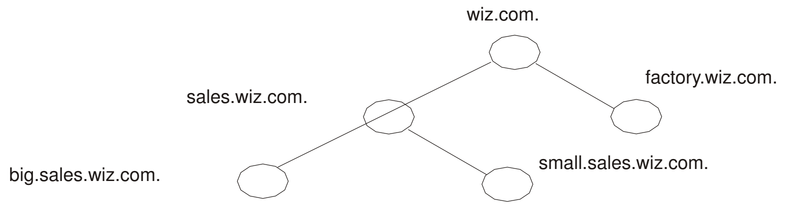
If you are unfamiliar with domain hierarchies, read NIS+ Namespace and Structure before designing your domain hierarchy. It describes NIS+ domain structure, information storage, and security.

Once you are familiar with the components of a domain hierarchy, make a diagram of how you expect the hierarchy to look when you are finished. The diagram will be a useful reference when you are in the midst of the setup procedure. At a minimum, you need to consider the following issues:

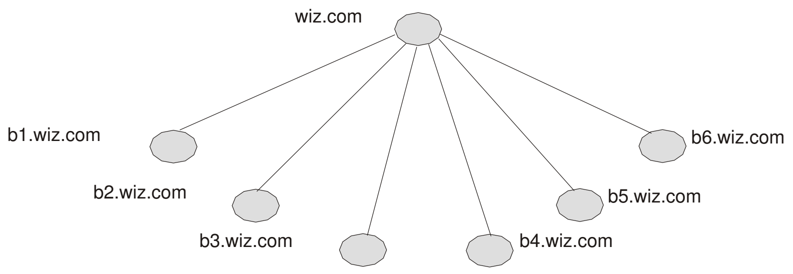
- Organizational or geographical mapping, on page 3-6
- Connection to higher domain, on page 3-7
- Client support in the root domain, on page 3-7
- Domain size compared to number of domains, on page 3-7
- Number of levels, on page 3-8
- Security levels, on page 3-8
- Replicas and number of replicas, on page 3-8
- Domains across time zones, on page 3-8
- Information management, on page 3-9

Organizational or Geographical Mapping

One of the major benefits of NIS+ is its capability of dividing the namespace into smaller, manageable parts. You could create a hierarchy of organizations, such as those of the hypothetical corporation, Wizard, Inc. (See figure.)



You could also organize the hierarchy by buildings instead of organizations, as shown in the following figure.



The scheme you select depends primarily on how you prefer to administer the namespace and how clients will tend to use the namespace. For example, if clients of `factory.wiz.com` will be distributed throughout the buildings of Wizard, Inc., you should not organize the namespace by building. Since the clients would constantly need to have access to other domains, you would need to add their credentials to the other domains and you would increase traffic flow through the root master server. A better scheme would be to arrange clients by organization. On the other hand, building–sized domains are immune to the reorganizations that require organization–based domains to be restructured.

You are not limited by the physical layout of the network, since an NIS+ namespace does not have to be congruent with the physical network, except where it has to support NIS clients. The number of domains your namespace needs depends on the kind of hierarchy you select.

Consider future expansion plans. Will today's NIS+ root domain be beneath another NIS+ domain in the future? Changing this arrangement would entail a great deal of work. Try to estimate the need for future domains in the namespace and design a structure that can accommodate them without disruption.

Connection to Higher Domains

Consider whether the NIS+ namespace will be connected to higher domains, such as those of the Internet or DNS. If you currently use NIS under a DNS hierarchy, do you want to replace only the NIS domains or do you want to replace the entire companywide DNS/NIS structure with an NIS+ namespace?

Client Support in the Root Domain

In the organizational and building–based domain hierarchies illustrated earlier, are all the clients placed in domains beneath the root domain? Or do some belong to the root domain? Is the purpose of the root domain to act only as the root for its subdomains or will it support its own group of clients? You could place all clients in the lowest layer of domains and only those used for administration in the root and any intermediate domains. For example, if you implemented the organizational domain plan, all clients would belong to the `big.sales.wiz.com`, `small.sales.wiz.com`, and `factory.wiz.com` domains, and only clients used for administration would belong to the `wiz.com` and `sales.wiz.com` domains.

Or you could place the clients of general–purpose departments in higher–level domains. For example, if the domain is organized by building, you could put the clients of the Facilities Department in the `wiz.com` domain. It is not recommended that you do so, however, because the root domain should be kept simple and relatively unpopulated.

Domain Size Compared With Number of Domains

The current NIS+ implementation is optimized for up to 1000 NIS+ clients per domain and for up to 10 replicas per domain. Such a domain would typically have 10,000 table entries. The limitations come from the current server discovery protocol. If you have more than 1000 NIS+ clients, you should divide your namespace into different domains and create a hierarchy.

Creating a hierarchy, however, may introduce more complexity than you are prepared to handle. You may still prefer to create larger domains rather than a hierarchy because one large domain requires less administration than multiple smaller domains do. Larger domains need fewer skilled administrators to service them, since tasks can be automated more readily (with scripts you create), thus lowering the administrative expense. Smaller domains provide better performance, and you can customize their tables more easily. You also achieve greater administrative flexibility with smaller domains.

Number of Levels

NIS+ was designed to handle multiple levels of domains. Although the software can accommodate almost any number of levels, a hierarchy with too many levels would be difficult to administer. For example, the names of objects could become long and unwieldy. Consider 20 to be the limit for the number of subdomains for any one domain and limit the levels of the NIS+ hierarchy to 5.

Security Level

Typically, you will run the namespace at security level 2. However, if you plan to use different security levels for different domains, you should identify them now. The section on Planning NIS+ Security Measures, on page 3-17, provides more information about security levels.

Replicas and Number of Replicas

Any one domain should have no more than 10 replicas because of the increased network traffic and server load that occur when information updates are propagated to the replicas. Determining the number of replicas a domain requires depends on other factors as well. They are:

- The physical location of the servers
- The number of subnets in a domain
- Whether there are NIS clients in the NIS+ namespace

You should create a minimum of two servers (one master and one replica) for every domain and at least one replica for every physical location. You do not need a replica for every subnet. NIS clients do not have access to servers that are not on the same subnet. The only exceptions are the NIS clients, which can use **ypinit** to specify a list of NIS servers. The netmask number in these cases would have to be set appropriately.

One way you can have a sufficient number of replicas per domain without using a multiplicity of machines is to create multihomed servers. A multihomed server is a machine with multiple ethernet or network interfaces. A multihomed server can serve multiple subnets in a domain.

If the domain hierarchy that you design spans a wide area network (WAN) link, it would be prudent to replicate the domain on either side of the WAN link—with a master server on one side and a replica on the other. This could possibly enable clients on the other side of the link to continue with NIS+ service even if the WAN link were temporarily disabled. Putting servers on either side of a WAN, however, does change the structure of a namespace that is organized by group function rather than by physical layout, since the replica might physically reside within the geographic perimeter of a different domain.

Domains Across Time Zones

Geographically dispersed organizations may determine that organizing their domain hierarchy by functional groups would cause a domain to span more than one time zone. It is *strongly* recommended that you do *not* have domains that span multiple time zones. If you do need to configure a domain across time zones, be aware that a replica's time will be taken from the master server, so the database updates will be synchronized properly, using Greenwich mean time (GMT). This may cause problems if the replica machine is used for other services that are time critical. To make domains across time zones work, the replica's timezone has to be locally set to the master server's time zone when you are installing

NIS+. Once the replica is running, some time critical programs may run properly and some may not, depending on whether these programs use universal or local time.

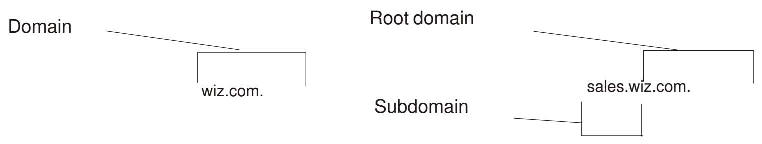
Information Management

It is best to use a model of local administration within centralized constraints for managing the information in an NIS+ namespace. Information should be managed, as much as possible, from within its home domain, but according to guidelines or policies set at the global namespace level. This provides the greatest degree of domain independence while maintaining consistency across domains.

Domain Names

Consider name length and complexity. First, choose names that are descriptive. For example, `Sales` is considerably more descriptive than `BW23A`. Second, choose short names. To make your administrative work easier, avoid long names such as `EmployeeAdministrationServices.WizardCorporation`.

A domain name is formed from left to right, starting with the local domain and ending with the root domain. (See figure.) Unlike NIS, an NIS+ domain name is case insensitive.



The root domain must always have at least two labels and must end in a dot. The second label can be an Internet domain name, such as `com`.

Also consider implications of particular names for electronic mail domains, both within the company and over the Internet.

Depending on the migration strategy chosen, a viable alternative could be to change domain names on NIS to the desired structure, then migrate to NIS+ domain by domain.

The Electronic Mail Environment

Because NIS+ can have a domain hierarchy while NIS has a flat domain space, changing to NIS+ can have effects on your mail environment. With NIS, only one mail host is required. If you use a domain hierarchy for NIS+, you may need one mail host for each domain in the namespace because names in separate domains may no longer be unique.

Therefore, the e–mail addresses of clients who are not in the root domain may change. As a general rule, client e–mail addresses can change when domain names change or when new levels are added to the hierarchy.

NIS+ provides several **sendmail** enhancements to make the task easier. In addition, NIS+ provides a **sendmailvars** table. The **sendmail** program first looks at the **sendmailvars** table (see the following table), then examines the local **sendmail.cf** file.

Note: Be sure that mail servers reside in the NIS+ domain whose clients they support. For performance reasons, do not use paths to direct mail servers to tables in other domains.

Consider the impact of the new mail addresses on DNS. You may need to adjust the DNS MX records.

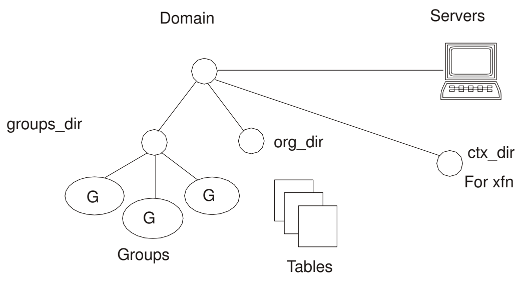
Selecting the Namespace Servers

Each NIS+ domain is supported by a set of NIS+ servers. The servers store the domain's directories, groups, and tables, and answer requests for access from users, administrators, and applications. Each domain is supported by only one set of servers. However, a single set of servers can support more than one domain.

Remember that a domain is not an object, but a reference to a collection of objects. Therefore, a server that supports a domain is not actually associated with the domain but with the domain's directories. A domain consists of four directories:

- *domain*
- **org_dir.***domain*
- **groups_dir.***domain*

as shown in the following figure.



Any AIX Version 4.3.3–based workstation that is installed with the appropriate NIS+ filesets can be an NIS+ server as long as it has available file system space. The software for both NIS+ servers and clients is included in the AIX product. Therefore, any workstation that has an AIX Version 4.3.3 release installed can become a server or a client, or both.

When selecting the servers that will support the NIS+ namespace, consider the following factors, discussed in the following sections:

- Supported Domains
- Server Load
- Disk Space and Memory Requirements

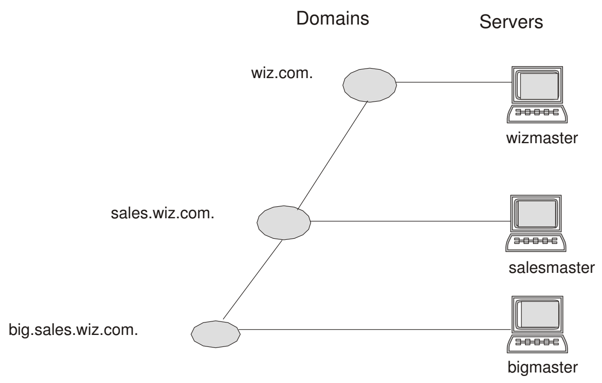
Supported Domains

When selecting servers, you must differentiate between the requirements imposed by the NIS+ service and those imposed by the traffic load of your namespace.

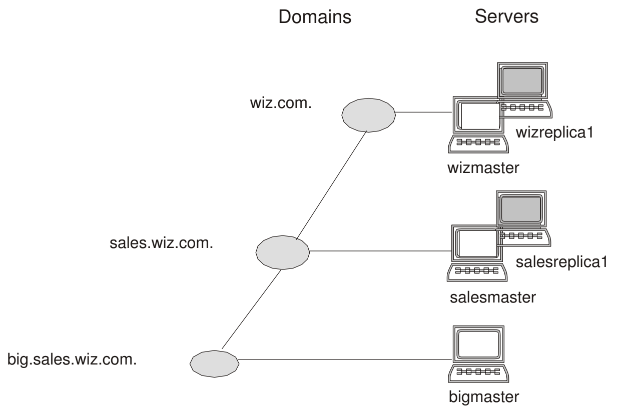
The NIS+ service requires you to assign at least one server, the master, to each NIS+ domain. (An NIS+ server is capable of supporting more than one domain, but use the one server for one domain configuration in small namespaces or testing situations.) How many other servers a domain requires is determined by the traffic load, the network configuration, and whether NIS clients are present.

The traffic loads you anticipate will determine the total number of servers used to support the namespace, how much storage and processing speed each will require, and whether a domain needs replicas to ensure its availability. How can you determine how many servers you need?

A good way to begin is by assigning one master server to each domain in the hierarchy.



If certain domains must always be available, add two or more replicas to them. Two replicas allow requests to still be answered even if one of the replicas is damaged. Requests may not be answered in a timely manner if a master has only one replica and that replica is being repaired or updated. A domain with only one replica loses 50 percent of its load capacity when the replica is down. Always add at least one replica to a domain. In small to medium domains, configurations with two to four replicas are normal.



In organizations with many distributed sites, each site often needs its own subdomain. Typically, the subdomain master is placed in a higher–level domain. As a result, there can be a great deal of traffic between point–to–point links. Creating local replicas can speed request response and minimize point–to–point traffic across the link. In this configuration, lookups may be handled locally.

Server Load

NIS+ master servers require fewer replicas than NIS servers did, since NIS+ does not depend on broadcasts on the local subnet.

Putting replicas on both sides of a weak network link (such as WAN links) is recommended. If the link breaks and the networks are decoupled, both sides of the network can still obtain service.

Do not put more than 10 replicas on one domain. If you can, put one on each subnet; otherwise, distribute the servers as best you can and optimize for the best performance. You do not need NIS+ servers on every subnet, unless they support NIS clients. In such cases, you may want to install NIS+ servers on multihomed machines.

Try to keep fewer than 1000 clients in a domain. NIS+ clients present a higher load on servers than NIS clients. A large number of clients served by only a few servers may impact network performance.

Disk Space and Memory Requirements

How much disk space you need depends on four factors:

- Disk space consumed by the base AIX Version 4.3.3 software
- Disk space for /**var**/**nis** (and /**var**/**yp**)
- Amount of memory
- Swap space required for NIS+ server processes

The base AIX Version 4.3.3 software requires at least 32 MBs of disk space. You should also count the disk space consumed by other software the server may use. For more details on the base operating system installation and requirements, see the *AIX 4.3 Installation Guide*.

Although NIS+ is part of the AIX Version 4.3.3 distribution, it is not automatically installed in the base installation. NIS+ directories, groups, tables, and client information are stored in /**var**/**nis**. The /**var**/**nis** directory uses about 5 KBs of disk space per client. For example purposes only, if a namespace has 1000 clients, /**var**/**nis** requires about 5 MBs of disk space. However, because transaction logs (also kept in /**var**/**nis**) can grow large, you may want additional space per client—an additional 10–15 MBs is recommended. In other words, for 1000 clients, allocate 15 to 20 MBs for /**var**/**nis**. You can reduce this if you checkpoint transaction logs regularly.

If you will use NIS+ concurrently with NIS, allocate space equal to the amount you are allocating to /**var**/**nis** for /**var**/**yp** to hold the NIS maps that you transfer from NIS.

You also need swap space equal to three times or more of the size of the NIS+ server process—in addition to the server's normal swap space requirements. The size of the **rpc.nisd** process as shown by the **ps –efl** command. Most of this space is used during callback operations or when directories are checkpointed (with **nisping –C**) or replicated, because during such procedures an entire NIS+ server process is forked.

Determine Table Configurations

NIS+ tables provide several features not found in simple text files or maps. They have a column–entry structure, accept search paths, can be linked together, and can be configured in several different ways. You can also create your own custom NIS+ tables. When selecting the table configurations for your domains, consider the following factors discussed in the following sections:

- Differences Between NIS+ Tables and NIS Maps
- Use of Custom NIS+ Tables
- Connections Between Tables

Differences Between NIS+ Tables and NIS Maps

NIS+ tables differ from NIS maps in many ways, but two of those differences should be kept in mind during your namespace design:

- NIS+ uses fewer standard tables than NIS
- NIS+ tables interoperate with /**etc** files differently than NIS maps do.

Review the standard NIS+ tables to make sure they suit the needs of your site. They are listed in the following table.

Note the new names of the automounter tables:

- auto_home (old name: **auto.home**)
- auto_master (old name: **auto.master**)

Table 4. Standard NIS+ Tables

NIS+ Table	Information in the Table
hosts	Network address and host name of every workstation in the domain
bootparams	Location of the root, swap, and dump partition of every diskless client in the domain
passwd	Password information about every user in the domain
cred	Credentials for principals who belong to the domain
group	The group password, group ID, and members of every group in the domain
netgroup	The netgroups to which workstations and users in the domain may belong
mail_aliases	Information about the mail aliases of users in the domain
timezone	The time zone of the domain
networks	The networks in the domain and their canonical names
netmasks	The networks in the domain and their associated netmasks
ethers	The ethernet address of every workstation in the domain
services	The names of IP services used in the domain and their port numbers
protocols	The list of IP protocols used in the domain
rpc	The RPC program numbers for RPC services available in the domain
auto_home	The location of all user's home directories in the domain
auto_master	Automounter map information
sendmailvars	The mail domain
client_info	Information about NIS+ clients

The following table lists the correspondences between NIS maps and NIS+ tables. Do not worry about synchronizing related tables. The NIS+ tables store essentially the same information as NIS maps, but they consolidate similar information into a single table (for example, the NIS+ hosts table stores the same information as the **hosts.byaddr** and **hosts.byname** NIS maps). Instead of the key–value pairs used in NIS maps, NIS+ tables use columns and rows. (See NIS+ Tables and Information, on page 4-18.) Key–value tables have two columns, with the first column being the key and the second column being the value. Therefore, when you update any information, such as host information, you need only update it in one place, such as the hosts table. You no longer have to worry about keeping that information consistent across related maps.

The dots were changed to underscores because NIS+ uses dots to separate directories. Dots in a table name will cause NIS+ to mistranslate names. For the same reason, machine names cannot contain any dots. You must change any machine name that contains a dot to something else. For example, a machine named `sales.alpha` is not allowed. You could change it to `sales_alpha` or `salesalpha` or any other name that does not contain a dot.

To make the transition from NIS to NIS+, you must change the dots in your NIS automounter maps to underscores. You may also need to do this on your clients' automounter configuration files.

Table 5. Correspondences Between NIS Maps and NIS+ Tables

NIS Map	NIS+ Table	Notes
auto.home	auto_home	
auto.master	auto_master	
bootparams	bootparams	
ethers.byaddr	ethers	
ethers.byname	ethers	
group.bygid	group	Not the same as NIS+ groups
group.byname	group	Not the same as NIS+ groups
hosts.byaddr	hosts	
hosts.byname	hosts	
mail.aliases	mail_aliases	
mail.byaddr	mail_aliases	
netgroup	netgroup	
netgroup.byhost	netgroup	
netgroup.byuser	netgroup	
netid.byname	cred	
netmasks.byaddr	netmasks	
networks.byaddr	networks	
networks.byname	networks	
passwd.byname	passwd	
passwd.byuid	passwd	
protocols.byname	protocols	
protocols.bynumber	protocols	
publickey.byname	cred	
rpc.bynumber	rpc	
services.byname	services	
ypservers		Not needed

Use of Custom NIS+ Tables

Determine which nonstandard NIS maps you use and their purpose. Can they be converted to NIS+ or replaced with NIS+ standard maps?

Some applications may rely on NIS maps. Will they still function the same way with NIS+, and can they function correctly in a mixed environment?

To build a custom table in NIS+, use **nistbladm**. Remember that you cannot use dots in the table names.

If you want to use NIS+ to support your custom NIS maps, you should create a key–value table, a table with two columns. The first column is the key and the second column is the value. If you then run the NIS+ servers in NIS–compatibility mode, the NIS clients will not notice any change in functionality.

Connections Between Tables

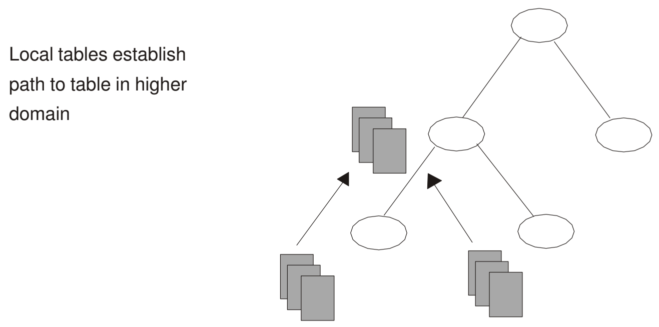
NIS+ tables contain information only about the resources and services in their home domain. If a client tries to find information that is stored in another domain, the client has to provide the other domain name. You can make this ”forwarding” automatic by connecting the local table to the remote table. NIS+ tables can be connected in two different ways:

- Through *paths*
- Through *links*

Paths and links should not be used if you are going to have NIS clients in the NIS+ namespace, because NIS clients are unable to follow the paths or links to find the appropriate information.

Paths

If information in a particular NIS+ table is often requested by clients in other domains, consider establishing a path from the local NIS+ table to the one in the other domain.



Such a path would have two main benefits. First, it would save clients in lower domains the trouble of explicitly searching through a second table. Second, it would allow the administrator in the higher–level domain to make changes in one table and render that change visible to clients in other domains. However, such a path would also hurt performance.

Performance is especially affected when searches are unsuccessful, because the NIS+ service must search through two tables instead of one. When you use paths, a table lookup now also depends upon the availability of other domains. This dependence can reduce the net availability of your domain. For these reasons, use paths only if you do not have any other solution to your problem.

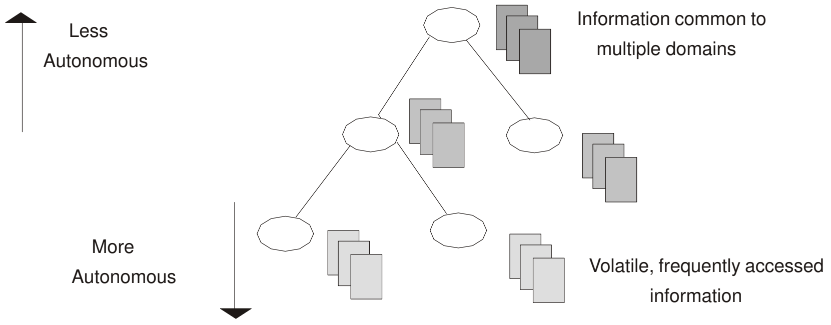
You should also be aware that since ”mailhost” is often used as an alias, when trying to find information about a specific mail host, you should use its fully qualified name in the search path (for example, `mailhost.sales.wiz.com`); otherwise NIS+ will return all the ”mailhosts” it finds in all the domains it searches through.

The path is established in the local table, with the **–p** option to the **nistbladm** command. To change a table's path, you must have modify access to the table object. To find out what a table's search path is, use the **niscat –o** command (you must have read access to the table).

Links

Links between tables produce an effect similar to paths, except that the link involves a search through only one table: the remote table. With a search path, NIS+ first searches the local table, and only if it is unsuccessful does it search the remote table. With a link, the search moves directly to the remote table. In fact, the remote table virtually replaces the local table.

The benefit of a link is that it allows a lower domain to access the information in a higher domain (see figure) without the need to administer its own table.



To create a link, use the **nisln** command. You must have modify rights to the table object.

Deciding whether to use a path or to link NIS+ tables in a domain is a complex decision, but here are some basic principles:

- Every domain must have access to every standard table.
- Volatile, frequently accessed data should be located lower in the hierarchy. Such data should be located closer to where it is used most often.
- Data that is accessed by several domains should be located higher in the hierarchy, unless the domains need to be independent.
- The lower in the hierarchy you place data, the easier it will be to administer autonomously.
- Only NIS+ clients can see tables connected by paths and links. They cannot be seen by NIS clients.

Resolving User/Host Name Conflicts

NIS+ cannot distinguish between a human principal and a workstation principal when requests are made. Therefore, all user names must be different from machine names in the same namespace. In other words, within a given namespace no user can have the same user name as a machine name, and no machine can have the same name as any user ID.

For example, under NIS it was acceptable to have a user with the login name of `irina` whose local machine is also named `irina`. Her network address would be `irina@irina`. This is not allowed under NIS+. When the site is converted to NIS+. either the user will have to change her login name or her machine name will have to be changed. Identical user and machine names are a problem even when the machine with the duplicate name does not belong to the user with the same name. The following examples illustrate duplicate name combinations not valid with NIS+:

- `jane@jane` in the same namespace
- `patna@peshawar` and `rani@patna` in the same namespace

The best solution to this problem is to check all /**etc** files and NIS maps before you use the data to populate NIS+ tables. If you find duplicate names, change the machine names rather than the login names, and later create an alias for the machine’s old name.

Planning NIS+ Security Measures

This section provides general guidelines and recommendations for making choices about security in your namespace. This section contains the following topics:

- Understanding the Impact of NIS+ Security, on page 3-17
- Selecting Credentials, on page 3-18
- Choosing a Security Level, on page 3-18
- Establishing Password–Aging Criteria, Principles, and Rules, on page 3-19
- Planning NIS+ Groups, on page 3-19
- Planning Access Rights to NIS+ Groups and Directories, on page 3-20
- Planning Access Rights to NIS+ Tables, on page 3-21

Understanding the Impact of NIS+ Security

Because NIS+ provides security that NIS did not, NIS+ security requires more administrative work. It may also require more work from users who are not used to performing **chkey**, **keylogin**, or **keylogout** procedures. Furthermore, the protection provided by NIS+ is not entirely secure. Given enough computing power and the right knowledge, the Diffie–Hellman public–key cryptography system can be broken. In addition, the secret key stored with the key server process is not automatically removed when a credentialed nonroot user logs out unless that user logs out with **keylogout**. (See the **keylogout** command description for more information.) The root user's key, created by **keylogin –r** and stored in /**etc**/**.rootkey**, remains until the **.rootkey** file is explicitly removed. The superuser cannot use **keylogout**.

How NIS+ Security Affects Users

NIS+ security benefits users because it improves the reliability of the information they obtain from NIS+, and it protects their information from unauthorized access. However, NIS+ security requires users to learn about security and requires them to perform some administrative steps.

Although NIS+ requires a network login, users are not required to perform an additional key login because the **login** command automatically gets the network keys for the client when the client has been correctly configured. Clients are correctly configured when their login password and their Secure RPC password are the same. The secret key for the user **root** is normally made available in the /**etc**/**.rootkey** file. When the NIS+ user password and credential are changed with the **passwd** command, the credential information is automatically changed for the user.

- To change the NIS+ machine's local root password, run the **passwd** command.
- To change the root credential, run the **chkey** command.

If your site allows users to maintain passwords in their local /**etc**/**passwd** files in addition to their Secure RPC passwords, and if these passwords are different from the Secure RPC passwords, then users must run **keylogin** each time they run **login**. For more information, see Administering Passwords, on page 6-36.

How NIS+ Security Affects Administrators

Because AIX Version 4.3.3 includes the DES encryption mechanism for authentication, administrators who need secure operation do not need to purchase a separate encryption package. However, administrators must train users how and when to use the **passwd** command.

Furthermore, setting up a secure NIS+ namespace is more complex than setting up a namespace without any security. The complexity comes not only from the extra steps required to set up the namespace, but from the job of creating and maintaining user and machine credentials for all NIS+ principals. Administrators have to remove obsolete credentials just as they remove inactive account information from the passwd and hosts tables. Also, when servers' public keys change, administrators have to update the keys throughout the namespace (using **nisupdkeys**). Administrators also have to add LOCAL credentials for users from other domains who want to log in remotely to this domain and have authenticated access to NIS+.

How NIS+ Security Affects Transition Planning

After you become familiar with the benefits and the administrative requirements of NIS+ security, you must decide whether to implement NIS+ security during or after the transition. It is recommended that you use full NIS+ security even if you operate some or all servers in a domain in NIS–compatibility mode. (All servers in a domain should have the same NIS–compatibility status.) However, this entails a heavy administrative burden. If you prefer a simpler approach, set up the NIS+ servers and namespace with NIS–compatible security, but decline to create credentials for NIS+ clients. Administrators and servers would still require credentials. The NIS+ clients would be relegated to the **nobody** class, along with the NIS clients. This reduces training and setup requirements, but it has the following drawbacks:

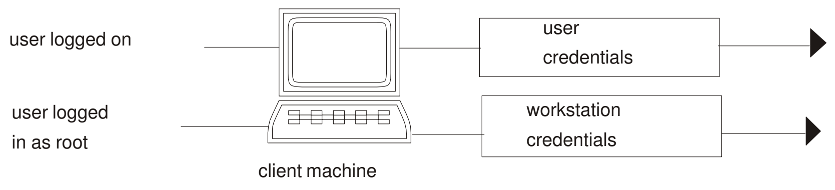
- Users lose the ability to update any NIS+ tables (but they retain their ability to change their login password).
- Users will not be able to verify that the name service information is coming from an authenticated NIS+ server.

Selecting Credentials

NIS+ provides two types of credential: LOCAL and DES. All NIS+ principals need at least one of these credentials. When the namespace is running at security level 2 (the default), all NIS+ principals (clients) must have DES credentials in their home domains. In addition, all users (not workstations) must have LOCAL credentials in their home domains and in every other domain for which they need login access.

To determine the credential needs of your namespace, consider the type of principal and the type of credential.

NIS+ principals can be users or the superuser identity on the client workstation.



When you determine the credentials you need to create, make sure you know which type of principal the credential is for. For instance, when you set up an NIS+ client with the **nisclient** script, you will create credentials for both the workstation and for the corresponding user. All users on that workstation must run the **nisclient** script to be configured as NIS+ users. Unless credentials for the user are also created, the user would only have the access rights granted to the **nobody** class. If you do not give some access rights to the **nobody** class, the namespace will not be available to users.

Note: The **nisclient** script fails unless the user has existing entries in the **passwd** and **cred** tables.

Choosing a Security Level

NIS+ is designed to be run at security level 2, which is the default. Security levels 0 and 1 are provided only for the purpose of testing and debugging. Do not run an operational network with real users at any level other than level 2. See Administering NIS+ Directories, on page 6-50 for more information on the NIS+ security levels.

Establishing Password–Aging Criteria, Principles, and Rules

Password–aging is a mechanism that you can use to force users to periodically change their passwords. Password–aging allows you to:

- Specify the maximum number of days that a password can be used before it has to be changed.
- Specify the minimum number of days that a password has to be in existence before it can be changed.
- Specify that a warning message be displayed whenever a user logs in a specified number of days before the user's password time limit is reached.
- Specify the maximum number of days that an account can be inactive. If that number of days pass without the user logging in to the account, the user's password will be locked.

Keep in mind that users who are already logged in when the various maximums or dates are reached are not affected by the above features. They can continue to work as normal. Password–aging limitations and activities are activated only when a user logs in or performs one of the following operations:

- **login**
- **rlogin**
- **telnet**
- **ftp**

These password–aging parameters are applied on a user–by–user basis, and you can have different password–aging requirements for different users. You can also set general default password–aging parameters that apply to all users except those you have individually set.

When planning your NIS+ namespace, decide which password–aging features you want to implement, and the default values you want to specify. For additional information on password–aging, see Administering Passwords, on page 6-36.

Planning NIS+ Groups

NIS+ uses groups as a means to provide NIS+ access rights to several NIS+ principals at one time; it is used only for NIS+ authorization.

An NIS+ group is one of the four authorization classes on which access rights are based. The four classes are:

Owner	Every NIS+ object has one owner who is a single user. The owner is usually the person who created the object, but ownership can be transferred to another user.
Group	A collection of users grouped together under a group name for the purpose of granting that collection of users specified NIS+ access rights.
World	All *authenticated* users. In other words, any user with valid DES credentials. By definition, an object's owner and members of an object's group are also part of the **world** class as long as their credentials are valid.
Nobody	Anyone who does not have a valid DES credential. If the credentials of some member of one of the other classes are not valid, missing, corrupted, or not found, then that user is placed in the **nobody** class.

The default name of the group created by NIS+ scripts for such purposes is the **admin** group. You can create other groups with different names and assign different groups to different NIS+ objects. This can be done as a default parameter by setting the environment variable **NIS_GROUPS** to whatever group you prefer.

Member users of an object's group usually have special privileges to that object, such as permission to make certain changes to the object. For example, you could add several junior administrators to an **admin** group so that they can only modify the passwd and hosts tables, but they would be unable to modify any other tables. By using an **admin** group, you can distribute administration tasks across many users and not just reserve them for the superuser of the entire hierarchy. The NIS+ **admin** group must have credentials created for its members even if you are running the domain in NIS–compatibility mode, because only authenticated users have permission to modify NIS+ tables.

After identifying the type of credentials you need, select the access rights that are required in the namespace. To make that task easier, first decide how many administrative groups you will need. Using separate groups is useful when you want to assign them different rights. Usually, you create groups by domain. Each domain should have only one admin group.

Planning Access Rights to NIS+ Groups and Directories

After arranging your principals into groups, determine the kind of access rights granted by the objects in the namespace to those groups, as well as to the other classes of principal (**nobody**, **owner**, **group**, and **world**). Planning these assignments ahead of time will help you establish a coherent security policy.

As shown in the following table, NIS+ provides different default access rights for different namespace objects.

Table 6. Default Access Rights for NIS+ Objects

Object	Nobody	Owner	Group	World
Root–directory object	r–––	rmcd	rmcd	r–––
Non–root directory object	r–––	rmcd	rmcd	r–––
groups_dir directory objects	r–––	rmcd	rmcd	r–––
org_dir directory objects	r–––	rmcd	rmcd	r–––
NIS+ groups	––––	rmcd	r–––	r–––
NIS+ tables (see Planning Access Rights to NIS+ Tables, on page 3-21)	varies	varies	varies	varies

You can use the default rights or assign your own. If you assign your own, consider how the objects in your namespace will be accessed. Keep in mind that the **nobody** class comprises all requests from NIS+ clients, whether authenticated or not. The **world** class comprises all authenticated requests from NIS+ clients. Therefore, if you do not want to provide namespace access to unauthenticated requests, do not assign any access rights to the **nobody** class; reserve them only for the **world** class. On the other hand, if you expect some clients—through applications, for instance—to make unauthenticated read requests, assign read rights to the **nobody** class. If you want to support NIS clients in NIS–compatibility mode, assign read rights to the **nobody** class.

Also consider the rights each type of namespace object will assign to the NIS+ groups you specified earlier. Depending on how you plan to administer the namespace, you can assign all or some of the available access rights to the group. A good solution is to have the user root on the master server be the owner of the admin group. The admin group should have create and destroy rights on the objects in the root domain. If you want only one administrator to create and modify the root domain, then put just that administrator in the admin group. You can always add additional members to the group. If several administrators may be involved in the setup process, put them all in the group and assign full rights to it. That is easier than switching ownership back and forth.

Finally, the owner of an object should have full rights, although this is not as important if the group does. A namespace is more secure if you give only the owner full rights, but it is easier to administer if you give the administrative group full rights.

Planning Access Rights to NIS+ Tables

NIS+ objects other than NIS+ tables are primarily structural. NIS+ tables, however, are informational objects. Access to NIS+ tables is required by all NIS+ principals and applications running on behalf of those principals. Therefore, their access requirements are somewhat different.

The following table lists the default access rights assigned to NIS+ tables. If any columns provide rights in addition to those of the table, they are also listed. You can change these rights at the table and entry level with the **nischmod** command, and at the column level with the **nistbladm –u** command. Protecting the Encrypted Passwd Field, on page 3-22 provides just one example of how to change table rights to accommodate different needs.

Table 7. Default Access Rights for NIS+ Tables and Columns

Table/Column	Nobody	Owner	Group	World
auto_home table	r——	rmcd	rmcd	r——
auto_master table	r——	rmcd	rmcd	r——
bootparams table	r——	rmcd	rmcd	r——
client_info table	——	rmcd	rmcd	r——
cred table	r——	rmcd	rmcd	r——
cname column	——	——	——	——
auth_type column	——	——	——	——
auth_name column	——	——	——	——
public_data column	——	–m—	——	——
private_data column	——	–m—	——	——
	r——	rmcd	rmcd	r——
ethers table				
group table	——	rmcd	rmcd	r——
name column	r——	——	——	——
passwd column	——	–m—	——	——
gid column	r——	——	——	——
members column	r——	–m—	——	——
hosts table	r——	rmcd	rmcd	r——
netmasks table	r——	rmcd	rmcd	r——
networks table	r——	rmcd	rmcd	r——
passwd table	——	rmcd	rmcd	r——
name column	r——	——	——	——
passwd column	——	–m—	——	——
uid column	r——	——	——	——
gid column	r——	——	——	——
gcos column	r——	rm—	r——	r——

Table/Column	Nobody	Owner	Group	World
home column	r——	——	——	——
shell column	r——	——	——	——
shadow column	——	–m—	——	——
protocols table	r——	rmcd	rmcd	r——
rpc table	r——	rmcd	rmcd	r——
sendmailvars table	——	rmcd	rmcd	r——
services table	r——	rmcd	rmcd	r——

Note: NIS–compatible domains give the **nobody** class read rights to the **passwd** table at the table level.

Protecting the Encrypted Passwd Field

As you can see in the previous table, read access is provided by default to the **nobody** class by all tables except the **passwd** table. NIS+ tables give the **nobody** class read access because many applications that need to access NIS+ tables run as unauthenticated clients. However, if this were also done for the passwd table, it would expose the encrypted passwd column to unauthenticated clients.

The configuration shown in the previous table is the default set of access rights for NIS–compatible domains. NIS–compatible domains must give the **nobody** class read access to the passwd column because NIS clients are unauthenticated and would otherwise be unable to access their passwd column. Therefore, in an NIS–compatible domain, even though passwords are encrypted, they are vulnerable to decoding. They would be much more secure if they were not readable by anyone except their owner.

Standard NIS+ domains (not NIS–compatible) provide that extra level of security. The default configuration (provided by **nissetup**) uses a column–based scheme to hide the passwd column from unauthenticated users while still providing access to the rest of the passwd table. At the table level, no unauthenticated principals have read access. At the column level, they have read access to every column except the passwd column.

Entry owners have both read and modify access to their own entries. They obtain read access by being a member of the **world** class. (Remember that at the table level, the **world** class has read rights.) They obtain modify access by explicit assignment at the column level.

Table owners and entry owners are rarely and not necessarily the same NIS+ principals. Thus, table–level read access for the owner does not imply read access for the owner of any particular entry.

For a more complete explanation and discussion of table–, entry–, and column level–security, see Security, on page 7-1.

Using NIS–Compatibility Mode

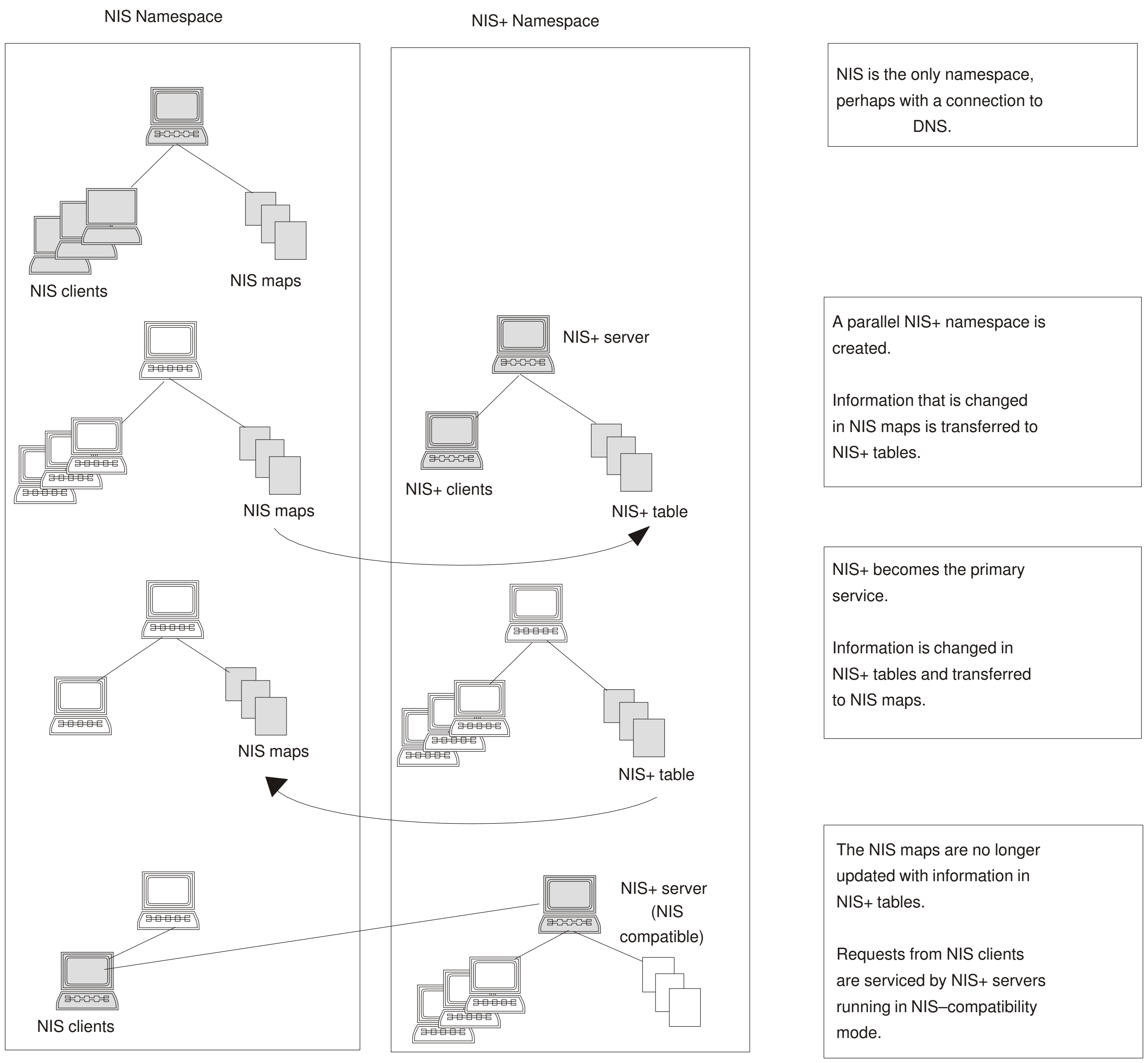
Deciding whether and how to run NIS+ in parallel to NIS—and when to stop—is probably one of the most difficult transition issues you will face. NIS+ provides several features that allow it to operate in parallel with NIS; notably, the NIS–compatibility mode.

If you plan to use NIS–compatibility mode, you have to consider:

- Selecting Your NIS–Compatible Domains, on page 3-24
- Determining NIS–Compatible Server Configuration, on page 3-25
- Deciding How to Transfer Information Between Services, on page 3-25
- Deciding How to Implement DNS Forwarding, on page 3-26
- NIS and NIS+ API Function Equivalents, on page 3-26
- NIS–Compatibility Mode Protocol Support, on page 3-26

The essential benefit provided by NIS–compatibility mode is that it does not require you to make any changes to NIS clients. The essential drawback is that you will not be able to take advantage of full NIS+ security and hierarchy and you may have to change those clients' domain names.

The following figure illustrates how you convert from an NIS–only namespace to an NIS–compatible namespace that responds to both NIS and NIS+ requests.



NIS is the only namespace, perhaps with a connection to DNS.

A parallel NIS+ namespace is created.

Information that is changed in NIS maps is transferred to NIS+ tables.

NIS+ becomes the primary service.

Information is changed in NIS+ tables and transferred to NIS maps.

The NIS maps are no longer updated with information in NIS+ tables.

Requests from NIS clients are serviced by NIS+ servers running in NIS–compatibility mode.

Selecting Your NIS–Compatible Domains

Make a list of your NIS clients and group them in their eventual NIS+ domains. If the NIS+ domain running in NIS–compatibility mode does not have the same name as its NIS clients' original NIS domain, you must change the NIS clients' domain name to the NIS+ domain name that is being supported by the NIS–compatible NIS+ server.

At first, NIS will no doubt be the primary service. As you become familiar with the intricacies of sharing information, you will be able to plan a transition to make NIS+ the primary service.

Some NIS+ users may want the capability of switching back and forth between the main NIS domain and the new NIS+ domain. The **nisclient** script can help them do this easily when backup files are made.

Determining NIS–Compatible Server Configuration

Take stock of your NIS servers, keeping in mind the requirements for your NIS+ servers. If you plan to eventually use them for the NIS+ service, upgrade them to the NIS+ recommendations. Identify which NIS servers will be used to support which NIS+ domains, and in what capacity (either master or replica). Remember that NIS+ servers belong to the domain **above** the one they support (except for the root domain servers). Since NIS+ servers do not belong to the domain they serve, you will not be able to use the same machines for other services that require domain–dependent information.

If possible, plan to use your NIS+ server machines only for NIS+. This arrangement could require you to transfer other network services, such as DNS name services, boot server, home directories, NFS servers, and so on, to non–NIS+ server machines.

At many sites, the NIS server plays multiple roles, such as NFS server, compute server, **rlogin** server, and mail host server. Because the NIS server uses the same information to resolve its names as do its clients, the NIS server can provide other services as well. As discussed in Domain Hierarchy, on page 3-5, except for root domains, all NIS+ servers live in the domain above the ones that they serve. So either do not run services on an NIS+ server that require access to the name service, or use other means, such as files in **irs.conf**, to acquire this same information. This problem would be solved if there were no hierarchy, the NIS+ root servers would live in the domain that they serve. The resource requirements of an NIS+ server are greater than those of an NIS server; therefore, it is advisable to not run other services along with NIS+.

Deciding How to Transfer Information Between Services

To keep information synchronized, be sure to make one namespace subordinate to the other. At first, the NIS namespace may be the dominant one, in which case you would make changes to the NIS maps and load them into the NIS+ tables. In effect, the NIS namespace would be the master database.

An NIS+ server in NIS–compatibility mode supports standard NIS maps. An exhaustive list of these maps is in the **ypfiles** command description. But there are some limitations on map support: The NIS+ server serves **ypmatch** requests only on the netgroup map, and not on the reverse maps. It does not support enumeration requests (for example, **ypcat**). The **passwd.adjunct** map is not supported, either.

Eventually, the NIS+ namespace should be dominant. When that is the case, you would make changes in the NIS+ tables and copy them to the NIS maps.

The NIS+ **nisaddent** command and the NIS+ **nispopulate** script transfer information between NIS maps and NIS+ tables, as summarized in the following table.

Table 8. NIS+ Data Transfer Commands Changing Information in the Passwd Table

NIS+ Command	Description
/usr/lib/nis/nisaddent –y	Transfers information from an NIS map to an NIS+ table after you run **ypxfr** to transfer maps from an NIS server to the local disk. Nonstandard NIS maps can be transferred to NIS+ tables if the information is in key–value pairs. Multicolumned maps will be not be transferred.
/usr/lib/nis/nisaddent –d	Copies information from an NIS+ table to a file, which can then be transferred to an NIS map with standard NIS utilities.
/usr/lib/nis/nispopulate –Y	Transfers information from NIS maps to NIS+ tables.

To properly implement the **passwd** command and password aging on your NIS+ network, the entries in the /**etc**/**security**/**user** file on every machine must be correct. These entries determine where the **passwd** command will go for information and where it will update password information.

In domains created with NIS–compatibility mode, the permissions are slightly different: permissions at the table level must be set to provide read rights to the world class, and at the column level permissions must provide read access to the nobody class.

Deciding How to Implement DNS Forwarding

NIS servers can forward DNS requests made from NIS clients. NIS+ servers running in NIS–compatibility mode also provide DNS forwarding.

If the DNS domains are repartitioned, you must redefine new DNS zone files. Clients, however, may require updates to their **/etc/resolv.conf** file. A client, if it is also a DNS client, can set up its name service switch configuration file to search for host information in either DNS zone files or NIS maps—in addition to NIS+ tables.

NIS and NIS+ API Function Equivalents

To completely convert your site to NIS+, you must both change the name service and port all applications to NIS+. Any internally created applications that make NIS calls have to be modified to use NIS+ calls. Otherwise, you will always have to run your NIS+ servers in NIS–compatibility mode, with all the drawbacks that this mode entails. External applications may force you to run your namespace in NIS–compatibility mode until they are updated as well.

The following table contains a list of the NIS API functions and their NIS+ API equivalents, if they exist.

Table 9. NIS API and NIS+ API Equivalent Functions

NIS API Functions	NIS+ API Functions
yp_get_default_domain()	**nis_local_directory()**
ypbind()	**N/A**
ypunbind()	**N/A**
ypmatch()	**nis_list()**
yp_first()	**nis_first_entry()**
yp_next()	**nis_next_entry()**
yp_all()	**nis_list()**
yp_master()	**nis_lookup()**
yperr_string()	**nis_perror() nis_sperrno()**
ypprot_err()	**nis_perror() nis_sperrno()**
yp_order()	**N/A**
yp_update()	**nis_add_entry(), nis_remove_entry(), nis_modify_entry()**

NIS–Compatibility Mode Protocol Support

The following table shows which NIS protocols are supported by NIS+ servers in NIS–compatibility mode.

Table 10. Support for NIS Protocols by NIS+ Servers

NIS Protocols	Compatibility Description
NIS client V2 protocol	Supported
NIS server–to–server protocol	Unsupported

NIS Protocols	Compatibility Description
NIS client update protocol	**yppasswd** protocol supported
NIS client V1 protocol	Not supported except for **YPPROC_NULL, YPPROC_DOMAIN,** and **YPPROC_DOMAIN_NONACK**

Prerequisites to Transition

This section describes several miscellaneous tasks that must be carried out before beginning the transition:

1. Gauge the Impact of NIS+ on Other Systems, on page 3-28
2. Train Administrators, on page 3-28
3. Write a Communications Plan, on page 3-29
4. Identify Required Conversion Tools and Processes, on page 3-29
5. Identify Administrative Groups Used for Transition, on page 3-29
6. Determine Who Will Own the Domains, on page 3-30
7. Determine Resource Availability, on page 3-30
8. Resolve Conflicts Between Login Names and Host Names, on page 3-30
9. Examine All Information Source Files, on page 3-31
10. Remove the ”.” from Host Names, on page 3-31
11. Remove the ”.” from NIS Map Names, on page 3-31
12. Document Your Current NIS Namespace, on page 3-31
13. Create a Conversion Plan for Your NIS Servers, on page 3-32

Gauge the Impact of NIS+ on Other Systems

Develop a formal introduction, testing, and familiarization program for your site, not only to train administrators, but also to uncover dependencies of other systems or applications on NIS that will be affected by a transition to NIS+.

For example, some applications may rely on some of the NIS maps. Will they function with standard or custom NIS+ tables? How will their need for access affect your overall security plan?

What nonstandard NIS maps are being used at your site? Can you convert them to NIS+ tables or create nonstandard NIS+ tables to store their information? Be sure to check their access rights.

Does your site use locally built applications that depend on NIS? Do you have commands or applications that make direct NIS calls, such as embedded **ypmatch** function calls? (See NIS and NIS+ API Function Equivalents, on page 3-26 for more information.)

Do you have any duplicate user and host names in your namespace? (See Resolving User/Host Name Conflicts, on page 3-16 for more information.)

How will the network installation procedures be affected by the transition to NIS+? Analyze the changes required, if any.

Gauging the impact of NIS+ on your site administrative practices will help uncover potential roadblocks.

Train Administrators

Another goal of the introduction and familiarization program discussed in Become Familiar with NIS+, on page 3-4 is to give the administrators at your site an opportunity to become familiar with NIS+ concepts and procedures. Classroom instruction alone is insufficient. Administrators need a chance to work in a safe test environment. The training should consist of:

- A formal course in NIS+ concepts and administration.
- Basic NIS+ troubleshooting information and practice.

- Information about your site's implementation strategy and plans.

Write a Communications Plan

Prepare a plan to communicate your intentions to users long before you actually begin converting clients to NIS+. Tell them about the implementation plan and give them a way to obtain more information. As mentioned in Transition Principles, on page 3-3, a typical transition goal is to keep the impact of the transition on clients to a minimum, but users might become concerned about the upcoming change. Send out e–mail notices, conduct informative seminars, and designate email aliases or individuals to whom users can send questions.

Identify Required Conversion Tools and Processes

Consider creating or obtaining transition tools to help with the implementation. If your site already uses automated tools to administer individual systems or network services, consider porting them to operate under the versions of AIX software and NIS+ that you will be using for the transition. Here are some suggestions for scripts you might want to write:

- A script to convert users to NIS+—make additions to the **nisclient** shell script
- A check script to verify the correctness of a user's NIS+ environment
- Backup and recovery scripts
- **crontab** entries for routine NIS+ maintenance
- Procedures for notification of outages

Scripts such as these will ensure that the transition is carried out uniformly across domains, speed the transition, and reduce complaints.

You should also prepare a set of standard configuration files and options, such as **/etc/irs.conf**, that all clients across the namespace can use.

Identify Administrative Groups Used for Transition

Be sure that the NIS+ groups created as part of your namespace design (see Establishing Password–aging Criteria, Principles, and Rules, on page 3-19) correspond to the administrative resources you have identified for the transition. You could require a different set of NIS+ groups for the transition than for routine operation of an NIS+ namespace. Consider adding remote administrators to your groups in case you need their help in an emergency.

Make sure that group members have the proper credentials, that namespace objects grant the proper access rights to groups, and that the right group is identified as the group owner of the right namespace objects.

The following table summarizes commands that operate on NIS+ groups and the group permissions that are useful,

Table 11. NIS+ Commands for Groups

Command	Description
nisgrpadm	Creates or deletes groups, adds, changes, lists, or deletes members
niscat –o	Displays the object properties of an NIS+ group
nissetup	Creates the basic structure of the directory in which a domain's groups are stored: **groups_dir**
nisls	Lists the contents of a directory
NIS_GROUP	Environment variable that overrides the value of **nisdefaults** for the shell in which it is set
nischmod	Changes an object's access rights

Command	Description
nischown	Changes the owner of an NIS+ object
nischgrp	Changes the group owner of an NIS+ object
nistbladm –u	Changes access rights to NIS+ table columns
nisdefaults	Displays or changes the current NIS+ defaults

Determine Who Will Own the Domains

To take complete advantage of the features inherent in a domain hierarchy, distribute the ownership of domains to the organizations they are dedicated to supporting. This will free the administrators of the root domain from performing rudimentary tasks at the local level.

Once you know who owns what, you can provide guidelines for creating administrative groups and setting their access rights to objects.

Consider how to coordinate the ownership of NIS+ domains with the ownership of DNS domains. Here are some guidelines:

- The administration of the DNS domain structure should remain the responsibility of the highest–level administrative group at the site.
- This same administrative group also owns the top–level NIS+ domain.
- Responsibility for the administration of lower–level DNS and NIS+ domains is delegated to individual sites by the top–level administrative group. If the NIS+ domains will be created along the same principles as the DNS domains (for instance, organized geographically), this delegation will be simple to explain.

Determine Resource Availability

Determine what administrative resources will be required for the implementation. These will be above and beyond the resources required for normal operation of NIS+. If your transition will involve a long period of NIS+ and NIS compatibility, additional resources may be required.

Consider not only the implementation of the namespace design but also the conversion the numerous clients and dealing with special requests or problems. Keep in mind that NIS+ has a steep learning curve. Administrators may be less efficient for a while at performing support functions with NIS+ than they were with NIS. Consider not only formal training but extensive lab sessions with hands–on experience.

Finally, even after the transition is complete, administrators will require extra time to become familiar with the everyday work flow of supporting NIS+.

Consider hardware resources. NIS servers are often used to support other network services such as routing, printing, and file management. Because of the potential load on an NIS+ server, you should use dedicated NIS+ servers. This load–balancing simplifies the transition because it makes troubleshooting and performance monitoring simpler. Of course, you incur the cost of additional systems. The question of how many servers you will need and how they should be configured is addressed in Designing the NIS+ Namespace, on page 3-5.

Remember, these servers are required *in addition to* the NIS servers. Although the NIS servers might be decommissioned or recycled after the transition is complete, the NIS+ servers will continue to be used.

Resolve Conflicts Between Login Names and Host Names

The NIS+ authentication scheme does not allow workstations and users to use the same names within a domain; for example, joe@joe is not permitted. Since NIS+ does not distinguish between credentials for hosts and login names, you can only use one credential type per name. If you have duplicate names in your namespace and you must keep the duplicate host name for some other reason make this change: retain the user login name and alias the duplicate host names. Create a new name for the host and use the old name as an alias for the new name. See Resolving User/Host Name Conflicts, on page 3-16 for examples of illegal name combinations.

You must resolve name conflicts before the implementation can begin, but you should also plan on permanently checking new workstations and user names during routine NIS+ operation. The **nisclient** script does name comparisons when you use it to create a client credential.

Examine All Information Source Files

Check all /**etc** files and NIS maps for empty fields or corrupted data before configuring NIS+. The NIS+ table–populating scripts and commands might not succeed if the data source files contain empty fields or extraneous characters. Fill blank fields or fix the data before you start. It is better to delete questionable users or machines from the /**etc** files or NIS maps before running NIS+ scripts, and then add them back later after NIS+ is installed, than to proceed with the scripts and possibly corrupt data.

Remove the "." from Host Names

Because NIS+ uses dots (periods) to delimit between machine names and domains and between parent and sub–domains, you cannot have a machine (host) name containing a dot. Before converting to NIS+ you must eliminate any dots in your host names. You can convert host name dots to underscores (_). For example, you cannot have a machine named **sales.alpha**. You can convert that name to **sales_alpha**. (See the **host** command description for detailed information on allowable host names.)

Remove the "." from NIS Map Names

As described in Designing the NIS+ Namespace, on page 3-5, NIS+ automounter tables have replaced the "." (dot) in the table name with an underscore. You also need to make this change to the names of NIS maps that you will use during the transition. If you do not, NIS+ will confuse the dot in the name with the periods that distinguish domain levels in object names.

Attention: Be sure to convert the dot to underscores for **all** NIS maps, not just those of the automounter. Be aware, however, that changing the names of nonstandard NIS maps from dots to underscores may cause applications that use those nonstandard maps to fail unless you also modify the applications to recognize NIS+ syntax.

Document Your Current NIS Namespace

Documenting your current configuration will give you a clear point of departure for the transition. Make a list of the following items:

- Name and location of all current NIS domains and networks
- Host name and location of all current NIS servers, both master and slave
- Configuration of all current NIS servers, including:
- Host name
- CPU type
- Memory size
- Disk space available
- Name of administrators with root access
- Nonstandard NIS maps

Correlate the list of your NIS clients with their eventual NIS+ domains. They will have to be upgraded to the AIX Version 4.3.3 release.

Create a Conversion Plan for Your NIS Servers

Take stock of your NIS servers. Although you can recycle them after the transition is complete, keep in mind that you will go through a stage in which you will need servers for **both** services. Therefore, you cannot simply plan to satisfy all your NIS+ server needs with your existing NIS servers.

You will find it helpful to create a detailed conversion plan for NIS servers, identifying which NIS servers will be used for NIS+ and when they will be converted. Do not use the NIS servers as NIS+ servers during the first stages of NIS–to–NIS+ transition. As described in Implementing the Transition, on page 3-33, the implementation is most stable when you check the operation of the entire namespace as a whole before you convert any clients to NIS+.

Assign NIS servers to NIS+ domains and identify each server's role (master or replica). Once you have identified the servers you plan to convert to NIS+ service, upgrade them to NIS+ requirements (see Disk Space and Memory Requirements, on page 3-12).

Implementing the Transition

At this point, verify that all your pretransition tasks have been completed and that your site's users are aware of your plans.

If you will be running NIS+ domains alongside DNS domains, you will set up one NIS+ sub–domain with each DNS domain. After you have set up a complete NIS+ namespace along with the first DNS domain and have verified that everything is working properly, you can set up the other NIS+ namespaces in parallel.

The major implementation phases are as follows:

- Phase I—Set Up the NIS+ Namespace, on page 3-33
- Phase II—Connect the NIS+ Namespace to Other Namespaces, on page 3-34
- Phase III—Make the NIS+ Namespace Fully Operational, on page 3-35
- Phase IV—Upgrade NIS–Compatible Domains, on page 3-36

Phase I—Set Up the NIS+ Namespace

Set up the namespace with full DES authentication, even if the domains will operate in NIS–compatibility mode. Use the NIS+ scripts described in NIS+ Setup Scripts, on page 5-7 to set up your namespace. See NIS+ Namespace and Structure, on page 4-1 for more explanation of NIS+ structure and concepts. Then perform the following steps:

1. Set up the root domain.

   If you are going to run the root domain in NIS–compatibility mode, use **nisserver –Y**. (If you choose not to use the setup scripts, see Setting Up the Root Domain.)

2. Populate the root domain tables.

   You can use **nispopulate** to transfer information from NIS maps or text files. You can also create entries one at a time with **nistbladm** or **nisaddent**.

3. Set up clients of the root domain.

   Set up a few clients in the root domain so that you can test its operation properly. Use full DES authentication. Some of these client machines will later be converted to root replica servers and some will serve as workstations for the administrators who support the root domain. NIS+ servers should never be an individual's workstation.

4. Create or convert site–specific NIS+ tables.

   If the new NIS+ root domain will require custom, site–specific NIS+ tables, create them, with **nistbladm** and transfer the NIS data into them with **nisaddent**.

5. Add administrators to root domain groups.

   Remember, the administrators must have LOCAL and DES credentials (use **nisaddcred**). Their workstations should be root domain clients, and their root identities should also be NIS+ clients with DES credentials.

6. Update the sendmailvars table, if necessary.

   If your e–mail environment has changed as a result of the new domain structure, populate the root domain's **sendmailvars** table with the new entries.

7. Set up root domain replicas.

   First convert the clients into servers (use **rpc.nisd** with **–Y** for NIS compatibility and if you want DNS forwarding, also use **–B**), and then associate the servers with the root domain by running **nisserver –R**.

8. Test the root domain's operation.

   Develop a set of installation–specific test routines to verify a client is functioning after the switch to NIS+. You should operate this test domain for about a week before you begin converting other users to NIS+.

9. Set up the remainder of the namespace.

   Do not convert any more clients to NIS+, but set up all the other domains beneath the root domain. This includes setting up their master and replica servers. Test each new domain as thoroughly as you tested the root domain, until you are sure your configurations and scripts work properly.

10. Test the operation of the namespace.

    Test all operational procedures for maintenance, backup, recovery, and other scenarios. Test the information–sharing process between all domains in the namespace. Do not proceed to Phase II until the entire NIS+ operational environment has been verified.

11. Customize the security configuration of the NIS+ domains.

    This may not be necessary if everything is working properly. If, however, you want to protect some information from unauthorized access, you can change the default permissions of NIS+ tables so that even NIS clients would be unable to access them. You could also rearrange the membership of NIS+ groups and the permissions of NIS+ structural objects to align with administrative responsibilities.

Phase II—Connect the NIS+ Namespace to Other Namespaces

1. [Optional] Connect the root domain to the DNS namespace.

   Workstations, if they are also DNS clients, can have their information retrieval system configuration (/**etc**/**irs.conf**) files set to search for information in DNS zone files, in addition to NIS+ tables or NIS maps.

   Configure each client's /**etc**/**resolv.conf** file properly. The /**etc**/**resolv.conf** lists the IP addresses of the client's DNS servers.

2. Test the joint operation of NIS+ with DNS.

   Verify that requests for information can pass between the namespaces without difficulty.

3. If operating NIS+ in parallel with NIS, test the transfer of information.

   Use the **nispopulate** script to transfer information from NIS to NIS+. To transfer data from NIS+ to NIS, run:

   ```
   # nisaddent –d –t hosts.org_dir > hosts
   # makedbm –b hosts hosts.byname
   ```

   Establish policies for keeping tables synchronized, particularly the hosts and passwd tables. Test the tools used to maintain consistency between the NIS and NIS+ environments. Decide when to make the NIS+ tables the actual source of information.

4. Test operation of NIS+ with both DNS and NIS.

   Test all three namespaces together to make sure the added links do not create problems.

Phase III—Make the NIS+ Namespace Fully Operational

1. Convert clients to NIS+.

   Convert clients one workgroup at a time, and convert all workgroups in a subnet before starting on those of another subnet. That way, when you convert all the clients in a subnet, you can eliminate the NIS service on that subnet.

   Use the **nisclient** script to convert NIS clients to NIS+ clients. If you need to modify the clients' DNS configuration, you will have to write your own scripts to automate that process. You will also need to edit the **/etc/security/user** file to specify **NISPLUS** authentication for users. For example:

   ```
           admin = false
           login = true
           su = true
           daemon = true
           rlogin = true
           sugroups = ALL
           admgroups =
           ttys = ALL
           auth1 = SYSTEM
           auth2 = NONE
           tpath = nosak
           umask = 022
           expires = 0
           SYSTEM = NISPLUS
           logintimes =
           pwdwarntime = 0
           account_locked = false
           loginretries = 0
           histexpire = 0
           histsize = 0
           minage = 0
           maxage = 0
           maxexpired = -1
           minalpha = 0
           minother = 0
           minlen = 0
           mindiff = 0
           maxrepeats = 8
           dictionlist =
           pwdchecks =
   ```

   You can also save time if your site has a shared, mounted central directory similar to **/usr/local**. You could put the script in the central directory and, on the day of conversion, send an e–mail message to clients asking them to run the script as superuser.

2. Edit the **/etc/security/login.cfg**, **/usr/lib/security/methods.cfg**, and **/usr/lib/security/methods.cfg** configuration files. (Add them, if they do not exist.) The following lines must exist (and not be commented) in each file:

   ```
   NISPLUS
     program=/usr/lib/security/NISPLUS
   ```

3. Monitor the status of the transition as clients are being converted.

   Track progress against your plan and all serious complications not anticipated in the planning stages. Communicate your status to interested parties.

4. Decommission NIS servers.

   When all the clients on a subnet are converted to NIS+, decommission the NIS servers. If a particular subnet has some clients that require NIS service, use the NIS–compatibility feature of the NIS+ servers, but do not retain the NIS servers.

5. Evaluate NIS+ performance.

   Once the implementation is complete, test to see that NIS+ is working correctly.

6. Optimize the NIS+ environment.

   Based on the results of your performance evaluation, modify the NIS+ environment as needed. These improvements could be as simple as adding selected replicas in domains with high loads or as involved as rearranging the storage of NIS+ information for a group of domains.

7. Clean up new domains.

   If you did not change old domain names during the transition for the sake of simplicity, upgrade them now to the new NIS+ naming scheme. For example, if you left some domains with geographic labels while you converted to an organizational hierarchy, you would change the geographic names to their organizational versions.

Phase IV—Upgrade NIS–Compatible Domains

1. Convert the last NIS clients to NIS+.

   As soon as you can, eliminate the need for NIS–compatible NIS+ domains. Upgrading the last NIS clients to NIS+ allows you to take advantage of NIS+ security features.

2. Adjust your security configuration.

   Once you have deleted all NIS clients, you can restart the NIS+ servers in standard mode and run **nischmod** on the NIS+ tables to change permission levels to eliminate the security hole caused by NIS compatibility. If you did not create credentials for NIS+ principals before, you must do that now. Restrict the access of unauthenticated principals.

3. Establish miscellaneous evaluation and improvement programs.

   Evaluate operational procedures to determine which ones can be improved, particularly procedures used to recover from problems. Plan for new NIS+ releases and possible functional enhancements. Track the development of AIX components that might require new NIS+ tables. Look for automated tools that enable you to perform NIS+ administration functions more efficiently.

Chapter 4. The NIS+ Namespace and Structure

The NIS+ name service is designed to conform to the shape of the organization that installs it, wrapping itself around the bulges and corners of almost any network configuration. This is implemented through the NIS+ namespace. This chapter describes the structure of the NIS+ namespace, the servers that support it, and the clients that use it.

NIS+ Files and Directories

The following table lists the directories used to store NIS+ files.

Table 12. Where NIS+ Files are Stored

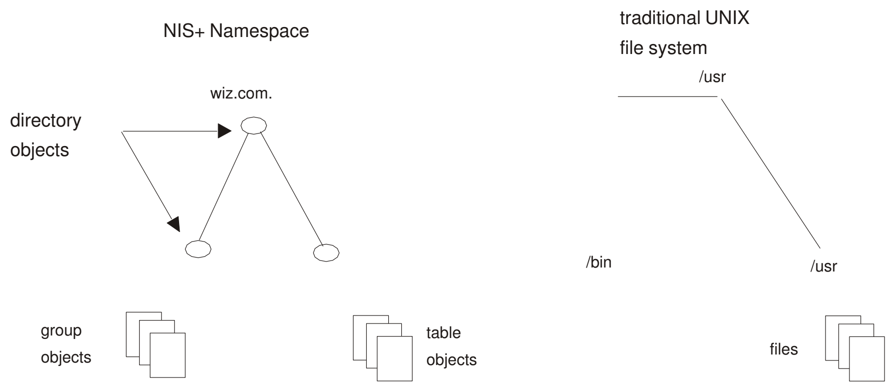
Directory	Where	Contains
/**usr**/**bin**	All machines	NIS+ user commands
/**usr**/**lib**/**nis**	All machines	NIS+ administrator commands
/**usr**/**sbin**	All machines	NIS+ daemons
/**usr**/**lib**/	All machines	NIS+ shared libraries
/**var**/**nis**/**data**	NIS+ server	Data files used by NIS+ server
/**var**/**nis**	NIS+ server	NIS+ working files
/**var**/**nis**	NIS+ client machines	Machine–specific data files used by NIS+

Attention: Do not rename the /**var**/**nis** or /**var**/**nis**/**data** directories or any of the files in these directories that were created by **nisinit** or any of the other NIS+ setup procedures. If any required file or directory is renamed, NIS+ cannot start or operate correctly.

Structure of the NIS+ Namespace

The NIS+ namespace is the arrangement of information stored by NIS+. The namespace can be arranged in a variety of ways to suit the needs of an organization. For example, if an organization had three divisions, its NIS+ namespace would likely be divided into three parts, one for each division. Each part would store information about the users, workstations, and network services in its division, but the parts could easily communicate with each other. Such an arrangement would make information easier for the users to access and for the administrators to maintain.

Although the arrangement of an NIS+ namespace can vary from site to site, all sites use the same structural components: directories, tables, and groups. These components are called NIS+ *objects*. NIS+ objects can be arranged into a hierarchy. For example, the illustration below shows, on the left, a namespace that consists of three directory objects, three group objects, and three table objects; on the right it shows a file system that consists of three directories and three files.

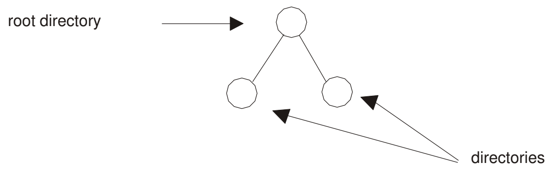


Although an NIS+ namespace resembles a traditional UNIX file system, it has five important differences:

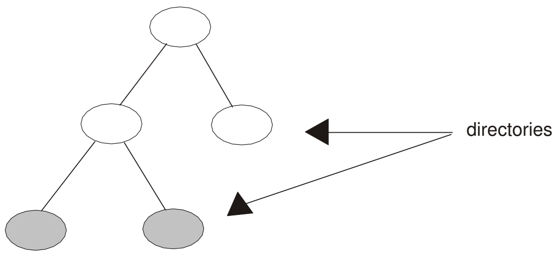
- Although both use directories, the other objects in an NIS+ namespace are tables and groups, not files.
- The NIS+ namespace is administered only through NIS+ administration commands (listed in a table within Using NIS+ Commands, on page 1-15) or graphical user interfaces designed for that purpose, such as Web-based System Manager or SMIT tools. It cannot be administered with standard UNIX file system commands or graphic user interfaces.
- The names of AIX file system components are separated by slashes (/**usr**/**bin**), but the names of NIS+ namespace objects are separated by dots (`wiz.com.`).
- The ”root” of an AIX file system is reached by stepping through directories from right to left (/**usr**/**src**/**file1**), while the root of the NIS+ namespace is reached by stepping from left to right (`sales.wiz.com.`).
- Because NIS+ object names are structured from left to right, a fully qualified name always ends in a dot. Any NIS+ object ending in a dot is assumed to be a fully qualified name. NIS+ object names that do not end in a dot are assumed to be relative names.

Directories

Directory objects are the skeleton of the namespace. When arranged into a tree–like structure, they divide the namespace into separate parts. A directory hierarchy is similar to an inverted tree, with the root of the tree at the top and the branches toward the bottom. The topmost directory in a namespace is the *root* directory. If a namespace is flat, it has only one directory, but that directory is nevertheless the root directory. The directory objects beneath the root directory are simply called *directories*.

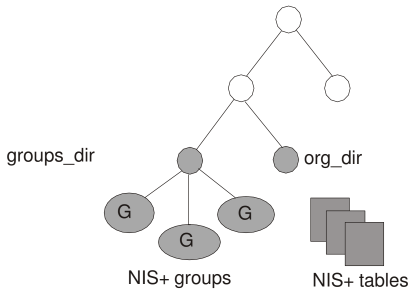


A namespace can have several levels of directories:



When identifying the relation of one directory to another, the directory beneath is called the *child* directory and the directory above is called the *parent* directory.

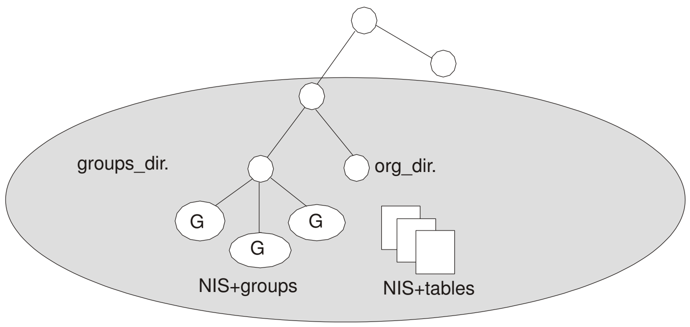
Whereas traditional UNIX directories are designed to hold traditional UNIX files, NIS+ directories are designed to hold NIS+ objects: other directories, tables and groups. Any NIS+ directory that stores NIS+ groups is named **groups_dir**. Any directory that stores NIS+ system tables is named **org_dir**.



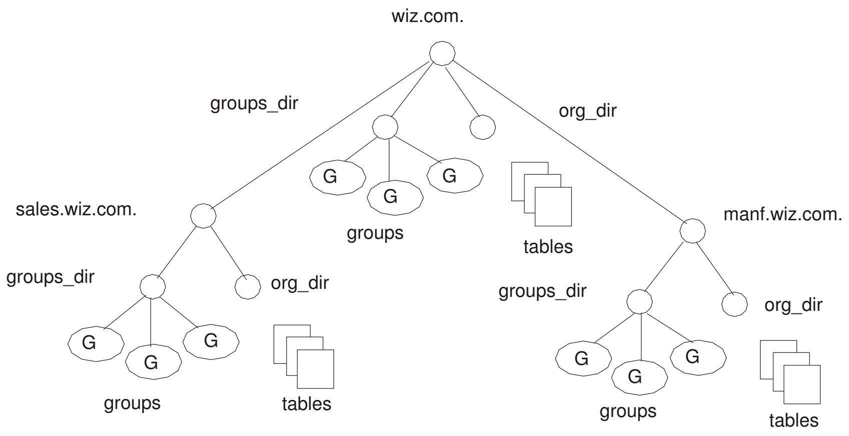
You can arrange directories, tables, and groups into any structure that you like. However, NIS+ directories, tables, and groups in a namespace are normally arranged into configurations called *domains*. Domains are designed to support separate portions of the namespace. For instance, one domain may support the Sales Division of a company, while another may support the Manufacturing Division.

Domains

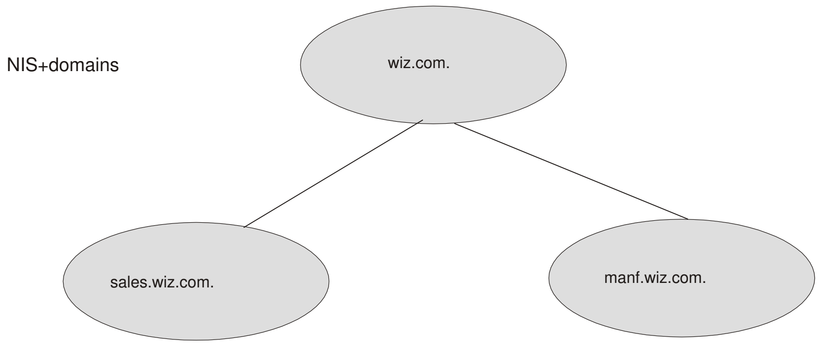
An NIS+ domain consists of a directory object, its **org_dir** directory, its **groups_dir** directory, and a set of NIS+ tables.



NIS+ domains are not *tangible* components of the namespace. They are simply a convenient way to *refer* to sections of the namespace that are used to support real–world organizations. For example, assume that the Wizard Corporation has a Sales division and an Manufacturing division. To support those divisions, its NIS+ namespace would most likely be arranged into three major directory groups, with a structure such as the one shown in the figure.



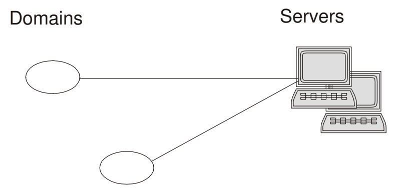
Instead of referring to such a structure as three directories, six subdirectories, and several additional objects, referring to it as three domains is more convenient.



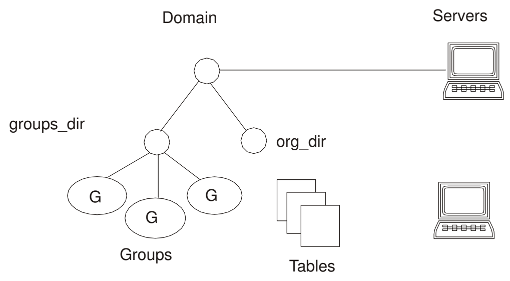
Setting Up the Root Domain, on page 5-32 and Setting Up a Nonroot Domain, on page 5-57 describe how to configure domains.

Servers

Every NIS+ domain is supported by a set of NIS+ servers. The servers store the domain's directories, groups, and tables, and answer requests for access from users, administrators, and applications. Each domain is supported by only one set of servers. However, a single set of servers can support more than one domain, as shown in the figure.



Remember that a domain is not an object but only refers to a collection of objects. Therefore, a server that supports a domain is not actually associated with the domain, but with the domain's main directory.

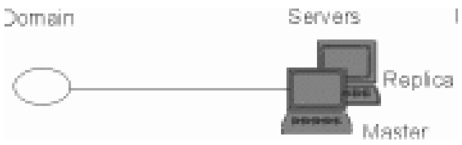


This connection between the server and the directory object is established during the process of setting up a domain. Although instructions are provided in NIS+ Installation and Configuration, on page 5-1, one thing is important to mention now: when that connection is established, the directory object stores the name and IP address of its server. This information is used by clients to send requests for service, as described later in this section.

Any AIX Version 4.3.3 workstation can be an NIS+ server. The software for both NIS+ servers and clients is bundled together into the release. Therefore, any workstation with the AIX Version 4.3.3 BOS installed can become a server or a client, or both. What distinguishes a client from a server is the role it is playing. If a workstation is providing NIS+ service, it is acting as an NIS+ server. If it is requesting NIS+ service, it is acting as an NIS+ client.

Because of the need to service many client requests, a workstation that will act as an NIS+ server might be configured with more computing power and more memory than the average client. And, because it needs to store NIS+ data, it might also have a larger disk. However, other than hardware to improve its performance, a server is not inherently different from an NIS+ client.

Two types of servers support an NIS+ domain: a master and its replicas (see figure).



The master server of the root domain is called the *root master* server. A namespace has only one root master server. The master servers of other domains are simply called master servers. Likewise, there are root replica servers and regular replica servers.

Both master and replica servers store NIS+ tables and answer client requests. The master, however, stores the master copy of a domain's tables. The replicas store only duplicates. The administrator loads information into the tables in the master server, and the master server propagates it to the replica servers.

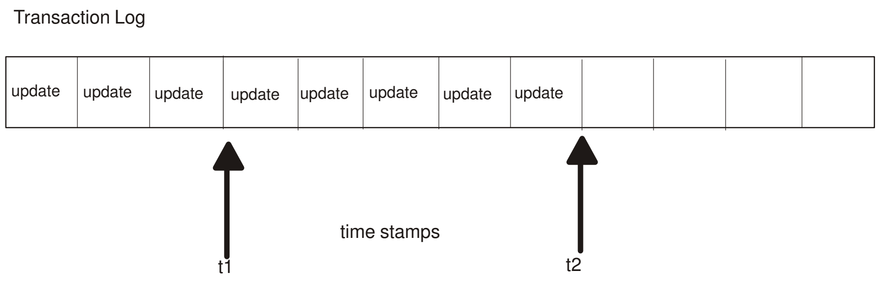
This arrangement has two benefits. First, it avoids conflicts between tables because only one set of master tables exists; the tables stored by the replicas are only copies of the masters. Second, it makes the NIS+ service much more available. If either the master or a replica is down, another server can act as a backup and handle the requests for service.

Note: If the root master server is unavilable and the NIS+ domain is being served solely by a replica, you can obtain information from the NIS+ tables, but changes to the original tables can be made only when the master server is available.

How Servers Propagate Changes

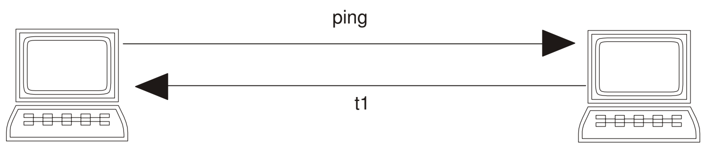
An NIS+ master server implements updates to its objects immediately; however, it tries to ”batch” several updates together before it propagates them to its replicas. When a master server receives an update to an object, whether a directory, group, link, or table, it waits about two minutes for any other updates that may arrive. Once it is finished waiting, it stores the updates in two locations: on disk and in a *transaction log* (it has already stored the updates in memory).

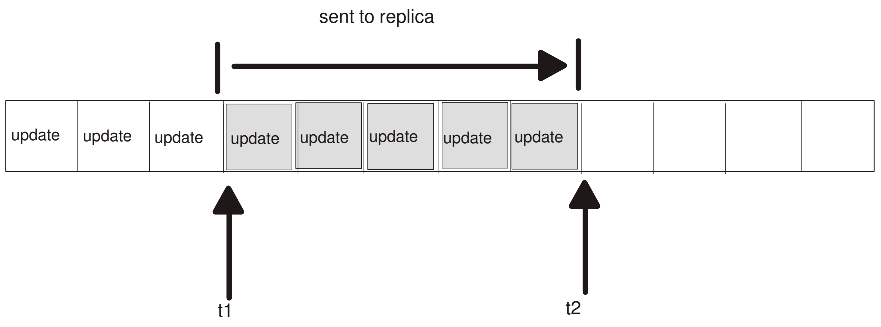
The transaction log is used by a master server to store changes to the namespace until they can be propagated to replicas. A transaction log has two primary components: updates and time stamps.



An update is an actual copy of a changed object. For instance, if a directory has been changed, the update is a complete copy of the directory object. If a table entry has been changed, the update is a copy of the actual table entry. The time stamp indicates the time at which an update was made by the master server.

After recording the change in the transaction log, the master sends a message to its replicas, telling them that it has updates to send them. Each replica replies with the time stamp of the last update it received from the master. The master then sends each replica the updates it has recorded in the log since the replica’s time stamp.





When the master server updates *all* its replicas, it clears the transaction log. In some cases, such as when a new replica is added to a domain, the master receives a time stamp from a replica that is before its earliest time stamp still recorded in the transaction log. In this situation, the master server performs a full *resynchronization*, or *resync*. A resync downloads all the objects and information stored in the master down to the replica. During a resync, both the master and replica are busy. The replica cannot answer requests for information; the master can answer read requests but cannot accept update requests. Both respond to requests with a `Server Busy - Try Again` or similar message.

Attention: The clocks of the server and clients *must* be synchronized to each other. If this is not done, credential clarification fails.

NIS+ Clients and Principals

NIS+ principals are the entities (clients) that submit requests for NIS+ services.

Principal

An NIS+ principal may be someone who is logged in to a client machine as a regular user or someone who is logged in as superuser (root). In the first instance, the request actually comes from the client user; in the second instance, the request comes from the client workstation. Therefore, an NIS+ principal can be a client user or a client workstation.

Note: An NIS+ principal can also be the entity that supplies an NIS+ service from an NIS+ server. Since all NIS+ servers are also NIS+ clients, much of this discussion also applies to servers.

Client

An NIS+ client is a workstation that has been set up to receive NIS+ service. Setting up an NIS+ client consists of establishing security credentials, making it a member of the proper NIS+ groups, verifying its home domain, and, finally, running the NIS+ initialization script.

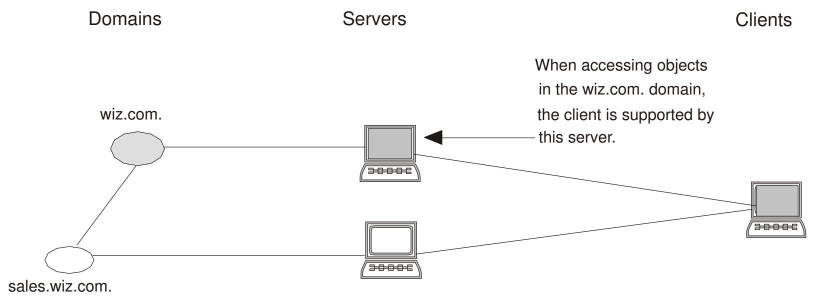
An NIS+ client can access any part of the namespace, subject to security constraints. In other words, if it has been authenticated and has been granted the proper permissions, it can access information or objects in any domain in the namespace.

Although a client can access the entire namespace, a client *belongs* to only one domain, which is referred to as its *home* domain. A client's home domain is usually specified during installation, but it can be changed or specified later. All the information about a client, such as its IP address and its credentials, is stored in the NIS+ tables of its home domain.

There is a subtle difference between being an NIS+ client and being listed in an NIS+ table. Entering information about a workstation into an NIS+ table does not automatically make that workstation an NIS+ client. It simply makes information about that workstation available to all NIS+ clients. That workstation cannot request NIS+ service unless it is actually set up as an NIS+ client.

Conversely, making a workstation an NIS+ client does not enter information about that workstation into an NIS+ table. It simply allows that workstation to receive NIS+ service. If information about that workstation is not explicitly entered into the NIS+ tables by an administrator, other NIS+ clients will not be able to get it.

When a client requests access to the namespace, it is actually requesting access to a particular domain in the namespace. Therefore, it sends its request to the server that supports the domain it is trying to access. Here is a simplified representation.

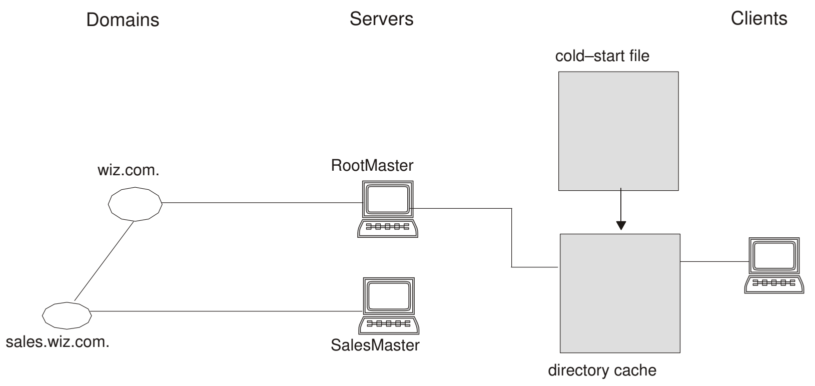

Domains
Servers
Clients
When accessing objects in the wiz.com. domain, the client is supported by this server.
wiz.com.
sales.wiz.com.

How does the client know which server that is? By trial and error. Beginning with its home server, the client tries first one server, then another, until it finds the right one. When a server cannot answer the client's request, it sends the client information to help locate the right server. Over time, the client builds up its own cache of information and becomes more efficient at locating the right server. The next section describes this process.

The Cold–Start File and Directory Cache

When a client is initialized, it is given a *cold–start file*. The cold–start file gives a client a copy of a directory object that it can use as a starting point for contacting servers in the namespace. The directory object contains the address, public keys, and other information about the master and replica servers that support the directory. Normally, the cold–start file contains the directory object of the client's home domain.

A cold–start file is used only to initialize a client's *directory cache*. The directory cache, managed by an NIS+ facility called the *cache manager*, stores the directory objects that enable a client to send its requests to the proper servers.

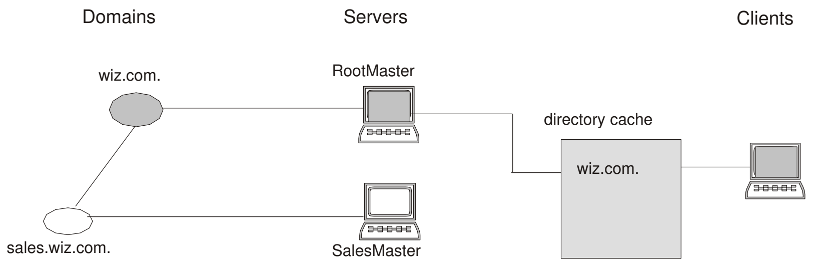


By storing a copy of the namespace's directory objects in its directory cache, a client can know which servers support which domains. (To view the contents of a client's cache, use the **nisshowcache** command.) The following table is a simplified example.

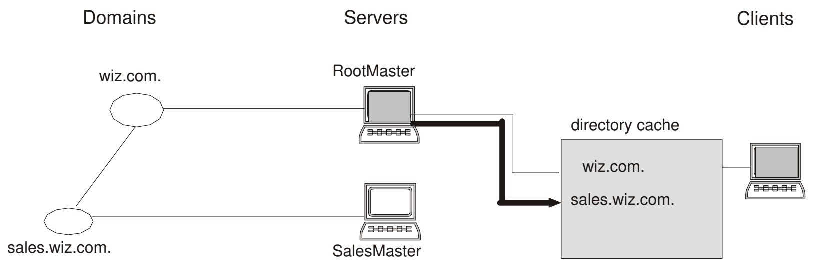
Domain	Directory Name	Supporting Server	IP Address
wiz.com.	**wiz.com.**	RootMaster	129.44.1.1
sales.wiz.com	**sales.wiz.com.**	SalesMaster	129.44.2.1
manf.wiz.com.	**manf.wiz.com.**	ManfMaster	129.44.3.1
int.sales.wiz.com.	**int.sales.wiz.com.**	IntlSalesMaster	129.44.2.11

To keep these copies up–to–date, each directory object has a **time–to–live** (TTL) field. Its default value is 12 hours. If a client looks in its directory cache for a directory object and finds that it has not been updated in the last 12 hours, the cache manager obtains a new copy of the object. You can change a directory object's time–to–live value with the **nischttl** command. However, keep in mind that the longer the time–to–live, the higher the likelihood that the copy of the object will be out of date; and the shorter the time to live, the greater the network traffic and server load.

How does the directory cache accumulate these directory objects? As mentioned above, the cold–start file provides the first entry in the cache. Therefore, when the client sends its first request, it sends the request to the server specified by the cold–start file. If the request is for access to the domain supported by that server, the server answers the request.

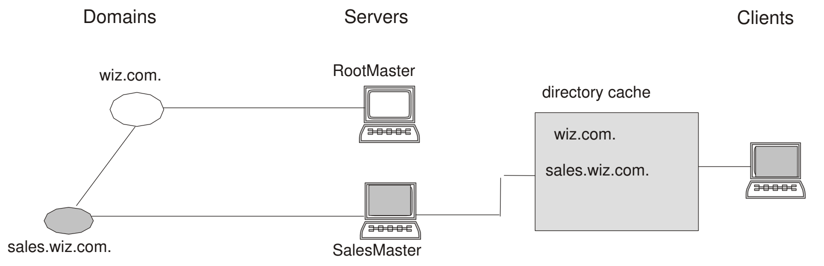


If the request is for access to another domain (for example, sales.wiz.com.), the server tries to help the client locate the proper server. If the server has an entry for that domain in its own directory cache, it sends a copy of the domain's directory object to the client. The client loads that information into its directory cache for future reference and sends its request to that server.



If the server does not support the domain the client is trying to access, it sends information to help the client find the right server.

In the unlikely event that the server does not have a copy of the directory object the client is trying to access, it sends the client a copy of the directory object for its own home domain, which lists the address of the server's parent. The client repeats the process with the parent server, and keeps trying until it finds the proper server or until it has tried all the servers in the namespace.



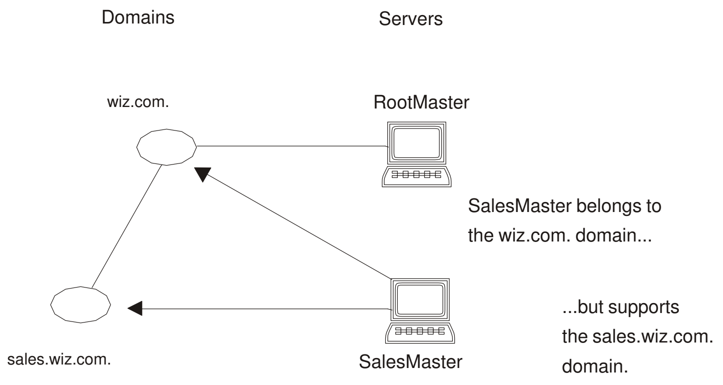
Over time, the client accumulates in its cache a copy of all the directory objects in the namespace and thus the IP addresses of the servers that support them. When it needs to send a request for access to another domain, it can usually find the name of its server in its directory cache and send the request directly to that server.

An NIS+ Server Is Also a Client

An NIS+ server is also an NIS+ client. In fact, before you can set up a workstation as a server, you must initialize it as a client. The only exception is the root master server, which has its own unique setup process.

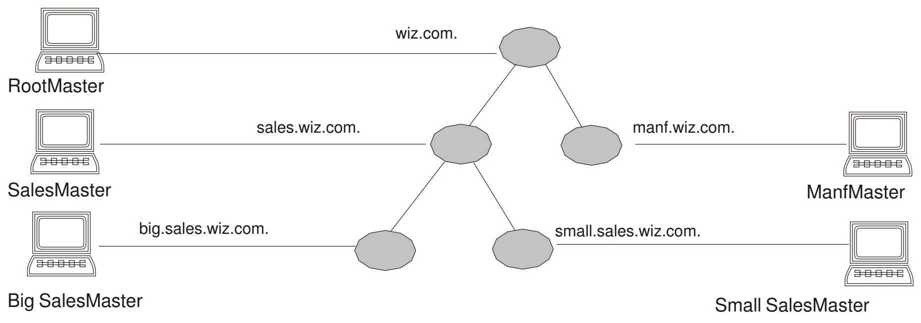
This means that in addition to *supporting* a domain, a server also *belongs* to a domain. In other words, by virtue of being a client, a server has a home domain. Its host information is stored in the Hosts table of its home domain, and its DES credentials are stored in the cred table of its home domain. Like other clients, it sends its requests for service to the servers listed in its directory cache.

An important point to remember is that—except for the root domain—a server's home domain is the *parent* of the domain the server supports.


Domains
Servers
wiz.com.
RootMaster
SalesMaster belongs to the wiz.com. domain...
...but supports the sales.wiz.com. domain.
sales.wiz.com.
SalesMaster

In other words, a server supports clients in one domain, but is a *client* of another domain. A server cannot be a client of a domain that it supports, with the exception of the root domain. Because they have no parent domain, the servers that support the root domain belong to the root domain itself.

For example, consider the namespace shown in the figure.



The following table lists which domain each server supports and which domain it belongs to:

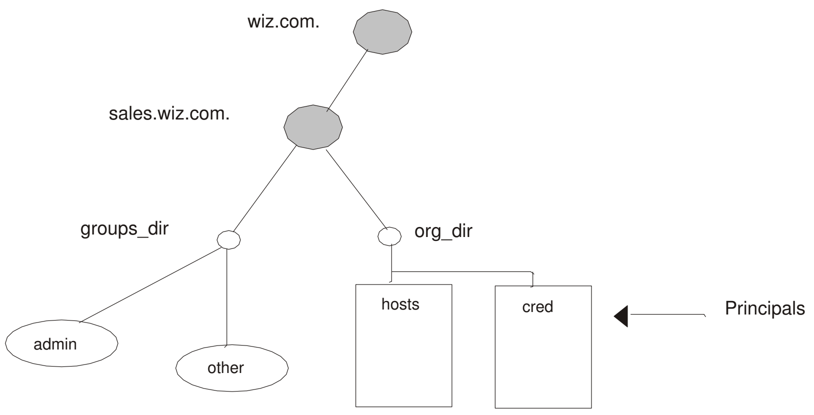
Server	Supports	Belongs to
RootMaster	wiz.com.	wiz.com.
SalesMaster	sales.wiz.com.	wiz.com.
BigSalesMaster	big.sales.wiz.com.	sales.wiz.com.
SmallSalesMaster	small.sales.wiz.com.	sales.wiz.com.
ManfMaster	manf.wiz.com.	wiz.com.

Naming Conventions

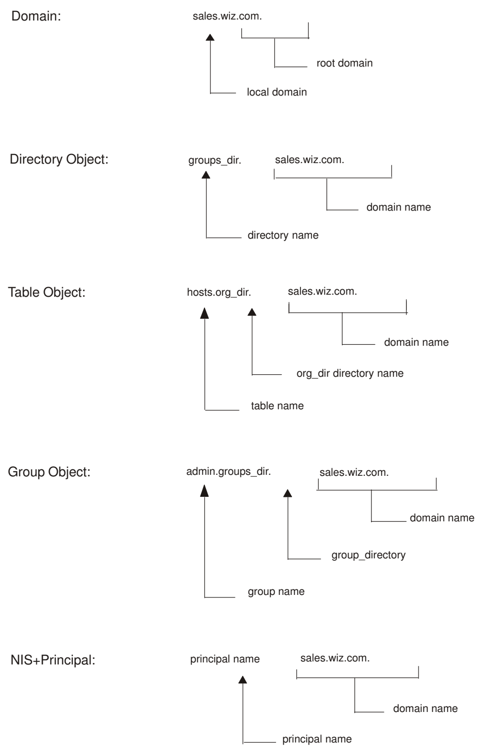
Objects in an NIS+ namespace can be identified with two types of names: *partially qualified* and *fully qualified*. A partially qualified name, also called a *simple* name, is simply the name of the object or any portion of the fully qualified name. If during any administration operation you type the partially qualified name of an object or principal, NIS+ will attempt to expand the name into its fully qualified version.

A fully qualified name is the complete name of the object, including all the information necessary to locate it in the namespace, such as its parent directory, if it has one, and its complete domain name, including a trailing dot.

This varies among different types of objects, so the conventions for each type, as well as for NIS+ principals, is described separately. This namespace will be used as an example in the following figure.



The fully qualified names for all the objects in this namespace, including NIS+ principals, are summarized in the following figure.



NIS+ Domain Names

A fully qualified NIS+ domain name is formed from left to right, starting with the local domain and ending with the root domain:

```
wiz.com.
sales.wiz.com.
intl.sales.wiz.com.
```

The first line shows the name of the root domain. The root domain must always have at least two labels and must end in a dot. The second label can be an Internet domain name, such as **com**. The second and third lines show the names of lower–level domains.

Directory Object Names

A directory's simple name is simply the name of the directory object. Its fully qualified name consists of its simple name plus the fully qualified name of its domain (which always includes a trailing dot):

`groups_dir` (simple name)

`groups_dir.manf.wiz.com.` (fully qualified name)

If you set up an unusual hierarchy in which several layers of directories do not form a domain, be sure to include the names of the intermediate directories. For example:

```
lowest_dir.lower_dir.low_dir.mydomain.com.
```

The simple name is normally used from within the same domain, and the fully qualified name is normally used from a remote domain. However, by specifying search paths in a domain's NIS_PATH environment variable, you can use the simple name from remote domains (see NIS+ Name Expansion), on page 4-16.

Tables and Group Names

Fully qualified table and group names are formed by starting with the object name and appending the directory name, followed by the fully qualified domain name. Remember that all system table objects are stored in an **org_dir** directory and all group objects are stored in a **groups_dir** directory. (If you create your own NIS+ tables, you can store them anywhere you like.) Here are some examples of group and table names:

```
admin.groups_dir.wiz.Inc.       admin.groups_dir.wiz.com.
admin.groups_dir.sales.wiz.Inc. admin.groups_dir.sales.wiz.com.
hosts.org_dir.wiz.Inc.          hosts.org_dir.wiz.com.
hosts.org_dir.sales.wiz.Inc.    hosts.org_dir.sales.wiz.com.
```

Table Entry Names

To identify an entry in an NIS+ table, you need to identify the table object and the entry within it. This type of name is called an *indexed* name. It has the following syntax:

```
[column=value,column=value,...],table-name
```

Column is the name of the table column. Value is the actual value of that column. Table–name is the fully qualified name of the table object. Here are a few examples of entries in the hosts table:

```
[addr=129.44.2.1,name=pine],hosts.org_dir.sales.wiz.com.
[addr=129.44.2.2,name=elm],hosts.org_dir.sales.wiz.com.
[addr=129.44.2.3,name=oak],hosts.org_dir.sales.wiz.com.
```

You can use as few column–value pairs inside the brackets as required to uniquely identify the table entry.

Some NIS+ administrative commands accept variations on this syntax. For details, see the **nistbladm**, **nismatch**, and **nisgrep** commands.

Host Names

Host names may contain up to 24 characters. Letters, numbers, the dash (**–**) and underscore (_) characters are allowed in host names. Host names are not case sensitive (that is, upper and lower case letters are treated as the same). The first character of a host name must be a letter of the alphabet. Blank spaces are not permitted in host names.

Note: Dots (.) are not permitted in host names. For example, a host name such as `myco.2` is not permitted. Dots are not allowed in host names even if they are enclosed in quotes. For example, `'myco.2'` is not permitted. Dot are only used as part of a fully qualified host name to identify the domain components. For example, `myco-2.sales.wiz.com` is a correct fully qualified host name.

Domains and hosts should not have the same name. For example, if you have a sales domain you should not have a machine named `sales`. Similarly, if you have a machine named `home`, you do not want to create a domain named home. This caution applies to subdomains, for example if you have a machine named `west` you do not want to create a `sales.west.myco.com` subdomain.

NIS+ Principal Names

NIS+ principal names are sometimes confused with Secure RPC netnames. Both types of names are described in the Security chapter, on page 7-1. However, one difference is worth pointing out now because it can cause confusion: NIS+ principal names *always* end in a dot and Secure RPC netnames *never* do:

`olivia.sales.wiz.com.` (NIS+ principal name)

`unix.olivia@sales.wiz.com` (Secure RPC netname)

Also, even though credentials for principals are stored in a cred table, neither the name of the cred table nor the name of the **org_dir** directory is included in the principal name.

Accepted Name Symbols

You can form namespace names from any printable character in the ISO Latin 1 set. However, the names cannot start with these characters:

```
@    <    >    +    [    ]    –    /
=    .    ,    :    ;
```

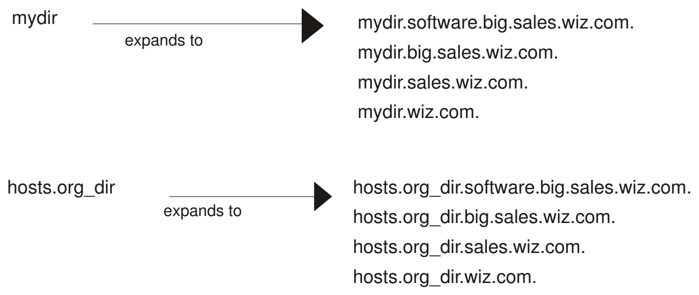
To use a string, enclose it in double quotes. To use a quote sign in the name, quote the sign too (for example, to use **o'henry**, type **o"'"henry**). To include white space (as in John Smith), use double quotes within single quotes, like this:

```
`"John Smith"`
```

See Host Names, on page 4-15 for restrictions that apply to host names.

NIS+ Name Expansion

Entering fully qualified names with your NIS+ commands can quickly become tedious. To ease the task, NIS+ provides a name–expansion facility. When you enter a partially qualified name, NIS+ attempts to find the object by looking for it under different directories. It starts by looking in the default domain. This is the home domain of the client from which you type the command. If it does not find the object in the default domain, NIS+ searches through each of the default domain's parent directories in ascending order until it finds the object. It stops after reaching a name with only two labels. Here are some examples (assume you are logged onto a client that belongs to the `software.big.sales.wiz.com.` domain).

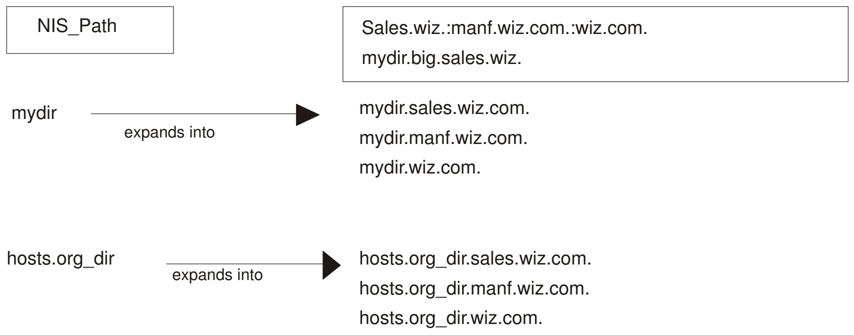

mydir
expands to
mydir.software.big.sales.wiz.com.
mydir.big.sales.wiz.com.
mydir.sales.wiz.com.
mydir.wiz.com.
hosts.org_dir
expands to
hosts.org_dir.software.big.sales.wiz.com.
hosts.org_dir.big.sales.wiz.com.
hosts.org_dir.sales.wiz.com.
hosts.org_dir.wiz.com.

NIS_PATH Environment Variable

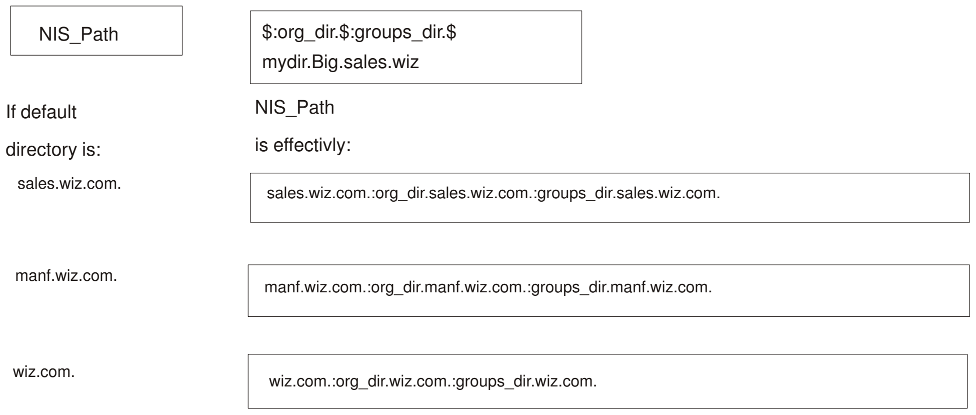
You can change or augment the list of directories NIS+ searches through by changing the value of the environment variable **NIS_PATH**. **NIS_PATH** accepts a list of directory names separated by colons:

```
NIS_PATH=directory1:directory2:directory3...;export NIS_PATH
```

NIS+ searches through these directories from left to right (see figure).



Like **$PATH** and **$MANPATH**, the **NIS_PATH** variable accepts the special symbol, **$**. You can append the **$** symbol to a directory name or add it by itself. If you append it to a directory name, NIS+ appends the default directory to that name (see figure).


NIS_Path
$:org_dir.$:groups_dir.$
mydir.Big.sales.wiz
If default
directory is:
NIS_Path
is effectivly:
sales.wiz.com.
sales.wiz.com.:org_dir.sales.wiz.com.:groups_dir.sales.wiz.com.
manf.wiz.com.
manf.wiz.com.:org_dir.manf.wiz.com.:groups_dir.manf.wiz.com.
wiz.com.
wiz.com.:org_dir.wiz.com.:groups_dir.wiz.com.

If you use the **$** sign by itself (for example, **org_dir.$:$**), NIS+ performs the standard name expansion described earlier: it starts looking in the default directory and proceeds through the parent directories. In other words, the default value of **NIS_PATH** is **$**.

Note: Keep in mind that additions and changes to your **NIS_PATH** may increase the number of lookups that NIS+ has to perform and thus slow down performance.

NIS+ Tables and Information

NIS+ stores a wide variety of network information in tables. This section describes the structure of those tables and provides a brief overview of how they can be set up.

NIS+ Table Structure

NIS+ tables provide several features not found in simple text files or maps. They have a column–entry structure, they accept search paths, they can be linked together, and they can be set up in several different ways. NIS+ provides preconfigured system tables, and you can also create your own tables. The following table lists the preconfigured NIS+ tables.

Table 13. NIS+ Tables

Table	Information in the Table
hosts	Network address and host name of every workstation in the domain
bootparams	Location of the root, swap, and dump partition of every diskless client in the domain
passwd	Password information about every user in the domain.
cred	Credentials for principals who belong to the domain
group	The group name, group password, group ID, and members of every UNIX group in the domain
netgroup	The netgroups to which workstations and users in the domain may belong
mail_aliases	Information about the mail aliases of users in the domain
timezone	The time zone of every workstation in the domain
networks	The networks in the domain and their canonical names
netmasks	The networks in the domain and their associated netmasks
ethers	The Ethernet address of every workstation in the domain
services	The names of IP services used in the domain and their port numbers
protocols	The list of IP protocols used in the domain
RPC	The RPC program numbers for RPC services available in the domain
auto_home	The location of all user's home directories in the domain
auto_master	Automounter map information
sendmailvars	The mail domain
client_info	Information about NIS+ clients

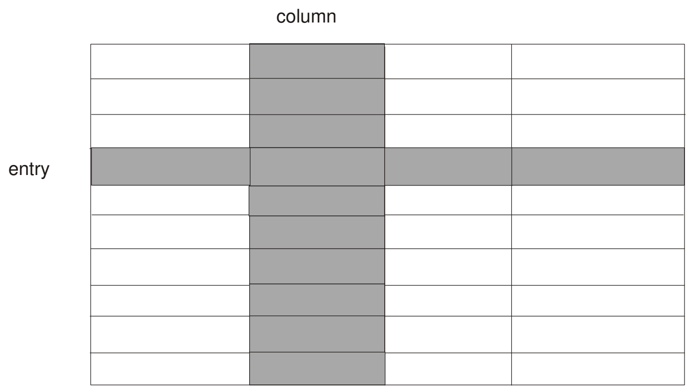
These tables store a wide variety of information, ranging from user names to Internet services. Most of this information is generated during a setup or configuration procedure. For instance, an entry in the passwd table is created when a user account is set up. An entry in the hosts table is created when a workstation is added to the network. And an entry in the networks table is created when a new network is set up.

Since this information is generated from such a wide field of operations, much of it is beyond the scope of this manual. However, as a convenience, the appendix of this book summarizes the information contained in each column of the tables, providing details only when necessary to keep things from getting confusing, such as when distinguishing groups from NIS+ groups and netgroups. For thorough explanations of the information, consult *AIX 4.3 System Management Guide: Communications and Networks*.

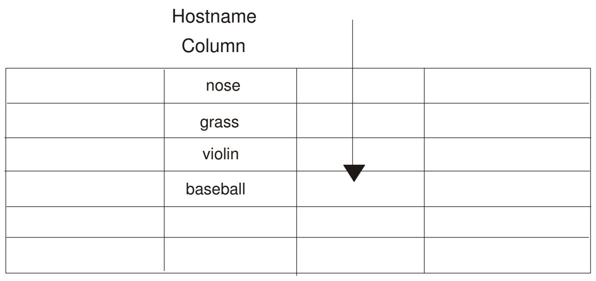
The Cred table, because it contains only information related to NIS+ security, is described in Administering NIS+ Credentials, on page 6-2.

Columns and Entries

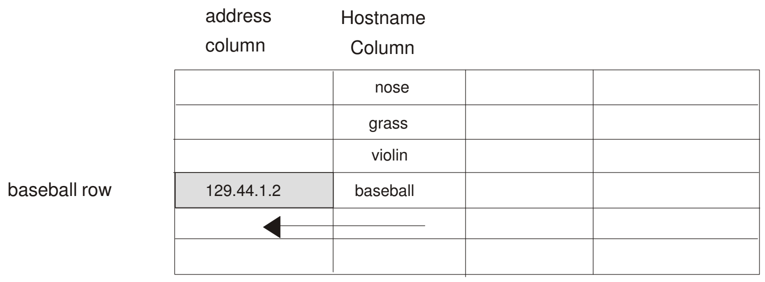
Although NIS+ tables store different types of information, they all have the same underlying structure; they are each made up of rows and columns (the rows are called *entries* or *entry objects*).



A client can access information by a key, or by any column that is searchable. For example, to find the network address of a workstation named **baseball**, a client could look through the hostname column until it found **baseball**.



it then would move along the baseball entry to find its network address.



Because a client can access table information at any level, NIS+ provides security mechanisms for all three levels. For instance, an administrator could assign read rights to everyone for a table at the object level, modify rights to the owner at the column level, and modify rights to the group at the entry level. Details about table security are provided in Administering NIS+ Access Rights, on page 6-21.

Search Paths

A table contains information only about its local domain. For instance, tables in the `wiz.com.` domain contain information only about the users, clients, and services of the `wiz.com.` domain. The tables in the `sales.wiz.com.` domain store information only about the users, clients, and services of the `sales.wiz.com.` domain. And so on.

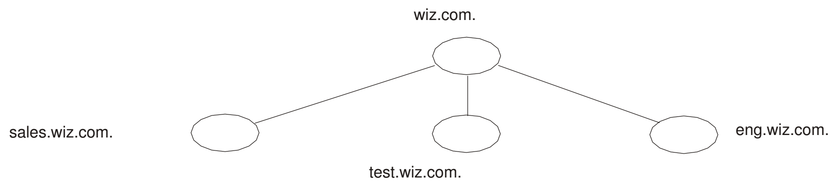
If a client in one domain tries to find information that is stored in another domain, it has to provide a fully qualified name. As described in NIS+ Name Expansion, on page 4-16, if the **NIS_PATH** environment variable is set up properly, the NIS+ service will do this automatically.

Every NIS+ table can also specify a **search path** that a server will follow when looking for information. The search path is an ordered list of NIS+ tables, separated by colons:

*table***:***table***:***table***...**

The table names in the search path do not have to be fully qualified; they can be expanded just like names entered at the command line. When a server cannot find information in its local table, it returns the table's search path to the client. The client uses that path to look for the information in every table named in the search path, in order, until it finds the information or runs out of names.

Here is an example that demonstrates the benefit of search paths. Assume a domain hierarchy as shown in the following figure.



The hosts tables of the lower three domains have the contents similar to that described in the following example. Assume now that a user logged onto a client in the `sales.wiz.com.` domain wants to log in remotely to another client. If that user does not provide a fully qualified name, it can only remotely log on to five workstations: **vermont**, **maine**, **cherry**, **apple**, and the **mailhost**.

sales.wiz.com.		test.wiz.com.		eng.wiz.com.	
127.0.0.1	localhost	127.0.0.1	localhost	127.0.0.1	localhost
129.44.2.10	vermont	129.44.4.10	nebraska	129.44.3.10	georgia
129.44.2.11	maine	129.44.4.11	oklahoma	129.44.3.11	florida
129.44.2.12	cherry	129.44.4.12	corn	129.44.3.12	orange
129.44.2.13	apple	129.44.4.13	wheat	129.44.3.13	potato
129.44.2.14	mailhost	129.44.4.14	mailhost	129.44.3.14	mailhost

Now assume that the search path of the hosts table in the sales.wiz.com. domain listed the hosts tables from the test.wiz.com. and eng.wiz.com. domains:

```
hosts.org_dir.test.wiz.com.:hosts.org_dir.eng.wiz.com.
```

Now a user in the `sales.wiz.com.` domain can enter something like **rlogin oklahoma**, and the NIS+ server will find it. It will first look for **oklahoma** in the local domain, but when it does not find a match, it will look in the `test.wiz.com.` domain. How does the client know how to find the `test.wiz.com.` domain? The information is stored in its directory cache. If it is not stored in its directory cache, the client will obtain the information by following the process described in this chapter.

There is a slight drawback, though, to specifying a search path. If the user were to enter an incorrect name, such as **rlogin potatoe**, the server would need to look through three tables—instead of just one—before returning an error message. If you set up search paths throughout the namespace, an operation may end up searching through the tables in 10 domains instead of just 2 or 3. Another drawback is a performance loss from having many clients contact more than one set of servers when they need to access NIS+ tables.

You should also be aware that since "mailhost" is often used as an alias, when trying to find information about a specific mailhost, you should use its fully qualified name (for example, `mailhost.sales.wiz.com.`), or NIS+ will return *all* the mailhosts it finds in all the domains it searches through.

You can specify a table's search path by using the **–p** option to the **nistbladm** command.

Ways to Set Up Tables

Setting up NIS+ tables involves three or four tasks:

1. Creating the **org_dir** directory
2. Creating the system tables
3. Creating nonsystem tables (optional)
4. Populating the tables with information

NIS+ system tables are stored under an **org_dir** directory. So, before you can create any tables:

- Use the **nisserver** script. The **nisserver** script creates the appropriate directories and a full set of system tables. Running the **nisserver** script is the recommended method.
- Use the **nismkdir** command. The **nismkdir** command simply creates the directory.
- Use the /**usr**/**lib**/**nis**/**nissetup** utility. The **nissetup** utility creates the **org_dir** and **groups_dir** directories and a full set of system tables.

The **nisserver** script and the **nissetup** and **nismkdir** utilities are described in their reference descriptions.

A benefit of the **nissetup** utility is its capability to assign the proper access rights to the tables of a domain whose servers are running in NIS–compatibility mode. When entered with the **–Y** flag, it assigns read permissions to the nobody class of the objects it creates, allowing NIS clients, who are unauthenticated, to get information from the domain's NIS+ tables.

The NIS+ system tables and the type of information they store are described in the appendix. To create them, you could use one of the three ways mentioned above. The **nistbladm** utility also creates and modifies NIS+ tables. You could conceivably create all the tables in a namespace with the **nistbladm** command, but you would have to type much more and you would have to know the correct column names and access rights. A much, much easier way is to use the **nisserver** script.

To create a nonsystem table—that is, a table that has not been preconfigured by NIS+—use the **nistbladm** command.

You can populate NIS+ tables in three ways: from NIS maps, from ASCII files (such as /**etc** files), and manually.

If you are upgrading from the NIS service, you already have most of your network information stored in NIS maps. You **do not** have to re–enter this information manually into NIS+ tables. You can transfer it automatically with the **nispopulate** script or the **nisaddent** utility.

If you are not using another network information service, but maintain network data in a set of /**etc** files, you **do not** have to re–enter this information, either. You can transfer it automatically, also using the **nispopulate** script or the **nisaddent** utility.

If you are setting up a network for the first time, you may not have much network information stored anywhere. In that case, you'll need to first get the information and then enter it manually into the NIS+ tables. You can do this with the **nistbladm** command. You can also do it by entering all the information for a particular table into an **input file**—which is essentially the same as an /**etc** file—and then transferring the contents of the file with the **nispopulate** script or the **nisaddent** utility.

How Tables Are Updated

When a domain is set up, its servers receive their first versions of the domain's NIS+ tables. These versions are stored on disk, but when a server begins operating, it loads them into memory. When a server receives an update to a table, it immediately updates its memory–based version of the table. When it receives a request for information, it uses the memory–based copy for its reply.

Of course, the server also needs to store its updates on disk. Since updating disk–based tables takes time, all NIS+ servers keep **log** files for their tables. The log files are designed to temporarily store changes made to the table, until they can be updated on disk. They use the table name as the prefix and append **.log**. For example:

```
hosts.org_dir.log
bootparams.org_dir.log
password.org_dir.log
```

You should update disk–based copies of a table on a daily basis so that the log files do not grow too large and take up too much disk space. This process is called *checkpointing*. To do this, use the **nisping –C** command.

Chapter 5. NIS+ Installation and Configuration

This chapter discusses prerequisites and procedures for installing and configuring NIS+.

Setting Up NIS+

This section describes two different methods of setting up an NIS+ namespace:

- **With the setup scripts**—how to set up NIS+ using the three NIS+ scripts: **nisserver**, **nispopulate**, and **nisclient**. This is the easiest method, and recommended, method.
- **With the** NIS+ **command set**—how to set up NIS+ using the NIS+ command set. While this method gives you more flexibility than the scripts method, it is more difficult. This method should be used only by experienced NIS+ administrators who need to set up a namespace with characteristics significantly different than those provided by the setup scripts.

See NIS+ Administration for information on how to remove an NIS+ directory or domain, an NIS+ server, or the NIS+ namespace.

Prerequisites for Installing and Configuring NIS+

Before you start to set up NIS+ at your site, you need to

1. Plan your NIS+ layout. See Planning Your NIS+ Layout, on page 5-2 and use the planning worksheets in Configuration Worksheets, on page 5-5. See Moving from NIS to NIS+ for a complete description of the planning process.
2. Prepare your existing namespace (if any). See Preparing the Existing Namespace, on page 5-4.
3. Choose a root domain name.
4. Choose a root server machine.
5. Make sure that you have at least one system already running at your site that can be used as your root master server. This machine must contain at least one user (root) in the system information files, such as /**etc**/**passwd**. (Machines usually come with root in the system files, so this should not be a problem.)

To create the sample namespace, you need only do steps 2, 4, and 5 above. The tutorial does the NIS+ layout planning for you and chooses a domain name.

Planning Your NIS+ Layout

To plan the structure of your NIS+ namespace:

- Determine your server requirements.
- Determine your disk space and memory requirements.
- Sketch the domain hierarchy.
- Select servers to be used for the namespace.
- Determine the administrative groups and their members.
- Determine access rights to the namespace.

See Designing the NIS+ Namespace, on page 3-5 for a full description of these steps and use the Configuration Worksheets, on page 5-5 to help plan your namespace.

You do not have to do any planning to work through the tutorial in Setting Up NIS+ With Scripts, on page 5-8. You just need a few networked machines to practice on. But be sure to plan your site's hierarchy before you move from the tutorial to setting up your real NIS+ namespace.

Determining Server Requirements

Once you have determined the domain structure of your namespace, you can choose the servers that will support them. You need to differentiate between the requirements imposed by NIS+ and those imposed by the traffic load of your namespace.

NIS+ requires you to assign at least one server, the master, to each NIS+ domain. Although you can assign any number of replicas to a domain, more than 10 per domain is not recommended. An NIS+ server is capable of supporting more than one domain, but this is not recommended except in small namespaces or testing situations. The number of servers a domain requires is determined by the traffic load and the configuration of its servers.

Here are some guidelines for determining how many servers you will need:

- Assign one master server per domain in the hierarchy.
- Add at least one replica server for each domain. (A replica can answer requests when the master is unavailable.)

Note: If the root master server is unavailable and the NIS+ domain is being served solely by a replica, you can obtain information from the NIS+ tables, but changes to the original tables can be made only when the master server is available.

- Calculate the disk space requirements of each server. The next section describes how to calculate disk space usage.

Evaluate Disk Space and Memory Requirements

Disk space requirements depend on four factors:

- Disk space consumed by the Solaris 2.5 software
- Disk space for /**var**/**nis** (and /**var**/**yp**)
- Amount of memory
- Swap space required for NIS+ processes

The base AIX Version 4.3.3 software requires at least 32 MBs of disk space. You should also count the disk space consumed by other software the server may use. For more details on the base operating system installation and requirements, see the *AIX 4.3 Installation Guide*.

Although NIS+ is part of the AIX Version 4.3.3 distribution, it is not automatically installed in the base installation. NIS+ directories, groups, tables, and client information are stored in /**var**/**nis**. The /**var**/**nis** directory uses about 5 KBs of disk space per client. For example purposes only, if a namespace has 1000 clients, /**var**/**nis** requires about 5 MBs of disk space. However, because transaction logs (also kept in /**var**/**nis**) can grow large, you may want additional space per client—an additional 10–15 MBs is recommended. In other words, for 1000 clients, allocate 15 to 20 MBs for /**var**/**nis**. You can reduce this if you checkpoint transaction logs regularly. You should create a separate file system for /**var**/**nis** on the servers. This separate file system will help during an operating system upgrade.

If you are going to load information into NIS+ from NIS maps, allocate an appropriate amount of space for /**var**/**yp** to hold those NIS maps.

You also need swap space equal to three times or more of the size of the NIS+ server process—in addition to the server's normal swap space requirements. The size of the **rpc.nisd** process as shown by the **ps –efl** command. Most of this space is used during callback operations or when directories are checkpointed (with **nisping –C**) or replicated, because during such procedures an entire NIS+ server process is forked. See Configuring a Client as an NIS+ Server, on page 5-22 and the **rpc.nisd** documentation for more information.

Preparing the Existing Namespace

If an NIS domain already exists at your site, you can use the same flat domain structure for your NIS+ namespace if you like. (You can change it later to a hierarchical structure.) Read before you start your transition from NIS to NIS+ for important planning and preparation information. The NIS+ scripts easily enable you to start NIS+ with data from NIS maps. Setting Up NIS+ With Scripts, on page 5-8, shows you how to use the NIS+ scripts to create a NIS+ namespace from either system files or NIS maps.

However, in order for the scripts to run smoothly, you must prepare your existing namespace (if you have one) for conversion to NIS+.

For your reference, key preparations are summarized below:

Domain and host names

Domains and hosts must not have the same name. For example, if you have a `sales` domain you cannot have a machine named `sales`. Similarly, if you have a machine named `home`, do not create a domain named `home`. This caution also applies to subdomains; for example, if you have a machine named `west,` you do not want to create a `sales.west.myco.com` subdirectory.

No dots in host names

Because NIS+ uses dots (periods) to delimit between machine names and domains and between parent and subdomains, you cannot have a machine name containing a dot. Before converting to NIS+ (before running the scripts) you must eliminate any dots in your host names. You should convert host name dots to hyphens. For example, you cannot have a machine named `sales.alpha`. You can convert that name to `sales-alpha`.

Root server must be running

The machine that will be designated the root server must be up and running and you must have superuser access to it.

View any existing local /etc files or NIS maps that you will be loading data from.

Make sure that there are no spurious or incorrect entries. Make sure that the right data is in the correct place and format. Remove any outdated, invalid, or corrupt entries. You should also remove any incomplete or partial entries. You can always add individual entries after setup is completed. That is easier than trying to load incomplete or damaged entries.

Configuration Worksheets

If you have more than one domain, make copies of the blank worksheets.

Table 14. Servers, Credentials, Directories, and Groups Worksheet

Domain:						
Servers	**Type**	**Name**				**Specifications**
	Master					
	First Replica					
	Second Replica					
Credentials	**Type of Principal**	**Type of Credential**				
	Servers					
	Clients					
	Administrators					
	Users					
Rights	**Types of Objects**	**Category & Rights**				
	Directories	**N**	**O**	**G**	**W**	**Use Defaults?**
	Groups	**N**	**O**	**G**	**W**	**Description**

Table 15. NIS+ Tables Worksheet

Domain:						
Rights	**Types of Objects**	**Category & Rights**				
	Tables	**N**	**O**	**G**	**W**	**Notes**
	bootparams					
	hosts					
	passwd					
	cred					
	group					
	netgroup					
	mail_aliases					
	timezone					
	networks					
	netmasks					
	ethers					
	services					
	protocols					
	rpc					
	auto_home					
	auto_master					
	sendmailvars					
	client_info					

NIS+ Setup Scripts

This section describes the NIS+ scripts, specifically:

- What the NIS+ Scripts Will Do, on page 5-8
- What the NIS+ Scripts Will Not Do, on page 5-8
- Setting Up NIS+ With Scripts, on page 5-8
- Creating a Sample NIS+ Namespace, on page 5-9
- Setting Up NIS+ Root Servers, on page 5-12
- Populating NIS+ Tables, on page 5-15
- Setting Up Root Domain NIS+ Client Machines, on page 5-20
- Initializing NIS+ Client Users, on page 5-21
- Setting Up NIS+ Servers, on page 5-22
- Designating Root Replicas, on page 5-23
- Creating a Subdomain, on page 5-25
- Populating the New Domain's Tables, on page 5-26
- Designating Replicas, on page 5-28
- Initializing Subdomain NIS+ Client Machines, on page 5-29
- Initializing Subdomain NIS+ Client Users, on page 5-30
- Summary of Commands for the Sample NIS+ Namespace, on page 5-30

Note: Before running the NIS+ setup scripts, make sure you have performed the steps described in Prerequisites for Installing and Configuring NIS+, on page 5-2.

The three NIS+ scripts—**nisserver**, **nispopulate**, and **nisclient**—enable you to set up an NIS+ namespace easily. The NIS+ scripts are shell scripts that execute groups of NIS+ commands so you do not have to type the NIS+ commands individually. The following table describes what each script does.

Table 16. NIS+ Scripts

NIS+ **Script**	**What It Does**
nisserver	Sets up the root master, nonroot master and replica servers with level 2 security (DES) by default. Additionally, can be used to remove a server.
nispopulate	Populates NIS+ tables in a specified domain from their corresponding system files or NIS maps.
nisclient	Creates NIS+ credentials for hosts and users; initializes NIS+ clients and users. Can be used to restore the previous environment or to remove a client.

What the NIS+ Scripts Will Do

In combination with a few NIS+ commands, you can use the NIS+ scripts to perform all the tasks necessary for setting up an NIS+ namespace. See the **nisserver**, **nispopulate**, and **nisclient** command descriptions for a complete discussion of these commands and their options. Setting Up NIS+ With Scripts, on page 5-8, shows you how to use the NIS+ scripts to set up an NIS+ namespace.

You can run each of the scripts without having the commands execute by using the **–x** option. This option lets you see what commands the scripts call and their approximate output without the scripts actually changing anything on your systems. First running the scripts with **–x** may minimize unexpected surprises.

What the NIS+ Scripts Will Not Do

While the NIS+ scripts reduce the effort required to create an NIS+ namespace, the scripts do not completely replace the individual NIS+ commands. The scripts only implement a subset of NIS+ features.

If you are unfamiliar with NIS+, you may wish to refer back to this section after you have created the sample NIS+ namespace.

The **nisserver** script will only set up an NIS+ server with the standard default tables and permissions (authorizations). This script does **not**:

- Set special permissions for tables and directories
- Add extra NIS+ principals to the NIS+ admin group

See Setting Up NIS+ With Scripts, on page 5-8, for how to use the **nisgrpadm** command instead of one of the NIS+ scripts to add extra NIS+ principals to the NIS+ admin group.

- Create private tables.
- Run an NIS+ server at security level 0. This is a temporary state.
- Start the **rpc.nisd**, **rpc.nispasswdd**, and **nis_cachemgr** daemons on remote servers. When starting the **rpc.nisd** daemon, use the **–r –S 0** options. This step is required to complete server installation.

See Setting Up NIS+ With Scripts, for how to use the **rpc.nisd** command instead of one of the NIS+ scripts to change NIS+ client machines into nonroot servers.

The **nisclient** script does not set up an NIS+ client to resolve host names using DNS. You need to explicitly set DNS for clients that require this option.

Setting Up NIS+ With Scripts

This section shows you how to set up a basic NIS+ namespace using the *nisserver*, *nispopulate*, and *nisclient* scripts in combination with a few NIS+ commands.

Note: Using these scripts is the recommended method of setting up and configuring an NIS+ namespace. It is much simpler to use these scripts than to try setting up an NIS+ namespace with the NIS+ command set.

See the **nisserver**, **nispopulate**, and **nisclient** command descriptions for more information about the scripts.

You should *not* use the small sample NIS+ namespace described in this tutorial as a basis for your actual NIS+ namespace. You should destroy the sample namespace once you are done exploring it, instead of ”adding on” to it. It is better to begin again and carefully plan your NIS+ hierarchy before you create your actual namespace.

The following table summarizes the recommended generic setup procedure. The left column lists the major setup activities, such as setting up the root domain or creating a client. The text in the middle describes the activities. The third column lists which script or NIS+ commands accomplish each step.

Table 17. Recommended NIS+ Setup Procedure Overview

Activity	**Description**	**Script**/NIS+ **Commands**
Plan your new NIS+ namespace	Plan your new NIS+ namespace. See Prerequisites for Installing and Configuring NIS+, on page 5-2 for a full discussion of planning requirements and steps. (If you are just following the NIS+ tutorial in a test–bed network, this step has been done for you.)	
Prepare your existing namespace	In order for the scripts to work best, your current namespace (if any) must be properly prepared. See Preparing the Existing Namespace, on page 5-4 for a description of necessary preparations. (If you are just following the NIS+ tutorial in a test–bed network, this step has been done for you.)	
Set up root Domain	Create the root domain. Set up and initialize the root master server. Create the root domain admin group.	**nisserver**
Populate tables	Populate the NIS+ tables of the root domain from text files or NIS maps. Create credentials for root domain clients. Create administrator credentials.	**nispopulate**, **nisgrpadm**, **nisping**
Set up root domain clients	Set up the client machines. (Some of them will subsequently be converted into servers.) Initialize users as NIS+ clients.	**nisclient**
Enable servers	Enable some clients of the root domain to become servers. Some servers will later become root replicas; others will support lower–level domains.	**nisserver**
Set up root replicas	Designate one or more of the servers you just set up as replicas of the root domain.	**nisserver –R**
Set up nonroot domains	Create a new domain. Designate previously enabled server as its master. Create its admin group and admin credentials.	**rpc.nisd**
Populate tables	Create credentials for clients of the new domain. Populate the NIS+ tables of the new domain from text files or NIS maps.	**nispopulate**
Set up nonroot domain clients	Set up the clients of the new domain. (Some may subsequently be converted into servers for lower–level domains.) Initialize users as NIS+ clients.	**nisclient**

The NIS+ scripts enable to you to skip most of the individual procedures included in the above activities.

Creating a Sample NIS+ Namespace

The procedures in this section show you how to create a sample NIS+ namespace. The sample NIS+ namespace will be created from /**etc** files and NIS maps. This sample shows you how to use the scripts both when your site is not running NIS and when NIS is running at your site. You can set your servers to NIS–compatibility mode if they will be serving NIS clients. See Using NIS–Compatibility Mode, on page 3-23 for more information on NIS–compatibility mode.

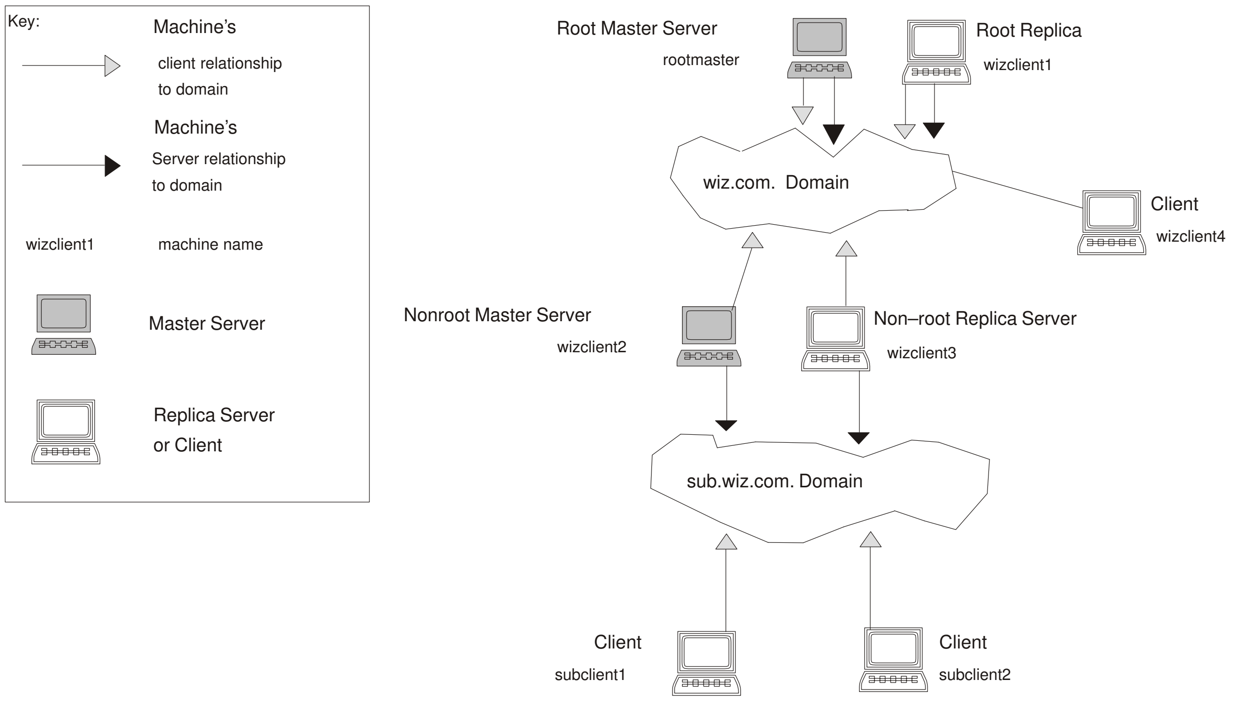
Note: Your site's actual NIS+ namespace and its domain hierarchy will probably differ from the sample namespace's, and yours will probably contain a different **number** of servers, clients, and domains. Do not expect any resemblance between your final domain configuration or hierarchy and the sample one. The sample namespace is merely an illustration of how to use the NIS+ scripts. Once you have created this sample namespace, you should have a clear idea about how to create domains, servers, and clients at your site.

The sample namespace will contain the following components:

- A root master server named `master` for the wiz.com. domain
- Four clients of the root domain, `wiz.com`:
- The first client, `wizclient1`, will become a root replica (for the wiz.com. domain).
- The second client, `wizclient2`, will become a master server for a new subdomain (for the subwiz.wiz.com. domain).
- The third client, `wizdlient3`, will become a nonroot replica server of the new subdomain (for the subwiz.wiz.com. domain).
- The fourth client, `wizclient4`, will remain solely a client of the root domain (wiz.com.).
- Two clients, `subclient1` and `subclient2`, of the subdomain (subwiz.wiz.com.)

This scenario shows the scripts being used to set up NIS+ at a site that uses both system information files, such as /**etc**/**hosts**, and NIS maps to store network service information. The sample NIS+ namespace uses such a mixed site purely for example purposes.

The figure shows the layout of the sample namespace. When you finish creating the sample domain, it should resemble the NIS+ domain in this figure. Notice that some machines are simultaneously servers and clients.



The following table contains the generic sequence of NIS+ scripts and commands you will use to create the NIS+ domains shown in the previous figure. Subsequent sections describe these command lines in detail. After you are familiar with the tasks required to create NIS+ domains, servers, and clients, use this table as a quick–reference guide to the appropriate command lines.The table summarizes the actual commands with the appropriate variables that you will type to create the sample NIS+ namespace.

Table 18. NIS+ Domains Setup Command Lines Summary

Action	**Machine**	**Command**

Include /**usr**/**lib**/**nis** in root's path.	Root master server and client machines as superuser	`PATH=$PATH:/usr/lib/nis; \`
		`  export PATH`

Create a root master server without or with NIS (YP) compatibility.	Root master server as superuser	`nisserver -r -d` *`newdomain`*`.`
		or
		`nisserver -Y -r -d` *`newdomain`*`.`

Populate the root master server tables from files or from NIS maps.	Root master server as superuser	`nispopulate -F -p` *`/files`* `\`
		`  -d` *`newdomain`*`.`
		or
		`nispopulate -Y -d` *`newdomain`*`. \`
		`  -h` *`NIS_servername`* `\`
		`  -a` *`NIS_server_ipaddress`*
		`  -y` *`NIS_domain`*

Add additional users to the NIS+ admin group.	Root master server as superuser	`nisgrpadm -a admin.`*`domain`*`.\`
		`  `*`name`*`.`*`domain`*`.`

Make a checkpoint of the NIS+ database	Root master server as superuser	`nisping -C` *`domain`*`.`

Initialize a new client to the server.	Server machine as superuser	`nisclient -c` *`clientname`*
		Note: If you get an error that this client does not exist, add the following entry to the hosts table:
		`nistbladm -a name=`*`clientname`*`\`
		`  cname=`*`clientname`* `\`
		`  addr=`*`ipaddress`* `hosts.org_dir`
		If *clientname* exists in /**etc**/**hosts**, you do not need to run this command.

Initialize a new client machine.	Client machine as superuser	`nisclient -i \`
		`  -d` *`domain`*`.`
		`  -h` *`rootmaster`*

Initialize user as an NIS+ client.	Client machine as user	`nisclient -u`

Start the **rpc.nisd** daemon—required to convert a client to a server without or with NIS (and DNS) compatibility.	Client machine as superuser	`startsrc -s rpc.nisd`
		or
		`startsrc -s rpc.nisd -a "-Y"`
		or
		`startsrc -s rpc.nisd -a "-Y-B"`

Convert a server to a root replica.	Root master server as superuser	`nisserver -R \`
		`  -d` *`domain`*`. \`
		`  -h` *`clientname`*

Convert a server to a nonroot master server.	Root master server as superuser	`nisserver -M \`
		`  -d` *`newsubdomain`*`.`*`domain`*`. \`
		`  -h` *`clientmachine`*

Action	Machine	Command

Populate the new master server tables from files or from NIS maps.	New subdomain master server as superuser	`nispopulate -F \`
		` -p /subdomaindirectory \`
		` -d newsubdomain.domain.`
		or
		`nispopulate -Y \`
		` -d newsubdomain.domain.\`
		` -h NIS_servername \`
		` -a NIS_server_ipaddress \`
		` -y NIS_domain`

Convert a client to a master server replica.	Subdomain master server as superuser	`nisserver -R \`
		` -d subdomain.domain.\`
		` -h clientname`

Initialize a new client of the subdomain. Clients can be converted to subdomain replicas or to another server.	New subdomain client machine as superuser	`nisclient -i \`
		` -d newsubdomain.domain.\`
		` -h subdomainmaster`

Initialize user as an NIS+ client.	Client machine as user	`nisclient -u`

Note: To see what commands an NIS+ script will call without actually having the commands execute, use the **–x** option. The **–x** option will cause the command names and their approximate output to echo to the screen as if you were actually running the script. Running the scripts for the first time with **–x** may minimize unexpected results.

Setting Up NIS+ Root Servers

Setting up the root master server is the first activity towards establishing NIS+ domain. This section shows you how to set up a root master server using the **nisserver** script with default settings. The root master server will use the following defaults:

- Security level 2 (DES)—the highest level of NIS+ security
- NIS compatibility set to OFF (instructions for setting NIS compatibility are included)
- System information files (/**etc**) or NIS maps as the source of name services information
- **admin**.*domainname* as the NIS+ group

Prerequisites

Check to see that the /**etc**/**passwd** file on the machine you want to be root master server contains an entry for root.

You need the following:

- The superuser password of the workstation that will become the root master server
- The name of the new root domain

In the following example, the machine that will be designated the root master server is called `rootmaster`, and wiz.com. will be the new root domain.

Attention: Domains and hosts should not have the same name. For example, if you have `wiz.com` as a root domain you should not have a machine named `wiz` in any of your domains. Similarly, if you have a machine named `home`, you do not want to create a domain named `home`. This caution applies to subdomains; for example, if you have a machine named `west`, you do not want to create a `sales.west.myco.com` subdomain. If names are not unique, NIS+ cannot parse addresses correctly. See the **chypdom** command description.

Procedure — Creating a Root Master Server

1. Set the superuser's **PATH** variable to include **/usr/lib/nis**. Add this path to root's **.profile** file or set the variable directly.
2. Type the following command as superuser (root) to set up a root master server.

   The **–r** option indicates that a root master server should be set up. The **–d** option specifies the NIS+ domain name.

   ```
   rootmaster# nisserver –r –d wiz.com.
   This script sets up this machine ”rootmaster” as an NIS+
   root master server for domain wiz.com.
   Domain name                  : wiz.com.
   NIS+ group                  : admin.wiz.com.
   NIS (YP) compatibility       : OFF
   Security level               : 2=DES
   Is this information correct? (type ’y’ to accept, ’n’ to change)
   ```

   ”NIS+ `group`” refers to the group of users who are authorized to modify the information in the wiz.com. domain. (Domain names always end with a period.) Modification includes deletion. **admin.***domainname* is the default name of the group. See Changing Incorrect Information, on page 5-14 for instructions on how to change this name.
   NIS compatibility refers to whether an NIS+ server will accept information requests from NIS clients. When set to **OFF**, the default setting, the NIS+ server will not fulfill requests from NIS clients. When set to **ON**, an NIS+ server will fulfill such requests. You can change the NIS–compatibility setting with this script. See Changing Incorrect Information, on page 5-14.

Note: This script only sets machines up at security level 2, the highest level of NIS+ security. You cannot change the security level when using this script. After the script has completed, you can change the security level with the appropriate NIS+ command. See NIS+ Security Levels, on page 7-4 and the **rpc.nisd** documentation for more information on changing security levels.

3. Type **y** (if the information shown on the screen is correct).

   Typing **n** causes the script to prompt you for the correct information. (See Changing Incorrect Information, on page 5-14 for what you need to do if you type **n**.)

   ```
   Is this information correct? (type ’y’ to accept, ’n’ to change) y
   This script will set up your machine as a root master server for domain
   wiz.com. without NIS compatibility at security level 2.
   Use ”nisclient –r” to restore your current network service environment.
   Do you want to continue? (type ’y’ to continue, ’n’ to exit the script)
   ```

4. Type **y** to continue the NIS+ setup.
   (Typing **n** safely stops the script.) If you interrupt the script after you have chosen **y** and while the script is running, the script stops running and leaves set up whatever it has created so far. The script does not do any automatic recovery or cleaning up. You can always rerun this script.

   ```
   Do you want to continue? (type ’y’ to continue, ’n’ to exit the script
   setting up domain information ”wiz.com.” ...
   The rpc.nisd Subsystem has been started.
   The nis_cachemgr Subsystem has been started.
   The rpc.nispasswdd Subsystem has been started.
   running nisinit ...
   This machine is in the wiz.com. NIS+ domain.
   Setting up root server ...
   All done.
   starting root server at security level 0 to create credentials...
   running nissetup ...
   (creating standard directories & tables)
   org_dir.wiz.com. created
   ...
   ...
   Enter login password:
   ```

The **nissetup** command creates the directories for each NIS+ table.

5. Type your machine's root password at the prompt and press Return.

   In this case, the user typed the **rootmaster** machine's root password.

   ```
   Wrote secret key into /etc/.rootkey
   setting NIS+ group to admin.wiz.com. ...
   restarting root server at security level 2 ...
   This system is now configured as a root server for domain wiz.com. You
   can now populate the standard NIS+ tables by using the nispopulate or
   /usr/lib/nis/nisaddent commands.
   ```

Your root master server is now set up and ready for you to populate the NIS+ standard tables. To continue with populating tables, skip to Populating NIS+ Tables, on page 5-15.

Procedure — Changing Incorrect Information

If you typed **n** because some or all of the information returned to you was wrong in 3 Step 3 in the above procedure, you will see the following:

```
Is this information correct? (type 'y' to accept, 'n' to change) n
Domain name: [wiz.com.]
```

1. Press Return if the domain name is correct; otherwise, type the correct domain name and press Return.

   In this example, Return was pressed, confirming that the desired domain name is wiz.com. The script then prompts for the NIS+ group name.

   ```
   Is this information correct? (type 'y' to accept, 'n' to change) n
   Domain name: [wiz.com.]
   NIS+ group: [admin.wiz.com.]
   ```

2. Press Return if NIS+ group is correct; otherwise, type the correct NIS+ group name and press Return.

   In this example, the name was changed. The script then prompts for NIS compatibility.

   ```
   NIS+ group: [admin.wiz.com.] netadmin.wiz.com.
   NIS (YP) compatibility (0=off, 1=on): [0]
   ```

3. Press Return if you do not want NIS compatibility; otherwise, type **1** and press Return.

   In this example, Return was pressed, confirming that NIS compatibility status is correct. Once again, the script asks you if the information is correct.

Note: If you choose to make this server NIS compatible, you also need to edit a file and restart the **rpc.nisd** daemon before it will work. See Configuring an NIS+ Server, on page 5-22 for more information.

   ```
   NIS (YP) compatibility (0=off, 1=on): [0]
   Domain name             : wiz.com.
   NIS+ group              : netadmin.wiz.com.
   NIS (YP) compatibility  : OFF
   Security level          : 2=DES
   Is this information correct? (type 'y' to accept, 'n' to change)
   ```

   Once the information is correct, continue with Creating a Root Master Server, on page 5-13. You can keep choosing **n** until the information is correct.

Note:

– If you make a mistake and want to start over, use the **nisserver –D** command. However, understand that this command removes *everything* from the /**var**/**nis** directory, including the directory itself. If you are running in **NIS**–compatibility mode, it also removes /**var**/**yp**/*ypdomain***.**

  – This script only sets machines up at security level 2. You cannot change the security level when using this script. After the script has completed, you can change the security level with the appropriate NIS+ command. See NIS+ Security Levels, on page 7-4 and the **rpc.nisd** documentation for more information on changing security levels.

Populating NIS+ Tables

Once the root master server has been set up, you should populate its standard NIS+ tables with name services information. This section shows you how to populate the root master server's tables with data from files or NIS maps using the **nispopulate** script with default settings. The script uses:

- The domain created in the previous example (wiz.com.)
- System information files or NIS maps as the source of name services
- The standard NIS+ tables: auto_master, auto_home, ethers, group, hosts, networks, passwd, protocols, services, rpc, netmasks, bootparams, netgroup, and aliases

Prerequisites

Before you can run the **nispopulate** script:

- View each local /**etc** file or NIS map that you will be loading data from. Make sure that there are no spurious or incorrect entries. Make sure that the right data is in the correct place and format. Remove any outdated, invalid, or corrupt entries. You should also remove any incomplete or partial entries. You can always add individual entries after setup is completed. That is easier than trying to load incomplete or damaged entries.
- The information in the files must be formatted appropriately for the table into which it will be loaded. NIS+ Tables and Information, on page 4-18 and Setting Up NIS+ Tables, on page 5-47, describe the format required for a text file to be transferred into its corresponding NIS+ table.
- Make sure that domain and host names are different. Domains and hosts cannot have the same name. For example, if you have a **sales** domain you cannot have a machine named sales. Similarly, if you have a machine named **home**, do not create a domain named home. This caution also applies to subdomains; for example, if you have a machine named **west** you do not want to create a sales.west.myco.com subdomain.
- Remove any dots in host names. Because NIS+ uses dots (periods) to delimit between machine names and domains and between parent and subdomains, you cannot have a machine name containing a dot. Before running the **nispopulate** script, you must eliminate any dots in your host names. You can convert host name dots to hyphens or underscores. For example, you cannot have a machine named **sales.alpha**. You can convert that name to **sales_alpha**.
- If you are setting up a network for the first time, you may not have much network information stored anywhere. In that case, you will first need to collect the information and then type it into the **input file**—which is essentially the same as an /**etc** file.
- For safety's sake, you should make copies of the /**etc** files and use the copies to populate the tables instead of the actual ones. (This example uses files in a directory called /**nis+files**, for instance.)

- Edit the following copied NIS files, **passwd**, **aliases,** and **hosts,** for security reasons. For example, you may want to remove the following lines from the copy of your local **passwd** file so they will not be distributed across the namespace:

```
root:!:0:0::/home/root:/bin/ksh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
guest:!:100:100::/home/guest:
nobody:!:4294967294:4294967294::/:
lpd:!:9:4294967294::/:
nuucp:*:6:5:uucp login user:/var/spool/uucppublic:/usr/sbin/uucp/uucico
ftp:*:200:1::/home/ftp:/usr/bin/ksh
anonymous:*:201:1::/home/ftp:/usr/bin/ksh
admin:!:202:1::/home/admin:/usr/bin/ksh
```

- The domain must have already been set up and its master server must be running.
- The domain’s server must have sufficient disk space to accommodate the new table information.
- You must be logged in as an NIS+ principal (a client with appropriate credentials) and have write permission to the NIS+ tables in the specified domain. In this example, you would have to be the user root on the machine **rootmaster**.

If populating from files, you need:

- The new NIS+ domain name
- The path of the appropriately edited text files whose data will be transferred
- Your root password

If populating from NIS maps, you need:

- The new NIS+ domain name
- The NIS domain name
- The NIS server’s name
- The IP address of the NIS server
- Your root password

Note: The NIS domain name is case–sensitive, while the NIS+ domain name is not.

Procedure

1. Perform either of the following alternatives to populate the root master server tables and then continue with the next step.

   The first alternative below shows you how to populate tables from files. The second shows you how to populate tables from NIS maps. Type these commands in a scrolling window; otherwise, the script's output may scroll off the screen.

   Note: The **nispopulate** script may fail if there is insufficient /**tmp** space on the system.

   – To populate from files, type the following command:

   ```
   rootmaster# nispopulate –F –p /nis+files –d wiz.com.
   NIS+ domain name        : wiz.com.
   Directory Path           : /nis+files
   Is this information correct? (type 'y' to accept, 'n' to change)
   ```

   The **–F** option indicates that the tables will take their data from files. The **–p** option specifies the directory search path for the source files. (In this case, the path is /**nis+files**.) The **–d** option specifies the NIS+ domain name. (In this case, the domain name is **wiz.com**.)

   The NIS+ principal user is root. You must perform this task as superuser in this instance because this is the first time that you are going to populate the root master server's tables. The **nispopulate** script adds credentials for all members of the NIS+ admin group.

   – Type the following command to populate the tables from NIS maps.

   ```
   rootmaster# nispopulate –Y –d wiz.com. –h corporatemachine \
     –a 130.48.58.111 –y corporate.wiz.com.
   NIS+ domain name              : wiz.com.
   NIS (YP) domain               : corporate.wiz.com
   NIS (YP) server hostname      : corporatemachine
   Is this information correct? (type 'y' to accept, 'n' to change)
   ```

   The **–Y** option indicates that the tables will take their data from NIS maps. The **–d** option specifies the NIS+ domain name. The **–h** option specifies the NIS server's machine name. (In this case, the NIS server's name is **corporatemachine**. You would have to insert the name of a real NIS server at your site to create the sample domain.) The **–a** option specifies the NIS server's IP address. (In this case, the address is 130.48.58.111. You would have to insert the IP address of a real NIS server at your site to create the sample domain.) The **–y** option specifies the NIS domain name. (In this case, the domain's name is **corporate.wiz.com**; you would have to insert the NIS domain name of the real NIS domain at your site to create the sample domain. Remember that NIS domain names are case sensitive.)

   The NIS+ principal user is root. You must perform this task as superuser in this instance because this is the first time that you are going to populate the root master server's tables. The **nispopulate** script also adds credentials for all members of the NIS+ admin group.

2. Type **y** (if the information returned on the screen is correct).

   Typing **n** causes the script to prompt you for the correct information. (See Changing Incorrect Information, on page 5-14 for what you need to do if the information is incorrect.)

3. If you populated the tables from files, you will see the following:

   ```
   Is this information correct? (type 'y' to accept, 'n' to change) y
   This script will populate the following NIS+ tables for domain
   wiz.com from the files in /nis+files:
   auto_master auto_home ethers group hosts networks passwd protocols
   services rpc netmasks bootparams netgroup aliases shadow
   **WARNING: Interrupting this script after choosing to continue
   may leave the tables only partially populated. This script does
   not do any automatic recovery or cleanup.
   ```

```
Do you want to continue? (type 'y' to continue, 'n' to exit this script)
```

4. If you populated the tables from maps, you will see the following:

```
Is this information correct? (type 'y' to accept, 'n' to change) y
This script will populate the following NIS+ tables for domain wiz.com.
from the NIS (YP) maps in domain corporate: auto_master auto_home ethers
group hosts networks passwd protocols services rpc netmasks bootparams
netgroup aliases
**WARNING: Interrupting this script after choosing to continue
may leave the tables only partially populated. This script does
not do any automatic recovery or cleanup.
Do you want to continue? (type 'y' to continue, 'n' to exit this script)
```

5. Type **y** to continue populating the tables.

   (Typing **n** safely stops the script.) If you interrupt the script after you have chosen **y**—while the script's running—the script stops running and may leave the tables only partially populated. The script does not do any automatic recovery or cleaning up. You can safely rerun the script, however, the tables will be overwritten with the latest information.

6. If you are populating tables from files, you will see messages like the following as the script uses hosts and passwd information to create the credentials for hosts and users:

```
Do you want to continue? (type 'y' to continue, 'n' to exit this script)y
populating auto_master table from file /nis+files/auto_master...
auto_master table done.
populating auto_home table from file /nis+files/auto_home...
auto_home table done.
....
....
Credentials have been added for the entries in the hosts and
passwd table(s). Each entry was given a default network password
(also known as a Secure-RPC password). This password is:
                   nisplus
Use this password when the nisclient script requests the network
password.
Done!
```

Note: Note and remember this Secure RPC password. Use the password when prompted for your network or Secure RPC password.

   The script continues until it has searched for all the files it expects and loads all the tables it can from the available files.

7. If you are populating tables from NIS maps, you will see messages like the following as the script uses **hosts** and **passwd** information to create the credentials for hosts and users:

```
Do you want to continue? (type 'y' to continue, 'n' to exit this script)y
populating auto_master table from corporate.wiz.com NIS(YP) domain...
auto_master table done.
populating auto_home table from file corporate.wiz.com NIS(YP) domain...
auto_home table done.
....
Credentials have been added for the entries in the hosts and
passwd table(s). Each entry was given a default network password
(also known as a Secure-RPC password). This password is:
                    nisplus
Use this password when the nisclient script requests the network
password.
Done!
```

Note: Note and remember this Secure RPC password. Use the password when prompted for your network or Secure RPC password.

   All the tables are now populated. You can ignore the **parse error** warnings shown above. The errors indicate that NIS+ found empty or unexpected values in a field of a particular NIS map. You may want to verify the data later after the script completes.

Note: The **nispopulate** command looks for certain tables or files and may not find all expected tables or files in your environment. It ends successfully if it populates at least one table or file.

8. (Optional) Add yourself and others to the root domain's admin group.

   For example, if your login ID is topadm and your co–worker's ID is secondadmin, you would enter:

   ```
   rootmaster# nisgrpadm –a admin.wiz.com. topadm.wiz.com.secondadm.wiz.com.
   Added ”topadm.wiz.com.” to group ”admin.wiz.com.”.
   Added ”secondadm.wiz.com.” to group ”admin.wiz.com.”.
   ```

   The **admin.wiz.com.** argument in the **nisgrpadm –a** command above is the group name which must come first. The remaining two arguments are the names of the administrators.

Note: This step is necessary only if you want to add additional users to the admin group now, which is a good time to add administrators to the root server. You can also add users to the admin group after you have set up NIS+.

You do not have to wait for the other administrators to change their default passwords to perform this step; however, they must already be listed in the passwd table before you can add them to the admin group. Members of the admin group will be unable to act as NIS+ principals until they add themselves to the domain. See Initializing an NIS+ User, on page 5-21 for more information on initializing users. The group cache also has to expire before the new members will become active.

9. Type the following command to checkpoint the domain.

   ```
   rootmaster# nisping –C wiz.com.
   Checkpointing replicas serving directory wiz.com.
   Master server is rootmaster.wiz.com.
           Last update occurred at date
   Master server is rootmaster.wiz.com.
   checkpoint scheduled on rootmaster.wiz.com.
   ```

   This step ensures that all the servers supporting the domain transfer the new information from their initialization**(.log**) files to the disk–based copies of the tables. Since you have just set up the root domain, this step affects only the root master server, as the root domain does not yet have replicas.

   Attention: If you do not have enough swap or disk space, the server will be unable to checkpoint properly, but it will not notify you. One way to make sure all goes well is to list the contents of a table with the **niscat** command. For example, to check the contents of the **rpc** table, type:

   ```
   rootmaster# niscat rpc.org_dir
   rpcbind rpcbind 100000
   rpcbind portmap 100000
   rpcbind sunrpc 100000
   ```

   If you do not have enough swap space, you'll see the following error message instead of the sort of output you see above.

   ```
   can't list table: Server busy, Try Again.
   ```

   Even though it may not *seem* to, this message indicates that you do not have enough swap space. Increase the swap space and checkpoint the domain again.

Setting Up Root Domain NIS+ Client Machines

Once the root master server's tables have been populated from files or NIS maps, you can initialize an NIS+ client machine. Since the root master server is an NIS+ client of its own domain, no further steps are required to initialize it. This section shows you how to initialize an NIS+ client by using the **nisclient** script with default settings. The NIS+ client machine is a different workstation than the NIS+ root server. The script will use:

- The domain used in previous examples, wiz.com.
- The Secure RPC password (also known as the network password) created by the **nispopulate** script in the previous example (**nisplus**, the default password)

Note: The **–i** option used in Initializing a New Client Machine, on page 5-20 does not set up an NIS+ client to resolve host names requiring DNS. You need to explicitly include DNS for clients in their name service switch files.

Prerequisites

Before you can use the **nisclient** script:

- The domain must have already been set up and its master server must be running.
- The master server of the domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must be logged in as superuser on the machine that is to become an NIS+ client. In this example, the new client machine is named `wizclient1`.

You need:

- The domain name
- The default Secure RPC password (**nisplus**)
- The root password of the workstation that will become the client
- The IP address of the NIS+ server (in the client's home domain)

Procedure — Initializing a New Client Machine

1. Type the following command to initialize the new client on the new client machine.

   The **–i** option initializes a client. The **–d** option specifies the new NIS+ domain name. (If the domain name is not specified, the default would be the current domain name.) The **–h** option specifies the NIS+ server's host name.

   ```
   wizclient1#nisclient –i –d wiz.com. –h rootmaster
   Initializing client wizclient1 for domain ”wiz.com.”.
   Once initialization is done, you will need to reboot your
   machine.
   Do you want to continue? (type ’y’ to continue, ’n’ to exit this script)
   ```

2. Type y.

   Typing **n** exits the script. The script only prompts you for the root server's IP address if there is no entry for it in the client's **/etc/hosts** file.

   ```
   Do you want to continue? (type ’y’ to continue, ’n’ to exit this script)y
   Type server rootmaster’s IP address:
   ```

3. Type the correct IP address, and press Return.

   This example uses the address 123.123.123.123.

   ```
   Type server rootmaster’s IP address: 123.123.123.123
   setting up the domain information...
   setting up the name service switch information...
   Client initialization completed!!
   Please reboot your machine for changes to take effect.
   ```

4. Reboot your new client machine.

   Your changes will not take effect until you reboot the machine.

   You can now have the users of this NIS+ client machine add themselves to the NIS+ domain.

Procedure — Creating Additional Client Machines

Repeat the preceding client–initiation procedure on as many machines as you like. To initiate clients for another domain, repeat the procedure but change the domain and master server names to the appropriate ones.

The sample NIS+ domain described in this section assumes that you will initialize four clients in the domain wiz.com. You are then going to configure two of the clients as non–root NIS+ servers and a third client as a root replica of the root master server of the wiz.com domain.

Note: You always have to make a system into a client of the parent domain before you can make the same system a server of any type.

Initializing NIS+ Client Users

Once a machine has become an NIS+ client, the users of that machine must add themselves to the NIS+ domain. Adding a user to the domain means changing the Secure RPC password to that user's login password. What actually happens is that the user's password and the Secure RPC password are bound together. This procedure uses the **nisclient** script.

Prerequisites

Before you can use the **nisclient** script to initialize a user:

- The domain must have already been set up and its master server must be running.
- The master server of the domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized a client machine in the domain.
- You must be logged in as a **user** on the client machine. In this example, the user is named user1.
- The user must also be an NIS+ user. See the **nisaddent** command description.

You need:

- A user's login name (**user1** in this example)
- The default Secure RPC password – (**nisplus** in this example)
- The login password of the user that will become the NIS+ client

Procedure

1. To become an NIS+ client, type the following command while logged in as the user.

   ```
   user1prompt% nisclient –u
   ```

2. At the prompt below, type the network password (also known as the Secure–RPC password) that you obtained either from your administrator or from running the nispopulate script.

   ```
   Please enter the Secure-RPC password for user1:
   ```

   Enter the Secure RPC password `nisplus`, in this case. The password does not echo on the screen.

3. Type the user's login password and press Return. The password does not echo on the screen.

```
Please enter the login password for user1:
Your network password has been changed to your login one.
Your network and login passwords are now the same.
```

This user is now an NIS+ client. All users must make themselves NIS+ clients.

Setting Up NIS+ Servers

Now that the client machines have been initialized, you can change any of them to NIS+ servers but not into root NIS+ servers. Root NIS+ servers are a special type of NIS+ server. See Setting Up NIS+ Root Servers, on page 5-12 for more information. You need NIS+ servers for three purposes:

- To be root replicas—to contain copies of the NIS+ tables that reside on the root master server
- To be master servers of subdomains of the root domain
- To be replicas of master servers of subdomains of the root domain

You can configure servers any of three different ways:

- Without NIS compatibility
- With NIS compatibility
- With NIS compatibility and DNS forwarding—you only need to set DNS forwarding if you are going to have DNS clients in your NIS+ namespace (see Using NIS–Compatibility Mode, on page 3-23 for more information).

Servers and their replicas should have the same NIS–compatibility settings. If they do not have the same settings, a client that needs NIS compatibility set to receive network information may not be able to receive it if either the server or replica it needs is unavailable.

Note: If the root master server is unavailable and the NIS+ domain is being served solely by a replica, you can obtain information from the NIS+ tables, but changes to the original tables can be made only when the master server is available.

This example shows the machine `wizclient1` being changed to a server. This procedure uses the NIS+ **mk_nisd** command instead of an NIS+ script.

Prerequisites

Before you can run **mk_nisd**:

- The domain must have already been set up and its master server must be running.
- The master server of the domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized the client machine in the domain.
- You must be logged in as root on the client machine. In this example, the client machine is named `wizclient1`.

You need the superuser password of the client that you will convert into a server.

Procedure — Configuring an NIS+ Server

Perform any of the following to alternative procedures to configure a client as a server. These procedures create a directory with the same name as the server and create the server's initialization files which are placed in **/var/nis**.

Note: All servers in the same domain must have the same NIS–compatibility setting. For example, if the master server is NIS compatible, then its replicas also should be NIS compatible.

To configure a server without NIS compatibility:

```
wizclient1# mk_nisd –B
```

To configure a server with NIS compatibility:

```
wizclient1# mk_nisd –B –y
```

To configure a server with DNS forwarding and NIS compatibility:

```
wizclient1# mk_nisd –B –y –b
```

You must have superuser authority to run **mk_nisd**. For more information, see the command description for **mk_nisd**.

Procedure — Creating Additional Servers

Repeat the preceding client–to–server conversion procedure on as many client machines as you like.

The sample NIS+ domain described in this section assumes that you will convert three clients to servers. You will then configure one of the servers as a root replica, another as a master of a new subdomain, and the third as a replica of the master of the new subdomain.

Designating Root Replicas

To have regularly available NIS+ service, you should always create root replicas. Having replicas may also speed network–request resolution because multiple servers are available to handle requests. The root replica server contains exact copies of the NIS+ tables on the root server.

Note: If the root master server is unavailable and the NIS+ domain is being served solely by a replica, you can obtain information from the NIS+ tables, but changes to the original tables can be made only when the master server is available.

Replication of the master's database starts a few minutes after you perform this procedure and can take anywhere from a few minutes to a couple of hours to complete, depending on the size of your tables.

This example shows the machine `wizclient1` being configured as a root replica. This procedure uses the NIS+ **nisserver** script.

Prerequisites

Before you can run **nisserver** to create a root replica:

- The domain must have already been set up and its master server must be running.
- The master server of the domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized the client machine in the domain.
- You must have started **rpc.nisd** on the client.
- You must be logged in as root on the root master server. In this example, the root master machine is named **rootmaster**.

You need:

- The domain name
- The client machine name; (`wizclient1`, in this example)
- The superuser password for the root master server

Procedure — Creating a Root Replica

1. To create a root replica, type the following command as superuser (root) on the NIS+ domain's root master server.

The **–R** option indicates that a replica should be set up. The **–d** option specifies the NIS+ domain name (wiz.com., in this example). The **–h** option specifies the client machine (`wizclient1`, in this example) that will become the root replica.

```
rootmaster# nisserver –R –d wiz.com. –h wizclient1
This script sets up an NIS+ replica server for domain wiz.com.
Domain name: :wiz.com.
NIS+ server: :wizclient1
Is this information correct? (type ’y’ to accept, ’n’ to change)
```

2. Type **y** to continue.

   Typing **n** causes the script to prompt you for the correct information. (See Changing Incorrect Information, on page 5-14 for what you need to do if you type **n**.)

```
Is this information correct? (type ’y’ to accept, ’n’ to change) y
This script will set up machine ”wizclient1” as an NIS+
replica server for domain wiz.com. without NIS compatibility.
The NIS+ server daemon, rpc.nisd, must be running on wizclient1
with the proper options to serve this domain. Do you want to continue?
(type ’y’ to continue, ’n’ to exit this script)
```

3. Type **y** to continue.

   Typing **n** safely stops the script. The script will exit on its own if **rpc.nisd** is **not** running on the client machine.

```
Is this information correct? (type y’ to continue, ’n’ to exit this script) y
The system wizclient1 is now configured as a replica server for domain wiz.com..
```

   The NIS+ server daemon, **rpc.nisd**, must be running on `wizclient1` with the proper options to serve this domain. If you want to run this replica in NIS–compatibility mode, use the **mk_nisd** command to uncomment the appropriate lines in the **/etc/rc.nfs** file and set the **–Y** option. For example, before the command, the **/etc/rc.nfs** file contains:

```
# if [-x/usr/sbin/rpc.nisd] then;
# startsrc –s rpc.nisd
#
```

   After running **mk_nisd**, the same lines appear as:

```
if [-x/usr/sbin/rpc.nisd] then;
startsrc –s rpc.nisd
```

   Then, restart **rpc.nisd**. These actions should be taken after this script completes.

Note: The **mk_nisd**, **mk_cachemgr**, **mk_nispasswdd**, **rm_nisd**, **rm_cachemgr**, and **rm_nispasswdd** commands alter the entries of daemon startup calls in **/etc/rc.nfs**, and they alter the default behavior of the daemon **src** entities. For example, if the **rpc.nisd** daemon is supposed to be started with the **–Y** option, it is not explicitly set in the **/etc/rc.nfs** entry for starting the **rpc.nisd** daemon. Instead, a **chssys** is executed to place the default options that are added (if any) to the daemons during startup. To verify that these options exist, use the **lssrc –S –s** *subsystem* command to show the default options.

The machine `wizclient1` is now an NIS+ root replica. The new root replica can handle requests from the clients of the root domain. Since there are now two servers available to the domain, information requests may be fulfilled faster.

Procedure — Creating Additional Replicas

Repeat the preceding server–to–replica conversion procedure for each additional server. For performance reasons, you should have no more than a few replicas per domain. Do create as many replicas, though, as is necessary to serve physically distant sites. For example, it may make sense from an organizational point of view to have two physically distant sites in the same NIS+ domain. If a root replica and the master of the domain are at the first site, there will be much network traffic between the first site and the second site of the domain. Creating an additional root replica at the second site should reduce network traffic.

The sample NIS+ domain described in this section includes only one root replica. One of the other clients of the wiz.com. domain will be converted to a replica of the subdomain created in the next section.

Creating a Subdomain

This section shows you how to create the master server of a new non–root domain. The new domain will be a subdomain of the wiz.com. domain. The hierarchical structure of NIS+ allows you to create a domain structure that parallels your organizational structure.

This example shows the machine **wizclient2** being converted to the master server of a new domain called subwiz.wiz.com. This procedure uses the NIS+ script **nisserver**.

Prerequisites

Before you can run **nisserver** to create a master server for a new nonroot domain:

- The parent domain must have already been set up and its master server must be running.
- The parent domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized the new client machine in the parent domain.
- You must have started **rpc.nisd** on the client.
- You must have adequate permissions to add the new domain. In this case, you must be logged in as root on the parent master server. In this example, the parent master machine is named **rootmaster**.

You need:

- A name for the new non–root domain—the name of the new domain includes the name of the parent domain with this syntax: *newdomain.rootdomain*.
- The client machine name (**wizclient2,** in this example)
- The superuser password for the parent master server

In the following example, the new nonroot domain is called subwiz.wiz.com.

Note: Any NIS+ client can be converted to an NIS+ master server as long as it is itself in a domain above the domain it will be serving. For example, an NIS+ client in domain subwiz.wiz.com can serve domains below it in the hierarchy, such as corp.subwiz.wiz.com or even east.corp.subwiz.wiz.com. This client cannot, however, serve the domain wiz.com, because wiz.com is above the domain subwiz.wiz.com in the hierarchy. Root replicas are the only exception to this rule. They are clients of the domain that they serve.

Procedure — Creating a New NonRoot Domain

1. Type the following command as superuser (root) on the NIS+ domain's root master server to create a new nonroot domain master server.

   The **–M** option indicates that a master server for a new nonroot domain should be created. The **–d** option specifies the **new** domain name, subwiz.wiz.com. in this instance. The **–h** option specifies the client machine, (**wizclient2,** in this example), that will become the master server of the new domain.

   ```
   rootmaster# nisserver –M –d subwiz.wiz.com. –h wizclient2
   This script sets up a non–root NIS+ master server for domain
   subwiz.wiz.com.
   Domain name             : subwiz.wiz.com.
   NIS+ server             : wizclient2
   NIS+ group              : admin.subwiz.wiz.com.
   NIS (YP) compatibility  : OFF
   Security level          : 2=DES
   Is this information correct? (type 'y' to accept, 'n' to change)
   ```

Master servers of new nonroot domains are created with the same set of default values as root servers. See Creating a Root Master Server, on page 5-13 for more information on NIS+ group, NIS compatibility, and security level.

2. Type **y** to continue.

   Typing **n** causes the script to prompt you for the correct information. (See Changing Incorrect Information, on page 5-14 for what you need to do if you type **n**.)

```
Is this information correct? (type 'y' to accept, 'n' to change) y
This script sets up machine "wizclient2" as an NIS+
non-root master server for domain subwiz.wiz.com.
Do you want to continue? (type 'y' to continue, 'n' to exit this script)
```

3. Type **y** to continue.

   Typing **n** safely exits the script. The script will exit on its own if **rpc.nisd** is **not** running on the client machine.

```
Do you want to continue?(type 'y' to continue, 'n' to exit this script)y
running nissetup ...
org_dir.subwiz.wiz.com. created
groups_dir.subwiz.wiz.com. created
...
...
setting NIS+ group admin.subwiz.wiz.com. ...
The system wizclient2 is now configured as a non-root server for domain
subwiz.wiz.com. You can now populate the standard NIS+ tables by using
the nispopulate or /usr/lib/nis/nisaddent commands.
```

The machine **wizclient2** is now the master server of the subwiz.wiz.com. domain. The subwiz.wiz.com. domain is a subdomain of the wiz.com. domain. The machine **wizclient2** is simultaneously still a client of the root domain wiz.com., and the master server of the subwiz.wiz.com. domain.

You can now populate the standard NIS+ tables on the new master server of the subwiz.wiz.com. domain.

Procedure — Creating Additional Domains

Repeat the preceding procedure for changing servers to master servers of new non–root domains on as many server machines as you like. Every new master server is a new domain. Plan your domain structure before you start creating an NIS+ namespace. See Configuration Worksheets, on page 5-5 for help with planning an NIS+ hierarchy.

Populating the New Domain's Tables

After you have created a new domain, you need to populate its master server's standard NIS+ tables. You use the same procedure to populate the new master server's tables as you used to populate the root master server's tables. The major difference is that the **nispopulate** script is run on the new master server instead of on the root master server. The domain names and file paths or NIS servers' names may change as well.

This example shows the tables of the new domain, subwiz.wiz.com., being populated.

Prerequisites

Before you can run the **nispopulate** script to populate the new master server's tables:

- The information in the files must be formatted appropriately for the table into which it will be loaded.
- Before proceeding, view each local /**etc** file or NIS map that you will be loading data from. Make sure that there are no spurious or incorrect entries. Make sure that the right data is in the correct place and format. Remove any outdated, invalid, or corrupt entries. You should also remove any incomplete or partial entries. You can always add individual entries after set up is completed. That is easier than trying to load incomplete or damaged entries.

- If you are setting up a network for the first time, you may not have much network information stored anywhere. In that case, you'll need to first get the information and then enter it manually into the **input file**—which is essentially the same as an /**etc** file.
- You should make copies of the /**etc** files and use the copies to populate the tables instead of the actual ones for safety reasons. (This example uses files in a directory called /**nis+files**, for instance.)
- Edit the following copied NIS files, **passwd**, **aliases,** and **hosts,** for security reasons. For example, you may want to remove the following lines from the copy of your local **passwd** file so they will not be distributed across the namespace:

```
root:!:0:0::/home/root:/bin/ksh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
guest:!:100:100::/home/guest:
nobody:!:4294967294:4294967294::/:
lpd:!:9:4294967294::/:
nuucp:*:6:5:uucp login user:/var/spool/uucppublic:/usr/sbin/uucp/uucico
ftp:*:200:1::/home/ftp:/usr/bin/ksh
anonymous:*:201:1::/home/ftp:/usr/bin/ksh
admin:!:202:1::/home/admin:/usr/bin/ksh
```

- The domain must have already been set up and its master server must be running.
- The domain's servers must have sufficient disk space to accommodate the new table information.
- You must be logged in as an NIS+ principal and have write permission to the NIS+ tables in the specified domain. In this example, you would have to be the user root on the machine **wizclient2**.

Note: The **nispopulate** script may fail if there is insufficient /**tmp** space on the system.

If populating from files, you need:

- The new NIS+ domain name (if not already set)
- The path of the appropriately edited text files whose data will be transferred
- The root password of the NIS+ master server

If populating from NIS maps, you need:

- The new NIS+ domain name
- The NIS domain name
- The NIS server's name
- The IP address of the NIS server
- The root password of the NIS+ master server

Note: The NIS domain name is case–sensitive, while the NIS+ domain name is not.

Procedure — Populating the Master Server Tables

This example shows you what to type to populate the tables of the new domain, `subwiz.wiz.com`.

Note: This script should be run on the new domain's master server, not the root master server.

There are two alternate methods of populating the master server tables on the new master server:

- You can populate master server tables from files.

- You can populate master server tables from NIS maps.

Whichever method you choose should be executed in a scrolling window as the script's output may otherwise scroll off the screen.

Populating the Tables From Files

To populate master server tables from files, type the following commands.

```
wizclient2# nispopulate –F –p /nis+files –d subwiz.wiz.com.
NIS+ domain name        : subwiz.wiz.com.
Directory Path          : /nis+files
Is this information correct? (type 'y' to accept, 'n' to change)
```

Populating the Tables From NIS Maps

To populate master server tables from NIS maps, type the following commands.

```
wizclient2# nispopulate –Y –d subwiz.wiz.com. –h businessmachine \
  –a IP_addr_of_NIS_server –y business.wiz.com
NIS+ Domain name              : subwiz.wiz.com.
NIS (YP) domain               : business.wiz.com
NIS (YP) server hostname      : businessmachine
Is this information correct? (type 'y' to accept, 'n' to change)
```

Designating Replicas

Just as you did in the wiz.com. domain, to have regularly available NIS+ service, you should always create replicas. Having replicas may also speed network–request resolution since multiple servers are available to handle requests. The replica server contains exact copies of the NIS+ tables on the master server of your new domain. Replication of the master's database starts a few minutes after you perform this procedure and can take anywhere from a few minutes to a couple of hours to complete, depending on the size of your tables.

You use the same procedure to create a replica as you do to create a root replica. The major difference between creating the root replica and this replica is that the machine you are going to convert to a replica will remain a client of the domain above the one it will be serving as a replica. This example shows you only what you would type to create a replica for the new domain. For the rest of the script's output, see Creating a Root Replica, on page 5-23.

Prerequisites

Before you can run **nisserver** to create a replica:

- The domain must have already been set up and its master server must be running.
- The domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized the client machine in the parent domain.
- You must have started **rpc.nisd** on the client.
- You must be logged in as root on the master server. In this example, the master machine is named **wizclient2**.
- The domain name
- The client machine name (**wizclient3,** in this example)
- The superuser password for the root master server

Procedure

Run the **nisserver –R** command as superuser (root) on the NIS+ domain's master server.

```
wizclient2# nisserver –R –d subwiz.wiz.com. –h wizclient3
This script sets up an NIS+ replica server for domain
subwiz.wiz.com.
Domain name:  subwiz.wiz.com.
NIS+ server:  wizclient3
```

```
Is this information correct? (type 'y' to accept, 'n' to change)
```

In this example, **wizclient2** is the master server. The **–R** option indicates that a replica should be set up. The **–d** option specifies the NIS+ domain name (subwiz.wiz.com. in this example). The **–h** option specifies the client machine (**wizclient3,** in this example) that will become the replica. Notice that this machine is still a client of the wiz.com. domain and not a client of the subwiz.wiz.com. domain.

See Creating a Root Replica, on page 5-23 for the rest of this script's output.

Initializing Subdomain NIS+ Client Machines

Once the master server's tables have been populated from files or NIS maps, you can initialize an NIS+ client machine. This section shows you how to initialize an NIS+ client in the new domain using the **nisclient** script with default settings. The NIS+ client machine is a different workstation than the NIS+ master server.

You use the same procedure to initialize a client in the new domain as you do to initialize a client in the root domain. This example shows you only what you would type to initialize a client for the new domain. For the rest of the script's output, see Initializing a New Client Machine, on page 5-20.

Prerequisites

Before you can use the **nisclient** script to initialize a user:

- The domain must have already been set up and its master server must be running.
- The master server of the domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized a client machine in the domain.
- You must be logged in as a **user** on the client machine. In this example, the user is named user1.

You need:

- The domain name (subwiz.wiz.com., in this example)
- The default Secure RPC password (**nisplus**)
- The root password of the workstation that will become the client
- The IP address of the NIS+ server (in the client's home domain) (in this example, the address of the master server wizclient2)

Procedure

Type the following command as superuser to initialize the new client on the new client machine.

```
subclient1# nisclient –i –d subwiz.wiz.com. –h wizclient2 –a
wizclient2ipaddress
initializing client subclient1 for domain "subwiz.wiz.com.".
Once initialization is done, you will need to reboot your machine.
Do you want to continue? (type 'Y' to continue, 'N' to exit this script)
```

The **–i** option initializes a client. The **–d** option specifies the new NIS+ domain name. (If the domain name is not specified, the default would be the current domain name.) The **–h** option specifies the NIS+ server's host name. The **–a** option specifies the IP address of `wizclient2`.

See Initializing a New Client Machine, on page 5-20 for the rest of this script's output.

Initializing Subdomain NIS+ Client Users

You use the same procedure (**nisclient**) to initialize a user in the new domain as you do to initialize a user in the root domain. All users must make themselves NIS+ clients. This example shows you only what you would type to initialize a user for the new domain. For the rest of the script's output, see Initializing an NIS+ User, on page 5-21.

Prerequisites

Before you can use the **nisclient** script to initialize a user:

- The domain must have already been set up and its master server must be running.
- The master server of the domain's tables must be populated. (At a minimum, the hosts table must have an entry for the new client machine.)
- You must have initialized a client machine in the domain.
- You must be logged in as a **user** on the client machine. In this example, the user is named user2.

You need:

- The user's login name (**user2,** in this example)
- The default Secure RPC password (**nisplus**)
- The login password of the user that will become the NIS+ client

Procedure

To become an NIS+ user, type the following command while logged in as the user.

```
user2prompt% nisclient –u
```

At the prompt below, type the network password (also known as the Secure–RPC password):

```
Please enter the Secure–RPC password for user2:
```

See Initializing an NIS+ User, on page 5-21 for the rest of this script's output.

Summary of Commands for the Sample NIS+ Namespace

The following table summarizes the actual commands that you typed to create the sample namespace. The prompt preceding each command indicates on which machine the command should be typed.

Table 19. Creating the Sample Namespace: Command Summary

Tasks	Commands
Set environment path to include **/usr/lib/nis**.	`# PATH=$PATH:/usr/lib/nis; export PATH`
Create root master server for wiz.com. domain.	`rootmaster# nisserver –r –d wiz.com.`

Populate the root master server's NIS+ tables—from files or from NIS maps.	`rootmaster# nispopulate –F \ –p /nis+files –d wiz.com.`
	or
	`rootmaster# nispopulate –Y –d wiz.com. \ –h corporatemachine \ –a 130.48.58.111 \ –y corporate.wiz.com`

Add additional members to the admin group.	`rootmaster# nisgrpadm –a admin.wiz.com. topadmin.wiz.com. \ secondadmin.wiz.com.`

Tasks	Commands
Make a checkpoint of the NIS+ database.	`rootmaster# nisping -C org_dir.wiz.com.`

Initialize an NIS+ client machine in the `wiz.com.` domain.	`wizclient1# nisclient -i -d wiz.com. \`
	`  -h rootmaster -a ipaddress`

Initialize user as an NIS+ client.	`wizclient1user1prompt% nisclient -u`

Convert NIS+ client to NIS+ server, without or with NIS compatibility or with NIS and DNS.	`wizclient1# mk_nisd [-I|-B|-N]`
	or
	`wizclient1# mk_nisd -y [-I|-B|-N]`
	or
	`wizclient1# mk_nisd -y -b [-I|-B|-N]`

Create a root replica.	`rootmaster# nisserver -R \`
	`  -d wiz.com. \`
	`  -h wizclient1`

Convert a server to a nonroot master server of the subwiz.wiz.com. domain.	`rootmaster# nisserver -M \`
	`  -d subwiz.wiz.com. \`
	`  -h wizclient2`

Populate the new master server's NIS+ tables—from files or from NIS maps.	`wizclient2# nispopulate -F \`
	`  -p /nis+files \`
	`  -d subwiz.wiz.com.`
	or
	`wizclient2# nispopulate -Y \`
	`  -d subwiz.wiz.com. \`
	`  -h businessmachine \`
	`  -a 130.48.58.242 \`
	`  -y business.wiz.com`

Create a master server replica.	`wizclient2# nisserver -R \`
	`  -d subwiz.wiz.com. \`
	`  -h wizclient3`

Initialize an NIS+ client in the `subwiz.wiz.com.` domain.	`subclient1# nisclient -i \`
	`  -d subwiz.wiz.com. \`
	`  -h wizclient2 \`
	`  -a ipaddress`

Initialize user as an NIS+ user.	`subclient1user2prompt% nisclient -u`

Setting Up the Root Domain

This section provides step–by–step instructions for one task: setting up the root domain with DES authentication using the NIS+ command set.

Note: It is much easier to perform this task with the NIS+ installation scripts as described in NIS+ Setup Scripts, on page 5-7 than with the NIS+ command set as described here. The methods described in this section should be used only by those administrators who are very familiar with NIS+ and who require some nonstandard features or configurations not provided by the installation scripts.

See Configuration Worksheets, on page 5-5, for worksheets that you can use to plan your NIS+ namespace.

This task describes how to set up the root domain with the root master server running at security level 2 (the normal level).

Setting up the root domain involves three major tasks:

- Preparing the root master server
- Creating the root domain
- Creating credentials for the root domain

However, setting up the root domain is not as simple as performing these three tasks in order; they are intertwined with each other. For instance, you must specify some security parameters before you create the root directory; the rest, after. To make the root domain easier to set up, this section separates these tasks into individual steps and arranges them into their most efficient order.

Standard versus NIS–Compatible Setup Procedures

The steps in this section apply to both a standard NIS+ root domain and an NIS–compatible root domain. There are, however, some important differences. The NIS+ daemon for an NIS–compatible domain must be started with the **–Y** option, which allows the root master server to answer requests from NIS clients.

An NIS–compatible domain also requires read rights to the passwd table for the nobody class, which allows NIS clients to access the information stored in the table's passwd column. This is accomplished with the **–Y** option to the **nissetup** command. The standard NIS+ domain version uses the same command but without the **–Y** option.

Establishing the Root Domain

The procedure describes each step in detail and provides related information. For those who do not need detailed instructions, a summary listing of the necessary commands is provided on Root Domain Setup Summary, on page 5-41.

Summary of Steps

Here is a summary of the entire setup process:

1. Log in as superuser to the root master server.
2. Check the root master server's domain name.
3. Check the root master server's /**etc**/**irs.conf** file.
4. Kill and restart **keyserv**.
5. Clean out leftover NIS+ material and processes.
6. Kill server daemons.
7. Name the root domain's admin group.
8. Create the root directory and initialize the root master server.

9. [NIS–compatibility Only] Start the NIS+ daemon with **–Y**. (See the **mk_nisd** command description.)

   or

   [Standard NIS+ Only] Start the NIS+ daemon. (See the **mk_nisd** command description.)
10. Verify that the root objects have been properly created.
11. Create the root domain's subdirectories and tables.
12. Create DES credentials for the root master server.
13. Create the root domain's admin group.
14. Add the root master to the root domain's admin group.
15. Update the root domain's public keys.
16. Start the NIS+ cache manager. (See the **mk_niscachemgr** command description.)
17. Stop the NIS+ daemon. (See the **rm_nisd** command description.)
18. Restart the NIS+ daemon with security level 2. (See the **mk_nisd** command description.)
19. Add your LOCAL credentials to the root domain.
20. Add your DES credentials to the root domain.
21. Add credentials for other administrators.
22. Add yourself and other administrators to the root domain's admin group.

Security Considerations

NIS+ provides preset security defaults for the root domain.

The default security level is level 2.

Attention:

- Operational networks with actual users should always be run at security level 2. Security levels 0 and 1 are for setup and testing purposes only. Do not run an operational network at level 0 or 1.
- The NIS+ security system is complex. If you are not familiar with NIS+ security, review the security–related chapters of this book before starting to set up your NIS+ environment.

Prerequisites

Before proceeding, make sure that

- The /**etc**/**passwd** file on the root master server must contain an entry for you and every other administrator whose credentials will be added to the root domain in this setup process.
- If the server will operate in NIS–compatibility mode and support DNS forwarding for clients, it must have a properly configured /**etc**/**resolv.conf** file.
- The server must have a unique machine name that does not duplicate any user ID.
- The server must have a machine name that does not contain any dots. For example, a machine named **sales.alpha** is not allowed. A machine named **sales–alpha** is allowed.

In order to complete this task you need to know

- The superuser password of the workstation that will become the root master server
- The name of the root domain
- The name of the root domain's admin group
- Your user ID and password

- The user ID of any administrator whose credentials you are adding to the root domain

Procedure

You can set up a root domain using Web-based System Manager, the System Management Interface Tool (SMIT), or the following procedure:

1. Log in as superuser on the machine to be the root master server.

   The examples in these steps use **rootmaster** as the root master server and wiz.com. as the root domain.

2. Check the root master server's domain name.

   Use the **domainname** command to make sure the root master server is using the correct domain name. The **domainname** command returns a workstation's current domain name.

   Attention: Domains and hosts should not have the same name. For example, if you have a sales domain you should not have a machine named `sales`. Similarly, if you have a machine named `home`, you do not want to create a domain named home. This caution applies to subdomains; for example, if you have a machine named `west,` you do not want to create a `sales.west.myco.com` subdirectory.

   If the name is not correct, change it.

   ```
   rootmaster# domainname
      strange.domain
   rootmaster# domainname wiz.com
   rootmaster# chypdom -I wiz.com.
   ```

   The above example changes the domain name of the root master server from `strange.domain` to `wiz.com`. When changing or establishing a domain name, make sure that the it has at least two labels; for example, `wiz.com` instead of `wiz`.

   Note: Do not include a trailing dot for the domain name in this instance. The **domainname** command is not an NIS+ command and does not follow the NIS+ convention of appending a dot to domain names.

   (More complete instructions are provided in Specifying a Domain Name After Installation, on page 5-62.)

3. Check the root master server's **/etc/irs.conf** file.

4. Now kill and restart **keyserv** as shown below.

   ```
   rootmaster# stopsrc -s keyserv
   rootmaster# startsrc -s keyserv
   ```

5. Clean out leftover NIS+ material and processes.

   If the workstation you are working on was previously used as an NIS+ server or client, remove any files that might exist in /**var**/**nis** and kill the cache manager, if it is still running. In this example, a cold–start file and a directory cache file still exist in /**var**/**nis**:

   ```
   rootmaster# stopsrc -g nisplus
   rootmaster# rm -rf /var/nis/*
   ```

   If running in NIS–compatibility mode, also enter the following command:

   ```
   rootmaster# rm -rf /var/yp/ypdomain
   ```

   This step makes sure files left in /**var**/**nis** or directory objects stored by the cache manager are completely erased so they do not conflict with the new information generated during this setup process. If you have stored any admin scripts in /**var**/**nis**, you may want to consider temporarily storing them elsewhere, until you finish setting up the root domain.

6. Kill server daemons.

   If the workstation you are working on was previously used as an NIS+ server, check to see if **rpc.nisd** or **rpc.nispasswdd** is running. If either of these daemons is running, kill them.

7. Name the root domain's admin group.

   Although you wo not actually create the admin group until later, you must identify it now. Identifying it now ensures that the root domain's **org_dir** directory object, **groups_dir** directory object, and all its table objects are assigned the proper default group when they are created.

   To name the admin group, set the value of the environment variable **NIS_GROUP** to the name of the root domain's admin group.

```
rootmaster# NIS_GROUP=admin.wiz.com.
rootmaster# export NIS_GROUP
```

8. Create the root directory and initialize the root master server.

   This step creates the first object in the namespace—the root directory—and converts the workstation into the root master server. Use the **nisinit –r** command, as shown below. (This is the only instance in which you will create a domain's directory object and initialize its master server in one step. In fact, **nisinit –r** performs an automatic **nismkdir** for the root directory. In any case except the root master, these two processes are performed as separate tasks.)

```
rootmaster# nisinit -r
This machine is in the wiz.com. NIS+ domain
Setting up root server ...
All done.
```

   A UNIX directory with the name **/var/nis/data** is created.

   Within the **/var/nis** directory is a file named **root.object.**

```
rootmaster# ls -l /var/nis/data
-rw-rw-rw- 1 root other 384 date  root.object
```

   This is not the root directory object; it is a file that NIS+ uses to describe the root of the namespace for internal purposes. The NIS+ root directory object will be created later in this procedure, then other files will be added beneath the directory. Although you can verify the existence of these files by looking directly into the directory, NIS+ provides more appropriate commands. They are called out where applicable in the following steps.

   Attention: Do not rename the **/var/nis** or **/var/nis/data** directories or any of the files in these directories that were created by **nisinit** or any of the other NIS+ setup procedures.

9. [NIS–Compatibility only] Start the NIS+ daemon with **–Y**.

   Perform this step only if you are setting up the root domain in NIS–compatibility mode. This step includes instructions for supporting the DNS forwarding capabilities of NIS clients.

10. Use **mk_nisd** with the **–I**, **–B**, or **–N** option.

    Note: Read the command description for **mk_nisd** and decide which option you should use before continuing with this procedure.

    For NIS compatibility with DNS forwarding, use:

```
rootmaster# mk_nisd -y -b [-I|-B|-N]
```

    For NIS compatibility without DNS forwarding, use:

```
rootmaster# mk_nisd -y [-I|-B|-N]
```

To start the NIS+ daemon without NIS compatibility or DNS forwarding, use:

```
rootmaster# mk_nisd [-I|-B|-N]
```

11. Verify that the root objects have been properly created.

    At this point in the procedure, your namespace should have:

    – A root directory object (**root.dir**)

    – A root master server (**rootmaster**) running the NIS+ daemon (**rpc.nisd**)

    – A cold start file for the master server (**NIS_COLD_START**)

    – A transaction log file (**trans.log**)

    – A table dictionary file (**data.dict**).

    The root directory object is stored in the /**var**/**nis**/**data** directory. Use the **ls** command to verify that it is there.

```
rootmaster# ls -l /var/nis/data
-rw-rw-rw-  1 root  other  384  date  root.object
-rw-rw-rw-  1 root  other  124  date  root.dir
```

    At this point, the root directory is empty; in other words, it has no subdirectories. You can verify this by using the **nisls** command.

```
rootmaster# nisls -l wiz.com.
wiz.com.:
```

    However, it has several **object** properties, which you can examine using **niscat –o**:

```
rootmaster# niscat -o wiz.com.
Object Name  : wiz
Owner      : rootmaster.wiz.com.
Group      : admin.wiz.com.
Domain     : com.
Access Rights : r---rmcdrmcdr---
.
.
.
```

    Note that the root directory object provides full (read, modify, create, and destroy) rights to both the owner and the group, while providing only read access to the world and nobody classes. (If your directory object does not provide these rights, you can change them using the **nischmod** command.)

    To verify that the NIS+ daemon is running, use the **ps** command.

```
rootmaster# ps -ef | grep rpc.nisd
root 1081    1 61 16:43:33 ?    0:01 rpc.nisd -S 0
root 1087 1004 11 16:44:09 pts/1 0:00 grep rpc.nisd
```

    The root domain's **NIS_COLD_START** file, which contains the IP address (and, eventually, public keys) of the root master server, is placed in /**var**/**nis**. Although there is no NIS+ command that you can use to examine its contents, its contents are loaded into the server's directory cache (**NIS_SHARED_DIRCACHE**). You can examine those contents with the /**usr**/**lib**/**nis**/**nisshowcache** command.

    Also created are a transaction log file (**trans.log**) and a dictionary file (**data.dict**). The transaction log of a master server stores all the transactions performed by the master server and all its replicas since the last update. You can examine its contents by using the **nislog** command. The dictionary file is used by NIS+ for internal purposes; it is of no interest to an administrator.

12. Create the root domain's subdirectories and tables.

    This step adds the **org_dir** and **groups_dir** directories, and the NIS+ tables, beneath the root directory object. Use the **nissetup** utility. For an NIS–compatible domain, be sure to include the **–Y** flag.

NIS–compatible only:

```
rootmaster# /usr/lib/nis/nissetup –Y
```

Standard NIS+ only:

```
rootmaster# /usr/lib/nis/nissetup
```

Each object added by the utility is listed in the output:

```
rootmaster# /usr/lib/nis/nissetup
org_dir.wiz.com. created
groups_dir.wiz.com. created
auto_master.org_dir.wiz.com. created
auto_home.org_dir.wiz.com. created
bootparams.org_dir.wiz.com. created
cred.org_dir.wiz.com. created
ethers.org_dir.wiz.com. created
group.org_dir.wiz.com. created
hosts.org_dir.wiz.com. created
mail_aliases.org_dir.wiz.com. created
sendmailvars.org_dir.wiz.com. created
client_info.org_dir.wiz.com. created
netmasks.org_dir.wiz.com. created
netgroup.org_dir.wiz.com. created
networks.org_dir.wiz.com. created
passwd.org_dir.wiz.com. created
protocols.org_dir.wiz.com. created
rpc.org_dir.wiz.com. created
services.org_dir.wiz.com. created
timezone.org_dir.wiz.com. created
```

The **–Y** option creates the same tables and subdirectories as for a standard NIS+ domain, but assigns read rights to the passwd table to the nobody class so that requests from NIS clients, which are unauthenticated, can access the encrypted password in that column.

Recall that when you examined the contents of the root directory with **nisls**, it was empty. Now, however, it has two subdirectories.

```
rootmaster# nisls wiz.com.
wiz.com.:
org_dir
groups_dir
```

You can examine the object properties of the subdirectories and tables by using the **niscat –o** command. You can also use the **niscat** option without a flag to examine the information in the tables, although at this point they are empty.

13. Create DES credentials for the root master server.

    The root master server requires DES credentials so that its own requests can be authenticated. To create those credentials, use the **nisaddcred** command as shown below. When prompted, enter the server's root password.

```
rootmaster# nisaddcred des
DES principal name: unix.rootmaster@wiz.com
Adding key pair for unix.rootmaster@wiz.com
           (rootmaster.wiz.com.).
Enter login password:
Wrote secret key into /etc/.rootkey
```

    If you enter a password that is different from the server's root password, you will get a warning message and a prompt to repeat the password:

```
Enter login password:
nisaddcred: WARNING: password differs from login password.
Retype password:
```

You can persist and retype the same password, and NIS+ will still create the credential. The new password will be stored in **/etc/.rootkey** and used by the keyserver when it starts up. To give the keyserver the new password right away, run **keylogin –r**, as described in Administering NIS+ Credentials, on page 6-2.

If you decide to use your login password after all, press Control–c and start the sequence over. If you were to simply retype your login password as encouraged by the server, you would get an error message designed for another purpose, but which in this instance could be confusing.

```
nisaddcred: WARNING: password differs from login password.
Retype password:
nisaddcred: password incorrect.
nisaddcred: unable to create credential.
```

As a result of this step, the root server's private and public keys are stored in the root domain's cred table (**cred.org_dir.wiz.com.**) and its secret key is stored in **/etc/.rootkey**. You can verify the existence of its credentials in the cred table by using the **niscat** command. Since the default domain name is wiz.com., you do not have to enter the cred table's fully qualified name; the **org_dir** suffix is enough. You can locate the root master's credential by looking for its secure RPC netname.

14. Create the root domain's admin group.

This step creates the admin group that was named earlier in this procedure. Use the **nisgrpadm** command with the **–c** option. The example below creates the admin.wiz.com. group.

```
rootmaster# nisgrpadm –c admin.wiz.com.
Group admin.wiz.com. created.
```

This step only creates the group—it does not identify its members. To observe the object properties of the group, use **niscat –o**, but be sure to append **groups_dir** in the group's name.

```
rootmaster# niscat –o admin.groups_dir.wiz.com.
Object Name  : admin
Owner      : rootmaster.wiz.com.
Group      : admin.wiz.com.
Domain     : groups_dir.wiz.com.
Access Rights : ----rmcdr---r---
Time to Live : 1:0:0
Object Type  : GROUP
Group Flags  :
Group Members :
```

15. Add the root master to the root domain's admin group.

Since at this point the root master server is the only NIS+ principal that has DES credentials, it is the only member you should add to the admin group. Use the **nisgrpadm** command again, but with the **–a** option. The first argument is the group name, the second is the name of the root master server. This example adds **rootmaster.wiz.com.** to the admin.wiz.com. group.

```
rootmaster# nisgrpadm –a admin.wiz.com. rootmaster.wiz.com.
Added rootmaster.wiz.com. to group admin.wiz.com.
```

To verify that the root master is indeed a member of the group, use the **nisgrpadm** command with the **–l** option (see Administering NIS+ Groups, on page 6-45).

Note: With group–related commands such as **nisgrpadm**, you do not have to include the **groups_dir** subdirectory in the name. You need to include that directory with commands like **niscat** because they are designed to work on NIS+ objects in general. The group–related commands are "targeted" at the **groups_dir** subdirectory.

```
rootmaster# nisgrpadm –l admin.wiz.com.
Group entry for admin.wiz.com. group:
  Explicit members:
    rootmaster.wiz.com.
  No implicit members
  No recursive members
  No explicit nonmembers
  No implicit nonmembers
  No recursive nonmembers
```

16. Update the root domain's public keys.

    Normally, directory objects are created by an NIS+ principal that already has DES credentials. In this case, however, the root master server could not acquire DES credentials until **after** it created the cred table (since there was no parent domain in which to store its credentials). As a result, three directory objects—**root, org_dir,** and **groups_dir**—do not have a copy of the root master server's public key. (You can verify this by using the **niscat –o** command with any of the directory objects. Look for the public key field. Instructions are provided in Administering NIS+ Directories, on page 6-50.

    To propagate the root master server's public key from the root domain's cred table to those three directory objects, use the /**usr**/**lib**/**nis**/**nisupdkeys** utility for each directory object.

    ```
    rootmaster# /usr/lib/nis/nisupdkeys wiz.com.
    rootmaster# /usr/lib/nis/nisupdkeys org_dir.wiz.com.
    rootmaster# /usr/lib/nis/nisupdkeys groups_dir.wiz.com.
    ```

    After each instance, you will see a confirmation message such as this one:

    ```
    Fetch Public key for server rootmaster.wiz.com.
     netname = 'unix.rootmaster@wiz.com.'
    Updating rootmaster.wiz.com.'s public key.
      Public key:
    ```

    Now, if you look in any of those directories (use **niscat** –o), you will see this entry in the public key field:

    ```
    Public key: Diffie–Hellman (192 bits)
    ```

17. Start the NIS+ cache manager.

    The cache manager maintains a local cache of location information for an NIS+ client (in this case, the root master server). It obtains its initial set of information from the client's cold–start file and downloads it into a file named **NIS_SHARED_DIRCACHE** in /**var**/**nis**.

    To start the cache manager, simply enter the **nis_cachemgr** command as shown below.

    ```
    rootmaster# startsrc –s nis_cachemgr
    ```

    Once the cache manager has been started, you have to restart it only if you have explicitly killed it. You do not have to restart it if you reboot, since the **NIS_COLD_START** file in /**var**/**nis** starts it automatically when the client is rebooted. For more information about the NIS+ cache manager, see Administering NIS+ Directories, on page 6-50.

18. Stop the NIS+ daemon. (See the **rm_nisd** command description for more information.

    ```
    rootmaster# stopsrc –s rpc.nisd
    ```

19. Restart the NIS+ daemon with security level 2.

    Now that the root master server has DES credentials and the root directory object has a copy of the root master's public key, you can restart the root master with security level 2 (the default).

    Standard NIS+ domain only

    ```
    rootmaster# startsrc –s rpc.nisd
    ```

For an NIS–compatible root domain, be sure to use the **–Y** flag:

```
rootmaster# startsrc –s rpc.nisd –a ″–Y″
```

For NIS–compatible NIS+ domain and DNS forwarding, use the **–Y** and **–B** flags:

```
rootmaster# startsrc –s rpc.nisd –a ″–Y –B″
```

Attention: Operational networks with actual users should always be run at security level 2. Security levels 0 and 1 are for setup and testing purposes only. Do not run an operational network at level 0 or 1 or you will be running in an unsecured NIS+ environment.

20. Add your LOCAL credentials to the root domain.

Since you do not have access rights to the root domain's cred table, you must perform this operation as superuser. In addition, the root master's /**etc**/**passwd** file must contain an entry for you. Use the **nisaddcred** command with the **–p** and **–P** flags as shown below.

```
nisaddcred –p uid –P principal-name local
```

The **principal–name** consists of the administrator's login name and domain name. This example adds a LOCAL credential for an administrator with a UID of 11177 and an NIS+ principal name of **topadmin.wiz.com.**

```
rootmaster# nisaddcred –p 11177 –P topadmin.wiz.com. local
```

For more information about the **nisaddcred** command, see Administering NIS+ Credentials, on page 6-2.

21. Add your DES credentials to the root domain.

Use the **nisaddcred** command again, but with the following syntax:

```
nisaddcred –p SecureRPC–netname –P principal–name des
```

The *SecureRPC–netname* consists of the prefix **UNIX** followed by your UID, the symbol @, and your domain name, but **without** a trailing dot. The *principal–name* is the same as for LOCAL credentials: your login name followed by your domain name, **with** a trailing dot.

```
rootmaster# nisaddcred –p unix.11177@wiz.com –P topadmin.wiz.com. des
Adding key pair for unix.11177@wiz.com (topadmin.wiz.com.).
Enter login password:
```

If after entering your login password you get a **password differs from login password** warning and yet the password you entered is your correct login password, ignore the error message. The message appears because NIS+ cannot read the protected /**etc**/**passwd** file that stores the password, as expected. The message would not have appeared if you had no user password information stored in the /**etc**/**passwd** file.

22. Add credentials for other administrators.

Add the credentials, both LOCAL and DES, of the other administrators who will work in the root domain.

– One way is to ask the other administrators to add their own credentials. However, they will have to do this as superuser. Here is an example that adds credentials for an administrator with a user ID of `33355` and a principal name of `miyoko.wiz.com.`

```
rootmaster# nisaddcred –p 33355 –P miyoko.wiz.com. local
rootmaster# nisaddcred –p unix.33355@wiz.com –P miyoko.wiz.com. des
Adding key pair for unix.33355@wiz.com (miyoko.wiz.com.).
Enter login password:
```

– Another way is for you to create temporary credentials for the other administrators, using dummy passwords. (Note that the other administrator, in this example `miyoko`, must have an entry in the NIS+ passwd table. If no such entry exists you must first create one with **nistbladm**. The example below includes that step.)

```
rootmaster# nistbladm -D owner=miyoko.wiz.com. name=miyoko uid=33355 \
  gcos=miyoko home=/home/miyoko shell=/bin/tcsh passwd.org_dir
rootmaster# nisaddent -a -f /etc/passwd.xfr passwd
rootmaster# nisaddent -a -f /etc/shadow.xfr shadow
rootmaster# nisaddcred -p 33355 -P miyoko.wiz.com. local
rootmaster# nisaddcred -p unix.33355@wiz.com -P miyoko.wiz.com. des
Adding key pair for unix.33355@wiz.com (miyoko.wiz.com.).
Enter miyoko's login password:
nisaddcred: WARNING: password differs from login passwd.
Retype password:
rootmaster# nischown miyoko.wiz.com. '[name=miyoko],passwd.org_dir'
```

In this case, the first instance of **nisaddent** populates the passwd table—except for the password column. The second instance populates the shadow column. Each administrator can later change his or her network password using the **chkey** command. Administering NIS+ Credentials, on page 6-2 describes how to do this.

23. Add yourself and other administrators to the root domain's admin group.

You do not have to wait for the other administrators to change their dummy passwords to perform this step. Use the **nisgrpadm** command with the **–a** option. The first argument is the group name, the remaining arguments are the names of the administrators. This example adds two administrators, `topadmin` and `miyoko`, to the `admin.wiz.com.` group:

```
rootmaster# nisgrpadm -a admin.wiz.com. topadmin.wiz.com. miyoko.wiz.com.
Added topadmin.wiz.com. to group admin.wiz.com.
Added miyoko.wiz.com. to group admin.wiz.com.
```

Root Domain Setup Summary

The following table summarizes the steps requires to set up a root domain. The summary assumes a simple case. Be sure you are familiar with the complete task descriptions before you use this summary as a reference. This summary does not show the server's responses to each command.

Table 20. Setting Up a Root Domain: Command Summary

Tasks	Commands

Log in as superuser to **rootmaster**.	`rootmaster% su`
	`Password:`

Check domain name	`# domainname`
Remove leftover NIS+ material.	`# rm -rf /var/nis*` If running in NIS–compatible mode, also remove NIS domain: `# rm -rf /var/yp/`*ypdomain*

Name the admin group.	`# NIS_GROUP=admin.wiz.com.; \`
	`  export NIS_GROUP`

Initialize the root master.
[NIS–compatibility with DNS forwarding only] Start daemon with **–Y –B, S 0**.
[NIS+ Only] Start daemon with –**S 0**.
	`# nisinit -r`
	`# startsrc -s rpc.nisd -a "-Y -B -S 0"`
	or
	`# startsrc -s rpc.nisd -a "-S 0"`

Create **org_dir**, **groups_dir**, tables.	`# /usr/lib/nis/nissetup [-Y]`

Tasks	Commands

Create DES credentials for root master.	`# nisaddcred des`
	`Enter login password:`

Create admin group.	`# nisgrpadm -c admin.wiz.com.`
Assign full group rights to root directory	`# nischmod g+rmcd wiz.com.`

Add root master to admin group.	`# nisgrpadm -a admin.wiz.com.`
	`rootmaster.wiz.com.`

Update root directory's keys.	`# /usr/lib/nis/nisupdkeys wiz.com.`
Update **org_dir**'s keys.	`# /usr/lib/nis/nisupdkeys org_dir.wiz.com.`
Update **groups_dir**'s keys.	`# /usr/lib/nis/nisupdkeys groups_dir.wiz.com.`
Start NIS+ cache manager	`# startsrc -s nis_cachemgr`
Kill existing NIS+ daemon.	`# stopsrc -s rpc.nisd`
Restart the NIS+ daemon. Use **–y** for NIS compatibility and **–b** for DNS forwarding.	`# mk_nisd [-y] [-b][-I|-B|-N]`

Add your LOCAL credentials.	`# nisaddcred -p 11177 \`
	` -P topadmin.wiz.com. local`

Add your DES credentials.	`# nisaddcred -p unix.11177@wiz.com  \`
	` -P topadmin.wiz.com. des`
	`Enter login password:`

Add credentials for other admins. Add other admins to admin group.	`# nisaddcred ...`
	`# nisgrpadm -a admin.wiz.com. member`
	` ...`

Setting Up NIS+ Servers

This section provides step–by–step procedures for using the NIS+ command set to perform three server–related tasks:

- Setting Up an NIS+ Server, on page 5-43
- Adding a Replica to an Existing Domain, on page 5-45

Note: It is much easier to perform this task with the NIS+ installation scripts as described in NIS+ Setup Scripts, on page 5-7 than with the NIS+ command set as described here. The methods described in this section should be used only by those administrators who are very familiar with NIS+ and who require some nonstandard features or configurations not provided by the installation scripts.

See Configuration Worksheets, on page 5-5, for worksheets that you can use to plan your NIS+ namespace.

A summary of each task is provided at the end of the section.

Setting Up an NIS+ Server

This section applies to any NIS+ server except the root master; that is, root replicas, nonroot masters, and nonroot replicas, whether running in NIS–compatibility mode or not.

Standard versus NIS–Compatible Setup Procedures

The differences between setting up an NIS–compatible and a standard NIS+ server are the same as the differences between setting up standard and NIS–compatible root master servers (see Standard versus NIS–Compatible Setup Procedures, on page 5-32). The NIS+ daemon for an NIS–compatible server must be started with the **–Y** option (and the **–B** option for DNS forwarding), which allows the server to answer requests from NIS clients.

Note: Whenever **rpc.nisd** is started with either the –**Y** or **–B** option, a secondary daemon named **rpc.nisd_resolv** is spawned to provide name resolution. This secondary daemon must be separately killed whenever you kill the primary **rpc.nisd** daemon.

Here is a summary of the entire setup process:

1. Logging in as superuser to the new replica server.
2. [NIS–Compatibility Only] Starting the NIS+ daemon with **–Y**.
3. [Standard NIS+ Only] Starting the NIS+ daemon.

Security Considerations

You must perform this operation as superuser on the server. The security level at which you start the server determines the credentials that its clients must have. For instance, if the server is set up with security level 2 (the default), the clients in the domain it supports must have DES credentials. If you have set up the client according to the instructions in this book, the client has DES credentials in the proper domain, and you can start the server with security level 2.

Note: Security level 0 is for administrator setup and testing purposes only. Security level 1 is not supported. Do not use level 0 or 1 in any environment where ordinary users are doing their normal work. Operating networks should always be run at security level 2.

Prerequisites

- The root domain must already be set up (see Setting Up the Root Domain, on page 5-32).
- The server must have already been initialized as an NIS+ client (see Setting Up NIS+ Clients, on page 5-62).

- If the server will run in NIS–compatibility mode and support DNS forwarding, it must have a properly configured **/etc/resolv.conf** file.

You need the superuser password of the client that you will convert into a server.

Procedure

1. Log in as superuser to the new replica server.

   The following steps assume you rebooted the workstation after you set it up as an NIS+ client, as instructed in Client Setup, on page 5-66. Rebooting starts the cache manager, which is a recommended prerequisite to the following step. If you did not reboot the workstation, restart the cache manager now, using **stopsrc –s nis_cachemgr** then **startsrc –s nis_cachemgr**.

2. [NIS–Compatibility Only] Start the NIS+ daemon with **–Y**.

   Perform this step only if you are setting up the server in NIS–compatibility mode; if setting up a standard NIS+ server, perform the following step instead.

   This step also includes instructions for supporting the DNS forwarding capabilities of NIS clients.

   This step has two parts. The first part starts the NIS+ daemon in NIS–compatibility mode. The second part makes sure that when the server is rebooted, the NIS+ daemon restarts in NIS–compatibility mode.

   – Start the NIS+ daemon with the **–Y** and **–B** flags.

     ```
     compatserver# startsrc –s rpc.nisd –a ″–Y –B″
     ```

     The **–Y** option invokes an interface that answers NIS requests in addition to NIS+ requests. The **–B** option supports DNS forwarding.

   – Run the **nissetup** command.

     This step creates a directory called **/var/nis/data** and a transaction log file called **trans.log**, which is placed in **/var/nis**. To verify that **/var/nis/data** exists:

     ```
     compatserver# ls –F /var/nis
     NIS_COLD_START   data/  trans.log  data.dict
     ```

     You can examine the contents of the transaction log by using the **nislog** command, described in Administering NIS+ Directories, on page 6-50.

     Attention: Do not rename the **/var/nis** directory or the **/var/nis/trans.log** or **/var/nis/data.dict** files. If any required file or directory is renamed, NIS+ will not start or operate correctly.

     Now this server is ready to be designated a master or replica of a domain, as described in Setting Up a Nonroot Domain, on page 5-57. This step completes this task.

3. [Standard NIS+ Only] Start the NIS+ daemon.

   ```
   server# startsrc –s rpc.nisd
   ```

   To verify that the NIS+ daemon is indeed running, use the **ps** command.

   ```
   server# ps –ef | grep rpc.nisd
   root 1081     1  0   16:43:33  –      0:01  rpc.nisd
   root 1087  1004  1   16:44:09  0      0:00  grep rpc.nisd
   ```

   This step creates a directory called **/var/nis/data** and a transaction log file called **trans.log** which is placed in **/var/nis**.

   ```
   compatserver# ls –F /var/nis
   NIS_COLD_START   data/  trans.log  data.dict
   ```

The **compatserver.log** file is a transaction log. You can examine the contents of the transaction log by using the **nislog** command, described in Administering NIS+ Directories, on page 6-50.

Attention: Do not rename the **/var/nis** directory or the **/var/nis/trans.log** or **/var/nis/data.dict** files. If any required file or directory is renamed, NIS+ will not start or operate correctly.

Now this server is ready to be designated a master or replica of a domain, as described in Setting Up a Nonroot Domain, on page 5-57. This step completes this task.

Adding a Replica to an Existing Domain

This section describes how to add a replica server to an existing domain using the raw NIS+ command, whether root or nonroot. Here is a list of the steps:

1. First set up the server as described in Setting Up an NIS+ Server, on page 5-43.
2. Log in to the domain's master server.
3. Add the replica to the domain.
4. Run **nisping** on the replica.

Note:

- If you have a domain that spans multiple subnets, it is a good idea to have at least one replica server within each subnet so that if the connection between nets is temporarily out of service, each subnet can continue to function until the connection is restored.
- An easier way to add a replica server is to use the `nisserver` script, as described in Setting Up NIS+ Servers.
- If the root master server is unavailable and the NIS+ domain is being served solely by a replica, you can obtain information from the NIS+ tables, but changes to the original tables can be made only when the master server is available.

Security Considerations

The NIS+ principal performing this operation must have modify rights to the domain's directory object.

Prerequisites

- The server that will be designated a replica must have already been set up.
- The domain must have already been set up and assigned a master server.

You need:

- The name of the server
- The name of the domain

Procedure

1. Log in to the domain's master server.
2. Add the replica to the domain.

   Run the **nismkdir** command with the **–s** option. The example adds the replica machine named **rootreplica** to the **Wiz.Com.** domain.

   ```
   rootmaster# nismkdir –s rootreplica Wiz.Com.
   rootmaster# nismkdir –s rootreplica org_dir.Wiz.Com.
   rootmaster# nismkdir –s rootreplica group_dir.Wiz.Com.
   ```

   When you run the **nismkdir** command on a directory object that already exists, it does not recreate the directory but simply modifies it according to the flags you provide. In this case, the **–s** flag assigns the domain an additional replica server. You can verify that the replica was added by examining the directory object's definition, using the **niscat** –o command.

Attention: Always run **nismkdir** on the master server. Never run **nismkdir** on the replica machine. Running **nismkdir** on a replica creates communications problems between the master and the replicas.

3. Run **nisping** on the directories

   This step sends a message (a "ping") to the new replica, telling it to ask the master server for an update. If the replica does not belong to the root domain, be sure to specify its domain name. (The example below includes the domain name only for completeness; since the example used throughout this task adds a replica to the root domain, the Wiz.Com. domain name in the example below is not necessary.)

```
rootmaster# nisping Wiz.Com.
rootmaster# nisping org_dir.Wiz.Com.
rootmaster# nisping group_dir.Wiz.Com.
```

   You should see results similar to these:

```
rootmaster# nisping Wiz.Com.
Pinging replicas serving directory wiz.com. :
Master server is rootmaster.wiz.com.
        No last update time
 Replica server is rootreplica.wiz.com.
        Last update seen was Wed Nov 18 11:24:32 1992
         Pinging ... rootreplica.wiz.com.
```

   If you have set up the domain's tables immediately after completing the domain setup, this step propagates the tables down to the replica. For more information about **nisping**, see Administering NIS+ Directories, on page 6-50.

Server Setup Summary

The following tables provide a summary of the tasks described in this section. They assume the simplest case, so be sure you are familiar with the more thorough task descriptions before you use this summary as a reference. This summary does not show the server's responses to each command.

Table 21. Starting Up a Nonroot Master Server: Command Summary

Tasks	Commands
Log in to the server as root.	`server% su`

NIS–compat only:
Start daemon with **–Y –B**.
	`server# startsrc –s rpc.nisd –a "–Y –B"`
	`server# nissetup`

NIS+–Only: Start daemon.	`server# startsrc –s rpc.nisd`

Table 22. Adding a Replica: Command Summary

Tasks	Commands
Log in as superuser to domain master.	`rootmaster% su`

Designate the new replica.	`# nismkdir –s rootreplica Wiz.Com.`
	`# nismkdir –s rootreplica org_dir.Wiz.Com.`
	`# nismkdir –s rootreplica groups_dir.Wiz.Com.`

Ping the replica.	`#/usr/lib/nis/nisping Wiz.Com.`
	`#/usr/lib/nis/nisping org_dir.Wiz.Com.`
	`#/usr/lib/nis/nisping groups_dir.Wiz.Com.`

Setting Up NIS+ Tables

This section provides step–by–step instructions for using the NIS+ command set to populate NIS+ tables on a root or nonroot master server from /**etc** files or NIS maps. This section also describes how to transfer information back from NIS+ tables to NIS maps, a procedure that may be required during a transition from NIS to NIS+. Finally, it includes two tasks that describe how to limit access to the passwd column of the passwd table:

- Populating NIS+ Tables From Files, on page 5-48
- Populating NIS+ Tables From NIS Maps, on page 5-51
- Transferring Information From NIS+ to NIS, on page 5-53
- Limiting Access to the Passwd Column to Owners and Administrators, on page 5-54

Note: It is much easier to perform this task with the NIS+ installation scripts than with the NIS+ command set as described here. The methods described in this section should be used only by those administrators who are very familiar with NIS+ and who require some nonstandard features or configurations not provided by the installation scripts.

See Configuration Worksheets, on page 5-5 for worksheets that you can use to plan your NIS+ namespace.

You can populate NIS+ tables in four ways:

- From files, as described in Populating NIS+ Tables From Files, on page 5-48
- From NIS maps, as described in Populating NIS+ Tables From NIS Maps, on page 5-51
- With the **nispopulate** script, as described in Populating NIS+ Tables, on page 5-15 and Populating the New Domain's Tables, on page 5-26

When populating tables from maps or files, the tables should have already been created in the process of setting up a root or subdomain as explained in Setting Up the Root Domain, on page 5-32, and Setting Up a Nonroot Domain, on page 5-57. Although you can populate a domain's tables at any time after they are created, it is recommended that you do so immediately after setting up the domain. This enables you to add clients more easily, since the required information about the clients should already be available in the domain's tables.

When you populate a table—whether from a file or an NIS map—you can use any of three options:

- **Replace** – With the replace option, NIS+ first deletes all existing entries in the table and then adds the entries from the source. In a large table, this adds a large set of entries into the master server's /**var**/**nis**/**trans.log** file (one set for removing the existing entries, another for adding the new ones), taking up space in /**var**/**nis**, Thus, propagation to replicas will take longer.
- **Append** – The append option simply adds the source entries to the NIS+ table.
- **Merge –** The merge option produces the same result as the replace option but uses a different process, one that can greatly reduce the number of operations that must be sent to the replicas. With the merge option, NIS+ handles three types of entries differently:
  - Entries that exist only in the source are added to the table
  - Entries that exist in both the source and the table are updated in the table
  - Entries that exist only in the NIS+ table are deleted from the table

When updating a large table with a file or map whose contents are not vastly different from those of the table, the merge option can spare the server a great many operations. Because it deletes only the entries that are not duplicated in the source (the replace option deletes **all** entries, indiscriminately), it saves one delete and one add operation for every duplicate entry. Therefore, this is the preferred option.

Populating NIS+ Tables From Files

This task transfers the contents of an ASCII file, such as /**etc**/**hosts**, into an NIS+ table.

Here is an outline of the procedure:

1. Checking the content of each file that you will be transferring data from.
2. Making a copy of each file. Using this copy to make the actual transfer from. (In this guide, copies of files to be transferred have names ending in **xfr** (for example, **hosts.xfr**).
3. Logging in to an NIS+ client. (You must have credentials and permissions allowing you to update the tables. See Security Considerations, on page 5-48.)
4. Adding /**usr**/**lib**/**nis** to the search path for this shell (if not already done).
5. Using **nisaddent** to transfer any of these files one at a time: **aliases**, **bootparams**, **ethers**, **group**, **hosts**, **netgroup**, **netmasks**, **networks**, **passwd**, **protocols**, **rpc**, **services**.
6. Transferring the **publickey** file.
7. Transferring the automounter information.
8. Checkpointing the tables.

Security Considerations

You can perform this task from any NIS+ client, including the root master server, as long as you have the proper credentials and access rights. If you are going to replace or merge the entries in the table with the entries from the text file, you must have create and destroy rights to the table. If you are going to append the new entries, you only need create rights.

Note: The NIS+ security system is complex. If you are not familiar with NIS+ security, you may wish to review the security–related sections of this book before starting to set up your NIS+ environment.

After you complete this operation, the table entries will be owned by the NIS+ principal that performed the operation and the group specified by the **NIS_GROUP** environment variable.

Prerequisites

- The domain must have already been set up and its master server must be running.
- The domain's servers must have enough swap space to accommodate the new table information. See Evaluate Disk Space and Memory Requirements, on page 5-3.
- The information in the file must be formatted appropriately for the table into which it will be loaded. See Prerequisites to running nispopulate, on page 5-15 for information on the format a text file must have to be transferred into its corresponding NIS+ table. Local /**etc** files are usually formatted properly, but may have several comments that you would need to remove.
- No duplicate machine and user names. All users and all machines must have unique names. You cannot have a machine with the same name as a user.
- Machine names cannot contain dots (periods). For example, a machine named **sales.alpha** is not allowed. A machine named **sales–alpha** is allowed.

You need the name and location of the text files that will be transferred.

Procedure

1. Check each file that you will be transferring data from.

   Make sure that there are no spurious or incorrect entries. Make sure that the right data is in the correct place and format properly. Remove any outdated, invalid, or corrupt entries. You should also remove any incomplete or partial entries. (It is easier to add incomplete entries after setup than to try transferring incomplete or damaged entries from the file.)

2. Make a working copy of each file you will be transferring.

   Use this working copy for the actual file transfer steps described in this section. Give each working copy the same filename extension (for example, **.xfr**).

   ```
   rootmaster% cp /etc/hosts /etc/hosts.xfr
   ```

3. Log in to an NIS+ client.

   You can perform this task from any NIS+ client—just be sure that the client belongs to the same domain as the tables into which you want to transfer the information. The examples in this task use the root master server. Since the administrator in these examples is logged on as superuser, the NIS+ principal actually performing this operation (and therefore needing the proper credentials and access rights) is the root master server.

4. Add /**usr**/**lib**/**nis** to the search path for this shell.

   Since you will be using the /**usr**/**lib**/**nis**/**nisaddent** command once per table, adding its prefix to the search path will save you the trouble of typing it each time.

   ```
   rootmaster# PATH=$PATH:/usr/lib/nis
   rootmaster# export PATH
   ```

5. Use **nisaddent** to transfer any of these files, one at a time:

   aliases, **auto_home**, **auto_master**, **bootparams**, **ethers**, **group**, **hosts**, **netgroup**, **netmasks**, **networks**, **protocols**, **rpc**, **services**

   The **publickey**, **automounter**, and **passwd** files require slightly different procedures; each is described separately in the following steps.

   By default, **nisaddent** appends the file information to the table information. To replace or merge instead, use the **–r** or **–m** options.

   To replace:

   ```
   rootmaster# nisaddent –r –f filename table [domain]
   ```

   To append:

   ```
   rootmaster# nisaddent –a –f filename table [domain]
   ```

   To merge:

   ```
   rootmaster# nisaddent –m –f filename table [domain]
   ```

   The best option for populating the tables for the first time is the **–a** option, the default. The best option to synchronize the NIS+ tables with NIS maps or /**etc** files is the **–m** (merge) option.

   – **filename** is the name of the file. The common convention is to append**.xfr** to the end of these file names to identify them as transfer files created with **nisaddent**.

   – **table** is the name of the NIS+ table. The **domain** argument is optional; use it only to populate tables in a different domain. Here are some examples, entered from the root domain's master server. The source files are simply edited versions of the /**etc** files:

     ```
     rootmaster# nisaddent –m –f /etc/hosts.xfr hosts
     rootmaster# nisaddent –m –f /etc/groups.xfr groups
     ```

   If you perform this operation from a non–root server, keep in mind that a non–root server belongs to the domain above the one it supports; therefore, it is a client of another domain. For example, the **Sales.Wiz.Com.** master server belongs to the Wiz.Com. domain. To populate tables in the Sales.Wiz.Com. domain from that master server, you would have to append the Sales.Wiz.Com. domain name to the **nisaddent** statement.

   ```
   salesmaster# nisaddent –f /etc/hosts.xfr hosts Sales.Wiz.Com.
   ```

If you performed this operation as a client of the Sales.Wiz.Com. domain, you would not need to append the domain name to the syntax.

6. To verify that the entries were transferred into the NIS+ table, use the **niscat** command.

```
rootmaster# niscat group.org_dir
root::0:root
other::1::
bin::2:root,bin,daemon
.
.
.
```

7. Transfer the **publickey** file.

   Since the domain's cred table already stores some credentials, you need to make sure they are not overwritten by the contents of the **publickey** text file that you transfer into the cred table. You can avoid this by removing those credentials from the **publickey** text file. For **rootmaster**, that line would be:

```
unix.rootmaster@Wiz.Com public-key:private-key
```

   Then you can transfer the contents of the **publickey** file to the cred table. Use **nisaddent**, with the **–a** (add) option.

```
rootmaster# nisaddent -a -f /etc/publickey.xfr -t cred.org_dir publickey\
  [domain]
```

   Note, however, that this operation only transfers DES credentials into the cred table. You will still need to create their LOCAL credentials to the cred table.

8. Transfer the automounter information.

   Although the **nissetup** utility creates auto_master and auto_home tables, they are not considered standard NIS+ tables. Therefore, transferring information into them requires a slightly different syntax; in particular, you must use the **–t** flag and specify that the table is of type key–value.

```
rootmaster# nisaddent -f auto.master.xfr -t auto_master.org_dir key-value
rootmaster# nisaddent -f auto.home.xfr   -t auto_home.org_dir   key-value
```

9. Build the NIS+ passwd table.

   The NIS+ passwd table is composed of data drawn from the /**etc**/**passwd** file.

```
rootmaster# nisaddent -m -f /etc/passwd.xfr passwd
```

10. Checkpoint the tables.

   This step ensures that all the servers supporting the domain transfer the new information from their **.log** files to the disk–based copies of the tables. If you have just set up the root domain, this step affects only the root master server, since the root domain does not yet have replicas. Use the **nisping** command with the **–C** (uppercase) option.

```
rootmaster# nisping -C org_dir
Checkpointing replicas serving directory org_dir.Wiz.Com. :
Master server is rootmaster.Wiz.Com.
     Last update occurred at July 14, 1994
Master server is rootmaster.Wiz.Com.
checkpoint succeeded.
```

   Attention: If you do not have enough swap space, the server cannot checkpoint properly, but it does not notify you. One way to make sure all went well is to list the contents of a table with the **niscat** command. If you do not have enough swap space, you see this error message:

```
can't list table: Server busy, Try Again.
```

   Even though it does not *seem* to, this message indicates that you do not have enough swap space. Increase the swap space and checkpoint the domain again.

Populating NIS+ Tables From NIS Maps

This task transfers the contents of an NIS map into an NIS+ table. Here is a list of the steps:

1. Checking the content of each NIS map that you will be transferring data from.
2. Logging in to an NIS+ client.
3. Adding /**usr**/**lib**/**nis** to the search path for this shell.
4. Using **nisaddent** to transfer any of these maps, one at a time: **aliases**, **auto_home**, **auto_master**, **bootparams**, **ethers**, **group**, **hosts**, **netgroup**, **netmasks**, **networks**, **passwd**, **protocols**, **rpc**, **services**.
5. Transferring the **publickey** map.
6. Transferring the automounter information.
7. Checkpointing the tables.

Security Considerations

You can perform this task from any NIS+ client as long as you (or superuser on the client) have the proper credentials and access rights. If you are going to replace or merge the entries in the table with the entries from the NIS map, you must have create and destroy rights to the table. If you are going to append the new entries, you only need create rights.

After you complete this operation, the table entries will be owned by the NIS+ principal that performed the operation (either you or, if logged on as superuser, the client) and the group specified by the **NIS_GROUP** environment variable.

Prerequisites

- The domain must have already been set up and its master server must be running.
- The **dbm** files (**.pag** and **.dir** files) for the NIS maps you are going to load into the NIS+ tables must already be in a subdirectory of /**var**/**yp**.
- No duplicate machine and user names. All users and all machines must have unique names. You cannot have a machine with the same name as a user.
- Machine names cannot contain dots (periods). For example, a machine named **sales.alpha** is not allowed. A machine named **sales–alpha** is allowed.

You need the name and location of the NIS maps.

Procedure

1. Check each NIS map that you will be transferring data from.

   Make sure that there are no spurious or incorrect entries. Make sure that the right data is in the correct place and format properly. Remove any outdated, invalid, or corrupt entries. You should also remove any incomplete or partial entries. (It is easier to add incomplete entries after setup than to try transferring incomplete or damages entries from the map.)

2. Log in to an NIS+ client.

   You can perform this task from any NIS+ client—so long as that client belongs to the same domain as the tables into which you want to transfer the information. The examples in this task use the root master server. Since the administrator in these examples is logged in as superuser, the NIS+ principal actually performing this operation (and therefore needing the proper credentials and access rights) is the root master server.

3. Add /**usr**/**lib**/**nis** to the search path for this shell.

   Since you will be using the /**usr**/**lib**/**nis**/**nisaddent**
   command once for each table, adding its prefix to the search path will save you the trouble of typing it each time.

```
rootmaster# PATH=$PATH:/usr/lib/nis
rootmaster# export PATH
```

4. Use **nisaddent** to transfer any of these maps, one at a time:

   aliases, **auto_home**, **auto_master**, **bootparams**, **ethers**, **group**, **hosts**, **netgroup**, **netmasks**, **networks**, **passwd**, **protocols**, **rpc**, **services**.

   By default, **nisaddent** appends the file information to the table information. To replace or merge instead, use the **–r** or **–m** options: To replace:

   ```
   rootmaster# nisaddent –r –y nisdomain   table
   ```

   To append:

   ```
   rootmaster# nisaddent –a –y nisdomain   table
   ```

   To merge:

   ```
   rootmaster# nisaddent –m –y nisdomain   table
   ```

   The best option for populating the tables for the first time is the **–a** option, which is the default. The best option to synchronize the NIS+ tables with NIS maps or /**etc** files is the **–m** (merge) option.

   The **–y** (lowercase) option indicates an NIS domain instead of a text file. The *nisdomain* argument is the name of the NIS domain whose map you are going transfer into the NIS+ table. You do not have to name the actual map; the **nisaddent** utility automatically selects the NIS map that correspond to the *table* argument. Here are some examples:

   ```
   rootmaster# nisaddent –m –y oldwiz hosts
   rootmaster# nisaddent –m –y oldwiz passwd
   rootmaster# nisaddent –m –y oldwiz groups
   ```

   The first example transfers the contents of the **hosts.byname** and **hosts.byaddr** maps in the oldwiz (NIS) domain to the NIS+ hosts table in the root domain (NIS+). The second transfers the NIS maps that store password–related information into the NIS+ passwd table. The third does the same with group–related information.

5. Transfer the **publickey** map.

   Since the domain's cred table already stores some credentials, you need to make sure they are not overwritten by the contents of the **publickey** map that you transfer into the cred table.

6. First, dump the **publickey** map to a file and then open that file with your text editor.

   ```
   rootmaster# makedbm –u /var/yp/oldwiz/publickey.byname /etc/publickey.xfr
   rootmaster# vi /tmp/publickey.tmp
   ```

7. Now remove the credentials of the workstation you are logged in to from the **publickey** map.

   For **rootmaster**, that line would be:

   ```
   unix.rootmaster@Wiz.Com public-key:private-key
   ```

8. Now you can transfer the contents of the **file**—not the map—into the cred table. Use **nisaddent**, with the **–a** (add) option.

   ```
   rootmaster# nisaddent –a –f /etc/publickey.xfr –t cred.org_dir Publickey
   ```

   Note, however, that this operation transfers only DES credentials into the cred table. You will still need to create their LOCAL credentials to the cred table.

9. Transfer the automounter information.

   Although the **nissetup** utility creates auto_master and auto_home tables, they are not considered standard NIS+ tables. Therefore, transferring information into them requires a slightly different syntax:

   ```
   rootmaster# nisaddent –y oldwiz –Y auto.master –t auto_master.org_dir key–value
   rootmaster# nisaddent –y oldwiz –Y auto.home –t auto_home.org_dir key–value
   ```

The **–m** and **–y** options are still required, as is the NIS domain name (in this instance, oldwiz). However, you must precede the name of the NIS map (for example, **auto.master**) with a **–Y** (uppercase).Then, as is required when transferring automounter **text files**, you must use the **–t** option, which indicates that this is a nonstandard NIS+ table. Its arguments are the name of the NIS+ directory object (**auto_master.org_dir**) and the type of table (key–value). Be sure to append the **org_dir** suffixes to the NIS+ table names.

10. Checkpoint the tables.

    This step ensures that all the servers supporting the domain transfer the new information from their **.log** files to the disk–based copies of the tables. If you just finished setting up the root domain, this step affects only the root master server, since the root domain does not yet have replicas. Use the **nisping** command with the **–C** (uppercase) option.

```
rootmaster# nisping –C org_dir
Checkpointing replicas serving directory org_dir.wiz.com. :
Master server is rootmaster.wiz.com.
     Last update occurred at July 14, 1994
Master server is rootmaster.wiz.com.
checkpoint succeeded.
```

    If you do not have enough swap space, the server will be unable to checkpoint properly, but it will not notify you. One way to make sure all went well is to use list the contents of a table with the **niscat** command. If you do not have enough swap space, you will see this error message:

```
can’t list table: Server busy, Try Again.
```

    Even though it does not **seem** to, this message indicates that you do not have enough swap space. Increase the swap space and checkpoint the domain again.

Transferring Information From NIS+ to NIS

This task transfers the contents of NIS+ tables into the NIS maps on an NIS master server. Here is a an outline of the procedure:

1. Logging into the NIS+ server.
2. Transferring the NIS+ tables into output files.
3. Transferring the contents of the output files to the NIS maps.

Security Considerations

To perform this task, you must have read access to each table whose contents you transfer.

Prerequisites

The maps must have already been built on the NIS server.

Procedure

1. Log into the NIS+ server.

   This example uses the server named **dualserver**.

2. Transfer the NIS+ tables into output files.

   Use the **nisaddent** command with the **–d** option, once for each table.

```
dualserver% /usr/lib/nis/nisaddent –d –t table tabletype > filename
```

   The **–d** option transfers the contents of *table* to *filename*, converting the contents back to standard /**etc** file format.

3. Transfer the contents of the output files into the NIS maps.

   The NIS+ output files are ASCII files that you can use as input files for the NIS maps. Copy them into the NIS master’s /**etc** directory, and then use **make** as usual.

```
dualserver# cd /var/yp
dualserver# make
```

Limiting Access to the Passwd Column to Owners and Administrators

This task describes how to limit read access to the password–related columns of the passwd table only to the entry owner and the table administrators without affecting the read access of other authenticated principals (including applications) to the remaining columns of the passwd table.

This task establishes the following rights:

```
                         Nobody  Owner  Group  World
Table Level Rights   :   ----    rmcd   rmcd   ----
Passwd Column Rights :   ----    rm--   rmcd   ----
Shadow Column Rights :   ----    rm--   rmcd   ----
```

Security Considerations

- The domain must **not** be running in NIS–compatibility mode.
- All clients of the domain must have DES credentials.
- All clients of the domain must be running AIX Version 4.3.3 or a later release.
- Users' network passwords (used to encrypt their DES credentials) must be the same as their login passwords.

Prerequisites

- The passwd table must have already been set up. It need not have any information in it, however.
- The NIS+ principal performing this task must have modify rights to the passwd table.

All you need is the name of the passwd table.

Procedure

1. Log in to the domain's master server.

   The examples in this task use the root master server, **rootmaster**.

2. Check the current table and column permissions.

   Use the **niscat –o** command.

   ```
   rootmaster# niscat -o passwd.org_dir
   ```

   This task assumes the existing permissions are:

   ```
   Access Rights    : ----rmcdrmcdr---
   Columns          :
           [0]  Name          : name
                Access Rights : r-----------r---
           [1]  Name          : passwd
                Access Rights : -----m----------
           [2]  Name          : uid
                Access Rights : r-----------r---
           [3]  Name          : gid
                Access Rights : r-----------r---
           [4]  Name          : gcos
                Access Rights : r----m------r---
           [5]  Name          : home
                Access Rights : r-----------r---
           [6]  Name          : shell
                Access Rights : r-----------r---
           [7]  Name          : shadow
                Access Rights : r-----------r---
   ```

If your permissions are different, you may need to use a different syntax. For instructions, see Administering NIS+ Access Rights, on page 6-21.

3. Change the table permissions.

   Use the **nischmod** command to change the table's object–level permissions to ––––**rmcdrmcd**––––

   ```
   rootmaster# nischmod og=rmcd,nw= passwd.org_dir
   ```

4. Change the column permissions.

   Use the **nistbladm** command with the **–u** option to change the permissions of the passwd and shadow columns to:

   ```
   passwd    ---- rm-- ---- ----
   shadow   ---- r--- ---- ----
   ```

   ```
   rootmaster# nistbladm -u passwd=o+r, shadow=o+r passwd.org_dir
   ```

5. Verify the new permissions.

   Use the **niscat –o** command as you did earlier in this procedure. The permissions should look the same as they did before.

Table Population Summaries

Following are summaries of the steps required to populate NIS+ tables. They assume the simplest case, so be sure you are familiar with the more thorough task descriptions before you use this summary as a reference. For brevity, these summaries do not show the server's responses to each command.

Table 23. Transferring NIS Files Into NIS+ Tables: Command Summary

Tasks	**Commands**
Log in to an NIS+ client.	`rootmaster%`

Create working copies of the files to be transferred.	`% cp /etc/hosts /etc/hosts.xfr`
	`% .`

Add /**usr**/**lib**/**nis** to search path.	`% PATH=$PATH:/usr/lib/nis; export PATH`

Transfer each file, one at a time.	`% nisaddent -m -f /etc/hosts.xfr hosts`
	`% .`
	`% .`

Remove old server credentials from **publickey** file.	`% vi /etc/publickey.xfer`
	`% .`

Transfer it to the cred table.	`% nisaddent -a -f /etc/publickey.xfr cred`

Transfer the automounter files.	`% nisaddent -f auto.master.xfr \`
	`  -t auto_master.org_dir key-value`
	`% nisaddent -f auto.home.xfr \`
	`  -t auto_home.org_dir key-value`

Checkpoint the table directory.	`% nisping -C org_dir`

Table 24. Transferring NIS Maps Into NIS+ Tables: Command Summary

Tasks	**Commands**
Log in to an NIS+ client.	`rootmaster%`
Add /**usr**/**lib**/**nis** to search path.	`% PATH=$PATH:/usr/lib/nis; export PATH`

Tasks	Commands

Transfer each map, one at a time.	`% nisaddent -m -y oldwiz hosts`
	`% .`
	`% .`
	`% .`

Dump **publickey** map to a file.	`% makedbm \`
	`  -u /var/yp/oldwiz/publickey.byname \`
	`  > /etc/publickey.xfr`

Remove new credentials.	`% vi /etc/publickey.xfr`
	`% .`
	`% .`
	`% .`

Transfer the **publickey** file.	`% nisaddent -a -f /etc/publickey.xfr \`
	`  -t cred.org_dir publickey`

Transfer the automounter maps.	`% nisaddent -y oldwiz \`
	`  -Y auto.master \`
	`  -t auto_master.org_dir key-value`
	`% nisaddent -y oldwiz \`
	`  -Y auto.home \`
	`  -t auto_home.org_dir key-value`

Checkpoint the table directory.	`% nisping -C org_dir`

Table 25.   Transferring NIS+ Tables to NIS Maps: Command Summary

Tasks	Commands
Log in to NIS+ server.	`dualserver%`

Transfer NIS+ tables to files.	`% /usr/lib/nis/nisaddent \`
	`  -d [-t` *`table`*`]` *`tabletype`* `\`
	`  >` *`filename`*
	`% .`
	`% .`
	`% .`

Transfer files to NIS maps.	`% makedbm` *`flags output-file NIS-dbm-file`*

Table 26.   Limiting Access to Passwd Column: Command Summary

Tasks	Commands
Log into the domain's master server.	`rootmaster#`
Check the table's existing rights.	`# niscat -o passwd.org_dir`
Assign the table new rights.	`# nischmod og=rmcd,nw=  passwd.org_dir`
Assign the columns new rights	`# nistbladm -u  passwd=o+r, shadow=n+r passwd.org_dir`
Verify the new rights.	`# niscat -o passwd.org_dir`

Setting Up a Nonroot Domain

This section provides step–by–step instructions for using NIS+ commands to set up a nonroot domain (also known as a subdomain). You should not set up a nonroot domain until **after** you have set up servers.

A summary of this task is provided by Subdomain Setup Summary, on page 5-61.

Note: It is much easier to perform this task with the NIS+ installation scripts than with the NIS+ command set as described here. The methods described in this section should be used only by those administrators who are very familiar with NIS+ and who require some nonstandard features or configurations not provided by the installation scripts.

See Configuration Worksheets, on page 5-5 for worksheets that you can use to plan your NIS+ namespace.

Setting up a nonroot domain involves the following tasks:

- Establishing security for the domain
- Creating the domain's directories
- Creating the domain's tables
- Designating the domain's servers

However, as with setting up the root domain, these tasks cannot be performed sequentially. To make the setup process easier to execute, they have been broken down into individual steps, and the steps have been arranged into the most efficient order.

Standard versus NIS–Compatible Setup Procedures

The differences between NIS–compatible and standard NIS+ servers in subdomains are the same as they are for servers in the root domain (see Standard versus NIS–Compatible Setup Procedures, on page 5-32).

The NIS+ daemon for each server in an NIS–compatible domain should have been started with the **–Y** option, as instructed in Setting Up NIS+ Servers, on page 5-43. An NIS–compatible domain also requires its tables to provide read rights for the nobody class, which allows NIS clients to access the information stored in them. NIS+ is made NIS compatible by using the **–Y** option with the **nissetup** command. (The standard NIS+ domain version uses the same setup command but without the **–Y** option.)

Here is a summary of the entire setup process:

1. Logging in to the domain's master server.
2. Naming the domain's administrative group.
3. Creating the domain's directory and designate its servers.
4. Creating the domain's subdirectories and tables.
5. Creating the domain's admin group.
6. Assigning full group access rights to the directory object.
7. Adding the servers to the domain's admin group.
8. Adding credentials for other administrators.
9. Adding the administrators to the domain's admin group.

Security Considerations

At most sites, to preserve the security of the parent domain, only the parent domain's master server or an administrator who belongs to the parent domain's admin group is allowed to create a domain beneath it. Although this is a policy decision and not a requirement of NIS+, the instructions in this section assume that you are following that policy. Of course, the parent domain's admin group must have create rights to the parent directory object. To verify this, use the **niscat** –o command.

```
rootmaster# niscat –o Wiz.Com.
Object Name   : wiz
Owner         : rootmaster
Group         : admin.wiz.com.
Access Rights : r---rmcdrmcdr---
:
```

If you are more concerned about convenience than security, you could simply make the new domain's master server a member of its parent domain's admin group and then perform the entire procedure from the server. Use the **nisgrpadm** command.

Prerequisites

- The parent domain must be set up and running.
- The server that will be designated as this domain's master must be initialized and running NIS+.
- If you will designate a replica server, the master server must be able to obtain the replica's IP address through its **/etc/hosts** file or from its NIS+ hosts table.

You also need the following information:

- The name of the new domain
- The name of the new domain's master and replica servers
- The name of the new domain's admin group
- User IDs (UID) of the administrators who will belong to the new domain's admin group

Procedure

1. Log in to the domain's master server.

   Log in to the server that you will designate as the new domain's master. The steps in this task use the server named **smaster**, which belongs to the Wiz.Com. domain, and will become the master server of the Sales.Wiz.Com. subdomain. The administrator performing this task is **nisboss.Wiz.Com.**, a member of the **admin.Wiz.Com.** group. That group has full access rights to the **Wiz.Com.** directory object.

2. Name the domain's administrative group.

   Although you won't actually create the admin group until later, you need to identify it now. This enables the **nismkdir** command used in the following step to create the directory object with the proper access rights for the group. It does the same for the **nissetup** utility.

   Set the value of the environment variable **NIS_GROUP** to the name of the domain's admin group. The following example sets **NIS_GROUP** to **admin.Sales.Wiz.Com.**

   ```
   smaster# NIS_GROUP=admin.Sales.Wiz.Com.
   smaster# export NIS_GROUP
   ```

3. Create the domain's directory and designate its servers.

   The **nismkdir** command, in one step, creates the new domain's directory and designates its supporting servers. It has the following syntax:

   ```
   nismkdir –m master –s replica domain
   ```

The **–m** flag designates its master server, and the **–s** flag designates its replica.

```
smaster# nismkdir –m smaster –s salesreplica Sales.Wiz.Com.
```

Attention: Always run **nismkdir** on the master server. Never run **nismkdir** on the replica machine. Running **nismkdir** on a replica creates communications problems between the master and the replica.

The directory is loaded into /**var**/**nis**. Use the **niscat –o** command to view it (do not use **cat** or **more**).

```
smaster# niscat –o Sales.Wiz.Com.
Object Name   : sales
Owner         : nisboss.wiz.com.
Group         : admin.sales.wiz.com.
Access Rights : ----rmcdr---r---
.
.
.
```

Unlike the root directory, this directory object **does** have the proper group assignment. As a result, you won't have to use **nischgrp**.

4. Create the domain's subdirectories and tables.

   This step adds the **org_dir** and **groups_dir** directories and the NIS+ tables beneath the new directory object. Use the **nissetup** utility, but be sure to add the new domain name. And, for an NIS–compatible domain, include the **–Y** flag.

   NIS compatible:

   ```
   smaster# /usr/lib/nis/nissetup –Y Sales.Wiz.Com.
   ```

   NIS+

   ```
   smaster# /usr/lib/nis/nissetup   Sales.Wiz.Com.
   ```

   Each object added by the utility is listed in the output:

   ```
   smaster# /usr/lib/nis/nissetup
   org_dir.Sales.Wiz.Com. created
   groups_dir.Sales.Wiz.Com. created
   auto_master.org_dir.Sales.Wiz.Com. created
   auto_home.org_dir.Sales.Wiz.Com. created
   bootparams.org_dir.Sales.Wiz.Com. created
   cred.org_dir.Sales.Wiz.Com. created
   ethers.org_dir.Sales.Wiz.Com. created
   group.org_dir.Sales.Wiz.Com. created
   hosts.org_dir.Sales.Wiz.Com. created
   mail_aliases.org_dir.Sales.Wiz.Com. created
   sendmailvars.org_dir.Sales.Wiz.Com. created
   client_info.org.dir.Sales.Wiz.Com. created
   netmasks.org_dir.Sales.Wiz.Com. created
   netgroup.org_dir.Sales.Wiz.Com. created
   networks.org_dir.Sales.Wiz.Com. created
   passwd.org_dir.Sales.Wiz.Com. created
   protocols.org_dir.Sales.Wiz.Com. created
   rpc.org_dir.Sales.Wiz.Com. created
   services.org_dir.Sales.Wiz.Com. created
   timezone.org_dir.Sales.Wiz.Com. created
   ```

   The **–Y** option creates the same tables and subdirectories as for a standard NIS+ domain, but assigns read rights to the nobody class so that requests from NIS clients, which are unauthenticated, can access information in the NIS+ tables.

   You can verify the existence of the **org_dir** and **groups_dir** directories by looking in your master server's /**var**/**nis**/**data** directory. They are listed along with the root object and other NIS+ tables. The tables are listed under the **org_dir** directory. You can examine the contents of any table by using the **niscat** command (although at this point the tables are empty).

5. Create the domain's admin group.

   This step creates the admin group named earlier. Use the **nisgrpadm** command with the **–c** option. This example creates the **admin.Sales.Wiz.Com.** group

```
smaster# nisgrpadm –c admin.Sales.Wiz.Com.
Group admin.Sales.Wiz.Com. created.
```

   This step only creates the group—it does not identify its members.

6. Assign full group access rights to the directory object.

   By default, the directory object only grants its group read access, which makes the group no more useful than the world class. To make the setup of clients and subdomains easier, change the access rights that the directory object grants its group from read to read, modify, create, and destroy. Use the **nischmod** command.

```
smaster# nischmod g+rmcd Sales.Wiz.Com.
```

7. Add the servers to the domain's admin group.

   At this point, the domain's group has no members. Add the master and replica servers, using the **nisgrpadm** command with the **–a** option. The first argument is the group name; the others are the names of the new members. This example adds **smaster.Wiz.Com.** and **salesreplica.Wiz.Com.** to the **admin.Sales.Wiz.Com.** group:

```
smaster# nisgrpadm –a admin.Sales.Wiz.Com.   smaster.Wiz.Com. \
  salesreplica.Wiz.Com.
Added smaster.Wiz.Com. to group admin.Sales.Wiz.Com.
Added salesreplica.Wiz.Com. to group admin.Sales.Wiz.Com.
```

   To verify that the servers are indeed members of the group, use the **nisgrpadm** command with the **–l** option.

```
smaster# nisgrpadm –l admin.Sales.Wiz.Com.
Group entry for admin.Sales.Wiz.Com. group:
    Explicit members:
        smaster.Wiz.Com.
        salesreplica.Wiz.Com.
    No implicit members
    No recursive members
    No explicit nonmembers
    No implicit nonmembers
    No recursive nonmembers
```

8. Add credentials for other administrators.

   Add the credentials of the other administrators who will work in the domain.

   For administrators who already have DES credentials in another domain, simply add LOCAL credentials. Use the **nisaddcred** command with both the **–p** and the **–P** flags.

```
smaster# nisaddcred –p 33355 –P nisboss.Wiz.Com. local
```

   For administrators that do not yet have credentials, you can proceed in two different ways.

   – One way is to ask them to add their own credentials. However, they will have to do this as superuser. Here is an example in which an administrator with a UID of 22244 and a principal name of **juan.Sales.Wiz.Com.** adds his own credentials to the Sales.Wiz.Com. domain.

```
smaster# nisaddcred –p 22244 –P juan.Sales.Wiz.Com. local
smaster# nisaddcred –p unix.22244@Sales.Wiz.Com –P juan.Sales.Wiz.Com. des
Adding key pair for unix.22244@Sales.Wiz.Com.
Enter login password:
```

   – The other way is for you to create temporary credentials for the other administrators, using dummy passwords (note that each administrator must have an entry in the NIS+ passwd table).

```
smaster# nisaddcred -p 22244 -P juan.Sales.Wiz.Com. local
smaster# nisaddcred -p unix.22244@Sales.Wiz.Com -P juan.Sales.Wiz.Com. des
Adding key pair for unix.22244@Sales.Wiz.Com.
Enter juan's login password:

nisaddcred: WARNING: password differs from login passwd.
Retype password:
```

Each administrator can later change his or her network password by using the **chkey** command.

Note: In the two examples shown above, the domain name following the lower case **–p** flag must **never** end in a trailing dot, while the domain name following the upper case **–P** flag must **always** end in a trailing dot.

9. Add the administrators to the domain's admin group.

   You don't have to wait for the other administrators to change their dummy passwords to perform this step. Use the **nisgrpadm** command with the **–a** option. The first argument is the group name, and the remaining arguments are the names of the administrators. This example adds the administrator **juan** to the **admin.Sales.Wiz.Com.** group:

```
smaster# nisgrpadm -a admin.Sales.Wiz.Com. juan.Sales.Wiz.Com.
Added juan.Sales.Wiz.Com. to group admin.Sales.Wiz.Com.
```

Subdomain Setup Summary

The following table is a summary of the steps required to set up a non–root domain. It assumes the simplest case, so be sure you are familiar with the more thorough task descriptions before you use this summary as a reference. This summary does not show the server's responses to each command.

Table 27. Setting Up a Subdomain Command Summary

Tasks	Commands
Log in as superuser to domain master.	`smaster% su`

Name the domain's admin group.	`# NIS_GROUP=admin.Sales.Wiz.Com.`
	`# export NIS_GROUP`

Create the domain's directory and designate its servers.	`# nismkdir -m smaster \`
	`  -s salesreplica Sales.Wiz.Com.`

Create **org_dir, groups_dir**, and tables. (For NIS–compatibility, use **–Y**.)	`# /usr/lib/nis/nissetup Sales.Wiz.Com.`
Create the admin group.	`# nisgrpadm -c admin.Sales.Wiz.Com.`
Assign full group rights to the domain's directory.	`# nischmod g+rmcd Sales.Wiz.Com.`

Add servers to admin group.	`# nisgrpadm -a admin.Sales.Wiz.Com.\`
	`  smaster.Wiz.Com. sreplica.Wiz.Com.`

Add credentials for other admins.	`# nisaddcred -p 22244 \`
	`  -P juan.Sales.Wiz.Com. local`
	`# nisaddcred -p unix.22244@Sales.Wiz.com.\`
	`  juan.Sales.Wiz.Com. des`

Add admins to domain's admin group.	`# nisgrpadm -a admin.Sales.Wiz.Com.`
	`  juan.Sales.Wiz.Com.`

Setting Up NIS+ Clients

This section provides step–by–step instructions for using the NIS+ command set to perform the following tasks:

- Changing a Workstation's Domain, on page 5-62
- Client Setup, on page 5-66
- Initializing an NIS+ Client, on page 5-63

Note: It is much easier to perform this task with the NIS+ installation scripts than with the NIS+ command set as described here. The methods described in this section should only be used by those administrators who are very familiar with NIS+ and who require some non–standard features or configurations not provided by the installation scripts.

See Configuration Worksheets, on page 5-5 for worksheets that you can use to plan your NIS+ namespace.

This section describes how to set up clients in both standard NIS+ domains and NIS–compatible domains.

The procedure describes each step in detail and provides related information. For those who do not need detailed instructions, a summary listing of the necessary commands is provided in NIS Client Setup Summary, on page 5-65.

Note that in the client setup instructions you must choose which of three methods to use: broadcast, host name, or cold–start file. Since each method is implemented differently, each has its own task description. After initializing a client by one of these methods, you can continue setting up the client.

The last task in the section describes how to change a workstation's domain.

Changing a Workstation's Domain

This task changes a workstation's domain name. Since a workstation's domain name is usually set during installation, you should check it (type **domainname** without an argument) before you decide to perform this task.

Specifying a Domain Name After Installation

A workstation is usually assigned to its domain during installation. On an operating network, the installation script usually obtains the domain name automatically and simply asks the installer to confirm it. During the installation proper, the workstation's domain name is assigned to a variable called **domainname**, which is stored in the kernel. There, it is made available to any program that needs it.

However, when a workstation is rebooted, the setting of the **domainname** variable is lost. As a result, unless the domain name is saved somewhere else, the operating system no longer knows which domain the workstation belongs to. To solve this problem, the domain name is stored in a file called **/etc/rc.nfs**.

Note: When the workstation is rebooted, the kernel automatically obtains the domain name from this file and resets the **domainname** variable. Thus, if you change or set a workstation's domain name, you must also edit the **/etc/rc.nfs** file or run the **chypdom** command; if you do not, after the next reboot, the workstation will revert to its previous domain name.

Security Considerations

You must perform this task as superuser on the workstation whose domain name you will change.

Preqrequisities

- The workstation's superuser password
- The new domain name

Procedure

1. Log in to the workstation and become superuser.

   The examples in this task use **client1** as the workstation and Wiz.Com. as the new domain name.

   ```
   client1% su
   Password:
   ```

2. Change the workstation's domain name.

   Type the new name with the **domainname** command. Do not use a trailing dot.

   ```
   client1# domainname Wiz.Com
   ```

   If the workstation was an NIS client, it may no longer be able to get NIS service.

3. Verify the result.

   Run the **domainname** command again, this time without an argument, to display the server's current domain.

   ```
   client1# domainname
   wiz.com.
   ```

4. Save the new domain name.

   ```
   client1# chypdom –I Wiz.Com.
   ```

5. At a convenient time, reboot the workstation.

   Even after entering the new domain name into the /**etc/rc.nfs** file, some processes may still operate with the old domain name. To ensure that all processes are using the new domain name, reboot the workstation.

   Since you may be performing this task in a sequence of many other tasks, examine the work remaining to be done on the workstation before rebooting. Otherwise, you might find yourself rebooting several times instead of just once.

Initializing an NIS+ Client

There are three different ways to initialize a NIS+ client:

1. Broadcast method (see Broadcast Initialization, on page 5-63)
2. Host–name method (see Host–Name Initialization, on page 5-64)
3. Cold–start file method (see Cold–Start File Initialization, on page 5-64)

Broadcast Initialization

This method **initializes** an NIS+ client by sending an IP broadcast on the client's subnet.

This is the simplest way to set up a client but is also the least secure. The NIS+ server that responds to the broadcast sends the client all the information that the client needs in its cold–start file, including the server's public key. Presumably, only an NIS+ server will respond to the broadcast. However, the client has no way of knowing whether the workstation that responded to the broadcast is indeed a trusted server. As a result, this method is only recommended for sites with small, secure networks.

Security Considerations

You must perform this task as superuser on the client.

Prerequisites

At least one NIS+ server must exist on the same subnet as the client.

Information You Need

You need the superuser password to the client.

Procedure

Initialize the client.

This step initializes the client and creates a **NIS_COLD_START** file in its /**var**/**nis** directory. Use the **nisinit** command with the **–c** and **–B** options.

```
client1# nisinit -c -B
This machine is in the Wiz.Com. NIS+ domain.
Setting up NIS+ client ...
All done.
```

An NIS+ server on the same subnet will reply to the broadcast and add its location information into the client's cold–start file.

Host–Name Initialization

Initializing a client by host name consists of explicitly identifying the IP address of its trusted server. This server's name, location information, and public keys are then placed in the client's cold–start file.

This method is more secure than the broadcast method because it actually specifies the IP address of the trusted server, rather than relying on a server to identify itself. However, if a router exists between the client and the trusted server, it could intercept messages to the trusted IP address and route them to an untrusted server.

Security Considerations

You must perform this operation as superuser on the client.

Prerequisites

- The NIS+ service must be running in the client's domain.
- The client must have an entry in its /**etc**/**hosts** file for the trusted server.

You need the name and IP address of the trusted server.

Procedure

1. Check the client's /**etc**/**hosts** file.

   Make sure the client has an entry for the trusted server.

2. Initialize the client.

   This step initializes the client and creates a **NIS_COLD_START** file in its /**var**/**nis** directory. Use the **nisinit** command with the **–c** and **–H** options. This example uses **rootmaster** as the trusted server.

   ```
   Client1# nisinit -c -H rootmaster
   This machine is in the Wiz.Com. NIS+ domain.
   Setting up NIS+ client ...
   All done.
   ```

   The **nisinit** utility looks for the server's address in the client's /**etc**/**hosts** file, so do not append a domain name to the server. If you do, the utility will not be able to find its address.

Cold–Start File Initialization

This task initializes an NIS+ client by using the cold–start file of another NIS+ client, preferably one from the same domain. This is the most secure method of setting up an NIS+ client. It ensures that the client obtains its NIS+ information from a trusted server, something that cannot be guaranteed by the host–name or broadcast method.

Security Considerations

You must perform this task as superuser on the client.

Prerequisites

The servers specified in the cold–start file must already be set up and running NIS+.

You need the name and location of the cold–start file you will copy.

Procedure

1. Copy the other client's cold–start file.

   Copy the other client's cold–start file into a directory in the new client. This may be easier to do while logged on as yourself rather than as superuser on the client. Be sure to switch back to superuser before initializing the client.

   Do not copy the **NIS_COLD_START** file into /**var**/**nis**, because that file gets overwritten during initialization. This example copies the cold–start file of previously initialized **client1** into the /**tmp** directory of uninitialized **client2**.

   ```
   client2# exit
   client2% rcp client1:/var/nis/NIS_COLD_START /tmp
   client2% su
   ```

2. Initialize the client from the cold–start file.

   Use the **nisinit** command with the **–c** and **–C** options.

   ```
   client2# nisinit –c –C /tmp/NIS_COLD_START
   This machine is in the Wiz.Com. NIS+ domain.
   Setting up NIS+ client ...
   All done.
   ```

NIS+ Client Setup Summary

The following table shows a summary of the steps required to set up a client. It assumes the simplest case, so be sure you are familiar with the more thorough task descriptions before you use this summary as a reference. For the sake of brevity, this summary does not show the responses to each command.

Table 28. Setting Up a Client: Command Summary

Tasks	Commands
Log in to domain's master.	`rootmaster%`

Create DES credentials for client.	`rootmaster% nisaddcred \`
	`  –p unix.client1.Wiz.Com \`
	`  –P client1.Wiz.Com. des`

Log in, as superuser, to the client.	`client1% su`
	`Password:`

Assign the client a domain name.	`client1# domainname Wiz.Com`
	`client1# chypdom –I Wiz.Com.`

Check the configuration file.	`client# view /etc/irs.conf`
Clean out /**var**/**nis**.	`client1# rm –rf /var/nis/*`
Initialize the client.	`client1# nisinit –c –H rootmaster`

Kill and restart the keyserver.	`client1# stopsrc –s keyserv`
	`client1# startsrc –s keyserv`

Run **keylogin** on the client.	`client1# keylogin –r`
	`Password:`

Reboot the client.	`client1# shutdown –Fr`

Client Setup

This section describes how to set up a typical NIS+ client in either the root domain or in a non–root domain. This procedure applies to regular NIS+ clients and to those clients that will later become NIS+ servers. It applies, as well, to clients in a standard NIS+ domain and those in an NIS–compatible domain.

Attention: Domains and hosts should not have the same name. For example, if you have a **sales** domain you should not have a machine named **sales**. Similarly, if you have a machine named **home**, you do not want to create a domain named home. This caution applies to subdomains; for example, if you have a machine named **west** you do not want to create a sales.west.myco.com subdirectory.

Setting up an NIS+ client involves the following tasks:

- Creating credentials for the client
- Preparing the workstation
- Initializing the workstation as an NIS+ client.

However, as with setting up the root domain, setting up a client is not as simple as carrying out these three tasks in order. To make the setup process easier to execute, these tasks have been broken down into individual steps, and the steps have been arranged in the most efficient order:

1. Logging in to the domain's master server
2. Creating DES credentials for the new client workstation
3. Logging in as superuser to the client
4. Assigning the client its new domain name
5. Cleaning out leftover NIS+ material and processes.
6. Initializing the client.
7. Killing and restarting the **keyserv** daemon.
8. Running **keylogin**.
9. Rebooting the client.

Security Considerations

Setting up a client has two main security requirements: both the administrator and the client must have the proper credentials and access rights. Otherwise, the only way for a client to obtain credentials in a domain running at security level 2 is for them to be created by an administrator who has valid DES credentials and modify rights to the cred table in the client's home domain. The administrator can either have DES credentials in the client's home domain or in the administrator's home domain.

Once an administrator creates the client's credentials, the client can complete the setup process. However, the client still needs read access to the directory object of its home domain. If you set up the client's home domain according to the instructions in either Setting Up the Root Domain, on page 5-32, or Setting Up a Nonroot Domain, on page 5-57, read access was provided to the world class by the NIS+ commands used to create the directory objects (**nisinit** and **nismkdir**, respectively).

You can check the directory object's access rights by using the **niscat –o** command. This command displays the properties of the directory, including its access rights:

```
rootmaster# niscat –o Wiz.Com.
ObjectName    : wiz
Owner         : rootmaster.wiz.com.
Group         : admin.wiz.com.
Access Rights : r---rmcdr---r---
.
.
.
```

You can change the directory object's access rights, provided you have modify rights to it yourself, by using the **nischmod** command, described in Administering NIS+ Access Rights, on page 6-21.

Prerequisites

The administrator setting up the client's credentials must have:

- A valid DES credential
- Modify rights to the cred table in the client's home domain

The client must have:

- Read rights to the directory object of its home domain
- The client's home domain must already be set up and running NIS+
- An entry in either the master server's /**etc**/**hosts** file or in its domain's hosts table
- A unique machine name that does not duplicate any user ID
- A machine name that does not contain any dots. (For example, a machine named **sales.alpha** is not allowed; a machine named **sales–alpha** is allowed)
- The name of the client's home domain
- The superuser password of the workstation that will become the client
- The IP address of an NIS+ server in the client's home domain

Procedure

1. Log into the domain's master server.

   You can log in as superuser or as yourself, depending on which NIS+ principal has the proper access rights to add credentials to the domain's cred table.

2. Create DES credentials for the new client workstation.

   Use the **nisaddcred** command with the **–p** and **–P** arguments. Here is the syntax:

   ```
   nisaddcred –p secure–RPC–netname –P principal–name des [domain]
   ```

   The **secure–RPC–netname** consists of the prefix **unix** followed by the client's host name, the symbol @ and the client's domain name, but without a trailing dot. The **principal–name** consists of the client's host name and domain name, with a trailing dot. If the client belongs to a different domain than the server from which you enter the command, append the client's domain name after the second argument.

   This example adds a DES credential for a client workstation named **client1** in the Wiz.Com. domain:

   ```
   rootmaster% nisaddcred –p unix.client1@Wiz.Com –P client1.Wiz.Com. des
   Adding key pair for unix.client1@Wiz.Com (client1.Wiz.Com.).
   Enter client1.Wiz.Com.'s root login passwd:
   Retype password:
   ```

   For more information about the **nisaddcred** command, see Administering NIS+ Credentials, on page 6-2.

3. Log in as superuser to the client.

   Now that the client workstation has credentials, you can log out of the master server and begin working from the client itself. You can do this locally or remotely.

4. Assign the client its new domain name.

   There are three ways to assign a new domain name to a client. Those methods are described in Changing a Workstation's Domain, on page 5-62. Use one of those methods to change the client's domain name and then return to the following step.

5. Clean out leftover NIS+ material and processes.

   If the workstation you are working on was previously used as an NIS+ server or client, remove any files that might exist in **/var/nis** and kill the cache manager, if it is still running. In this example, a cold–start file and a directory cache file still exist in **/var/nis**.

   ```
   client1# ls /var/nis
   NIS_COLD_START     NIS_SHARED_CACHE
   client1# rm –rf /var/nis/*
   client1# stopsrc –s nis_cachemgr
   ```

   This step makes sure that files left in **/var/nis** or directory objects stored by the cache manager are completely erased so that they do not conflict with the new information generated during this setup process. If you have stored any admin scripts in **/var/nis**, you may want to consider temporarily storing them elsewhere, until you finish setting up the root domain.

6. Initialize the client.

   Initialize a client in one of three different ways: by host name, by cold–start file, or by broadcast (see Initializing an NIS+ Client, on page 5-63).

7. Kill and restart the **keyserv** daemon.

   This step stores the client's secret key on the keyserver.

   a. Kill the **keyserv** daemon.

      ```
      stopsrc –s keyserv
      ```

      This also has the side effect of updating the key server's switch information about the client.

   b. Remove the **/etc/.rootkey** file.

   c. Restart the keyserver.

      ```
      startsrc –s keyserv
      ```

8. Run **keylogin –r**.

   This step stores the client's secret key with the keyserver. It also saves a copy in **/etc/.rootkey**, so that the superuser on the client does not have to run **keylogin** to use NIS+. Use **keylogin** with the **–r** option. When prompted for a password, type the client's superuser password. It must be the same as the password supplied to create the client's DES credentials:

   ```
   client1# keylogin –r
   Password:
   Wrote secret key into /etc/.rootkey
   ```

9. Reboot the client.

Chapter 6. NIS+ Administration

This chapter describes the system administration tasks for NIS+. Topics are:

Administering NIS+ Credentials

This section assumes that you have an adequate understanding of the NIS+ security system in general, and in particular of the role that credentials play in that system (see the Security, on page 7-1 chapter for this information)

This section provides the following general information about credentials.

- How Credentials Work, on page 6-2
- Credential versus Credential Information, on page 6-2
- Authentication Components, on page 6-3
- How Principals are Authenticated, on page 6-3
- The DES Credential in Detail, on page 6-5
- Where Credential–Related Information Is Stored, on page 6-8
- The Cred Table in Detail, on page 6-9

This section then describes how to use the NIS+ credential administration commands to perform the following tasks.

- Creating Credential Information, on page 6-10
- The nisaddcred Command, on page 6-10
- How nisaddcred Creates Credential Information, on page 6-10
- Secure RPC Netname and NIS+ Principal Name, on page 6-11
- Creating Credential Information for the Administrator, on page 6-12
- Creating Credential Information for NIS+ Principals, on page 6-12
- Updating Your Own Credential Information, on page 6-14
- Removing Credential Information, on page 6-14

How Credentials Work

This section describes how the credential and authentication process works.

The credential/authentication system prevents someone from assuming some other user's identity. That is, it prevents someone with root privileges on one machine from using the **su** command to assume the identity of a second user who is either not logged in at all or logged in on another machine and then accessing NIS+ objects with the second user's NIS+ access privileges.

Note: NIS+ cannot prevent someone who knows another user's login password from assuming that other user's identity and the other user's NIS+ access privileges, nor can NIS+ prevent a user with root privileges from assuming the identity of another user who is currently logged in on the **same** machine.

See Security, on page 7-1 for a description of how NIS+ credentials and authentication work with authorization and access rights to provide security for the NIS+ namespace.

Credential versus Credential Information

To understand how DES credentials are created and how they work, you need to distinguish between the credential itself and the information that is used to create and verify it.

- **Credential information:** The data that is used to generate a DES credential and by the server to verify that credential.

- **DES credential**: The bundle of numbers that is sent by the principal to the server to authenticate the principal. A principal's credential is generated and verified each time the principal makes an NIS+ request. See The DES Credential in Detail, on page 6-5 for a detailed description of the DES credential.

Authentication Components

For the credential/authentication process to work, the following components must be in place:

- **Principal's DES credential information**. This information is initially created by an NIS+ administrator for each principal. It is stored in the cred table of the principal's home domain. A principal's DES credential information consists of:
  - **Principal name.** This would be a user's fully qualified login ID or a machine's fully qualified host name.
  - **Principal's Secure RPC netname**. Each principal has a unique Secure RPC netname. (See DES Credential Secure RPC Netname, on page 6-5 for more information on Secure RPC netnames.)
  - **Principal's public key**.
  - **Principal's encrypted private key**.
- **Principal's LOCAL credential.**
- **Server's public keys**. Each directory object stores copies of the public keys of all the servers in that domain. Note that each server's DES credentials are also stored in the cred table.
- **Keyserver copy of principal's private key**. The keyserver has a copy of the private key of the principal that is currently logged in (user or machine).

How Principals are Authenticated

There are three phases to the authorization process:

- **Preparation phase.** This consists of the setup work performed by an NIS+ administrator prior to the user logging in; for example, creating credential information for the user.
- **Login phase**. This consists of the actions taken by the system when a user logs in.
- **Request phase**. This consists of the actions taken by the software when a NIS+ principal makes a request for a NIS+ service or access to a NIS+ object.

These three phases are described in detail in the following subsections.

Credentials Preparation Phase

Prior to an NIS+ principal logging in, an NIS+ administrator must create DES credential information for that principal (user or machine). (The easiest way for an NIS+ administrator to create credential information for users is to use the nisclient script as described in the **nisclient** command description.) The administrator must:

- Create a public key and an encrypted private key for each principal. These keys are stored in the principal's home domain cred table. This can be done with the **nisaddcred** command as described in Creating Credential Information for NIS+ Principals, on page 6-12.
- Create server public keys. (See Updating Public Keys, on page 6-20.)

Login Phase — Detailed Description

When a principal logs into the system the following steps are automatically performed:

1. The **keylogin** program is run for the principal. The **keylogin** program gets the principal's encrypted private key from the cred table and decrypts it using the principal's login password.

Attention: When a principal's login password is different from his or her Secure RPC password, **keylogin** cannot decrypt it and the user starts getting "cannot decrypt" errors or the command fails without a message. For a discussion of this problem, see Secure RPC Password versus Login Password Problem, on page 6-7.

2. The principal's decrypted private key is passed to the keyserver which stores it for use during the request phase.

   Note: The decrypted private key remains stored for use by the keyserver until the user does an explicit **keylogout**. If the user simply logs out (or goes home for the day without logging out), the decrypted private key remains stored in the server. If someone with root privileges on a user's machine switched to the user's login ID, that person would then have use of the user's decrypted private key and could access NIS+ objects using the user's access authorization. Thus, for added security, users should be cautioned to perform an **explicit keylogout** when they cease work. If they also log out of the system, all they need do is log back in when they return.

Request Phase — Detailed Description

Every time a NIS+ principal requests access to an NIS+ object, the NIS+ software performs a multistep process to authenticate that principal:

1. NIS+ checks the cred table of the object's domain. If:
   - The principal has LOCAL credential information, NIS+ uses the domain information contained in the LOCAL credential to find the principal's home domain cred table where it obtains the information used later.
   - The principal has no credential information, the rest of the process is aborted and the principal is given the authorization access class of nobody.

   NIS+ gets the user's DES credential from the cred table of the user's home domain. The encrypted private key is decrypted with the user's password and saved by the keyserver.
2. NIS+ obtains the server's public key from the NIS+ directory object.
3. The keyserver takes the principal's decrypted private key and the public key of the object's server (the server where the object is stored) and uses them to create a **common key**.
4. The common key is then used to generate an encrypted **DES key**. To do this, Secure RPC generates a random number which is then encrypted using the common key. For this reason, the DES key is sometimes referred to as the **random key** or the **random DES key**.
5. NIS+ then takes the current time of the principal's server and creates a time stamp that is encrypted using the DES key.
6. NIS+ then creates a 15–second window, which is encrypted with the DES key. This **window** is the maximum amount of time that is permitted between the time stamp and the server's internal clock.
7. NIS+ then forms the principal's DES credential, which is composed of the following:
   - The principal's Secure RPC netname (**unix.** identifier@domain) from the principal's cred table (see DES Credential Secure RPC Netname, on page 6-5 for more detail on the netname).
   - The principal's encrypted DES key from the keyserver
   - The encrypted time stamp
   - The encrypted window
8. NIS+ then passes the following information to the server where the NIS+ object is stored:
   - The access request (whatever it might be)
   - The principal's DES credential

– Window verifier (encrypted), which is the encrypted window plus one

9. The object's server receives this information.
10. The object's server uses the Secure RPC netname portion of the credential to look up the principal's public key in the cred table of the principal's home domain.
11. The server then uses the principal's public key and the server's private key to regenerate the common key. This common key must match the common key that was generated by the principal's private key and the server's public key.
12. The common key is used to decrypt the DES key that arrived as part of the principal's credential.
13. The server decrypts the principal's time stamp with the newly decrypted DES key and verifies it with the window verifier.
14. The server then compares the decrypted and verified time stamp with the server's current time and proceeds as follows:
    a. If the time difference at the server **exceeds** the window limit, the request is denied and the process aborts with an error message. For example, suppose the time stamp is 9:00am and the window is one minute. If the request is received and decrypted by the server after 9:01am, it is denied.

       If the time stamp is within the window limit, the server checks to see if the time stamp is **greater** than the one previously received from the principal. This ensures that NIS+ requests are handled in the correct order.

    b. Requests received out of order are rejected with an error message. For example, if the time stamp is 9:00am and the most recently received request from this principal had a time stamp of 9:02am, the request would be rejected.

       Requests that have a time stamp equal to the previous one are rejected with an error message. This ensures that a replayed request is not acted on twice. For example, if the time stamp is 9:00am and the most recently received request from this principal also had a time stamp of 9:00am, this request would be rejected.

       If the time stamp is within the window limit, and greater than the previous request from that principal, the server accepts the request.
15. The server then complies with the request and stores the time stamp from this principal as the most recently received and acted on request.
16. To confirm to the principal that the information received from the server in answer to the request comes from a trusted server, the server encrypts the time stamp with the principal's DES key and sends it back to the principal along with the data.
17. At the principal's end, the returned time stamp is decrypted with the principal's DES key.
    – If the decryption succeeds, the information from the server is returned to the requester.
    – If the decryption fails for some reason, an error message is displayed.

The DES Credential in Detail

The DES credential consists of:

- The principal's **Secure RPC netname** (see Secure RPC Netname below).
- A **verification** field (see Verification Field below).

DES Credential Secure RPC Netname

This portion of the credential is used to identify the NIS+ principal. (Remember that an NIS+ principal name *always* has a trailing dot, while a Secure RPC netname *never* does.) Every Secure RPC netname contains three components:

Prefix	The prefix is always the word **unix**.

Identifier	If the principal is a client user, the ID field is the user's UID. If the principal is a client workstation, the ID field is the workstation's hostname.
Domain name	The domain name is the name of the domain that contains the principal's DES credential (in other words, the principal's home domain).

Table 29. Secure RPC Netname Format

Principal	**Prefix**	**Identifier**	**Domain**	**Example**
User	unix	UID	Domain containing user's password entry and the DES credential itself	unix.24601@sales.wiz.com
Workstation	unix	hostname	The domain name returned by executing the **domainname** command on that workstation	unix.machine7@sales.wiz.com

Verification Field

The verification field is used to make sure the credential is not forged. It is generated from the credential information stored in the cred table.

The verification field is composed of:

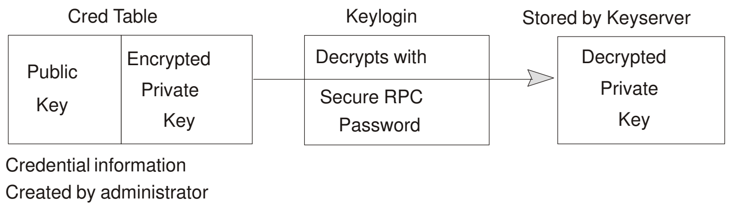
- The principal's encrypted DES key, generated from the principal's private key and the NIS+ server's public key
- The encrypted time stamp
- The time window

How the DES Credential Is Generated

To generate its DES credential, the principal depends on the **keylogin** command, which must have been executed **before** the principal tries to generate its credential. The **keylogin** command (often referred to simply as a **keylogin**) is executed automatically when an NIS+ principal logs in.

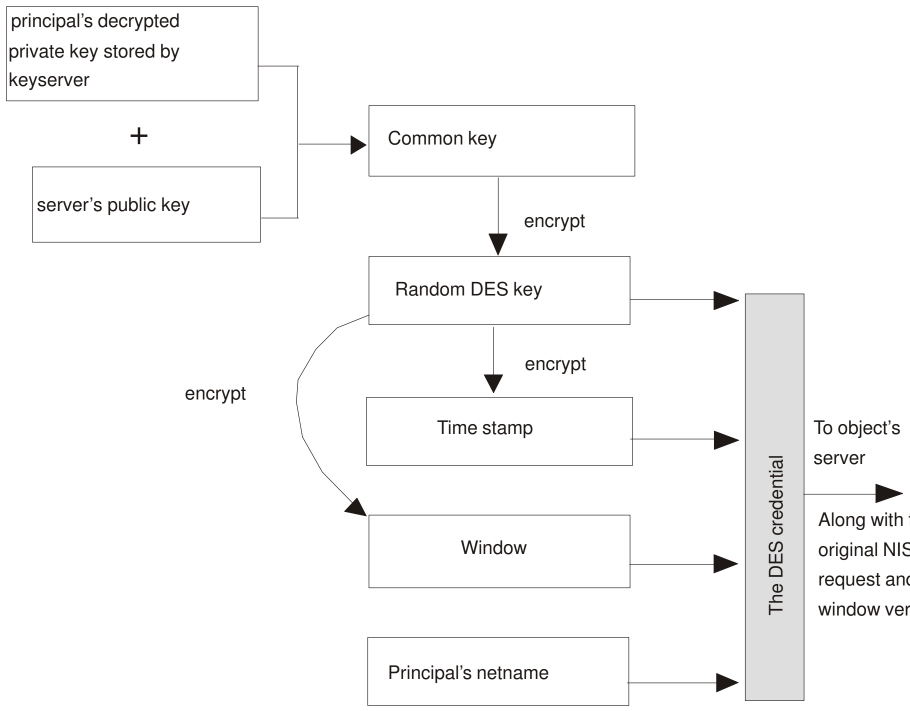
Note: If the principal's login password is different from the principal's Secure RPC password, a successful keylogin cannot be performed. See Secure RPC Password versus Login Password Problem, on page 6-7 for a discussion of this situation.

The purpose of the keylogin is to give the principal access to the principal's private key. **keylogin** fetches the principal's private key from the cred table, decrypts it with the principal's **Secure RPC password** (remember that the private key was originally encrypted with the principal's Secure RPC password), and stores it locally with the keyserver for future NIS+ requests. See figure for more information.



To generate its DES credential, the principal still needs the public key of the server to which it will send the request. This information is stored in the principal's directory object. Once the principal has this information, it can form the verification field of the credential.

First, the principal generates a random DES key for encrypting various credential information. The principal uses its own private key (stored in the keyserver) and the server's public key to generate a common key that is used to generate and encrypt the random DES key. It then generates a time stamp that is encrypted with the DES key and combines it with other credential–related information into the verification field. See figure for more information.



Secure RPC Password versus Login Password Problem

When a principal's login password is different from his or her Secure RPC password, **keylogin** cannot decrypt it at login time because **keylogin** defaults to using the principal's login password, and the private key was encrypted using the principal's Secure RPC password.

When this occurs, the principal can log in to the system, but for NIS+ purposes the principal is placed in the authorization class of nobody because the keyserver does not have a decrypted private key for that user. Since most NIS+ environments are set up to deny the nobody class create, destroy, and modify rights to most NIS+ objects, this results in "permission denied" errors when the user tries to access NIS+ objects.

To be placed in one of the other authorization classes, a user in this situation must explicitly run the **keylogin** program and give the principal's Secure RPC password when **keylogin** prompts for a password. (Note that in this context, *network password* is sometimes used as a synonym for Secure RPC password. When prompted for your network password, type your Secure RPC password.) (See the **keylogin** section, on page 6-16.)

But an explicit **keylogin** provides only a temporary solution that is good only for the current login session. The keyserver now has a decrypted private key for the user, but the private key in the user's cred table is still encrypted using the user's Secure RPC password, which is different than the user's login password. The next time the user logs in, the same problem recurs. To permanently solve the problem the user needs to re–encrypt the private key in the cred table to one based on the user's login ID rather than the user's Secure RPC password. To do this, the user needs to run **chkey –p** as described in Changing Keys for a NIS+ Principal, on page 6-16.

Thus, to permanently solve problems related to a difference in Secure RPC password and login password, the user (or an administrator acting for the user) must perform these steps:

1. Login using the login password.

2. Run the **keylogin** program to temporarily get a decrypted private key stored in the keyserver and thus gain temporary NIS+ access privileges.
3. Run **chkey –p** to permanently change the encrypted private key in the cred table to one based on the user's login password.
4. When you are ready to finish this login session, run **keylogout**.
5. Log off the system with **logout**.

Cached Public Keys Problems

Occasionally, you may find that even though you have created the proper credentials and assigned the proper access rights, some principal requests still get denied. The most common cause of this problem is the existence of stale objects with old versions of a server's public key. You can usually correct this problem by:

- Running **nisupdkeys** on the domain you are trying to access. (See **nisupdkeys** command description and Stale and Outdated Credential Information, on page 8-16 for information on how to correct this type of problem.)
- Killing the **nis_cachemgr** on your machine, removing **/var/nis/NIS_SHARED_DIRCACHE**, and then restarting **nis_cachemgr**.

Where Credential–Related Information Is Stored

This section describes where credential–related information is stored throughout the NIS+ namespace.

Credential–related information, such as public keys, is stored in many locations throughout the namespace. NIS+ updates this information periodically, depending on the time–to–live values of the objects that store it, but sometimes, between updates, it gets out of sync. As a result, you may find that operations that should work, do not. The following table lists all the objects, tables, and files that store credential–related information and how to reset it.

Table 30. Where Credential–Related Information Is Stored

Item	**Stores**	**To Reset or Change**
cred table	NIS+ principal's public key and private key. These are the master copies of these keys.	Use **nisaddcred** to create new credentials; it updates existing credentials. An alternative is **chkey**.
directory object	A copy of the public key of each server that supports it.	Run the **/usr/lib/nis/nisupdkeys** command on the directory object.
keyserver	The secret key of the NIS+ principal that is currently logged in.	Run **keylogin** for a principal user or **keylogin –r** for a principal workstation.

NIS+ daemon	Copies of directory objects, which in turn contain copies of their servers' public keys.	Kill the **rpc.nisd** daemon and the cache manager:
		`stopsrc –s rpc.nisd`
		`stopsrc –s nis_cachemgr`
		then remove **NIS_SHARED_DIRCACHE** from **/var/nis**. Then restart both.
		`startsrc –s rpc.nisd`
		`startsrc –s nis_cachemgr`

Item	Stores	To Reset or Change
Directory cache	A copy of directory objects, which in turn contain copies of their servers' public keys.	Kill the NIS+ cache manager and restart it. The **–i** option resets the directory cache from the cold–start file and restarts the cache manager. `stopsrc –s nis_cachemgr` `startsrc –s nis_cachemgr –a "–i"`
cold–start file	A copy of a directory object, which in turn contains copies of its servers' public keys.	On the root master, kill the NIS+ daemon and restart it. The daemon reloads new information into the existing **NIS_COLD_START** file. On a client workstation, first remove the **NIS_COLD_START** and **NIS_SHARED_DIRCACHE** files from /**var**/**nis**, and use **stopsrc –s nis_cachemgr** to kill the cache manager. Then re–initialize the principal with **nisinit –c**. The principal's trusted server reloads new information into the workstation's **NIS_COLD_START** file. Start the **nis_cachemgr** using **startsrc –s nis_cachemgr**.
passwd table	A user's password.	Use the **passwd** command. It changes the password in the NIS+ passwd table and updates it in the cred table.
passwd file	A user's password or a workstation's superuser password.	Use the **passwd** command, whether logged in as super user or as yourself, whichever is appropriate.

The Cred Table in Detail

Credential information for principals is stored in a **cred table**. The cred table is one of the standard NIS+ tables. Each domain has one cred table, which stores the credential information of client workstations that belong to that domain and client users who are allowed to log into them. (In other words, the principals of that domain.) The cred tables are located in their domains' **org_dir** subdirectory.

Attention: Never link a cred table. NIS+ does not operate correctly with linked cred tables. Each **org_dir** directory should have its own cred table. Do not use a link to some other **org_dir** cred table.

For users, the cred table stores LOCAL credential information for all users who are allowed to log into any of the machines in the domain. The cred table also stores DES credential information for those users that have the domain as their home domain.

You can view the contents of a cred table with the **niscat** command, described in Administering NIS+ Tables, on page 6-60.

The **cred** table, as shown in the following table, has five columns:

Table 31. Cred Table Credential Information

	NIS+ Principal Name	Authentication Type	Authentication Name	Public Data	Private Data
Column Name	cname	auth_type	auth_name	public_data	private_data
User	Fully qualified principal name	LOCAL	UID	GID list	
Machine	Fully qualified principal name	DES	Secure RPC netname	Public key	Encrypted Private key

The Authentication Type column, determines the types of values found in the other four columns.

- **LOCAL**. If the authentication type is LOCAL, the other columns contain a principal user's name, UID, and GID; the last column is empty.
- **DES**. If the authentication type is DES, the other columns contain a principal's name, Secure RPC netname, public key, and encrypted private key. These keys are used in conjunction with other information to encrypt and decrypt a DES credential.

Creating Credential Information

There are several methods of creating and administering credential information:

- Use Web-based System Manager or SMIT. They provide easier methods of credential administration and are recommended for administering individual credentials.
- Use the **nisclient** script. This is another easy method of creating or altering credentials for a single principal. Because of its convenience, this is a recommended method of administering individual credentials. NIS+ Setup Scripts gives step by step instructions on using the **nisclient** script to create credential information.
- Use the **nispopulate** script. This is an easy method of creating or altering credentials for a one or more principals who already have information on them stored in NIS maps or /**etc** files. Because of its convenience, this is a recommended method of administering credentials for groups of NIS+ principals. NIS+ Setup Scripts of gives step by step instructions on using the **nispopulate** script to create credential information.
- Use the **nisaddcred** command. The section below describes how credentials and credential information are created using **nisaddcred**.

How nisaddcred Creates Credential Information

When used to create LOCAL credential information, **nisaddcred** simply extracts the principal user's UID (and GID) from the principal's login record and places it in the domain's cred table.

When used to create DES credential information, **nisaddcred** goes through a two–part process:

1. Forming the principal's Secure RPC netname. A Secure RPC netname is formed by taking the principal's user ID number from the password record, attaching the `unix` prefix to it, and combining it with the domain name (**unix.1050@wiz.com**, for example).

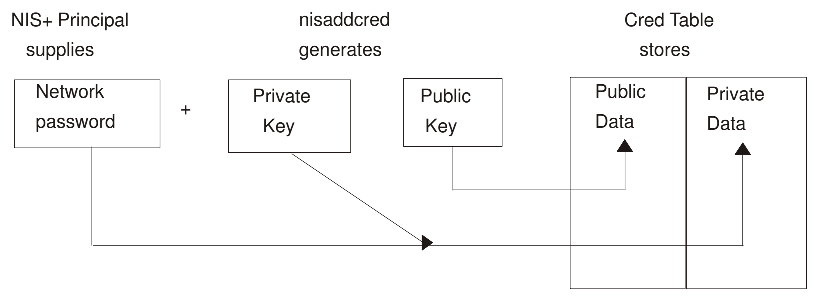
2. Generating the principal's private and public keys.

   To encrypt the private key, **nisaddcred** needs the principal's Secure RPC password. When the **nisaddcred** command is invoked with the **des** argument, it prompts the principal for a Secure RPC password. Normally, this password is the same as the principal's login password. (If it is different, the user will have to perform additional steps when logging in, as described in Secure RPC Password versus Login Password Problem, on page 6-7.)

The **nisaddcred** command generates a pair of random, but mathematically related 192–bit authentication keys using the Diffie–Hellman cryptography scheme. These keys are called the Diffie–Hellman key–pair, or simply **key–pair** for short.

One of these is the **private key**, and the other is the **public key**. The public key is placed in the public data field of the cred table. The private key is placed in the private data field, but only after being encrypted with the principal's Secure RPC password. See figure.

nisaddcred:



The principal's private key is encrypted as a security precaution because the cred table, by default, is readable by all NIS+ principals, even unauthenticated ones.

Secure RPC Netname and NIS+ Principal Name

When creating credential information, you will often have to enter a principal's **SecureRPC–netname** and **principal–name**. Each has its own syntax:

Secure RPC netname
A Secure RPC netname is a name whose syntax is determined by the Secure RPC protocol. Therefore, it does not follow NIS+ naming conventions:

- For users, the syntax is: **unix**.uid@ domain
- For machines, the syntax is: **unix**.hostname@ domain

If a Secure RPC netname identifies a user, it requires the user's UID. If it identifies a workstation, it requires the workstation's host name. (When used with the **nisaddcred** command it is always preceded by the **–p** (lowercase) flag.)

A Secure RPC netname always begins with the **unix** (all lowercase) prefix and ends with a domain name. However, because it follows the Secure RPC protocol, the domain name **does not** contain a trailing dot.

Principal name An NIS+ principal follows the normal NIS+ naming conventions, but it must always be fully qualified. the syntax is: principal.domain.

Whether it identifies a client user or a client workstation, it begins with the principal's **name**, followed by a dot and the complete domain name, ending in a dot. (When used with **nisaddcred** to create credential information, it is always preceded by the **–P** (uppercase) flag. When used to remove credential information, it does not use the **–P** flag.)

Creating Credential Information for the Administrator

When a namespace is first set up, credential information is created first for the administrators who will support the domain. Once they have credential information, they can create credential information for other administrators, client workstations, and client users.

When you try to create your own credential information, you run into a problem of circularity: you cannot create your own credential information unless you have Create rights to your domain's cred table, but if the NIS+ environment is properly set up, you cannot have such rights until you have credentials. You have to step out of the loop somehow. You can do this in one of two ways:

- By creating your credential information while logged in as superuser to your domain's master server
- By having another administrator create your credential information using a dummy password, then changing your password with the **chkey** command.

In either case, your credential information is thus created by another NIS+ principal. To create your own credential information, follow the instructions in Creating Credential Information for NIS+ Principals, on page 6-12.

Creating Credential Information for NIS+ Principals

Credential information for NIS+ principals can be created any time after their domain has been set up; in other words, once a cred table exists.

To create credential information for an NIS+ principal:

- You must have Create rights to the cred table of the principal's home domain.
- The principal must be recognized by the server. This means that:
  - If the principal is a user, the principal must have an entry either in the domain's NIS+ passwd table or in the server's /**etc**/**passwd** file.
  - If the principal is a workstation, it must have an entry either in the domain's NIS+ Hosts table or in the server's /**etc**/**hosts** file.

Once those conditions are met, you can use the **nisaddcred** command with both the **–p** and **–P** options:

For LOCAL credentials:

```
nisaddcred -p uid -P principal-name local
```

For DES credentials:

```
nisaddcred -p  SecureRPC-netname -P  principal-name des
```

Remember these principles:

- You can create both LOCAL and DES credential information for a principal user.
- You can only create DES credential information for a principal workstation.
- You can create DES credential information only in the principal's home domain (user or machine).
- You can create LOCAL credential information for a user in both the user's home domain and in other domains.

The following example creates both LOCAL and DES credential information for an NIS+ user named morena who has a UID of 11177. She belongs to the sales.wiz.com. domain, so this example enters her credential information from a principal machine of that domain:

```
salesclient# nisaddcred -p 11177 -P morena.sales.wiz.com. local
salesclient# nisaddcred -p unix.11177@sales.wiz.com -P
morena.sales.wiz.com. des
Adding key pair for unix.11177@sales.wiz.com (morena.sales.wiz.com.).
Enter login password:
```

The proper response to the **Enter login password:** prompt is morena's login password. If you do not know her login password, you can use a dummy password that she can later change using **chkey**. The following table shows how another administrator, whose credential information you create using a dummy password, can then use **chkey** to change his or her own password. In this example, you create credential information for an administrator named `eiji` who has a UID of 119. `eiji` belongs to the root domain, so you would enter his credential information from the root master server which is named **rmaster**.

Table 32. Creating Administrator Credentials: Command Summary

Tasks	**Commands**
Create LOCAL credential information for eiji.	`rmaster#` **`nisaddcred \ –p 119 –P eiji.wiz.com. local`**
Create DES credential information for eiji.	`rmaster#` **`nisaddcred \ –p unix.119@wiz.com \ –P eiji.wiz.com. des`** `Adding key pair for unix.119@wiz.com (eiji.wiz.com.).`
Type dummy password for eiji.	`Enter eiji's login password:`
Re–enter dummy password.	`nisaddcred: WARNING: password differs from login passwd. Retype password:`
You tell eiji the dummy password that you used.	
eiji logs into rmaster.	`rmaster login:` **`eiji`**
eiji enters real login password.	`Password:`
eiji gets error message but is allowed to log in anyway.	`Password does not decrypt secret key for unix.119@wiz.com.`
eiji runs keylogin.	`rmaster%` **`keylogin`**
eiji types dummy password.	`Password:` *`dummy–password`*
eiji runs **chkey –p**.	`rmaster%` **`chkey –p`** `Updating nisplus publickey database Generating new key for 'unix.119@wiz.com'.`
eiji types real login password.	`Enter login password:`
eiji re–types real login password.	`Retype password: Done.`

First, you would create eiji's credential information in the usual way, but using a dummy login password. NIS+ would warn you and ask you to re–type it. When you did, the operation would be complete. The domain's cred table would contain eiji's credential information based on the dummy password. The domain's passwd table (or /**etc**/**passwd** file), however, would still have his login password entry so that he can log on to the system.

Then, eiji would log in to the domain's master server, typing his **correct** login password (since the login procedure checks the password entry in the passwd table or /**etc**/**passwd** file). From there, eiji would first run **keylogin**, using the dummy password (since a **keylogin** checks the cred table), and then use the **chkey –p** command to change the cred entry to the real thing.

The two previous examples created credential information for a principal user while the principal user was logged in to the master server of the principal's home domain. However, if you have the proper access rights, you can create credential information in another domain. Simply append the domain name to this syntax:

For LOCAL credentials:

```
nisaddcred –p uid –P principal-name local domain-name
```

For DES credentials:

```
  nisaddcred –p SecureRPC-netname –P principal-name des domain-name
```

The following example first creates LOCAL and DES credential information for an administrator named chou in her home domain, which happens to be the root domain, then adds her LOCAL credential information to the sales.wiz.com domain. chou's UID is 11155. This command is typed on from the root master server. For simplicity, it assumes you are entering chou's correct login password.

```
rmaster# nisaddcred –p 11155 –P chou.wiz.com. local
rmaster# nisaddcred –p unix.11155@wiz.com –P chou.wiz.com. des
Adding key pair for unix.11155@wiz.com (chou.wiz.com.).
Enter login password:
rmaster# nisaddcred –p 11155 –P chou.wiz.com. local sales.wiz.com.
```

LOCAL credential information maps a UID to an NIS+ principal name. Although an NIS+ principal that is a client user can have different user IDs in different domains, it can have only one NIS+ principal name. So, if an NIS+ principal such as chou will be logging in from a domain other than her home domain, not only should she have a password entry in that domain, but also a LOCAL credential in that domain's cred table.

This example creates credential information for a principal **workstation**. Its host name is starshine1 and it belongs to the root domain. Therefore, its credential information is created from the root master server. In this example, you create them while logged in as root to the root master; however, if you already have valid credential information and the proper access rights, you could create them while logged in as yourself.

```
rmaster# nisaddcred –p unix.starshine1@wiz.com –P starshine1.wiz.com. des
Adding key pair for unix.starshine1@wiz.com
       (starshine1.wiz.com.).
Enter starshine1.wiz.com.'s root login password:Retype password:
```

The proper response to the password prompt is the principal workstation's superuser password. Of course, you could use a dummy password that would later be changed by someone logged in as superuser to that principal workstation.

Administering NIS+ Credential Information

The following sections describe how to use the **nisaddcred** command to administer existing credential information. You must have create, modify, read, and destroy rights to the cred table to perform these operations.

Updating Your Own Credential Information

Updating your own credential information is considerably easier than creating it. Just type the simple versions of the **nisaddcred** command while logged in as yourself:

```
  # nisaddcred des
  # nisaddcred local
```

To update credential information for someone else, you simply perform the same procedure that you would use to create that person's credential information.

Removing Credential Information

The **nisaddcred** command removes a principal's credential information, but only from the local domain where the command is run.

Thus, to completely remove a principal from the entire system, you must explicitly remove that principal's credential information from the principal's home domain and all domains where the principal has LOCAL credential information.

To remove credential information, you must have modify rights to the local domain's cred table. Use the **–r** option and specify the principal with a full NIS+ principal name:

```
  # nisaddcred –r principal-name
```

The following two examples remove the LOCAL and DES credential information of the administrator morena.wiz.com. The first example removes both types of credential information from her home domain (wiz.com.), the second removes her LOCAL credential information from the sales.wiz.com. domain. Note how they are each entered from the appropriate domain's master servers.

```
rmaster# nisaddcred –r morena.wiz.com.
salesmaster# nisaddcred –r morena.wiz.com.
```

To verify that the credential information was indeed removed, run **nismatch** on the cred table, as shown below. For more information about **nismatch**, see Administering NIS+ Tables, on page 6-60.

```
rmaster# nismatch morena.wiz.com. cred.org_dir
salesmaster# nismatch morena.wiz.com. cred.org_dir
```

Administering NIS+ Keys

This section describes how to use the **keylogin**, **chkey**, and **nisupdkeys** commands to administer keys. (The **nisaddcred** command also performs some key–related operations.

This section assumes that you have an adequate understanding of the NIS+ security system in general, and in particular of the role that keys play in that system (see Security, for this information).

- Keylogin, on page 6-16
- Changing Keys for a NIS+ Principal, on page 6-16
- Updating Public Keys, on page 6-20
- Updating IP Addresses, on page 6-20

Keylogin

When a principal logs in, the login process prompts for a password. That password is used to pass the user through the login security gate and give the user access to the network. The login process also decrypts the user's private key stored in the user's home domain cred table and passes that private key to the keyserver. The keyserver then uses that decrypted private key to authenticate the user each time the user accesses an NIS+ object.

Normally, this is the only time the principal is asked to provide a password. However, if the principal's private key in the cred table was encrypted with a password that was **different** from the user's login password, **login** cannot decrypt it using the login password at login time, and thus cannot provide a decrypted private key to the keyserver. (This most often occurs when a user's private key in the cred table was encrypted with a Secure RPC password different from the user's login password. Note also that in this context, *network password* is sometimes used as a synonym for *Secure RPC password*.)

To temporarily remedy this problem, the principal must perform a keylogin, using the **keylogin** command, after every login. (The **–r** flag is used to keylogin the superuser principal and to store the superuser's key in /**etc**/**.rootkey** on a host.)

For a principal user:

```
# keylogin
```

For a principal machine (only once):

```
# keylogin –r
```

Note, however, that performing an explicit keylogin with the original password provides only a temporary solution good for the current login session only. The private key in the cred table is still encrypted with a password different than the user's login password so the **next** time the user logs in the problem will reoccur. To permanently solve this problem, the user must run **chkey** to change the password used to encrypt the private key to the user's login password (see Changing Keys for a NIS+ Principal, on page 6-16).

Changing Keys for a NIS+ Principal

The **chkey** command changes an NIS+ principal's public and private keys that are stored in the cred table. It does not affect the principal's entry either in the passwd table or in the /**etc**/**passwd** file.

The **chkey** command:

- Generates new keys and encrypts the private key with the password. If run with the **–p** option, **chkey** re–encrypts the existing private key with a new password.

- Generates a new Diffie–Hellman key pair and encrypts the private key with the password you provide. However, in most cases you do not want a new keypair, you want to re–encrypt your **current** existing private key with the new password. To do this, you must use the **–p** flag: **chkey –p**.

See the command descriptions for more information on these subjects.

Note: In a NIS+ environment, when you change your login password with any of the current administration tools or the **passwd** command, your private key in the cred table is automatically re–encrypted with the new password for you. Thus, you do not need to explicitly run **chkey** after a change of login password.

The **chkey** command interacts with the keyserver, the cred table, and the passwd table. In order to run **chkey**, you:

- Must have an entry in the passwd table of your home domain. Failure to meet this requirement will result in an error message.
- Must run **keylogin** to make sure that the keyserver has a decrypted private key for you.
- Must have modify rights to the cred table. If you do not have modify rights you will get a ”permission denied” type of error message.
- Must know the original password with which the private key in the cred table was encrypted. (In most cases, this your Secure RPC password.)

To use the **chkey** command to re–encrypt your private key with your login password, you first run **keylogin** using the original password, and then use **chkey –p** as shown in the following table, which illustrates how to perform a **keylogin** and **chkey** for a principal user:

Table 33. Re–encrypting Your Private Key : Command Summary

Tasks	**Commands**
Log in.	`Sirius%` **`login`** *`Login-name`*
Provide login password.	`Password:`
If login password and Secure RPC password are different, perform a **keylogin.**	`Sirius%` **`keylogin`**
Provide the original password that was used to encrypt the private key.	`Password:` *`Secure RPC password`*

Run **chkey.**	`Sirius%` **`chkey -p`**
	`Updating nisplus publickey database`
	`Updating new key for 'unix.1199@Wiz.Com'.`

Enter login password.	`Enter login password:` *`login-password`*

Re–enter login password.	`Retype password:`
	`Done`

Changing Root Keys of an NIS+ Principal

The following sections describe how to change the keys of an NIS+ principal.

Changing Root Keys From Root

The following table shows how to change the keys for the root master server from the root master (as root).

Table 34. Changing a Root Master's Keys: Command Summary

Tasks	Commands
Create new DES credentials	`rootmaster#` **`nisaddcred des`**
Kill the NIS+ daemon	`rootmaster#` **`stopsrc –s rpc.nisd`**
Restart NIS+ daemon with no security	`rootmaster#` **`startsrc –s rpc.nisd –a ”–S0”`**
Perform a **keylogout** (previous **keylogin** is not out of date).	`rootmaster#` **`keylogout –f`**
Update the keys in the directories served by the master	`rootmaster#` **`nisupdkeys`** *`dirs`*
Kill the NIS+ daemon	`rootmaster#` **`stopsrc –s rpc.nisd`**
Restart NIS+ daemon with default security	`rootmaster#` **`startsrc –s rpc.nisd`**
Perform a keylogin	`rootmaster#` **`keylogin`**

Where:

dirs	Are the directory objects you wish to update (that is, the directory objects that are served by **rootmaster**).

In the first step of the process outlined in the previous table, **nisaddcred** updates the cred table for the root master, updates /**etc.rootkey** and performs a keylogin for the root master. At this point the directory objects served by the master have not been updated and their credential information is now out of synch with the root master. The subsequent steps described in the table are necessary to successfully update all the objects.

Changing Root Keys From Another Machine

To change the keys for the root master server from some other machine you must have the required NIS+ credentials and authorization to do so.

Table 35. Remotely Changing Root Master Keys: Command Summary

Tasks	Commands

Create the new DES credentials.	`othermachine% nisaddcred \`
	 `–p` *`SecureRPC–netname`* `\`
	 `–P` *`nisprincipal`* **`des`**`\`

Update the directory objects.	`othermachine%` **`nisupdkeys`** *`dirs`*
Update /**etc.roootkey.**	`othermachine%` **`keylogin –r`**
Reinitialize **othermachine** as client	`othermachine%` **`nisinit –c –H`** *`hostname`*

Where:

SecureRPC–netname
: Is the root machine’s Secure RPC netname. For example: `unix.rootmaster@wiz.com` (no dot at the end).

nisprincipal
: Is the root machine’s NIS+ principal name. For example, `rootmaster.wiz.com.` (a dot at the end).

dirs
: Are the directory objects you wish to update (that is, the directory objects that are served by **rootmaster**).

When running **nisupdkeys** be sure to update all relevant directory objects at the same time. In other words, do them all with one command. Separate updates may result in an authentication error.

Changing the Keys of a Root Replica from the Replica

To change the keys of a root replica from the replica, use these commands:

```
replica# nisaddcred des
replica# nisupdkeys dirs
```

Where:

dirs
: Are the directory objects you wish to update (that is, the directory objects that are served by *replica*).

When running **nisupdkeys** be sure to update all relevant directory objects at the same time. In other words, do them all with one command. Separate updates may result in an authentication error.

Changing the Keys of a Nonroot Server

To change the keys of a nonroot server (master or replica) from the server, use these commands:

```
subreplica# nisaddcred des
subreplica# nisupdkeys parentdir dirs
```

Where:

parentdir
: Is the non–root server’s parent directory (that is, the directory containing **subreplica**’s NIS+ server).

dirs
: Are the directory objects you wish to update (that is, the directory objects that are served by sub**replica**).

When running **nisupdkeys** be sure to update all relevant directory objects at the same time. In other words, do them all with one command. Separate updates may result in an authentication error.

Updating Public Keys

The public keys of NIS+ servers are stored in several locations throughout the namespace. When new credential information is created for the server, a new key pair is generated and stored in the cred table. However, namespace directory objects still have copies of the server's **old** public key. The **nisupdkeys** command is used to update those directory object copies.

Table 36. Updating a Public Key: Command Summary

Tasks	Commands
Update all keys of all servers of the current domain (Wiz.Com).	`rootmaster# /usr/lib/nis/nisupdkeys`

Update keys of all servers supporting the Sales.Wiz.Com domain directory object.	`Fetch Public key for server rootmaster.Wiz.Com.`
	`    netname='unix.rootmaster@Wiz.Com'`
	` Updating rootmaster.Wiz.Com.'s public key.`
	`    Public key: public-key`
	` salesmaster# nisupdkeys Sales.Wiz.Com`
	`  (Screen notices not shown)`

Update keys for a server named **server7** in all the directories that store them.	`rootmaster# nisupdkeys –H server7`
Clear the keys stored by the Sales.Wiz.Com directory object.	`rootmaster# nisupdkeys –C Sales.Wiz.Com`
Clear the keys for the current domain directory object for the server named **server7.**	`rootmaster# nisupdkeys –C –H server7`

Updating IP Addresses

If you change a server's IP address, or add additional addresses (multihome), you need to run **nisupdkeys** to update NIS+ address information.

To update the IP addresses of one or more servers, use the **nisupdkeys** command **–a** option.

To update the IP addresses of servers of a given domain

```
rootmaster# nisupdkeys –a domain
```

To update the IP address of a particular server

```
rootmaster# nisupdkeys –a –H server
```

Administering NIS+ Access Rights

This section assumes that you have an adequate understanding of the NIS+ security system in general, and in particular of the role that access rights play in that system.

This section provides the following general information about access rights:

- Concatenation of Access Rights, on page 6-21
- How Access Rights Are Assigned and Changed, on page 6-21
- Access Rights and Table Security, on page 6-22
- Where Access Rights Are Stored, on page 6-25
- Viewing an NIS+ Object's Access Rights, on page 6-25
- Default Access Rights, on page 6-25
- How a Server Grants Access Rights to Tables, on page 6-25
- Specifying Access Rights in Commands, on page 6-25

If you need to review authorization classes or access rights in general, see NIS+ Namespace and Structure, on page 4-1

Concatenation of Access Rights

Authorization classes are concatenated. In other words, the higher class usually belongs to the lower class and automatically gets the rights assigned to the lower class. It works like this:

Owner Class	An object's owner may, or may not, belong to the object's group. If the owner does belong to the group, then the owner gets whatever rights are assigned to the group. The object's owner automatically belongs to the world and nobody classes, so the owner automatically gets whatever rights that object assigns to those two classes.
Group Class	Members of the object's group automatically belong to the world and nobody classes, so the group members automatically get whatever rights that object assigns to world and nobody.
World Class	The world class automatically gets the same rights to an object that are given to the nobody class.
Nobody Class	The nobody class only gets those rights an object specifically assigns to the nobody class.

The basic principle that governs this is that access rights override the absence of access rights. In other words, a higher class can have *more* rights than a lower class, but not *fewer* rights. (The one exception to this rule is that if the owner is not a member of the group, it is possible to give rights to the group class that the owner does not have.)

How Access Rights Are Assigned and Changed

When you create an NIS+ object, NIS+ assigns that object a default set of access rights for the owner and group classes. By default, the owner is the NIS+ principal who creates the object. The default group is the group named in the **NIS_GROUP** environment variable. See Default Access Rights, on page 6-25 for details.

Specifying Different Default Rights

NIS+ provides two different ways to change the default rights that are automatically assigned to an NIS+ object when it is created.

- The **NIS_DEFAULTS** environment variable. **NIS_DEFAULTS** stores a set of security–related default values, one of which is access rights. These default access rights are the ones automatically assigned to an object when it is created. See Displaying NIS+ Defaults, on page 6-25 for details.

  If the value of the **NIS_DEFAULTS** environment variable is changed, objects created after the change are assigned the new values. However, previously created objects are not affected.

- The **–D** option, which is available with several NIS+ commands. When you use the **–D** option as part of the command to create an NIS+ object, it overrides the default rights specified by the **NIS_DEFAULTS** environment variable and allows you to explicitly specify an initial set of rights for that object. See Specifying Nondefault Security Values at Creation Time, on page 6-25 for details.

Changing Access Rights to an Existing Object

When an NIS+ object is created, it comes into existence with a default set of access rights (from either the **NIS_DEFAULTS** environment variable or as specified with the **–D** option). These default rights can be changed with the

- `nischmod` command
- `nistbladm` command for table columns

Access Rights and Table Security

NIS+ tables allow you to specify access rights on the table three ways: you can specify access rights to the table as a whole, to each table column individually, and to each entry (row) by itself. (A *field* is a single cell in the matrix or, in other words, the intersection between a column and an entry (row). All data values are entered in fields.)

Table 37. Table Entries and Columns

		(Column)			
(Entry)		FIELD			

These column– and entry level access rights allow you to specify *additional* access to individual rows and columns that override table level restrictions, but column and entry level rights cannot be *more* restrictive than the table as a whole:

- ***Table:*** The table level is the base level. Access rights assigned at the table level apply to every piece of data in the table unless specifically modified by a column or entry exception. Thus, the table level rights should be the *most* restrictive. Remember that authorization classes concatenate. Higher class gets the rights assigned to the lower class. (See Concatenation of Access Rights, on page 6-21.)

- ***Column:*** Column–level rights allow you to grant additional access rights on a column–by–column basis. For example, suppose the table level granted no access rights whatsoever to the world and nobody classes. In such a case, no one in those two classes could read, modify, create, or destroy any data in the table. You could use column–level rights to override that table level restriction and permit members of the world class the right to view data in a particular column.

  On the other hand, if the table level grants table–wide read rights to the owner and group classes, you cannot use column–level rights to prevent the group class from having read rights to that column.

Keep in mind that a column's group does not have to be the same as the table's group or an entry's group. They can all have different groups.

- ***Entry (row):*** entry level rights allow you to grant additional access rights on a row–by–row basis. For example, this allows you to permit individual users to change entries that apply to them, but not entries that apply to anyone else.

  Keep in mind that an entry's group does not have to be the same as the table's group or a column's group. They can all have different groups. This means that you can permit members of a particular group to work with one set of entries while preventing them from affecting entries belonging to other groups.

Table, Column, Entry Example

Column– or entry–level access rights can provide additional access in two ways: by extending the rights to additional principals or by providing additional rights to the same principals. Of course, both ways can be combined. Following are some examples.

Assume a table object granted read rights to the table's owner:

	Nobody	**Owner**	**Group**	**World**
Table Access Rights:	——	r——	——	——

This means that the table's owner could read the contents of the entire table but no one else could read anything. You could then specify that the following table grant read rights to the group class:

	Nobody	**Owner**	**Group**	**World**
Table Access Rights:	——	r——	——	——
Entry–2 Access Rights:	——	——	r——	——

Although only the owner could read all the contents of the table, any member of the table's group could read the contents of that particular entry. Now, assume that a particular column granted read rights to the world class:

	Nobody	**Owner**	**Group**	**World**
Table Access Rights:	——	r——	——	——
Entry–2 Access Rights:	——	——	r——	
Column–1 Access Rights:	——	——	——	r——

Members of the world class could now read that column **for all entries** in the table (see following example). Members of the group class could read everything in Column–1 (because members of the group class are also members of the world class) and also all columns of Entry–2. Neither the world nor the group classes could read any cells marked in the following example as ***NP*** (for Not Permitted).

	Col 1	**Col 2**	**Col 3**
Entry–1	contents	*NP*	*NP*
Entry–2	contents	contents	contents
Entry–3	contents	*NP*	*NP*
Entry–4	contents	*NP*	*NP*
Entry–5	contents	*NP*	*NP*

Rights at Different Levels

This section describes how the four different access rights (read, create, modify, and destroy) work at the four different access levels (directory, table, column, and entry).

The objects that these various rights and levels act on are summarized in the following table:

	Directory	**Table**	**Column**	**Entry**
Read	List directory contents	View table contents	View column contents	View entry (row) contents
Create	Create new directory or table objects	Add new entries (rows)	Enter new data values in a column	Enter new data values in an entry (row)
Modify	Move objects and change object names	Change data values anywhere in table	Change data values in a column	Change data values in an entry (row)
Destroy	Delete directory objects such as tables	Delete entries (rows)	Delete data values in a column	Delete data values in an entry (row)

Read Rights

- ***Directory:*** If you have read rights to a directory, you can list the contents of the directory.
- ***Table:*** If you have read rights to a table, you can view all the data in that table.
- ***Column:*** If you have read rights to a column, you can view all the data in that column.
- ***Entry:*** If you have read rights to an entry, you can view all the data in that entry.

Create Rights

- ***Directory:*** If you have create rights at the directory level, you can create new objects in the directory such as new tables.
- ***Table:*** If you have create rights at the table level, you can create new entries. (You cannot add new columns to an existing table regardless of what rights you have.)
- ***Column:*** If you have create rights to a column, you can enter new data values in the fields of that column. You cannot create new columns.
- ***Entry:*** If you have create rights to an entry, you can enter new data values in the fields of that row. (Entry level create rights do not permit you to create new rows.)

Modify Rights

- ***Directory:*** If you have modify rights at the directory level, you can move or rename directory objects.
- ***Table:*** If you have modify rights at the table level, you can change any data values in the table. You can create (add) new rows, but you cannot create new columns. If an existing field is blank, you can enter new data in it.
- ***Column:*** If you have modify rights to a column, you can change the data values in the fields of that column.
- ***Entry:*** If you have modify rights to an entry, you can change the data values in the fields of that row.

Destroy Rights

- ***Directory:*** If you have destroy rights at the directory level, you can destroy existing objects in the directory such as tables.
- ***Table:*** If you have destroy rights at the table level, you can destroy existing entries (rows) in the table but not columns. You cannot destroy existing columns in a table: you can only destroy entries.
- ***Column:*** If you have destroy rights to a column, you can destroy existing data values in the fields of that column.
- ***Entry:*** If you have destroy rights to an entry, you can destroy existing data values in the fields of that row.

Where Access Rights Are Stored

An object's access rights are specified and stored as part of the object's definition. This information is not stored in an NIS+ table.

Viewing an NIS+ Object's Access Rights

The access rights can be viewed by using the **niscat** command.

```
niscat –o objectname
```

Where *objectname* is the name of the object whose access rights you want to view.

This command returns the following information about an NIS+ object:

- ***Owner:*** The single NIS+ principal who has ownership rights. This is usually the person who created the object, but it could be someone to whom the original owner transferred ownership rights.
- ***Group:*** The object's NIS+ group.
- ***Nobody class access rights:*** The access rights granted to everyone, whether they are authenticated (have a valid DES credential) or not.
- ***Owner class access rights:*** The access rights granted to the object's owner.
- ***Group class access rights:*** The access rights granted to the principals in the object's group.
- ***World class access rights:*** The access rights granted to all authenticated NIS+ principals.

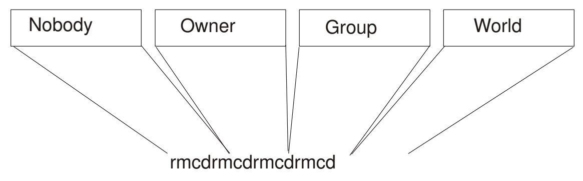
Access rights for the four authorization classes are displayed as a list of 16 characters, like this:

```
r---rmcdr---r---
```

Each character represents a type of access right:

r	represents read rights.
m	represents modify rights.
d	represents destroy rights.
c	represents create rights.
–	represents no access rights.

The first four characters represent the access rights granted to nobody, the next four to the owner, the next four to the group, and the last four to the world.



Note: Unlike UNIX file systems, the first set of rights is for nobody, not for the owner.

Default Access Rights

When you create an object, NIS+ assigns the object a default owner and group, and a default set of access rights for all four classes. The default owner is the NIS+ principal who creates the object. The default group is the group named in the **NIS_GROUP** environment variable. Initially, the default access rights are:

Table 38. Default Access Rights

Nobody	Owner	Group	World
–	read	read	read
–	modify	–	–
–	create	–	–
–	destroy	–	–

If you have the **NIS_DEFAULTS** environment variable set, the values specified in **NIS_DEFAULTS** will determine the defaults that are applied to new objects. When you create an object from the command line, you can use the **–D** flag to specify values other than the default values.

How a Server Grants Access Rights to Tables

This section discusses how a server grants access to tables objects, entries, and columns during each type of operation: read, modify, destroy, and create.

Note: At security level 0, a server enforces no NIS+ access rights and all clients are granted full access rights to the table object. Security level 0 is only for administrator setup and testing purposes. Do not use level 0 in any environment where ordinary users are performing their normal work.

The four factors that a server must consider when deciding whether to grant access are:

- The type of operation requested by the principal
- The table, entry, or column the principal is trying to access
- The authorization class the principal belongs to for that particular object
- The access rights that the table, entry, or column has assigned to the principal's authorization class

After authenticating the principal making the request by making sure the principal has a valid DES credential, an NIS+ server determines the type of operation and the object of the request.

- ***Directory:*** If the object is a directory or group, the server examines the object's definition to see what rights are granted to the four authorization classes, determines which class the principal belongs to, and then grants or denies the request based on the principal's class and the rights assigned to that class.
- ***Table:*** If the object is a table, the server examines the table's definition to see what table level rights are granted to the four authorization classes, and determines which class the principal belongs to. If the class to which the principal belongs does not have table level rights to perform the requested operation, the server then determines which row or

column the operation concerns and determines if there are corresponding row– or column–level access rights permitting the principal to perform the requested operation.

Specifying Access Rights in Commands

The following paragraphs describe how to specify access rights, as well as owner, group owner, and object, when using any of the commands described in this section. (This section assumes an NIS+ environment running at the default security level 2.)

Class, Operator, and Rights Syntax

Access rights, whether specified in an environment variable or a command, are identified with three types of arguments: **class**, **operator**, and **right**.

Class

Class refers to the type of NIS+ principal (authorization class) to which the **rights** will apply.

Table 39. Access Rights Syntax—Class

Class	Description
n	Nobody: all unauthenticated requests
o	The owner of the object or table entry
g	The group owner of the object or table entry
w	World: all authenticated principals
a	All: shorthand for owner, group, and world (this is the default)

Operator

The operator indicates the kind of operation that will be performed with the **rights**

Table 40. Access Rights Syntax—Operator

Operator	Description
+	Adds the access rights specified by **right**
–	Revokes the access rights specified by **right**
=	Explicitly changes the access rights specified by **right**; in other words, revokes all existing rights and replaces them with the new access rights.

Rights

The rights are the access rights themselves. The accepted values for each are listed below.

Table 41. Access Rights Syntax—Rights

Right	Description
r	Reads the object definition or table entry
m	Modifies the object definition or table entry
c	Creates a table entry or column
d	Destroys a table entry or column

You can combine operations on a single command line by separating each operation from the next with a comma (,).

Table 42. Class, Operator, and Rights Syntax—Examples

Operations	Syntax
Add read access rights to the **owner** class	`o+r`
Change owner. group, and world classes' access rights to modify only from whatever they were before.	`a=m`
Add read and modify rights to the world and nobody classes	`wn+m`
Remove all four rights from the group, world, and nobody classes.	`gwn-rmcd`
Add create and destroy rights to the owner class and add read and modify rights to the world and nobody classes	`o+cd,wn+rm`

Syntax for Owner and Group

- ***Owner:*** To specify an owner, use an NIS+ principal name.
- ***Group:*** To specify an NIS+ group, use an NIS+ group name with the domain name appended. Remember that, by definition, principal names are fully qualified (*principalname.domainname*).

For owner:

```
principalname
```

For group:

```
groupname.domainname
```

Syntax for Objects and Table Entries

Objects and table entries use different syntaxes.

- Objects use simple object names.
- Table entries use indexed names.

For objects:

```
objectname
```

For table entries:

```
[columnname=value],tablename
```

Note: In this case, the brackets are part of the syntax.

Indexed names can specify more than one column–value pair. If so, the operation applies only to the entries that match **all** the column–value pairs. The more column–value pairs you provide, the more stringent the search.

For example:

Table 43. Object and Table Entry—Examples

Type	Example
Object	**hosts.org_dir.sales.wiz.com.**
Table entry	**'[uid=33555],passwd.org_dir.Eng.wiz.com.'**
Two–value table entry	**'[name=sales,gid=2],group.org_dir.wiz.com.'**

Columns use a special version of indexed names. Because you can only work on columns with the **nistbladm** command, see its command description for more information.

Displaying NIS+ Defaults

The **nisdefaults** command displays the seven default values currently active in the namespace. These default values are either:

- Preset values supplied by the NIS+ software
- The defaults specified in the **NIS_DEFAULTS** environment variable (if you have **NIS_DEFAULTS** values set)

Any object that you create on this machine will automatically acquire these default values unless you override them with the **–D** option of the command you are using to create the object.

Table 44. The Seven NIS+ Default Values and **nisdefaults** Options

Default	Option	From	Description
Domain	**–d**	**/bin/domainname**	Displays the home domain of the workstation from which the command was entered.
Group	**–g**	**NIS_GROUP** environment variable	Displays the group that would be assigned to the next object created from this shell.
Host	**–h**	**uname –n**	Displays the workstation's host name.
Principal	**–p**	**gethostbyname()**	Displays the fully qualified user name or host name of the NIS+ principal who entered the **nisdefaults** command.
Access Rights	**–r**	**NIS_DEFAULTS** environment variable	Displays the access rights that will be assigned to the next object or entry created from this shell. Format:——**rmcdr——r——**
Search path	**–s**	**NIS_PATH** environment variable	Displays the syntax of the search path, which indicate the domains that NIS+ will search through when looking for information. Displays the value of the **NIS_PATH** environment variable if it is set.
Time–to–live	**–t**	**NIS_DEFAULTS** environment variable	Displays the time–to–live that will be assigned to the next object created from this shell. The default is 12 hours.
All (terse)	**–a**		Displays all seven defaults in terse format.
Verbose	**–v**		Display specified values in verbose mode.

You can use these options to display all default values or any subset of them:

- To display all values in verbose format, type the **nisdefaults** command without arguments.

```
master% nisdefaults
Principal Name : topadmin.wiz.com.
Domain Name    : wiz.com.
Host Name      : rootmaster.wiz.com.
Group Name     : salesboss
Access Rights  : ----rmcdr---r---
Time to live   : 12:00:00:00:00
Search Path    : wiz.com.
```

- To display all values in terse format, add the **–a** option.
- To display a subset of the values, use the appropriate options. The values are displayed in terse mode. For example, to display the rights and search path defaults in terse mode, type:

```
rootmaster% nisdefaults -rs
----rmcdr---r---
wiz.com.
```

- To display a subset of the values in verbose mode, add the **–v** flag.

Setting Default Security Values

This section describes how to perform tasks related to the **nisdefaults** command, the **NIS_DEFAULTS** environment variable, and the **–D** option. The **NIS_DEFAULTS** environment variable specifies the following default values:

- Owner
- Group
- Access rights
- Time–to–live

The values that you set in the **NIS_DEFAULTS** environment variable are the default values applied to all NIS+ objects that you create using that shell (unless overridden by using the **–D** option with the command that creates the object).

You can specify the default values (owner, group, access rights, and time–to–live) specified with the **NIS_DEFAULTS** environment variable. Once you set the value of **NIS_DEFAULTS**, every object you create from that shell will acquire those defaults, unless you override them by using the **–D** option when you invoke a command.

Displaying the Value of NIS_DEFAULTS

You can check the setting of an environment variable by using the **echo** command, as shown below:

```
client% echo $NIS_DEFAULTS
owner=butler:group=gamblers:access=o+rmcd
```

You can also display a general list of the NIS+ defaults active in the namespace by using the **nisdefaults** command.

Changing Defaults

You can change the default access rights, owner, and group, by changing the value of the **NIS_DEFAULTS** environment variable. Use the environment command that is appropriate for your shell with the following arguments:

- **access=***right*, where *right* represents the access rights using the formats described in Specifying Access Rights in Commands .
- **owner=***name*, where *name* is the user name of the owner.
- **group=***group*, where *group* is the name of the default group.

You can combine two or more arguments into one line, separated by colons:

```
owner=principal-name:group=group-name
```

Table 45. Changing Defaults—Examples

Tasks	Examples
This command grants owner read access as the default access right.	`client% export NIS_DEFAULTS="access=o+r"`
This command sets the default owner to be the user abe whose home domain is Wiz.com.	`client% export NIS_DEFAULTS="owner=abe.wiz.com."`
This command combines the first two examples on one code line.	`client% export NIS_DEFAULTS="access=o+r:owner=abe.wiz.com."`

All objects and entries created from the shell in which you changed the defaults will have the new values you specified. You cannot specify default settings for a table column or entry; the columns and entries simply inherit the defaults of the table.

Resetting the Value of NIS_DEFAULTS

You can reset the NIS_DEFAULTS variable to its original values, by typing the name of the variable without arguments, using the following format:

```
client$ NIS_DEFAULTS=; export NIS_DEFAULTS
```

Specifying Nondefault Security Values at Creation Time

You can specify different (that is, nondefault) access rights, owner, and group, any time that you create an NIS+ object or table entry with any of the following NIS+ commands:

nismkdir	Creates NIS+ directory objects
nisaddent	Transfers entries into an NIS+ table
nistbladm	Creates entries in an NIS+ table

To specify security values other than the default values, insert the **–D** option into the syntax of those commands, as described in Specifying Access Rights in Commands, on page 6-25.

As when setting defaults, you can combine two or more arguments into one line. Remember that column and entry's owner and group are always the same as the table, so you cannot override them.

For example, to use the **nismkdir** command to create a sales.wiz.com directory and override the default access right by granting the owner only read rights you would type:

```
client% nismkdir –D access=o+r sales.wiz.com
```

Changing Object and Entry Access Rights

The **nischmod** command operates on the access rights of an NIS+ object or table entry. It does not operate on the access rights of a table column; for columns, use the **nistbladm** command with the **–D** option. For all **nischmod** operations, you must already have modify rights to the object or entry.

To add rights for an object, use:

```
nischmod class+right object-name
```

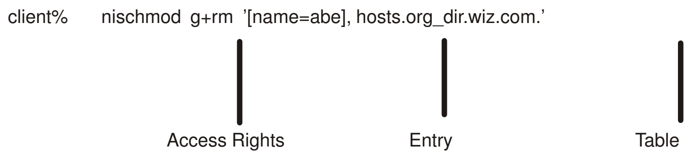
To add rights for a table entry, use:

```
nischmod class+right [column-name=value],table-name
```

For example, to add read and modify rights to the group of the `sales.wiz.com.` directory object you would type:

```
client% nischmod g+rm sales.wiz.com.
```

To add read and modify rights to group for the `name=abe` entry in the `hosts.org_dir.wiz.com.` table you would type:



```
client%     nischmod  g+rm  '[name=abe], hosts.org_dir.wiz.com.'
```

To remove rights for an object, use:

```
nischmod class-right object-name
```

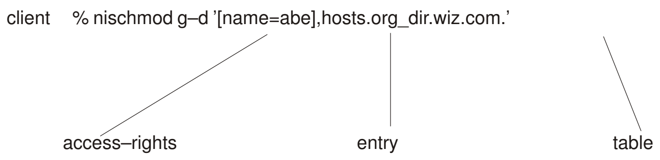
To remove rights for a table entry, use:

```
nischmod class-right [column-name=value],table-name
```

For example, to remove create and destroy rights from the group of the `sales.wiz.com.` directory object you would type:

```
client% nischmod g–cd sales.wiz.com.
```

For example to remove destroy rights from group for the `name=abe` entry in the `hosts.org_dir.wiz.com.` table, you would type:



```
client    % nischmod g–d '[name=abe],hosts.org_dir.wiz.com.'
```

Specifying Column Access Rights

The **nistbladm** command performs a variety of operations on NIS+ tables. However, two of its options, **–c** and **–u**, enable you to perform some security–related tasks:

- *The **–c** option:* The **–c** option allows you to specify initial column access rights when creating a table with the **nistbladm** command.
- *The **–u** option:* The **–u** option allows you to change column access rights with the **nistbladm** command.

Setting Column Rights When Creating a Table

When a table is created, its columns are assigned the same rights as the table object. These table level, rights are derived from the **NIS_DEFAULTS** environment variable, or are specified as part of the command that creates the table. You can also use the **nistbladm –c** option to specify initial column access rights when creating a table with **nistbladm**. To use this option you must have create rights to the directory in which you will be creating the table. To set column rights when creating a table use:

```
nistbladm –c type 'columname=[flags] [,access]... tablename'
```

Where:

- *type* is a character string identifying the kind of table. A table's type can be anything you want it to be.
- *columnname* is the name of the column.
- *flags* is the type of column. Valid flags are: **S** for searchable, **I** for case insensitive, **C** for encrypted, **B** for binary data, and **X** for XDR encoded data.
- *access* is the access rights for this column that you specify using the syntax described in Specifying Access Rights in Commands, on page 6-25.
- *...* indicates that you can specify multiple columns each of the own type and with their own set of rights.

- *tablename* is the fully qualified name of the table you are creating.

To assign a column its own set of rights at table creation time, append access rights to each column's equal sign after the column type and a comma. Separate the columns with a space:

```
column=type,rights column=type,rights column=type,rights
```

The example below creates a table named **depts** in the Wiz.com directory, of type **div**, with three columns (**Name**, **Site**, and **Manager)**, and adds modify rights for the group to the second and third columns:

```
rootmaster% nistbladm -c div Name=S Site=S,g+m Manager=S,g+m depts.wiz.com.
```

For more information about the **nistbladm** and the **–c** option, see Administering NIS+ Tables.

Adding Rights to an Existing Table Column

The **nistbladm –u** option allows you to add additional column access rights to an existing table column with the **nistbladm** command. To use this option you must have modify rights to the table column. To add additional column rights use:

```
   nistbladm -u [column=access,...],tablename
```

Where:

- *column* is the name of the column.
- *access* is the access rights for this column that you specify using the syntax described in Specifying Access Rights in Commands, on page 6-25.
- *...* indicates that you can specify rights for multiple columns.
- *tablename* is the fully qualified name of the table you are creating.

Use one *column=access* pair for each column whose rights you want to update. To update multiple columns, separate them with commas and enclose the entire set with square brackets:

```
   [column=access,column=access,column=access]
```

The full syntax of this option is described in the nistbladm command description.

The example below adds read and modify rights to the group for the **name** and **addr** columns in the `hosts.org_dir.wiz.com.` table.

```
client% nistbladm -u '[name=g+rm,addr=g+rm],hosts.org_dir.wiz.com.'
```

Removing Rights to a Table Column

To remove access rights to a column in an NIS+ table, you use the –**u** option as described in the previous section except that you subtract rights with a minus sign (rather than adding them with a plus sign).

The example below removes group's read and modify rights to the hostname column in the `hosts.org_dir.wiz.com.` table.

```
client% nistbladm -u 'name=g-rm,hosts.org_dir.wiz.com.'
```

Changing Ownership of Objects and Entries

The **nischown** command changes the owner of one or more objects or entries. To use it, you must have modify rights to the object or entry. The **nischown** command cannot change the owner of a column, since a table's columns belong the table's owner. To change a column's owner, you must change the table's owner.

To change an object's owner, use the following syntax:

```
   nischown new-owner object
```

Where:

- *new–owner* is the fully qualified user ID of the object's new owner.
- *object* is the fully qualified name of the object.

Be sure to append the domain name to both the object name and new owner name.

The example below changes the owner of the hosts table in the Wiz.com. domain to the user named lincoln whose home domain is Wiz.com.:

```
   client% nischown lincoln.wiz.com. hosts.org_dir.wiz.com.
```

The syntax for changing a table entry's owner uses an indexed entry to identify the entry, as shown below:

```
   nischown new-owner [column=value,...],tablename
```

Where:

- *new–owner* is the fully qualified user ID of the object's new owner.
- *column* is the name of the column whose value will identify the particular entry (row) whose owner is to be changed.
- *value* is the data value that identified the particular entry (row) whose owner is to be changed.
- *...* indicates that you can specify ownership changes for multiple entries.
- *tablename* is the fully qualified name of the tables containing the entry whose owner is to be changed.

Be sure to append the domain name to both the new owner name and the table name.

The example below changes the owner of an entry in the Hosts table of the Wiz.com. domain to **takeda** whose home domain is Wiz.com. The entry is the one whose value in the name column is **virginia**.

```
client% nischown takeda.wiz.com. '[name=virginia],hosts.org_dir.wiz.com.'
```

Changing an Object or Entry's Group

The **nischgrp** command changes the group of one or more objects or table entries. To use it, you must have modify rights to the object or entry. The **nischgrp** command cannot change the group of a column, since the group assigned to a table's columns is the same as the group assigned to the table. To change a column's group owner, you must change the table's group owner.

To change an object's group, use the following syntax:

```
   nischgrp group object
```

Where:

- *group* is the fully qualified name of the object's new group.
- *object* is the fully qualified name of the object.

Be sure to append the domain name to both the object name and new group name.

The example below changes the group of the hosts table in the `Wiz.com.` domain to `admins.wiz.com.`:

```
   client% nischgrp admins.wiz.com. hosts.org_dir.wiz.com.
```

The syntax for changing a table entry's group uses an indexed entry to identify the entry. In the following example, from *column* through *tablename* is the indexed entry.

```
   nischgrp new-group [column=value,...],tablename
```

Where:

- *new–group* is the fully qualified name of the object's new group.
- *column* is the name of the column whose value will identify the particular entry (row) whose group is to be changed.
- *value* is the data value that identified the particular entry (row) whose group is to be changed.
- *tablename* is the fully qualified name of the tables containing the entry whose group is to be changed.
- *...* indicates that you can specify group changes for multiple entries.

Be sure to append the domain name to both the new group name and the table name.

The example below changes the group of an entry in the Hosts table of the `Wiz.com.` domain to `sales.wiz.com.`. The entry is the one whose value in the host name column is **virginia**.

```
client% nischgrp sales.wiz.com. '[name=virginia],hosts.org_dir.wiz.com.'
```

Administering Passwords

This section describes how to use the **passwd** command from the point of view of an ordinary user and of a system administrator. You can also use Web-based System Manager or SMIT to do these tasks, both of which can be more convenient than running the **passwd** command as described in this section. This section assumes you have adequate understanding of the NIS+ security system in general—in particular of the role that login passwords play in that system (see the Security chapter). This section covers:

- Logging In, on page 6-36
- Changing Your Password, on page 6-37
- Choosing a Password, on page 6-38
- The passwd Command, on page 6-39
- The nistbladm Command, on page 6-40
- Changing Passwords, on page 6-40
- Managing Password Aging, on page 6-41
- Specifying Password Criteria and Defaults, on page 6-44

Logging In

Logging in to a system is a two–step process:

1. Type your login ID at the **Login:** prompt.
2. Type your password at the **Password:** prompt.

   (To maintain password secrecy, your password is not displayed on your screen when you type it.)

If your login is successful you will see your system's message of the day (if any) and then your command–line prompt, windowing system, or normal application.

The Login incorrect Message

The **Login incorrect** message indicates that:

- You have entered the wrong login ID or the wrong password. This is the most common cause of the **Login incorrect** message. Check your spelling and repeat the process. Note that most systems limit to three the number of unsuccessful login tries you can make.
- Another possible cause of the **Login incorrect** message is that an administrator has locked your password and you cannot use it until it is unlocked. If you are sure that you are entering your login ID and password correctly, and you still get a **Login incorrect** message, contact your system administrator.
- Another possible cause of the **Login incorrect** message is that an administrator has expired your password privileges and you cannot use your password until your privileges are restored. If you are sure that you are entering your login ID and password correctly, and you still get a **Login incorrect** message, contact your system administrator.

The password expired Message

If you receive a **Your password has expired** message it means that your password has reached its age limit and expired. In other words, the password has been in use for too long and you must choose a new password at this time. (See Choosing a Password, on page 6-38, for criteria that a new password must meet.)

In this case, choosing a new password is a three–step process:

1. Type your old password at the **Enter login password** (or similar) prompt.

Your keystrokes are not shown on your screen.

2. Type your new password at the **Enter new password** prompt.

   Your keystrokes are not shown on your screen.

3. Type your new password again at the **Re–enter new password** prompt.

   Your keystrokes are not shown on your screen.

The will expire Message

If you receive a **Your password will expire in** *N* **days** message (where *N* is a number of days), or a **Your password will expire within 24 hours** message, it means that your password will reach its age limit and expire in that number of days (or hours).

In essence, this message is telling you to change your password now. (See Changing Your Password, on page 6-37.)

The Permission denied Message

After entering your login ID and password, you may get a **Permission denied** message and be returned to the **login:** prompt. This means that your login attempt has failed because an administrator has either locked your password, or terminated your account, or your password privileges have expired. In these situations a user cannot log in until an administrator unlocks the user password or reactivates the account or privileges.

Changing Your Password

To maintain security, you should change your password regularly. (See Choosing a Password, on page 6-38 for password requirements and criteria.) Note that you can also use Web-based System Manager or SMIT to change your password. You may find that more convenient than running the passwd command as described in this manual.

Changing your password is a four–step process:

1. Run the **passwd** command at a system prompt.
2. Type your old password at the **Enter login password** (or similar) prompt.
3. Your keystrokes are not shown on your screen.
4. Type your new password at the **Enter new password** prompt

   Your keystrokes are not shown on your screen.

At this point the system checks to make sure that your new password meets the requirements:

- If it does meet the requirements, you are asked to enter it again.
- If your new password does not meet the system requirements, a message is displayed informing you of the problem. You must then enter a new password that does meet the requirements.

  See Password Requirements, on page 6-38 for the requirements a password must meet.

  – Type your new password again at the **Re–enter new password** prompt.

    Your keystrokes are not shown on your screen.

  – If your second entry of the new password is not identical to your first entry, you are prompted to repeat the process.

Note: When changing root's password, you must always run **chkey –p** immediately after changing the password. (See Changing Root Keys From Root, on page 6-18 and Changing Root Keys From Another Machine, on page 6-18 for information on using **chkey –p** to change root's keys.) Failure to run **chkey –p** after changing root's password will result in root being unable to properly log in.

Password Change Failures

Some systems limit either the number of failed attempts you can make in changing your password or the total amount of time you can take to make a successful change. (These limits are implemented to prevent someone else from changing your password by guessing your current password.)

If you (or someone posing as you) fails to successfully log in or change your password within the specified number of tries or time limit, you will get a **Too many failures – try later** or **Too many tries: try again later** message. You will not be allowed to make any more attempts until a certain amount of time has passed. (That amount of time is set by your administrator.)

Choosing a Password

Many breaches of computer security involve guessing another user's password. While the **passwd** command enforces some criteria for making sure the password is hard to guess, a clever person can sometimes figure out a password just by knowing something about the user. Thus, a good password is one that is easy for you to remember but hard for someone else to guess. A bad password is one that is so hard for you to remember that you have to write it down (which you are not supposed to do), or that is easy for someone who knows about you to guess.

Password Requirements

A password should meet the following suggested requirements:

- **Length**. By default, a password must have at least six characters. Only the first eight characters are significant. (In other words, you can have a password that is longer than eight characters, but the system only checks the first eight.) Because the minimum length of a password can be changed by a system administrator, it may be different on your system.
- **Characters**. A password must contain at least two letters (either upper– or lower–case) and at least one numeral or symbol such as **@,#,%**. For example, you can use **dog#food** or **dog2food** as a password, but you cannot use **dogfood**.
- **Not your login ID**. A password cannot be the same as your login ID, nor can it be a rearrangement of the letters and characters of your login ID. (For the purpose of this criteria, upper and lower case letters are considered to be the same.) For example, if your login ID is **Claire**2 you cannot have **e2clair** as your password.
- **Different from old password**. Your new password must differ from your old one by at least three characters. (For the purpose of this criterion, upper– and lower–case letters are considered to be the same.) For example, if your current password is **Dog#fooD** you can change it to **dog#Meat** but you cannot change it to **daT#Food**.

For further information, see the **/etc/security/user** file description.

Bad Choices for Passwords

Bad choices for passwords include:

- Any password based on your name
- Names of family members or pets
- Car license numbers
- Telephone numbers
- Social Security numbers
- Employee numbers
- Names related to a hobby or interest
- Seasonal themes, such as Santa in December
- Any word that is in a standard dictionary

Good Choices for Passwords

Good choices for passwords include:

- Phrases plus numbers or symbols (**beam#meup**)
- Nonsense words made up of the first letters of every word in a phrase plus a number or symbol (**swotrb7** for SomeWhere Over The RainBow)
- Words with numbers, or symbols substituted for letters (**sn00py** for **snoopy**)

The passwd Command

The **passwd** command performs various operations regarding passwords. The **yppasswd** command is still retained with all of its functionality for the purpose of backward compatibility.

The passwd Command and Credentials

When run in an NIS+ environment, the **passwd** command is designed to function with or without credentials. Users without credentials are limited to changing their own password. Other password operations can only be performed by users who have credentials (are authenticated) and who have the necessary access rights (are authorized).

The passwd Command and Permissions

In this discussion of authorization and permissions, it is assumed that everyone referred to has the proper credentials.

By default, in a normal NIS+ environment the owner of the passwd table can change password information at any time and without constraints. In other words, the owner of the passwd table is normally granted full read, modify, create, and destroy authorization (permission) for that table. An owner can also:

- Assign table ownership to someone else with the **nischown** command.
- Grant some or all of read, modify, create, and destroy rights to the table's group, or even to the world or nobody class. (Of course, granting such rights to world or nobody seriously weakens NIS+ security.)
- Change the permissions granted to any class with the **nisdefaults, nischmod,** or **nistbladm** commands.

Note: Regardless of what permissions they have, everyone in the world, and nobody classes are forced to comply with password–aging constraints. In other words, they cannot change a password for themselves or anyone else unless that password has aged past its minimum. Nor can members of the group, world, and nobody classes avoid having to change their own passwords when the age limit has been reached. However, age constraints do not apply to the owner of the passwd table.

To use the **passwd** command in an NIS+ environment, you must have the required authorization (access rights) for the operation:

Table 46. Access Rights for **passwd** Command

This Operation	Requires These Rights	To This Object
Displaying information	read	passwd table entry
Changing Information	modify	passwd table entry
Adding New Information	modify	passwd table

The passwd Command and Keys

If you use **passwd** in a NIS+ environment to change a principal's password, it tries to update the principal's private (secret) key in the cred table.

- If you have modify rights to the DES entry in the cred table and if the principal's login and Secure RPC passwords are the same, **passwd** will update the private key in the cred table.
- If you do not have modify rights to the DES entry in the cred table or if the principal's login and Secure RPC passwords are not the same, the **passwd** command will change the password, but not change the private key.

If you do not have modify rights to the DES entry, it means that the private key in the cred table will have been formed with a password that is now different from the one stored in the passwd table. In this case, the user will have to change keys with the **chkey** command or run **keylogin** after each login.

The nistbladm Command

The **nistbladm** command allows you to create, change, and display information about any NIS+ table, including the passwd table.

It is possible to use the **nistbladm** command to:

- Create new passwd table entries
- Delete an existing entry
- Change the UID and GID fields in the passwd table
- Change access rights and other security–related attributes of the passwd table
- Set expiration and inactivity periods for a user's account (see Password Privilege Expiration, on page 6-41 and Specifying Maximum Number of Inactive Days, on page 6-43.)
- Set password parameters for multiple users at once (see Setting Password Aging Criteria for Multiple Users, on page 6-44).

Changing Passwords

New passwords must meet the criteria described in Password Requirements, on page 6-38.

Changing Your Own Password

To change your password, type

```
station1% passwd
```

You will be prompted for your old password and then the new password and then the new password a second time to confirm it.

Changing Someone Else's Password

To change someone else' password, use:

To change another user's password in the same domain

```
passwd username
```

When using the **passwd** command in a NIS+ environment to change someone else's password you must have modify rights to that user's entry in the passwd table (this usually means that you are a member of the group for the passwd table and the group has modify rights). You do not have to enter either the user's old password or your password. You will be prompted to enter the new password twice to make sure that they match. If they do not match, you will be prompted to enter them again.

Changing Root’s Password

When changing root’s password, you must always run **chkey –p** immediately after changing the password with the **passwd** command. Failure to run **chkey –p** after changing root’s password will result in root being unable to properly log in.

To change a root password, follow these steps:

1. Log in as root.
2. Change root’s password using **passwd**.

   Do not use **nispasswd**.
3. Run **chkey –p**.

   You **must** use the **–p** option.

Managing Password Aging

Password aging is a mechanism you can use to force users to periodically change their passwords.

Password aging encompasses the following tasks:

- Force a user to choose a new password the next time the user logs in.
- Specify a maximum number of days that a password can be used before it has to be changed.
- Specify a minimum number of days that a password has to be in existence before it can be changed.
- Specify that a warning message be displayed whenever a user logs in a specified number of days before the user’s password time limit is reached.
- Specify a maximum number of days that an account can be inactive. If that number of days pass without the user logging in to the account, the user’s password will be locked.
- Specify an absolute date after which a user’s password cannot be used, thus denying the user the ability to log on to the system.

Keep in mind that users who are already logged in when the various maximums or dates are reached are not affected by the above tasks. They can continue to work as normal.

Password aging limitations and activities are only activated when a user logs in or performs one of the following operations:

- **login**
- **rlogin**
- **telnet**
- **ftp**

These password aging parameters are applied on user–by–user basis. You can have different password aging requirements for different users. (You can also set general default password aging parameters as described in the **/etc/security/user** file description.)

Password Privilege Expiration

You can set a specific date on which a user’s password privileges expires. When a user’s password privilege expires, that user can no longer have a valid password at all. In effect, this locks the user out of the system after the given date because after that date the user can no longer log in.

For example, if you specify an expire date of December 31, 1995, for a user named petew, on January 1, 1996 he will not be able to log in under that user ID regardless of what password he uses. After each login attempt he will receive a **Login incorrect** message.

Password Aging versus Expiration

Expiration of a user’s password privilege is not the same as password aging.

- **Password aging**. A password that has not been changed for longer than the aging time limit is sometimes referred to as an **expired password**. But that password can still be used to log in **one** more time. As part of that last login process the user is forced to choose a new password.
- **Expiration of password privilege**. When a user’s password **privilege** expires, the user cannot log in at all with **any** password.) In other words, it is the user’s permission to log in to the network that has expired.

Setting an Expiration Date

Password privilege expiration dates only take effect when the user logs in. If a user is **already** logged in, the expiration date has no affect until the user logs out or tries to use **rlogin** or **telnet** to connect to another machine at which time the user will not be able to log in again. Thus, if you are going to implement password privilege expiration dates, you should require your users to log out at the end of each day’s work session.

Note: Security Tip: Whenever possible, use Web-based System Manager or SMIT instead of **nistbladm** to set an expiration date. Web-based System Manager or SMIT are easier to use and provides less chance for error.

To set an expiration date with the **nistbladm** command:

```
nistbladm –m ’shadow=n:n:n:n:n:n6:n’ [name=login],passwd.org_dir
```

Where:

- login is the user’s login ID
- **n** indicates the values in the other fields of the shadow column.
- n6 is the date on which the user’s password privilege expires. This date is entered as a number of days since January 1, 1970. **n6** can be one of the following values:
  - **Minus one (–1**). A value of minus one (–1) turns off the expiration feature. If a user’s password has already expired, changing this value to –1 restores (un–expires) it. If you do not want to set any expiration date, type **–1** in this field.
  - **Greater than zero**. A value greater than zero sets the expiration date to that number of days since 1/1/70. If you enter today’s date or earlier, you immediately expire the user’s password.

For example, to specify an expiration date for the user **petew** of December 31, 1995 you would type:

```
station1% nistbladm –m ’shadow=n:n:n:n:n:9493:n’[name=petew],passwd.org_dir
```

Note: All of the fields must be filled in with valid values.

Turning Off Password Privilege Expiration

To turn off or deactivate password privilege expiration, you must use the **nistbladm** command to place a **–1** in this field. For example, to turn off privilege expiration for the user huck, you would type:

```
station1% nistbladm –m ’shadow=n:n:n:n:n:–1:n’ [name=huck],passwd.org_dir
```

Or you can use the **nistbladm** command reset the expiration date to some day in the future by entering a new number of days in the **n6** field.

Specifying Maximum Number of Inactive Days

You can set a maximum number of days that a user can go without logging in on a given machine. Once that number of days passes without the user logging in, that machine will no longer allow that user to log in. In this situation, the user will receive a **Login incorrect** message after each login attempt.

This feature is tracked on a machine–by–machine basis, not a network–wide basis. That is, in an NIS+ environment, you specify the number of days a user can go without logging in by placing an entry for that user in the passwd table of the user's home domain. That number applies for that user on all machines on the network. However, the date on which a user last logged in to a given machine is maintained on a machine–by–machine basis in the machine's **/var/adm/utmp** and/or **/var/adm/wtmp** (if it exists) file.

For example, suppose you specify a maximum inactivity period of 10 days for the user samh. On January 1, samh logs in to both machine–A and machine–B, and then logs off both machines. Four days later on January 4, samh logs in on machine–B and then logs out. Nine days after that on January 13, samh can still log –n to machine–B because only 9 days have elapsed since the last time he logged in on that machine, but he can no longer log in to machine–A because thirteen days have passed since his last log in on that machine.

Keep in mind that an inactivity maximum cannot apply to a machine the user has never logged in to. No matter what inactivity maximum has been specified or how long it has been since the user has logged in to some other machine, the user can always log in to a machine that the user has never logged in to before.

Note: Do not set inactivity maximums unless your users are instructed to log out at the end of each workday. The inactivity feature only relates to logins; it does not check for any other type of system use. If a user logs in and then leaves the system up and running at the end of each day, that user will soon pass the inactivity maximum because there has been no login for many days. When that user finally does reboot or log out, he or she will not be able to log in.

Note: Whenever possible, use Web-based System Manager or SMIT instead of **nistbladm** to set an inactivity maximum. Web-based System Manager or SMIT are easier to use and provides less chance for error. to set an inactivity maximum.

To set a login inactivity maximum, you must use the **nistbladm** command in the format:

```
nistbladm -m 'shadow=n:n:n:n:n5:n:n' [name=login],passwd.org_dir
```

Where:

- login is the user's login ID
- n indicates the values in the other fields of the shadow column.
- n5 is the number of days the user is allowed to go between logins. **Inactive** can be one of the following values:
  - **Minus one (–1**). A value of minus one (–1) turns off the inactivity feature. The user can be inactive for any number of days without losing login privileges. This is the default.
  - **Greater than zero**. A value greater than zero sets the maximum inactive period to that number of days.

  For example, to specify that the user **samh** must log in at least once every seven days, you would type:

  ```
  station1% nistbladm -m 'shadow=n:n:n:n:n:7:n:n'[name=samh],passwd.org_dir
  ```

  To clear an inactivity maximum and allow a user who has been prevented from logging in to log in again, use **nistbladm** to set the inactivity value to –1.

Setting Password Aging Criteria for Multiple Users

You can use the **nistbladm** command globally specify password max, min, warn, inactive, and expire, values for all principals listed in a given passwd table.

To globally change password aging values for all users listed in a given password table, you use the **nistbladm** command without an indexed entry between the square brackets. For example, to globally set a minimum of 7 days, a maximum of 30 days, a warning period of 5 days, and no inactivity limit or expire date you would type:

```
station1% nistbladm –m 'shadow=n:7:30:5:–1:–1:0' [],passwd.org_dir
```

You can also use the **nistbladm** command to turn off password aging for all users in a given password table by globally setting their max value to –1 or 0.

Note: The value you enter in the lastchange field (the first field) will be applied to all the users. In effect, you will be resetting everyone's last change date to that value.

Specifying Password Criteria and Defaults

Read the /**etc**/**security**/**user** file description for information on password–related defaults and general criteria that you can specify.

Password Failure Limits

You can specify a number–of–tries limit or an amount–of–time limit (or both) for a user's attempt to change passwords. These limits are specified by adding arguments when starting the **rpc.nispasswdd** daemon.

Limiting the number of attempts or setting a time frame provides a limited (but not foolproof) defense against unauthorized persons attempting to change a valid password to one that they discover through trial and error.

Maximum Number of Tries

Security Tip: To set the maximum number of times a user can try to change a password without succeeding, use the **–a** number argument with **rpc.nispasswdd,** where number is the number of allowed tries. (You must have superuser privileges on the NIS+ master server to run **rpc.nispasswdd**.)

For example, to limit users to no more than four attempts (the default is 3), you would type:

```
   station1# rpc.nispasswdd –a 4
```

In this case, if a user's fourth attempt at logging in is unsuccessful, the message **Too many failures – try later** is displayed. No further attempts are permitted for that user ID until a specified period of time has passed.

Maximum Login Time Period

To set the maximum amount a time a user can take to successfully change a password, use the **–c** minutes argument with **rpc.nispasswdd,** where minutes is the number of minutes a user has to log in. (You must have superuser privileges on the NIS+ master server to **run rpc.nispasswdd**.)

For example, to specify that users must successfully log in within 2 minutes, you would type:

```
   station1# rpc.nispasswdd –c 2
```

In this case, if a user is unable to successfully change a password within 2 minutes, the message is displayed at the end of the two–minute period. No further attempts are permitted for that user ID until a specified period of time has passed.

Administering NIS+ Groups

A NIS+ group is a set of NIS+ principals. NIS+ groups are used to assign a set of access rights to NIS+ objects to the members of the group.

This section describes how to use NIS+ group administration commands to perform the following tasks:

- Specifying Group Members, on page 6-45
- Listing the Object Properties of a Group, on page 6-46
- Creating an NIS+ Group, on page 6-47
- Deleting an NIS+ Group, on page 6-48
- Adding Members to an NIS+ Group, on page 6-48
- Listing the Members of an NIS+ Group, on page 6-49
- Removing Members From an NIS+ Group, on page 6-49
- Testing for Membership in an NIS+ Group, on page 6-49

Related Commands

The **nisgrpadm** command performs most group administration tasks, but several other commands affect groups as well:

Table 47. Commands That Affect Groups

Command	Description
nissetup	Creates, among other things, the directory in which a domain's groups are stored: **groups_dir**.
nisls	Lists the contents of the **groups_dir** directory; in other words, all the groups in a domain.
nischgrp	Changes or assigns a group to any NIS+ object.
nisdefaults	Lists, among other things, the group that will be assigned to any new NIS+ object.

Specifying Group Members

NIS+ groups can have three types of members: explicit, implicit, and recursive; and three types of nonmembers, also explicit, implicit, and recursive. These member types are used when adding or removing members of a group.

Member Types

Explicit An individual principal. Identified by principal name. The name does not have to be fully qualified if entered from its default domain.

Implicit All the NIS+ principals who belong to an NIS+ domain. They are identified by their domain name, preceded by the * symbol and a dot. The operation you select applies to all the members in the group.

Recursive All the NIS+ principals that are members of another NIS+ group. They are identified by their NIS+ group name, preceded by the @ symbol. The operation you select applies to all the members in the group.

Nonmember Types

NIS+ groups also accept nonmembers in all three categories: explicit, implicit, and recursive. Nonmembers are principals specifically excluded from a group that they otherwise would be part of.

Nonmembers are identified by a minus sign in front of their name:

Explicit–nonmember
Identified by a minus sign in front of the principal name.

Implicit nonmember
Identified by a minus sign, * symbol, and dot in front of the domain name.

Recursive nonmember
Identified by a minus sign and @ symbol in front of the group name.

Using Member Types

Note: The order in which inclusions and exclusions are entered does not matter. Exclusions always take precedence over inclusions. Thus, if a principal is a member of an included implicit domain and **also** a member of an excluded recursive group, then that principal is not included.

Thus, when using the **nisgrpadm** command, you can specify group members and nonmembers as shown in the following table:

Table 48. Specifying Group Members and Nonmembers

Type of member	Syntax
Explicit member	*username.domain*
Implicit member	***.***domain*
Recursive member	*@groupname.domain*
Explicit nonmember	*–username.domain*
Implicit nonmember	–***.***domain*
Recursive nonmember	*–@groupname.domain*

Using niscat With Groups

The **niscat –o** command can be used to list the object properties and membership of an NIS+ group.

Listing the Object Properties of a Group

To list the object properties of a group, you must have read access to the **groups_dir** directory in which the group is stored. Use **niscat –o** and the group's fully qualified name, which must include its **groups_dir** subdirectory:

```
niscat –o group-name.groups_dir.domain-name
```

For example:

```
rootmaster# niscat –o sales.groups_dir.wiz.com.
Object Name   : sales
Owner         : rootmaster.wiz.com.
Group         : sales.wiz.com.
Access Rights : ----rmcdr---r---
Time to Live  : 1:0:0
Object Type   : GROUP
Group Flags   :
Group Members : rootmaster.wiz.com.
                topadmin.wiz.com.
                @.admin.wiz.com.
                *.sales.wiz.com.
```

Several of the group's properties are inherited from the **NIS_DEFAULTS** environment variable, unless they were overridden when the group was created. The group flags field is currently unused. In the list of group members, the * symbol identifies member domains and the @ symbol identifies member groups.

The nisgrpadm Command

The **nisgrpadm** command creates, deletes, and performs miscellaneous administration operations on NIS+ groups. To use **nisgrpadm**, you must have access rights appropriate for the operation,

Table 49. Rights Required for **nisgrpadm** Command

This Operation	Requires This Access Right	To This Object
Create a group	Create	**groups_dir** directory
Destroy a group	Destroy	**groups_dir** directory
List the Members	Read	the group object
Add Members	Modify	the group object
Remove Members	Modify	the group object

The **nisgrpadm** has two main forms, one for working with groups and one for working with group members.

To create or delete a group, or to lists its members use this form:

```
nisgrpadm –c group–name.domain–name
nisgrpadm –d group–name
nisgrpadm –l group–name
```

To add or remove members, or determine if they belong to the group use this form (where member... can be any combination of the six membership types):

```
nisgrpadm –a group–name member...
nisgrpadm –r group–name member...
nisgrpadm –t group–name member...
```

All operations except create (**–c**) accept a partially qualified group names. However, even for the **–c** option, **nisgrpadm** does not require the use of **groups_dir** in the *group–name* argument. In fact, it will not accept it.

Creating an NIS+ Group

To create an NIS+ group, you must have create rights to the **groups_dir** directory of the group's domain. Use the **–c** option and a fully qualified group name:

```
nisgrpadm –c group–name.domain–name
```

A newly created group contains no members. See Adding Members to an NIS+ Group, on page 6-48 for information on how to specify who belongs to a group.

The example below creates three groups named admin. The first is in the Wiz.com. domain, the second in sales.wiz.com., and the third in manf.wiz.com. All three are created on the master server of their respective domains.

```
rootmaster# nisgrpadm –c admin.wiz.com.
Group admin.wiz.com. created.
salesmaster# nisgrpadm –c admin.sales.wiz.com.
Group admin.sales.wiz.com. created.
engmaster# nisgrpadm –c admin.manf.wiz.com.
Group admin.manf.wiz.com. created.
```

The group you create will inherit all the object properties specified in the **NIS_DEFAULTS** variable; that is, its owner, owning group, access rights, and time–to–live. You can view these defaults by using the **nisdefaults** command (described in Administering NIS+ Access Rights, on page 6-21). Used without options, it provides this output:

```
rootmaster# nisdefaults
Principal Name : rootmaster.wiz.com.
Domain Name    : Wiz.com.
Host Name      : rootmaster.wiz.com.
Group Name     :
Access Rights  : ----rmcdr---r---
Time to live   : 12:0:0
Search Path    : Wiz.com.
```

The owner is listed in the Principal Name: field. The owning group is listed only if you have set the **NIS_GROUP** environment variable.

Of course, you can override any of these defaults at the time you create the group by using the **–D** option:

```
salesmaster# nisgrpadm –D group=special.sales.wiz.com.–c
admin.sales.wiz.com.
Group admin.sales.wiz.com. created.
```

Deleting an NIS+ Group

To delete an NIS+ group, you must have destroy rights to the **groups_dir** directory in the group's domain. Use the **–d** option:

```
nisgrpadm –d group-name
```

If the default domain is set properly, you do not have to fully–qualify the group name. However, you should check first (use **nisdefaults**), because you could unintentionally delete a group in another domain. The example below deletes the `test.sales.wiz.com.` group.

```
salesmaster% nisgrpadm –d test.sales.wiz.com.
Group 'test.sales.wiz.com.' destroyed.
```

Adding Members to an NIS+ Group

To add members to an NIS+ group you must have modify rights to the group object. Use the **–a** option:

```
nisgrpadm –a group-name    members. . .
```

As described in Specifying Group Members, on page 6-45, you can add principals (explicit members), domains (implicit members), and groups (recursive members). You do not have to fully qualify the name of the group or the name of the members who belong to the default domain. This example adds the NIS+ principals panza and valjean, both from the default domain, sales.wiz.com., and the principal **makeba**, from the manf.wiz.com. domain, to the group Ateam.sales.wizCom.

```
client% nisgrpadm –a Ateam panza valjean makeba.manf.wiz.com.
Added panza.sales.wiz.com to group Ateam.sales.wiz.com
Added valjean.sales.wiz.com to group Ateam.sales.wiz.com
Added makeba.manf.wiz.com to group Ateam.sales.wiz.com
```

To verify the operation, use the **nisgrpadm –l** option. Look for the members under the Explicit members heading.

This example adds all the NIS+ principals in the Wiz.com. domain to the Staff.wiz.com. group. It is entered from a client in the wiz.com. domain. Note the * symbol and the dot in front of the domain name.

```
client% nisgrpadm –a Staff *.wiz.com.
Added *.wiz.com. to group Staff.manf.wiz.com.
```

This example adds the NIS+ group admin.wiz.com. to the admin.manf.wiz.com. group. It is entered from a client of the manf.wiz.com. domain. Note the @ symbol in front of the group name.

```
client% nisgrpadm -a admin @admin.wiz.com.
Added @admin.wiz.com. to group admin.manf.wiz.com.
```

Listing the Members of an NIS+ Group

To list the members of an NIS+ group, you must have read rights to the group object. Use the **–l** option:

```
nisgrpadm -l group-name
```

This example lists the members of the admin.manf.wiz.com. group. It is entered from a client in the manf.wiz.com. group:

```
client% nisgrpadm -l admin
Group entry for admin.manf.wiz.com. group:
    No explicit members
    No implicit members:
    Recursive members:         @admin.wiz.com.
    No explicit nonmembers
    No implicit nonmembers
    No recursive nonmembers
```

Removing Members From an NIS+ Group

To remove members from an NIS+ group, you must have modify rights to the group object. Use the **–r** option:

```
nisgrpadm -r group-name   members. . .
```

This example removes the NIS+ principals **allende** and **hugo.manf.wiz.com**. from the Ateam.sales.wiz.com group. It is entered from a client in the sales.wiz.com. domain:

```
client% nisgrpadm -r Ateam allende hugo.manf.wiz.com.
Removed allende.sales.wiz.com. from group Ateam.sales.wiz.com.
Removed hugo.manf.wiz.com. from group Ateam.sales.wiz.com.
```

This example removes the admin.wiz.com. group from the admin.manf.wiz.com. group. It is entered from a client in the manf.wiz.com. domain:

```
client% nisgrpadm -r admin @admin.wiz.com.
Removed @admin.wiz.com. from group admin.manf.wiz.com.
```

Testing for Membership in an NIS+ Group

To find out whether an NIS+ principal is a member of a particular NIS+ group you must have read access to the group object. Use the **–t** option:

```
nisgrpadm -t group-name   members. . .
```

This example tests whether the NIS+ principal topadmin belongs to the admin.wiz.com. group. It is entered from a client in the Wiz.com. domain.

```
client% nisgrpadm -t admin topadmin
topadmin.wiz.com. is a member of group admin.wiz.com.
```

This example tests whether the NIS+ principal jo, from the sales.wiz.com. domain, belongs to the admin.sales.wiz.com. group. It is entered from a client in the wiz.com. domain.

```
client% nisgrpadm -t admin.sales.wiz.com. jo.sales.wiz.com.
jo.sales.wiz.com. is a member of group admin.sales.wiz.com.
```

Administering NIS+ Directories

This section describes how to use the NIS+ directory administration commands to perform the following tasks:

- Listing the Object Properties of a Directory, on page 6-50
- Listing the Contents of a Directory—Terse, on page 6-51
- Listing the Contents of a Directory—Verbose, on page 6-51
- Creating a Directory, on page 6-52
- Adding a Replica to an Existing Directory, on page 6-53
- Removing a Directory, on page 6-53
- Disassociating a Replica From a Directory, on page 6-54
- Removing Nondirectory Objects, on page 6-54
- Starting a NIS–Compatible Daemon, on page 6-55
- Starting a DNS–Forwarding NIS–Compatible Daemon, on page 6-55
- Stopping the NIS+ Daemon, on page 6-55
- Initializing a Client, on page 6-55
- Initializing the Root Master Server, on page 6-56
- Starting the Cache Manager, on page 6-56
- Displaying the Contents of the NIS+ Cache, on page 6-57

Using the niscat Command With Directories

The **niscat –o** command can be used to list the object properties of an NIS+ directory. To use it, you must have read access to the directory object itself.

Listing the Object Properties of a Directory

To list the object properties of a directory, use **niscat –o** and the directory's name:

```
niscat –o directory-name
```

For example:

```
rootmaster# niscat –o wiz.com.
Object Name   : wiz
Owner         : rootmaster.wiz.com.
Group         :
Access Rights : r---rmcdr---r---
Time to Live  : 24:0:0
Object Type   : DIRECTORY
.
.
.
```

The nisls Command

The **nisls** command lists the contents of an NIS+ directory. To use it, you must have read rights to the directory object.

To display in terse format, use:

```
nisls [–dgLmMR] directory-name
```

To display in verbose format, use:

```
nisls –l [–gm] [–dLMR] directory-name
```

Table 50. Options for the **nisls** Command

Option	Purpose
–d	Directory object. Instead of listing a directory's contents, treat it like another object.
–L	Links. If the directory name is actually a link, the command follows the link and displays information about the linked directory.
–M	Master. Get the information from the master server only. Although this provides the most up–to–date information, it may take longer if the master server is busy.
–R	Recursive. List directories recursively. That is, if a directory contains other directories, their contents are displayed as well.
–l	Long. Display information in long format. Long format displays an object's type, creation time, owner, and access rights.
–g	Group. When displaying information in long format, display the directory's group owner instead of its owner.
–m	Modification time. When displaying information in long format, display the directory's modification time instead of its creation time.

Listing the Contents of a Directory—Terse

To list the contents of a directory in the default short format, use one or more of the options listed below and a directory name. If you do not supply a directory name, NIS+ will use the default directory.

```
nisls [–dLMR]
nisls [–dLMR] directory–name
```

For example, this instance of **nisls** is entered from the root master server of the root domain wiz.com.:

```
rootmaster% nisls
wiz.com.:
org_dir
groups_dir
```

Here is another example entered from the root master server:

```
rootmaster% nisls –R Sales.wiz.com.
Sales.wiz.com.:
org_dir
groups_dir

groups_dir.Sales.wiz.com.:
admin

org_dir.Sales.wiz.com.:
auto_master
auto_home
bootparams
cred
.
.
.
```

Listing the Contents of a Directory—Verbose

To list the contents of a directory in the verbose format, use the **–l** option and one or more of the options listed below. The **–g** and **–m** options modify the attributes that are displayed. If you don't supply a directory name, NIS+ will use the default directory.

```
nisls –l [–gm] [–dLMR]
nisls –l [–gm] [–dLMR] directory–name
```

Here is an example, entered from the master server of the root domain wiz.com.:

```
rootmaster% nisls -l
wiz.com.:
D r---rmcdr---r--- rootmaster.wiz.com. date org_dir
D r---rmcdr---r--- rootmaster.wiz.com. date groups_dir
```

The nismkdir Command

The **nismkdir** command creates a nonroot NIS+ directory and associates it with a master server. (To create a root directory, use the **nisinit –r** command.) The **nismkdir** command can also be used to add a replica to an existing directory.

This section desribes how to add a nonroot directory and its master server to an existing system using the **nismkdir** command. An easier way to do this is with the **nisserver** script, described in NIS+ Setup Scripts.

To create a directory, use:

```
nismkdir [–m master-server] directory-name
```

To add a replica to an existing directory, use:

```
nismkdir –s replica-server directory-name
nismkdir –s replica-server org_dir. directory-name
nismkdir –s replica-server groups_dir. directory-name
```

Creating a Directory

To create a directory, you must have create rights to its parent directory on the domain master server. First use the **–m** option to identify the master server and then the **–s** option to identify the replica, use:

```
nismkdir –m master directory
nismkdir –s replica directory
```

Attention: Always run **nismkdir** on the master server. Never run **nismkdir** on the replica machine. Running **nismkdir** on a replica creates communications problems between the master and the replica.

The following example creates the `Sales.wiz.com.` directory and specifies its master server, `smaster.wiz.com.` and its replica, `rep1.wiz.com.` It is entered from the root master server.

```
rootmaster% nismkdir –m smaster.wiz.com. Sales.wiz.com.
rootmaster% nismkdir –m smaster.wiz.com. org_dir.Sales.wiz.com.
rootmaster% nismkdir –m smaster.wiz.com. groups_dir.Sales.wiz.com.
rootmaster% nismkdir –s rep1.wiz.com. Sales.wiz.com.
rootmaster% nismkdir –s rep1.wiz.com. org_dir.Sales.wiz.com.
rootmaster% nismkdir –s rep1.wiz.com. groups_dir.Sales.wiz.com.
```

The **nismkdir** command allows you to use the parent directory's servers for the new directory instead of specifying its own. However, this should not be done except in the case of small networks. See the following examples:

- The first example creates the `Sales.wiz.com.` directory and associates it with its parent directory's master and replica servers.

```
rootmaster% nismkdir Sales.wiz.com
```

- The second example creates the `Sales.wiz.com.` directory and specifies its own master server, `smaster.wiz.com.`

```
rootmaster% nismkdir –m smaster.wiz.com. Sales.wiz.com.
```

Since no replica server is specified, the new directory will have only a master server until you use **nismkdir** again to assign it a replica. If the `Sales.wiz.com.` domain already existed, the **nismkdir** command as shown above would have made `salesmaster.wiz.com.` its new master server and would have relegated its old master server to a replica.

Adding a Replica to an Existing Directory

This section describes how to add a replica server to an existing system using the nismkdir command. An easier way to do this is with the **nisserver** script, described in NIS+ Setup Scripts.

To assign a new replica server to an existing directory, use **nismkdir** on the master server with the **–s** option and the name of the existing directory, **org_dir**, and **groups_dir**:

```
nismkdir –s replica-server existing-directory-name
nismkdir –s replica-server org_dir. existing-directory-name
nismkdir –s replica-server groups_dir. existing-directory-name
```

The **nismkdir** command realizes that the directory already exists, so it does not recreate it. It only assigns it the additional replica. Here is an example in which `rep1` is the name of the new replica machine:

```
rootmaster% nismkdir –s rep1.wiz.com. wiz.com.
rootmaster% nismkdir –s rep1.wiz.com. org_dir.wiz.com.
rootmaster% nismkdir –s rep1.wiz.com. groups_dir.wiz.com.
```

Note that you cannot assign a server to support its parent domain—unless, of course, it belongs to the root domain.

Attention: Always run **nismkdir** on the master server. Never run **nismkdir** on the replica machine. Running **nismkdir** on a replica creates communications problems between the master and the replica.

After running the three iterations of **nismkdir** as shown above, you need to run **nisping** from the master server on the three directories:

```
rootmaster# nisping wiz.com.
rootmaster# nisping org_dir.wiz.com.
rootmaster# nisping group_dir.wiz.com.
```

You should see results similar to these:

```
rootmaster# nisping wiz.com.
Pinging replicas serving directory wiz.com. :
Master server is rootmaster.wiz.com.
        Last update occurred at Wed Nov 18 19:54:38 1995
Replica server is rep1.wiz.com.
        Last update seen was Wed Nov 18 11:24:32 1995
        Pinging ... rep1.wiz.com.
```

It is good practice to include **nisping** commands for each of these three directories in the master server's **/etc/crontab** file so that each directory is pinged at least once every 24 hours after being updated.

The nisrmdir Command

The **nisrmdir** command can remove a directory or simply dissociate a replica server from a directory. When it removes a directory, NIS+ first disassociates the master and replica servers from the directory, and then removes the directory. To remove the directory, you must have destroy rights to its parent directory. To dissociate a replica server from a directory, you must have modify rights to the directory.

Removing a Directory

To remove an entire directory and dissociate its master and replica servers, use the **nisrmdir** command without any options:

```
nisrmdir directory-name
```

The following example removes the `manf.wiz.com.` directory from beneath the `wiz.com.` directory:

```
rootmaster% nisrmdir manf.wiz.com.
```

Disassociating a Replica From a Directory

To dissociate a replica server from a directory, use the **nisrmdir** command with the **–s** option:

```
nisrmdir –s servername directory
```

The following example disassociates the `manfreplica1` server from the `manf.wiz.com.` directory:

```
rootmaster% nisrmdir –s manfreplica1 manf.wiz.com.
```

The nisrm Command

The **nisrm** command is similar to the standard **rm** system command. It removes any NIS+ object from the namespace, except directories and nonempty tables. To use the **nisrm** command, you must have destroy rights to the object. However, if you don't, you can use the **–f** option, which tries to force the operation in spite of permissions.

You can remove group objects with the **nisgrpadm –d** command (see Deleting an NIS+ Group), and you can empty tables with **nistbladm –r** or **nistbladm –R**.

To remove a nondirectory object, use:

```
nisrm [–if] object–name
```

Table 51. **nisrm** Syntax Options

Option	Purpose
–i	Inquire. Asks for confirmation prior to removing an object. If the object–name you provide is not fully qualified, this option is used automatically.
–f	Force. Attempts to force a removal even if you don't have the proper permissions. It attempts to change the permission by using the **nischmod** command, and then tries to remove the object again.

Removing Nondirectory Objects

To remove nondirectory objects, use the **nisrm** command and provide the object names:

```
nisrm object–name...
```

The following example removes a group and a table from the namespace:

```
rootmaster% nisrm –i admins.wiz.com. groups.org_dir.wiz.com.
Remove admins.wiz.com.? y
Remove groups.org_dir.wiz.com.? y
```

The rpc.nisd Command

The **rpc.nisd** command starts the NIS+ daemon. The daemon can run in NIS–compatibility mode, which enables it to answer requests from NIS clients as well. You do not need any access rights to start the NIS+ daemon, but you should be aware of all its prerequisites and related tasks.

By default, the NIS+ daemon starts with security level 2.

To start the daemon, use:

```
startsrc –s rpc.nisd
```

To start the daemon in NIS–compatibility mode, use:

```
startsrc –s rpc.nisd –a '–Y'
```

To start an NIS–compatible daemon with DNS forwarding capabilities, use:

```
startsrc –s rpc.nisd –a '–Y –B'
```

Table 52. Other **rpc.nisd** Syntax Options

Option	Purpose
–S security–level	Specifies a security level, where **0** means no NIS+ security and **2** provides full NIS+ security. (Level 1 is not supported.)
–F	Forces a checkpoint of the directory served by the daemon. This has the side effect of emptying the directory's transaction log and freeing disk space.

To start the NIS+ daemon on any server, use the command without options:

```
startsrc –s rpc.nisd
```

The daemon starts with security level 2, which is the default.

To start the daemon with security level 0, use the **–S** flag:

```
startsrc –s rpc.nisd –a '–S 0'
```

Starting a NIS–Compatible Daemon

You can start the NIS+ daemon in NIS–compatibility mode in any server, including the root master. Use the **–Y** (uppercase) option:

```
startsrc –s rpc.nisd –a "–Y"
```

If the server is rebooted, the daemon will not restart in NIS–compatibility mode unless you also edit the server's /**etc**/**rpc.nfs** file with **mk_nisd –I –y –b**.

Starting a DNS–Forwarding NIS–Compatible Daemon

You can add DNS forwarding capabilities to an NIS+ daemon running in NIS–compatibility mode by adding the **–B** option to **rpc.nisd**:

```
startsrc –s rpc.nisd –a "–Y –B"
```

If the server is rebooted, the daemon will not restart in DNS–forwarding NIS–compatibility mode unless you also edit the server's /**etc**/**rpc.nfs** file with **mk_nisd –I –y –b**.

Stopping the NIS+ Daemon

To stop the NIS+ daemon, whether it is running in normal or NIS–compatibility mode, use:

```
rootmaster# stopsrc –s rpc.nisd
```

The nisinit Command

This section describes how to initialize a workstation client using the **nisinit** command. An easier way to do this is with the **nisclient** script, described in NIS+ Setup Scripts.

The **nisinit** command initializes a workstation to be an NIS+ client. As with the **rpc.nisd** command, you do not need any access rights to use the **nisinit** command, but you should be aware of its prerequisites and related tasks.

To initialize a client, use:

```
nisinit –c –B
nisinit –c –H hostname
nisinit –c –C filename
```

To initialize a root master server, use:

```
nisinit –r
```

Initializing a Client

You can initialize a client in three different ways:

- By host name
- By broadcast

- By cold–start file

Each way has different prerequisites and associated tasks. For instance, before you can initialize a client by host name, the client's **/etc/hosts** file must list the host name you will use and the **irs.conf** file must have **files** as the first choice on the **hosts** line. Following is a summary of the steps that use the **nisinit** command.

To initialize a client by host name, use the **–c** and **–H** options, and include the name of the server from which the client will obtain its cold–start file:

```
nisinit –c –H hostname
```

To initialize a client by cold–start file, use the **–c** and **–C** options, and provide the name of the cold–start file:

```
nisinit –c –C filename
```

To initialize a client by broadcast, use the **–c** and **–B** options:

```
nisinit –c –B
```

Initializing the Root Master Server

To initialize the root master server, use the **nisinit –r** command:

```
nisinit –r
```

The nis_cachemgr Command

The **nis_cachemgr** command starts the NIS+ cache manager program, which should run on all NIS+ clients. The cache manager maintains a cache of location information about the NIS+ servers that support the most frequently used directories in the namespace, including transport addresses, authentication information, and a time–to–live value.

At start–up the cache manager obtains its initial information from the client's cold–start file, and downloads it into the **/var/nis/NIS_SHARED_DIRCACHE** file.

The cache manager makes requests as a client workstation. Make sure the client workstation has the proper credentials, or instead of improving performance, the cache manager will degrade it.

Starting the Cache Manager

To start the cache manager, enter the **nis_cachemgr** command (with or without the **–i** option):

```
client% startsrc –s nis_cachemgr
client% startsrc –s nis_cachemgr –a "–i"
```

Without the **–i** option, the cache manager is restarted but it retains the information in the **/var/nis/NIS_SHARED_DIRCACHE** file. The information in the cold–start file is simply appended to the existing information in the file. The **–i** option clears the cache file and re–initializes it from the contents of the client's cold–start file.

To stop the cache manager, kill it using **stopsrc –s nis_cachemgr**.

The nisshowcache Command

The **nisshowcache** command displays the contents of a client's directory cache.

Displaying the Contents of the NIS+ Cache

The **nisshowcache** command is located in /**usr**/**lib**/**nis**. It displays only the cache header and the directory names. Here is an example entered from the root master server:

```
rootmaster# /usr/lib/nis/nisshowcache -v
 Cold Start directory:
 Name : wiz.com.
 Type : NIS
 Master Server :
   Name       : rootmaster.wiz.com.
   Public Key : Diffie-Hellman (192 bits)
   Universal addresses (3)
   .
   .
   .
 Replicate:
   Name       : rootreplica1.wiz.com.
   Public Key : Diffie-Hellman (192 bits)
   Universal addresses (3)
   .
   .
   .
Time to live : 12:0:0
Default Access Rights :
```

The nisping Command

The **nisping** command pings replica servers, telling them to ask the master server for updates immediately. (The replicas normally wait a couple of minutes before executing this request.) Before pinging, the command checks the time of the last update received by each replica. If it is the same as the last update sent by the master, it does not ping the replica.

The **nisping** command can also checkpoint a directory. This consists of telling each server in the directory, including the master, to update its information on disk from the domain's transaction log.

To display the time of the last update, use:

```
/usr/lib/nis/nisping -u [domain]
```

To ping replicas, use:

```
/usr/lib/nis/nisping [domain]
/usr/lib/nis/nisping -H hostname [domain]
```

To checkpoint a directory, use:

```
/usr/lib/nis/nisping -C hostname [domain]
```

For more information, read the **nisping** command description.

The nislog Command

The **nislog** command displays the contents of the transaction log.

```
/usr/sbin/nislog
/usr/sbin/nislog –h [number]
/usr/sbin/nislog –t [number]
```

Table 53. Options for the **nislog** Command

Option	Purpose
–h [num**]**	Display transactions starting with the head (beginning) of the log. If the number is omitted, the display begins with the first transaction. If the number 0 is entered, only the log header is displayed.
–t [num**]**	Display transactions starting backward from the end (tail) of the log. If the number is omitted, the display begins with the last transaction. If the number 0 is entered, only the log header is displayed.
–v	Verbose mode

For more information, see the **nislog** command description.

The nischttl Command

The **nischttl** command changes the time–to–live value of objects or entries in the namespace. This time–to–live value is used by the cache manager to determine when to expire a cache entry. You can specify the time–to–live in total number of seconds or in a combination of days, hours, minutes, and seconds.

The time–to–live values you assign objects or entries should depend on the stability of the object. If an object is prone to frequent change, give it a low time–to–live value. If it is steady, give it a high one. A high time–to–live is a week; a low one is less than a minute. Password entries should have time–to–live values of about 12 hours to accommodate one password change per day. Entries in tables that don't change much, such as those in the RPC table, can have values of several weeks.

To change the time–to–live of an object, you must have modify rights to that object. To change the time–to–live of a table entry, you must have modify rights to the table, entry, or columns you wish to modify.

To display the current time–to–live value of an object or table entry, use the **nisdefaults –t** command

To change the time–to–live value of objects, use:

```
nischttl time-to-live object-name
nischttl [-L] time-to-live object-name
```

To change the time–to–live value of entries, use:

```
nischttl time-to-live [column=value,...], table-name
nischttl [-ALP] time-to-live [column=value,...],table-name
```

Where time–to–live is expressed as:

- **Number of seconds.** A number with no letter is interpreted as a number of seconds. Thus, **1234** for TTL would be interpreted as 1234 seconds. A number followed by the letter **s** is also interpreted as a number of seconds. Thus, **987s** for TTL would be interpreted as 987 seconds. When seconds are specified in combination with days, hours, or minutes, you must use the letter **s** to identify the seconds value.
- **Number of minutes.** A number followed by the letter **m** is interpreted as a number of minutes. Thus, **90m** for TTL would be interpreted as 90 minutes.
- **Number of hours.** A number followed by the letter **h** is interpreted as a number of hours. Thus, **9h** for TTL would be interpreted as 9 hours.

- **Number of days.** A number followed by the letter **d** is interpreted as a number of days. Thus, **7d** for TTL would be interpreted as 7 days.

These values may be used in combination. For example, a TTL value of **4d3h2m1s** would specify a time to live of four days, three hours, two minutes, and one second.

Table 54. **nischttl** Syntax Options

Option	Purpose
–A	All. Apply the change to all the entries that match the column**=** value specifications that you supply**.**
–L	Links. Follow links and apply the change to the linked object or entry rather than the link itself.
–P	Path. Follow the path until there is one entry that satisfies the condition.

Administering NIS+ Tables

This section describes how to use the following NIS+ table administration commands:

- **nistbladm Command**, on page 6-60
- **niscat**, on page 6-60
- **nismatch**, on page 6-61
- **nisgrep**, on page 6-61
- **nisln**, on page 6-63
- **nissetup**, on page 6-63
- **nisaddent**, on page 6-64

Using the nistbladm Command

The **nistbladm** command is the primary NIS+ table administration command. With it, you can create, modify, and delete NIS+ tables and entries. To create a table, its directory must already exist. To add entries to the table, the table and columns must already be defined.

To create a table, you must have create rights to the directory under which you will create it. To delete a table, you must have destroy rights to the directory. To modify the contents of a table, whether to add, change, or delete entries, you must have modify rights to the table or the entries.

To create a table, use:

```
nistbladm –c table-type columnspec...tablename
```

To delete a table, use:

```
nistbladm –d tablename columnspec ::=  column=[CSI,rights]
```

To add entries, use:

```
nistbladm –a
nistbladm –A entry
```

To modify entries, use:

```
nistbladm –m new-entry old-entry
```

To remove entries, use:

```
nistbladm –r
nistbladm –R [entry or table] entry::=column=value ... tablename
[column=value,...],tablename
```

The **columnspec** syntax is explained in the command description for **nistbladm**.

Using the niscat Command

The **niscat** command displays the contents of an NIS+ table. However, you can also use it to display the object properties of the table. You must have read rights to the table, entries, or columns that you wish to display.

To display the contents of a table, use:

```
niscat [–hM] tablename
```

To display the object properties of a table, use:

```
niscat –o tablename
niscat –o entry
```

Table 55. **niscat** Syntax Options

Option	Description
–h	Header. Displays a header line above the table entries, listing the name of each column.
–M	Master. Displays only the entries of the table stored on the Master server. This ensures you get the most up–to–date information and should be used only for debugging.
–o	Object. Displays object information about the table, such as column names, properties, and servers.

For more information, see the command description for **niscat**.

Using the nismatch and nisgrep Commands

The **nismatch** and **nisgrep** commands search through NIS+ tables for entries that match a particular string or regular expression, respectively. They display either the entries themselves or a count of how many entries matched. The differences between the **nismatch** and **nisgrep** commands are highlighted in following table.

Table 56. Comparison of **nisgrep** and **nismatch**

Characteristics	nismatch	nisgrep
Search criteria	Accepts text only	Accepts regular expressions
Speed	Faster	Slower
Searches through	Searchable columns only	All columns, whether searchable or not
Syntax of search criteria	*column=string* ... *tablename* [*column=string*,...],*tablename*	*column=exp* ... *tablename*

The tasks and examples in this section describe the syntax for both commands.

To use either command, you must have read access to the table you are searching through.

The examples in this section are based on the values in the following table, named depts.wiz.com. Only the first two columns are searchable.

Table 57. depts.wiz.com. Example Table

Name (S)	Site (S)	Manager
R&D	SanFran	kuznetsov
Sales	SanFran	jhill
Manf–1	Emeryville	hosteen
Manf–2	Sausalito	lincoln
Shipping–1	Emeryville	tsosulu
Shipping–2	Sausalito	katabami
Service	Sparks	franklin

About Regular Expressions

Regular expressions are combinations of text and symbols that you can use to search for special configurations of column values. For example, the regular expression **’Hello’** searches for a value that begins with **Hello**:

```
rootmaster% nisgrep –h greeting=’Hello’ phrases.wiz.com.
```

The regular expression symbols are summarized in the following table.

Table 58. Regular Expression Symbols

Symbol	Description
^*string*	Find a value that begins with *string*.
string$	Find a value that ends with *string*.
.	Find a value that has a number characters equal to the number of periods.
[*chars*]	Find a value that contains any of the characters in the brackets.
**expr*	Find a value that has zero or more matches of the **expr.**
+	Find something that appears one or more times.
?	Find any value.
\'*s–char*'	Find a special character, such as **?** or **$**.
x | *y*	Find a character that is either *x* or *y.*

Syntax

To search through the first column, use:

```
nismatch string tablename
nisgrep reg-exp tablename
```

To search through a particular column, use:

```
nismatch column=string tablename
nisgrep column=reg-exp tablename
```

To search through multiple columns, use:

```
nismatch column=string ... tablename
nismatch [column=string,...],tablename
nisgrep  column=reg-exp ... tablename
```

Table 59. **nismatch** and **nisgrep** Syntax Options

Option	Description
–c	Count. Instead of the entries themselves, displays a count of the entries that matched the search criteria.
–h	Header. Displays a header line above the entries, listing the name of each column.
–M	Master. Displays only the entries of the table stored on the master server. This ensures you get the most up–to–date information and should be used only for debugging.

Searching the First Column

To search for a particular value in the first column of a table, simply enter the first column value and a **tablename**. In **nismatch**, the value must be a string. In **nisgrep**, the value must be a regular expression.

```
nismatch [-h] string tablename
nisgrep [-h] reg-expression tablename
```

This example searches through the **depts** table for all the entries whose first column has a value of **R&D**:

```
rootmaster% nismatch -h 'R&D' depts.wiz.com.
rootmaster% nisgrep -h 'R&D' depts.wiz.com.
```

Note that quotes are used in the R&D expression to prevent the shell from interpreting the ampersand (&) as a metacharacter.

Searching a Particular Column

To search through a particular column other than the first, use the following syntax:

```
nismatch column=string tablename
nisgrep column=reg-expression tablename
```

This example searches through the depts table for all the entries whose second column has a value of **SanFran**:

```
rootmaster% nismatch -h Site=SanFran depts.wiz.com
rootmaster% nisgrep -h Site=SanFran depts.wiz.com
```

Searching Multiple Columns

To search for entries with matches in two or more columns, use the following syntax:

```
nismatch [-h] column=string ... tablename
nismatch [-h] [column=string,...],tablename
nisgrep [-h] column=reg-exp ... tablename
```

This example searches for entries whose second column has a value of SanFran and whose third column has a value of **jhill**:

```
rootmaster% nismatch -h [Site=SanFran,Manager=jhill],  depts.wiz.com.
rootmaster% nisgrep -h Site=SanFran Manager=jhill depts.wiz.com.
```

Using the nisln Command

The **nisln** command creates symbolic links between NIS+ objects and table entries. You can use it to link objects to objects or objects to table entries. (You cannot create a link that originates with a table entry.) All NIS+ administration commands accept the **–L** flag, which directs them to follow links between NIS+ objects.

To create a link to another object or entry, you must have modify rights to the source object; that is, the one that will point to the other object or entry.

Attention: Never link a cred table. Each **org_dir** directory should have its own cred table. Do not use a link to some other **org_dir** cred table. NIS+ cannot operate correctly with linked cred tables.

To create a link, use:

```
nisln source target
```

Table 60. **nisln** Syntax Options

Option	Description
–L	Follow links. If the **source** is itself a link, the new link will not be linked to it, but to that link's original source.
–D	Defaults. Specify a different set of defaults for the linked object. Defaults are described in Specifying Nondefault Security Values at Creation Time .

For more information, see the command description for **nisln**.

Using the nissetup Command

The **nissetup** command expands an existing NIS+ directory object into a domain by creating the **org_dir** and **groups_dir** directories, and a full set of NIS+ tables. It does not, however, populate the tables with data. For that, you will need the **nisaddent** command. Expanding a directory into a domain is part of the process of setting up a domain.

The **nissetup** command can expand a directory into a domain that supports NIS clients as well.

To use **nissetup**, you must have modify rights to the directory under which you'll store the tables.

To expand a directory into an NIS+ domain, use:

```
/usr/lib/nis/nissetup
/usr/lib/nis/nissetup directory-name
```

To expand a directory into an NIS–compatible NIS+ domain, use:

```
/usr/lib/nis/nissetup –Y
/usr/lib/nis/nissetup –Y directory-name
```

For more information, see the command description for **nissetup**.

Using the nisaddent Command

The **nisaddent** command loads information from text files or NIS maps into NIS+ tables. It can also dump the contents of NIS+ tables back into text files.

You can use **nisaddent** to transfer information from one NIS+ table to another (for example, to the same type of table in another domain), but not directly. First, you need to dump the contents of the table into a file, and then load the file into the other table. Be sure, though, that the information in the file is formatted properly. Appendix A, Information in NIS+ Tables, describes the format required for each table.

When you load information into a table, you can use any of three options: replace, append, or merge. The append option simply adds the source entries to the NIS+ table. With the replace option, NIS+ first deletes all existing entries in the table and then adds the entries from the source. In a large table, this adds a large set of entries into the table's **.log** file (one set for removing the existing entries, another for adding the new ones), taking up space in /**var**/**nis** and making propagation to replicas time consuming.

The merge option produces the same result as the replace option but uses a different process, one that can greatly reduce the number of operations that must be sent to the replicas. With the merge option, NIS+ handles three types of entries differently:

- Entries that exist only in the source are **added** to the table
- Entries that exist in both the source and the table are **updated** in the table
- Entries that exist only in the NIS+ table are **deleted** from the table

When updating a large table with a file or map whose contents are not greatly different from those of the table, the merge option can spare the server a great many operations. Because the merge option deletes only the entries that are not duplicated in the source (the replace option deletes **all** entries, indiscriminately), it saves one delete and one add operation for every duplicate entry.

If you are loading information into the tables for the first time, you must have create rights to the table object. If you are overwriting information in the tables, you must have modify rights to the tables.

To load information from text files, use:

```
/usr/lib/nis/nisaddent –f filename table-type [domain]
/usr/lib/nis/nisaddent –f filename –t tablename table-type [domain]
```

To load information from NIS maps, use:

```
/usr/lib/nis/nisaddent –y NISdomain table-type [domain]
/usr/lib/nis/nisaddent –y NISdomain –t tablename table-type [domain]
/usr/lib/nis/nisaddent –Y map table-type [domain]
/usr/lib/nis/nisaddent –Y map –t tablename table-type [domain]
```

To dump information from an NIS+ table to a file, use:

```
/usr/lib/nis/nisaddent –d [–t tablename ] tabletype > filename
```

Another way to load information from files is to use **stdin** as the source. However, you cannot use the **–m** option with **stdin**. You can use redirect (**>**) or pipe (**|**), but you cannot pipe into another domain.

Table 61. Using **cat** with **nisaddent**

Task	Command
Redirect	**`cat`** *`filename`* **`> nisaddent`** *`table-type`*
Redirect with append option	**`cat`** *`filename`* **`> nisaddent -a`** *`table-type`*
Redirect with append into another domain	**`cat`** *`filename`* **`> nisaddent -a`** *`table-type`* NIS+*`domain`*
Pipe	**`cat`** *`filename`* **`| nisaddent`** *`table-type`*

Pipe with append option	**`cat`** *`filename`* **`| nisaddent -a`** *`table-type`*
	If the NIS+ table is one of the automounter tables or a nonstandard table, add the **–t** option and the complete name of the NIS+ table. For example:
	`master#` **`nisaddent -f /etc/auto_home.xfr`**
	`-t auto_home.org_dir.wiz.com.` *`key-value`*
	`master#` **`nisaddent -f /etc/auto_home.xfr`**
	`-t auto_home.org_dir.wiz.com.` *`key-value`* **`sales.wiz.com.`**

Table 62. **nisaddent** Syntax Options

Option	Description
–a	Append. Contents of the source are appended to contents of the table.
–r	Replace. Contents of the source replace contents of the table.
–m	Merge. Contents of the source are merged with contents of the table.
–d	Dump. Contents of the NIS+ table are dumped to **stdout**.
–v	Verbose. The command prints verbose status messages.
–P	Follow path. If the command was unable to find a table, follow the search paths specified in the environment variable **NIS_PATH**.
–A	All data. Apply the operation to all the tables in the search path.
–M	Master server. Use the tables only in the master server of the domain.
–D	Override defaults. For the new data being loaded into the tables, override existing defaults.

For more information, see the command description for **nisaddent**.

Removing NIS+

This section describes how to use the NIS+ directory administration commands to perform the following tasks:

- Removing NIS+ From a Client Machine, on page 6-66
- Removing NIS+ From a Server, on page 6-66
- Removing the NIS+ Namespace, on page 6-66

Removing NIS+ From a Client Machine

This section describes how to remove NIS+ from a client machine. Keep in mind that removing NIS+ from a client machine does not remove the NIS+ name service from your network. See Removing the NIS+ Namespace, on page 6-66 for information on removing the NIS+ name service from a network and returning to either NIS or /**etc** files for name purposes.

To remove NIS+ from a client machine that was set up as an NIS+ client using the **nisaddcred**, **domainname**, and **nisinit** commands, use **nisclient –D**.

Note: The **nisclient –D** command does not restore the previous environment as **nisclient –r** does.

Restoring to Previous Network Environment

To remove NIS+ from a client machine that was set up as an NIS+ client using the **nisclient –i** script, simply run **nisclient** with the **–r** option:

```
clientmachine# nisclient –r
```

nisclient –r simply undoes the most recent iteration of **nisclient –i**; it restores the previous naming system used by the client, such as NIS or /**etc** files.

Removing NIS+ From a Server

This section describes how to remove NIS+ from an NIS+ server.

Keep in mind that removing NIS+ from a server does not remove the NIS+ name service from your network. See Removing the NIS+ Namespace, on page 6-66 for information on removing the NIS+ name service from a network and returning to either NIS or /**etc** files for naming purposes.

To remove NIS+ from a server, use **nisserver –D**.

Removing the NIS+ Namespace

To remove the NIS+ namespace and return to using either NIS or /**etc** files for name services, follow these steps:

1. Remove the **.rootkey** file from the root master.

   ```
   rootmaster# rm –f /etc/.rootkey
   ```

2. Remove the **groups_dir** and **org_dir** subdirectories from the root master root domain.

   ```
   rootmaster# nisrmdir –f groups_dir.domainname
   rootmaster# nisrmdir –f org_dir.domainname
   ```

   Where **domainname** is the name of the root domain, for example, **wiz.com**.

3. Remove the root domain.

   ```
   rootmaster# nisrmdir –f domainname
   ```

   Where **domainname** is the name of the root domain, for example, **wiz.com**.

4. Locate and kill the **keyserv**, **rpc.nisd**, and **nis_cachemgr** processes.

   ```
   rootmaster# stopsrc –g nisplus
   rootmaster# stopsrc –s keyserv
   ```

5. Create a new domain.

   ```
   rootmaster# chypdom [–I|–B|–N] name
   ```

   Where *name* is the name of the new domain; for example, the name of the domain before you installed NIS+.

6. Restart the **keyserv** process.

   ```
   rootmaster# startsrc –s keyserv
   ```

7. Remove the /**var**/**nis** directory and files.

   ```
   rootmaster# rm –rf /var/nis/*
   ```

   If you are running in NIS–compatibility mode, also enter the following:

   ```
   rootmaster# rm –rf /var/yp/ypdomain
   ```

8. Now restart your other name service (NIS or /**etc** files).

Chapter 7. Security

This chapter gives an overview of how NIS+ protects its namespace.

AIX Security Mechanisms

In essence, AIX security is provided by gates that users must pass through in order to enter the AIX environment, and permission matrixes that determine what they are able to do once inside. In some contexts, *Secure RPC* passwords have been referred to as *network passwords*.

The overall system is composed of four gates and two permission matrixes.

Dialup gate To access a given AIX environment from the outside through a modem and phone line, you must provide a valid login ID and dialup password.

Login gate To enter a given AIX environment you must provide a valid login ID and user password.

File and Directory Matrix
Once you have gained access to a AIX environment, your ability to read, execute, modify, create, and destroy files and directories is governed by the applicable permissions matrix.

Root gate To gain access to root privileges, you must provide a valid super user (root) password.

Secure RPC gate
In an NIS+ environment running at security level 2 (the default), when you try to use NIS+ services and gain access to NIS+ objects (servers, directories, tables, table entries, and so forth) your identity is confirmed by NIS+ using the Secure RPC process.

To enter the Secure RPC gate requires presentation of a Secure RPC password. Your Secure RPC password and your login password normally are identical and when that is the case you are passed through the gate automatically without having to re–enter your password. (In some contexts, *Secure RPC* passwords have been referred to as *network passwords*. See Secure RPC Password versus Login Password Problem, on page 6-7 for information about handling two passwords that are not the same.)

A set of *credentials* are used to automatically pass your requests through the Secure RPC gate. The process of generating, presenting, and validating your credentials is called *authentication* because it confirms who you are and that you have a valid Secure RPC password. This authentication process is automatically performed every time you request a NIS+ service.

In an NIS+ environment running in NIS–compatibility mode (also known as YP–compatibility mode), the protection provided by the Secure RPC gate is significantly weakened because everyone has read rights for all NIS+ objects and modify rights for those entries that apply to them regardless of whether or not they have a valid credential (that is, regardless of whether or not the authentication process has confirmed their identity and validated their Secure RPC password). Since that allows anyone to have read rights for all NIS+ objects and modify rights for those entries that apply to them, an NIS+ network running in compatibility mode is less secure than one running in normal mode. (In Secure RPC terminology, any user without a valid creditial is considered a member of the **nobody** class. See Authorization Classes, on page 7-7 for a description of the four classes.)

For details on how to create and administer NIS+ authentication and credentials, see Administering NIS+ Credentials, on page 6-2.

NIS+ **objects matrix**
Once you have been properly authenticated to NIS+ your ability to read, modify, create, and destroy NIS+ objects is governed by the applicable permissions matrix. This process is called NIS+ *authorization*.

For details NIS+ permissions and authorization, see Administering NIS+ Access Rights, on page 6-21.

NIS+ Security Mechanisms

NIS+ security is an integral part of the NIS+ namespace. You cannot set up security and the namespace independently. For this reason, instructions for setting up security are woven through the steps used to set up the other components of the namespace. Once an NIS+ security environment has been set up, you can add and remove users, change permissions, reassign group members, and all other routine administrative tasks needed to manage an evolving network.

The security features of NIS+ protect the information in the namespace, as well as the structure of the namespace itself, from unauthorized access. Without these security features, any NIS+ client could obtain and change information stored in the namespace or even damage it.

NIS+ security does two things:

Authentication	Authentication is used to identify NIS+ principals. Every time a principal (user or machine) tries to access an NIS+ object, the user's identity and Secure RPC password is confirmed and validated.
Authorization	Authorization is used to specify access rights. Every time NIS+ principals try to access NIS+ objects, they are placed in one of four authorization classes (owner, group, world, nobody). The NIS+ security system allows NIS+ administrators to specify different read, modify, create, or destroy rights to NIS+ objects for each class. Thus, for example, a given class could be permitted to modify a particular column in the passwd table but not read that column, or a different class could be allowed to read some entries of a table but not others.

In essence, then, NIS+ security is a two–step process:

1. **Authentication**. NIS+ uses credentials to confirm that you are who you claim to be.

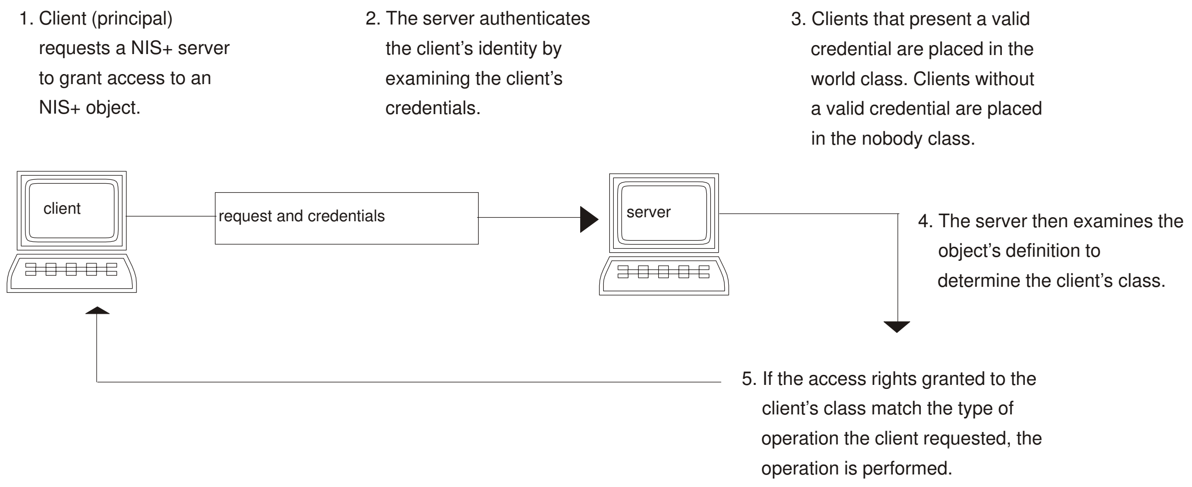
   (You should not have to enter a password as part of the authentication process. However, if for some reason your Secure RPC password is different than your login password, you will have to perform a **keylogin** the first time you try to access NIS+ objects or services. To perform a **keylogin**, you must provide a valid Secure RPC password. See Secure RPC Password versus Login Password Problem, on page 6-7.)

2. **Authorization**. Once your identity is established by the authentication process, NIS+ determines your class. What you can do with a given NIS+ object or service depends on which class you belong to. For example, a given NIS+ table may allow one class to both read and modify the information in the table, but a different class is only allowed to read the information, and a third class is not even allowed to do that. This is similar in concept to the standard UNIX file and directory permissions system. (See Authorization Classes, on page 7-7 for more information on classes.)

This process, for example, prevents someone with root privileges on machine A from using the **su** command to assume the identity of a second user who is either not logged in at all or logged in on machine B, and then accessing NIS+ objects with the second user's NIS+ access privileges.

Note, however, that NIS+ cannot prevent someone who knows another user's login password from assuming that other user's identity and NIS+ access privileges. Nor can NIS+ prevent a user with root privileges from assuming the identity of another user who is logged in from the **same** machine.

The following figure details this process.


1. Client (principal) requests a NIS+ server to grant access to an NIS+ object.
2. The server authenticates the client's identity by examining the client's credentials.
3. Clients that present a valid credential are placed in the world class. Clients without a valid credential are placed in the nobody class.
client
request and credentials
server
4. The server then examines the object's definition to determine the client's class.
5. If the access rights granted to the client's class match the type of operation the client requested, the operation is performed.

NIS+ Principals

NIS+ principals are the entities (clients) that submit requests for NIS+ services. An NIS+ principal may be someone who is logged in to a client machine as a regular user, someone who is logged in as superuser, or any process that runs with superuser permission on an NIS+ client machine. Thus, an NIS+ principal can be a client user or a client workstation.

A NIS+ principal can also be the entity that supplies an NIS+ service from an NIS+ server. Since all NIS+ servers are also NIS+ clients, much of this discussion also applies to servers.

NIS+ Security Levels

NIS+ servers operate at one of two security levels. These levels determine the types of credential principals must submit for their requests to be authenticated. NIS+ is designed to run at the most secure level, which is security level 2. Level 0 is provided only for testing, setup, and debugging purposes. These security levels are summarized in the following table.

Note: Use Web-based System Manager, SMIT, or the **passwd** command to change your own password regardless of security level or credential status.

Table 63. NIS+ Security Levels

Security Level	Description
0	Security level 0 is designed for testing and setting up the initial NIS+ namespace. An NIS+ server running at security level 0 grants any NIS+ principal full access rights to all NIS+ objects in the domain. Level 0 is for setup purposes only and should only be used by administrators for that purpose. Level 0 should not be used on networks in normal operation by regular users.
1	Security level 1 uses AUTH_SYS security. This level is not supported by NIS+ and should not be used.
2	Security level 2 is the default. It is the highest level of security currently provided by NIS+. It authenticates only requests that use DES credentials. Requests with no credentials are assigned to the nobody class and have whatever access rights that have been granted to that class. Requests that use invalid DES credentials are retried. After repeated failure to obtain a valid DES credential, requests with invalid credentials fail with an authentication error. (A credential might be invalid for a variety of reasons such as the principal making the request is not keylogged in on that machine, the clocks are out of synch, there is a key mismatch, and so forth.)

NIS+ Authentication and Credentials

The purpose of NIS+ credentials is to *authenticate* (confirm) the identity of each principal requesting a NIS+ service or access to a NIS+ object. In essence, the NIS+ credential/authorization process is an implementation of the Secure RPC system.

The credential/authentication system prevents someone from assuming some other user's identity. That is, it prevents someone with root privileges on one machine from using the **su** command to assume the identity of a second user who is either not logged in at all or logged in on another machine and then accessing NIS+ objects with the second user's NIS+ access privileges.

Note: NIS+ cannot prevent someone who knows another user's login password from assuming that other user's identity and the other user's NIS+ access privileges. Nor can NIS+ prevent a user with root privileges from assuming the identity of another user who is currently logged in on the **same** machine.

Once a server authenticates a principal, it then checks the NIS+ object that the principal wants to access to see what activities that principal is authorized to perform. (See NIS+ Authorization and Access, on page 7-7 for further information on authorization.)

User and Machine Credentials

There are two basic types of principal, *users* and *machines,* and thus two different types of credentials:

User credentials
When someone is logged in to an NIS+ client as a regular user, requests for NIS+ services include that person's user credentials.

Machine credentials
When a user is logged in to an NIS+ client as superuser, request for services use the client workstation's credentials.

DES versus LOCAL Credentials

NIS+ principals can have two types of credential: DES and LOCAL.

DES Credentials

Data Encryption Standard (DES) credentials are the type of credential that provide secure authentication. When this guide refers to NIS+ checking a credential to authenticate a NIS+ principal, it is the DES credential that NIS+ is validating. (Note that DES credentials are only one method of achieving authentification. In the future, other methods may be available. Thus, do not equate DES credentials with NIS+ credentials.)

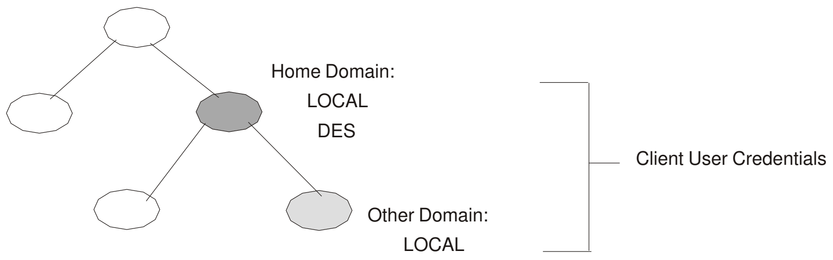
Each time a principal requests a NIS+ service or access to a NIS+ object, the software uses the credential information stored for that principal to generate a credential for that principal. DES credentials are generated from information created for each principal by a NIS+ administrator, as explained in Administering NIS+ Credentials, on page 6-2.

- When the validity of a principal's DES credential is confirmed by NIS+, that principal is *authenticated*.
- A principal must be authenticated in order to be placed in the owner, group, or world authorization classes. In other words, you must have a valid DES credential in order to be placed in one of those classes. (Principals that do not have a valid DES credential are automatically placed in the nobody class.)
- DES credential information is always stored in the cred table of the principal's home domain, regardless of whether that principal is a client user or a client workstation.

LOCAL Credentials

LOCAL credentials are simply a map between a user's User ID number and NIS+ principal name which includes their home domain name. When users log in, the system looks up their LOCAL credential, which identifies their home domain where their DES credential is stored. The system uses that information to get the user's DES credential information.

When users log in to a remote domain, those requests use their LOCAL credential which points back to their home domain; NIS+ then queries the user's home domain for that user's DES credential information. This allows a user to be authenticated in a remote domain even though the user's DES credential information is not stored in that domain. For more information, see figure.



Credentials and Domains

LOCAL credential information can be stored in any domain. In fact, in order to log into a remote domain and be authenticated, a client user **must** have a LOCAL credential in the cred table of the remote domain. If a user does not have a LOCAL credential in a remote domain the user is trying to access, NIS+ will be unable to locate the user's home domain to obtain the user's DES credential. In such a case the user would not be authenticated and would be placed in the nobody class.

User Types and Credential Types

A user can have both types of credential, but a machine can only have a DES credential.

Root cannot have NIS+ access, as root, to other machines because the root UID of every machine is always zero. If root (UID=0) of machine A tried to access machine B as root, that would conflict with machine B's already existing root (UID=0). Thus, a LOCAL credential does not make sense for a client **workstation**; so it is allowed only for a client **user**.

Table 64. Types of Credentials

Type of Credential	Client User	Client Workstation
DES	Yes	Yes
LOCAL	Yes	No

NIS+ Authorization and Access

The basic purpose of NIS+ authorization is to specify the access rights that each NIS+ principal has for each NIS+ object and service.

Once the principal making an NIS+ request is authenticated, NIS+ places them in an authorization class. The access rights (permissions) that specify which activities a principal may do with a given NIS+ object are assigned on a class basis. In other words, one authorization class may have certain access rights while a different class has different rights.

Authorization classes
There are four authorization classes: owner, group, world, and nobody. (See Authorization Classes, on page 7-7 for details.

Access rights There are four types of access rights (permissions): create, destroy, modify, and read. (See NIS+ Access Rights, on page 7-10 for details.)

Authorization Classes

NIS+ objects do not grant access rights directly to NIS+ principals. Instead, they grant access rights to four *classes of principal*:

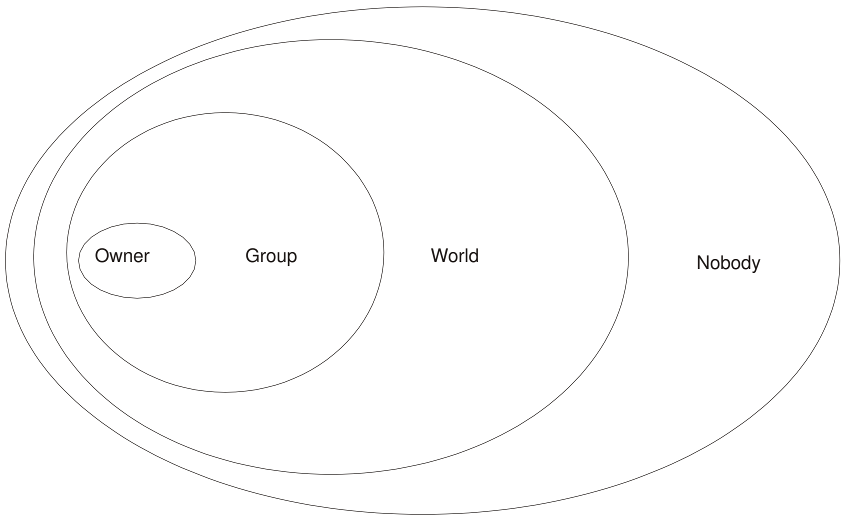
Owner The principal who happens to be the object's owner gets the rights granted to the owner class.

Group Each NIS+ object has one group associated with it. The members of an object's group are specified by the NIS+ administrator. The principals who belong to the object's group class get the rights granted to the group class. (In this context, *groups* refers to NIS+ groups, not UNIX or net groups. See Group Class, on page 7-8 for a description of NIS+ groups.)

World The world class encompasses all NIS+ principals that a server has been able to authenticate. (That is, everyone who has been authenticated but who is not in either the owner or group classes.)

Nobody Everyone belongs to the nobody class even those who are not authenticated.

See figure for an illustration of the classes.



For any NIS+ request, the system determines which class the requesting principal belongs to and the principal then can use whatever access rights belonging to that class.

An object can grant any combination of access rights to each of these classes. Normally, however, a higher class is assigned the same rights as all the lower classes, plus possible additional rights.

For instance, an object could grant read access to the nobody and world classes, both read and modify access to the group class, and read, modify, create, and destroy access to the owner class.

The four classes are described in detail below.

Owner Class

The owner is a **single** NIS+ principal.

A principal making a request for access to an NIS+ object must be authenticated (present a valid DES credential) before being granted owner access rights.

By default, an object's owner is the principal that created the object. However, an object's owner can cede ownership to another principal in two ways:

- One way is for the principal to specify a different owner at the time the object is created (see Specifying Access Rights in Commands, on page 6-25).
- A second way is for the principal to change the ownership of the object after it is created (see Changing Ownership of Objects and Entries, on page 6-33).

Once a principal gives up ownership, that principal gives up all owner's access rights to the object and keeps only the rights the object assigns to either the group, the world, or nobody.

Group Class

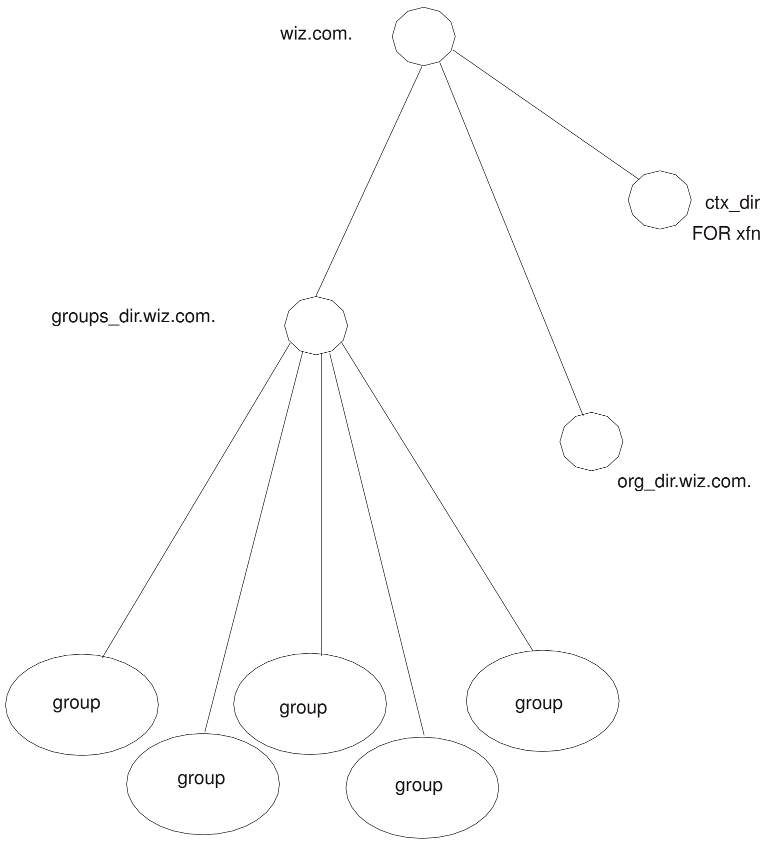
The object's group is a **single** NIS+ group. (In this context, *group* refers to NIS+ groups, not UNIX or net groups.)

A principal making a request for access to an NIS+ object must be authenticated (present a valid DES credential) and belong to the group before being granted group access rights.

An NIS+ group is a collection of NIS+ principals, grouped together as a convenience for providing access to the namespace. The access rights granted to an NIS+ group apply to all the principals that are members of that group. (An object's owner, however, does not need to belong to the object's group.)

When an object is created it may be assigned a default group. A nondefault group can be specified for an object when it is created or later. An object's group may be changed at any time.

Information about NIS+ groups is stored in NIS+ group **objects**, under the **groups_dir** subdirectory of every NIS+ domain. (Note that information about NIS+ groups is not stored in the NIS+ group table. That table stores information about UNIX groups.)



Instructions for administering NIS+ groups are provided in Administering NIS+ Groups, on page 6-45.

World Class

The world class contains all NIS+ principals that are authenticated by NIS+. In other words, everyone in the owner and group class, plus everyone else who presents a valid DES credential.

Access rights granted to the world class apply to all authenticated principals.

Nobody Class

The nobody class is composed of anyone who is not properly authenticated. In other words, anyone who does not present a valid DES credential.

Authorization Classes and the NIS+ Object Hierarchy

There is a hierarchy of NIS+ objects and authorization classes that can apply independently to each level. The standard default NIS+ directory hierarchy is:

Directory level In each NIS+ domain there are two NIS+ directory objects: **groups_dir** and **org_dir**. Each **groups_dir** directory object contains various groups. Each **org_dir** directory object contains various tables.

Group level or table level
Groups contain individual entries and possibly other groups. Tables contain both columns and individual entries.

Column level A given table will have one or more columns.

Entry (row) level
A given group or table will have one or more entries.

The four authorization classes apply at each level. Thus, a directory object will have its own owner and group. The individual tables within a directory object will have their own individual owners and groups which may be different than the owner and group of the directory object.

Within a table, a column or an entry may have its own individual owner or group which may be different than the owner and group of the table as a whole or the directory object as a whole.

NIS+ Access Rights

NIS+ objects specify their access rights as part of their object definitions. (You can examine these by using the **niscat –o** command.)

NIS+ objects specify access rights for NIS+ principals in the same way that UNIX files specify permissions for UNIX users. Access rights specify the types of operations that NIS+ principals are allowed to perform on NIS+ objects.

NIS+ operations vary among different types of objects, but they all fall into one of the four access rights categories: read, modify, create, and destroy.

Read	A principal with read rights to an object can view the contents of that object.
Modify	A principal with modify rights to an object can change the contents of that object.
Destroy	A principal with destroy rights to an object can destroy or delete the object.
Create	A principal with create rights to a higher level object can create new objects within that level. In other words, if you have create rights to a NIS+ directory object, you can create new tables within that directory. If you have create rights to a NIS+ table, you can create new columns and entries within that table.

Every communication from an NIS+ client to an NIS+ server is, in effect, a request to perform one of these operations on a specific NIS+ object. For instance, when an NIS+ principal requests the IP address of another workstation, it is effectively requesting read access to the **hosts** table object, which stores that type of information. When a principal asks the server to add a directory to the NIS+ namespace, it is actually requesting **modify** access to the directory's parent object.

Keep in mind that these rights logically evolve down from directory to table to table column and entry levels. For example, to create a new table, you must have create rights for the NIS+ directory object where the table will be stored. When you create that table, you become its default owner. As owner, you can assign yourself create rights to the table which allows you to create new entries in the table. If you create new entries in a table, you become the default owner of those entries. As table owner, you can also grant table–level create rights to others. For example, you can give your table's group class table–level create rights. In that case, any member of the table's group can create new entries in the table. The individual member of the group who creates a new table entry becomes the default owner of that entry.

NIS+ Security and Administrative Rights

NIS+ does not enforce any requirement that there be a single NIS+ administrator. Whoever has administrative rights over an object—that is, the authority to create, destroy, and for some objects, modify rights—is considered to be an NIS+ administrator for that object.

Whoever creates an NIS+ object sets the initial access rights to that object. If the creator restricts administrative rights to the object's owner (initially the creator), then only the owner has administrative power over that object. On the other hand, if the creator grants administrative rights to the object's group, then everyone in that group has administrative power over that object.

Theoretically, you could grant administrative rights to the world class, or even the nobody class. The software allows you to do that. But granting administrative rights beyond the group class effectively nullifies NIS+ security. Thus, if you grant administrative rights to either the World or the nobody class you are, in effect, defeating the purpose of NIS+ security.

NIS+ Security Reference

Use the following commands to administer passwords, credentials, and keys (see the appropriate command descriptions for more information):

chkey	Changes a principal's Secure RPC key pair. Do not use **chkey** unless necessary, use **passwd** instead.
keylogin	Decrypts and stores a principal's secret key with the **keyserv**.
keylogout	Deletes stored secret key from **keyserv**.
keyserv	Enables the server for storing private encryption keys.
newkey	Creates a new key pair in public–key database.
nisaddcred	Creates credentials for NIS+ principals.
nisupdkeys	Updates public keys in directory objects.
passwd	Changes and administers principal's password.

Chapter 8. NIS and NIS+ Troubleshooting

This chapter describes some of the problems you may encounter while administering NIS and/or an NIS+ namespace. Problems are grouped according to type. For each problem there is a list of common symptoms, a description of the problem, and one or more suggested solutions.

In addition to this chapter, there is also a book that lists AIX error messages. If you are responding to a specific error message, check *AIX Messages Guide and Reference* first.

This chapter covers the following types of NIS+ problems:

- Troubleshooting NIS–Related Problems, on page 8-2
- Troubleshooting Namespace Administration Problems, on page 8-6
- Troubleshooting Namespace Database Problems, on page 8-9
- Troubleshooting NIS Compatibility Problems, on page 8-10
- Troubleshooting Object Not Found Problems, on page 8-12
- Troubleshooting Ownership and Permission Problems, on page 8-14
- Troubleshooting Security Problems, on page 8-16
- Troubleshooting Slow Performance and System Hang Problems, on page 8-23
- Troubleshooting System Resource Problems, on page 8-26
- Troubleshooting User Problems, on page 8-27
- Troubleshooting Other NIS+ Problems, on page 8-29

Troubleshooting NIS–Related Problems

The approach to troubleshooting a Network Information Service (NIS) problem depends on whether the problem is at the NIS client or the NIS server.

Identifying NIS Client Problems

The most common NIS client problems occur at the following times:

- When commands hang
- When NIS service is unavailable
- When the **ypbind** daemon fails
- When the **ypwhich** command is inconsistent

Note: When attempting to solve a map problem, be aware the same problem may occur in other maps as well. See Files Where NIS Appends Map Information, on page 2-15 for a more detailed explanation.

Using rsh

When an AIX machine has two interfaces and they both are given the same name, **gethostbyname** lookups for **rsh** command will fail if NIS is being used. This is because AIX NIS will not return both addresses, but only the first one found. This is an implementation limitation imposed by the New Database Manager (NDBM) and performance considerations. The error message will be:

```
0826-825: there is a host address that does not match
```

When Commands Hang

The most common problem occurring at an NIS client node is for a command to hang. Sometimes a command appears to hang, even though the system seems fine and other commands run. In such a case, a message like the following can be generated at the console:

```
   NIS: server not responding for domain <wigwam>. Still trying
```

This error message indicates that the **ypbind** daemon on the local machine is unable to communicate with the **ypserv** daemon in the `wigwam` domain. This results when systems that run the **ypserv** daemon have failed. It may also occur if the network or the NIS server machine is so overloaded that the **ypserv** daemon cannot get a response back to your **ypbind** daemon within the time–out period.

Under these circumstances, all the other NIS clients on your network show the same or similar problems. The condition is usually temporary. The messages go away when the NIS server machine reboots and the **ypserv** daemon restarts, or else when the load on the NIS server and the network decreases.

If the **ypbind** daemon is communicating with the **ypserv** daemon and the NIS server is not overloaded, one of the following problems may exist:

- The `domainname` on the NIS client machine is not set or is set incorrectly. Clients must use a domain name that the NIS servers know. The domain name is case–sensitive and initially set with lowercase letters. In this case, set the domain correctly.
- Your local network may not have an NIS server machine. You can automatically bind only to a **ypserv** process on your local network. Binding on servers residing on other networks through a gateway requires using the **ypset** command. Two or more NIS servers (one master and several slave servers) can improve availability and response characteristics for NIS services.
- The NIS server may not be up and running. Check other machines on your local network. If several clients have problems simultaneously, the server may be the cause.

Find a client system that is working normally and try the **ypwhich** command. If the **ypwhich** command never returns an answer, stop it. Then type the following at the NIS server machine:

```
ps –ef | grep yp
```

Look for the **ypserv** and **ypbind** processes. If the server's **ypbind** daemon is not running, start it using the instructions in Starting and Stopping the NIS Daemons, on page 2-13.

If a **ypserv** process is running, issue the **ypwhich** command on the NIS server machine. If this command returns no answer, the **ypserv** daemon is probably hung and should be restarted. Stop and restart the **ypserv** daemon by following the instructions in Starting and Stopping the NIS Daemons, on page 2-13.

When NIS Service Is Unavailable

When other machines on the network appear to have no problems, but NIS service becomes unavailable on your system, a variety of symptoms can show up:

- Some commands may operate correctly while others terminate and print an error message about the unavailability of NIS.
- Some commands limp along in a backup–strategy mode particular to the program involved.
- Some commands or daemons crash with error messages or no message at all.

  For example, messages such as the following might be generated:

  ```
  ypcat myfile
  ypcat: can't bind to NIS server for domain <wigwam>
  Reason: can't communicate with ypbind.
  ```

  OR

  ```
  /usr/etc/yp/yppoll myfile
  RPC: timed out
  ```

When symptoms like these occur, issue the **ls –l** command on a directory containing files owned by many users, including users not in the local machine's /**etc**/**passwd** file. Use the following format:

```
ls –l
```

If the **ls –l** command reports file owners that are not in the local machine's /**etc**/**passwd** file as numbers, rather than names, it means that NIS service is not working.

These symptoms usually indicate that your **ypbind** daemon is not running. You can use the **ps –ef** command to check for one. If you do not find a **ypbind** daemon, start it by following the instructions in Starting and Stopping NIS Daemons, on page 2-13.

When the ypbind Daemon Becomes Inoperable

If the **ypbind** daemon repeatedly crashes immediately after it is started, you should look for a problem in some other part of the system.

- Check for the presence of the **portmap** daemon by typing:

  ```
  ps –ef | grep portmap
  ```

  If the daemon is not running, reboot the system.

- If the **portmap** daemon is running but will not stay up or responds strangely, check the network software.

  Try to communicate with the **portmap** daemon on your machine from a different machine that is operating normally. From such a machine, type:

  ```
  rpcinfo –p client
  ```

  where *client* is the hostname of the machine.

- If the **portmap** daemon is up and running, the output appears in a format similar to the following:

```
program   vers proto  port
100007     2    tcp   1024  ypbind
100007     2    udp   1028  ypbind
100007     1    tcp   1024  ypbind
100007     1    udp   1028  ypbind
100021     1    tcp   1026  nlockmgr
100024     1    udp   1052  status
100020     1    udp   1058  llockmgr
100020     1    tcp   1028  llockmgr
100021     2    tcp   1029  nlockmgr
100012     1    udp   1083  sprayd
100011     1    udp   1085  rquotad
100005     1    udp   1087  mountd
100008     1    udp   1089  walld
100002     1    udp   1091  rusersd
100002     2    udp   1091  rusersd
100001     1    udp   1094  rstatd
100001     2    udp   1094  rstatd
100001     3    udp   1094  rstatd
```

  If the daemons are not listed, the **ypbind** daemon is unable to register its services. Reboot the machine.

  If the daemons are there, but they change each time you try to restart the **ypbind** daemon, reboot the system (even though the **portmap** daemon is up).

When the ypwhich Command Is Inconsistent

When you use the **ypwhich** command several times at the same client node, the response varies because the status of the NIS server changes. The status changes are normal.

The binding of NIS client to NIS server changes over time on a busy network, when the NIS servers are busy. Whenever possible, the system stabilizes so that all clients get acceptable response time from the NIS servers. The source of an NIS service is not important, because an NIS server machine often gets its own NIS services from another NIS server on the network.

Identifying NIS Server Problems

The most common NIS server problems occur at the following times:

- When different versions of an NIS map exist
- When the **ypserv** daemon crashes

When Different Versions of an NIS Map Exist

Because NIS works by propagating maps among servers, you can sometimes find different versions of a map at the network servers. This is normal if temporary and abnormal otherwise.

Normal update is prevented when an NIS server or a router between NIS servers is down during a map transfer attempt. When all the NIS servers and all the routers between them are up and running, the **ypxfr** command should succeed.

If a particular slave server has problems updating a map, you can log in to that server and run the **ypxfr** command interactively. If this command fails, an error message returns to tell you why, so that you can fix the problem. If the command succeeds, but you want to check it regardless, create a log file to enable logging of messages by typing the following:

```
cd /var/yp
touch ypxfr.log
```

This saves all output from the **ypxfr** command. The output looks much like what the **ypxfr** command creates when it is run interactively, but each line in the log file is time stamped.

The time stamp tells when the **ypxfr** command began its work. It is normal to see unusual orderings in the time stamps. If copies of the **ypxfr** command ran simultaneously but their work took differing amounts of time, the summary status line may be written to the log files in an order that differs from the order in which they were invoked.

Any pattern of intermittent failure shows up in the log. After you fix the problem, turn off logging by removing the log file. If you forget to remove the log file, it grows without limit.

While you are logged in to the NIS slave server, inspect the system /**etc**/**crontab** entries, and the **ypxfr** shell scripts it invokes.

Make sure that the NIS slave server is in the **ypservers** map. If not, the **yppush** command will not notify the slave server when a new copy of a map exists.

When the ypserv Daemon Becomes Inoperable

When the **ypserv** process repeatedly crashes immediately after starting and does not stay up with repeated activations, the debugging process is similar to that described for **ypbind** crashes. First, you should check for the **portmap** daemon:

```
ps -ef | grep portmap
```

If you do not find the **portmap** daemon, reboot the server. If there is a **portmap** daemon, type:

```
rpcinfo -p speed
```

where *speed* is the hostname of the NIS server.

On your particular machine, the port numbers will be different. The four entries that represent the **ypserv** daemon are:

```
100004     2   udp   1027  ypserv
100004     2   tcp   1024  ypserv
100004     1   udp   1027  ypserv
100004     1   tcp   1024  ypserv
```

If these entries do not exist, the **ypserv** daemon is unable to register its services. Reboot the machine. If the **ypserv** entries are present, but they change each time you try to restart the **ypserv** daemon, reboot the machine again.

Troubleshooting NIS+ Namespace Administration Problems

This section describes problems that may be encountered in the course of routine namespace administration work.

Symptoms

- **Illegal object type for operation** message.
- Other ”object problem” error messages
- Initialization failure
- Checkpoint failures
- Difficulty adding a user to a group
- Logs too large/lack of disk space/difficulty truncating logs
- Cannot delete **groups_dir** or **org_dir**

Illegal Object Problems

Symptoms

- **Illegal object type for operation** message
- Other ”object problem” error messages

Possible Causes

There are a number of possible causes for this error message:

- You have attempted to create a table without any searchable columns.
- A database operation has returned the status of **DB_BADOBJECT**.
- You are trying to add or modify a database object with a length of zero.
- You attempted to add an object without an owner.
- The operation expected a directory object, and the object you named was not a directory object.
- You attempted to link a directory to a LINK object.
- An object that was not a group object was passed to the **nisgrpadm** command.
- An operation on a group object was expected, but the type of object specified was not a group object.
- An operation on a table object was expected, but the object specified was not a table object.

nisinit Fails

Make sure that:

- You can **ping** the NIS+ server to check that it is up and running as a machine.
- The NIS+ server that you specified with the **–H** option is a valid server and that it is running the NIS+ software.
- **rpc.nisd** is running on the server.
- The nobody class has read permission for this domain.
- The netmask is properly set up on this machine.

Checkpoint Keeps Failing

If checkpoint operations with a **nisping –C** command consistently fail, make sure you have sufficient swap and disk space. Check for error messages in **syslog**. Check for **core** files filling up space.

Cannot Add User to a Group

A user must first be an NIS+ principal client with a LOCAL credential in the domain's **cred** table before the user can be added as a member of a group in that domain. A DES credential alone is not sufficient.

Logs Grow too Large

Failure to regularly checkpoint your system with **nisping –C** causes your log files to grow too large. Logs are not cleared on a master until **all** replicas for that master are updated. If a replica is down or otherwise out of service or unreachable, the master's logs for that replica cannot be cleared. Thus, if a replica is going to be down or out of service for a period of time, you should remove it as a replica from the master with **nisrmdir –f –s** for all directories, including **groups_dir** and **org_dir**.

Lack of Disk Space

Lack of sufficient disk space will cause a variety of different error messages. (See Insufficient Disk Space, on page 8-26 for additional information.

Cannot Truncate Transaction Log File

First, check to make sure that the file in question exists and is readable and that you have permission to write to it.

- You can use **ls –l /var/nis/trans.log** to display the transaction log.
- You can use **nisls –l** and **niscat** to check for existence, permissions, and readability.
- You can use **syslog** to check for relevant messages.

The most likely cause of inability to truncate an existing log file for which you have the proper permissions is lack of disk space. (The checkpoint process first creates a duplicate temporary file of the log before truncating the log and then removing the temporary file. If there is not enough disk space for the temporary file, the checkpoint process cannot proceed.) Check your available disk space and free up additional space if necessary.

Domain Name Confusion

Domain names play a key role in many NIS+ commands and operations. To avoid confusion, you must remember that except for root servers, all NIS+ masters and replicas are clients of the domain **above** the domain that they serve. If you make the mistake of treating a server or replica as if it were a client of the domain that it serves, you may get **Generic system error** or **Possible loop detected in namespace** directoryname**:** domainname error messages.

For example, the machine **aladin** might be a client of the subwiz.wiz.com. domain. If the master server of the subwiz.wiz.com. subdomain is the machine **merlin**, then **merlin** is a client of the wiz.com. domain. When using, specifying, or changing domains, remember these rules to avoid confusion:

1. Client machines belong to a given domain or subdomain.
2. Servers and replicas that serve a given subdomain are clients of the domain above the domain they are serving.

The only exception to Rule 2 is that the root master server and root replica servers are clients of the same domain that they serve. In other words, the root master and root replicas are all clients of the root domain.

Thus, in the example above, the fully qualified name of the **aladin** machine is aladin.subwiz.wiz.com. The fully qualified name of the **merlin** machine is merlin.wiz.com.

The name merlin.subwiz.wiz.com. is wrong and will cause an error because **merlin** is a client of wiz.com., not subwiz.wiz.com.

Cannot Delete org_dir or groups_dir

Always delete **org_dir** and **groups_dir** *before* deleting their parent directory. If you use **nisrmdir** to delete the domain before deleting the domain's **groups_dir** and **org_dir**, you will not be able to delete either of those two subdirectories.

Troubleshooting NIS+ Namespace Database Problems

This section covers problems related to the namespace database and tables.

Symptoms

Error messages with operative clauses such as:

”Abort_transaction: Internal database error”

”Abort_transaction: Internal Error, log entry corrupt”

”Callback: – select failed”

”CALLBACK_SVC: bad argument”

See also Ownership and Permission Problems, on page 8-14.

Multiple rpc.nisd Parent Processes

Symptoms

Various Database and transaction log corruption error messages containing the terms:

”Corrupt log”

”Log corrupted”

”Log entry corrupt”

”Corrupt database”

”Database corrupted”

Possible Causes

You have multiple *independent* **rpc.nisd** daemons running. In normal operation, **rpc.nisd** may spawn other child **rpc.nisd** daemons. This causes no problem. However, if two parent **rpc.nisd** daemons are running at the same time on the same machine, they will overwrite each other’s data and corrupt logs and databases. (Normally, this could only occur if someone started running **rpc.nisd** by hand.)

Diagnosis

Run **ps –ef | grep rpc.nisd**. Make sure that you have no more than one parent **rpc.nisd** process.

Solution

If you have more than one parent **rpc.nisd** entries, you must kill all but one of them. Use **kill –9** *process–id*, then run the **ps** command again to make sure it has died. (If you started **rpc.nisd** with the –B option, you must also kill the **rpc.nisd_resolv** daemon.)

If an NIS+ database is corrupt, you will also have to restore it from your most recent backup that contains an uncorrupted version of the database. You can then use the logs to update changes made to your namespace since the backup was recorded. However, if your logs are also corrupted, you will have to recreate by hand any namespace modifications made since the backup was taken.

Troubleshooting NIS Compatibility Problems

This section describes problems having to do with NIS compatibility with NIS+ and earlier systems and the switch configuration file.

Symptoms

The **/etc/irs.conf** file fails to perform correctly.

Error messages with operative clauses such as:

"Unknown user"

"Permission denied"

"Invalid principal name"

User Cannot Log In After Password Change

Symptoms

New users, or users who recently changed their password are unable to log in at all, or able to log in on one or more machines but not on others. The user may see error messages with operative clauses such as:

"Unknown user: *username*"

"Permission denied"

"Invalid principal name"

First Possible Cause

Password was changed on NIS machine.

If a user or system administrator uses the **yppasswd** command to change a password on a NIS machine running NIS in a domain served by NIS+ namespace servers, the user's password is changed only in that machine's **/etc/passwd** file. If the user then goes to some other machine on the network, the user's new password will not be recognized by that machine. The user will have to use the old password stored in the NIS+ passwd table.

Diagnosis

Check to see if the user's old password is still valid on another NIS+ machine.

Solution

Use **passwd** on a machine running NIS+ to change the user's password.

Second Possible Cause

Password changes take time to propagate through the domain.

Diagnosis

Namespace changes take a measurable amount of time to propagate through a domain and an entire system. This time might be as short as a few seconds or as long as many minutes, depending on the size of your domain and the number of replica servers.

Solution

You can simply wait the normal amount of time for a change to propagate through your domains. Or you can use the **nisping org_dir** command to resynchronize your system.

/etc/irs.conf File Fails to Perform Correctly

A modified (or newly installed) /**etc**/**irs.conf** file fails to work properly.

Symptoms

You install a new /**etc**/**irs.conf** file or make changes to the existing file, but your system does not implementy the changes.

Possible Cause

Each time the /**etc**/**irs.conf** file is installed or changed, you must reboot the machine for your changes to take effect.

Solution

Check your /**etc**/**irs.conf** file against the information contained in the /**etc**/**irs.conf** file description. Correct the file if necessary, then reboot the machine.

Troubleshooting Object Not Found Problems

This section describes problem in which NIS+ was unable to find some object or principal.

Symptoms

Error messages with operative clauses such as:

”Not found”

”Not exist”

”Can’t find suitable transport for name”

”Cannot find”

”Unable to find”

”Unable to stat”

Syntax or Spelling Error

The most likely cause of some NIS+ object not being found is that you mistyped or misspelled its name. Check the syntax and make sure that you are using the correct name.

Incorrect Path

A likely cause of an ***object*** **not found** problem is specifying an incorrect path. Make sure that the path you specified is correct. Also make sure that the **NIS_PATH** environment variable is set correctly.

Domain Levels Not Correctly Specified

Remember that all servers are clients of the domain above them, not the domain they serve. There are two exceptions to this rule:

- The root masters and root replicas are clients of the root domain.
- NIS+ **domain names end with a period.**

  When using a fully qualified name you must end the domain name with a period. If you do not end the domain name with a period, NIS+ assumes it is a partially qualified name. However, the domain name of a machine should not end with a dot in the **/etc/defaultdomain** file. If you add a dot to a machine’s domain name in the **/etc/defaultdomain** file, you will get **Could not bind to server serving domain** name error messages and encounter difficulty in connecting to the net on boot up.

Object Does Not Exist

The NIS+ object may not have been found because it does not exist, either because it has been erased or not yet created. Use **nisls –l** in the appropriate domain to check that the object exists.

Lagging or Out–of–Sync Replica

When you create or modify an NIS+ object, there is a time lag between the completion of your action and the arrival of the new updated information at a given replica. In ordinary operation, namespace information may be queried from a master or any of its replicas. A client automatically distributes queries among the various servers (master and replicas) to balance system load. This means that at any given moment you do not know which machine is supplying you with namespace information. If a command relating to a newly created or modified object is sent to a replica that has not yet received the updated information from the master, you will get an ”object not found” type of error or the old out–of–date information. Similarly, a general command such as **nisls** may not list a newly created object if the system sends the **nisls** query to a replica that has not yet been updated. (Unlike NIS, there is no binding with NIS+.)

You can use **nisping** to resync a lagging or out of sync replica server.

Alternatively, you can use the **–M** option with most NIS+ commands to specify that the command must obtain namespace information from the domain's master server. In this way you can be sure that you are obtaining and using the most up–to–date information. (However, you should use the **–M** option only when necessary because a main point of having and using replicas to serve the namespace is to distribute the load and thus increase network efficiency.)

Files Missing or Corrupt

One or more of the files in /**var**/**nis**/**data** directory has become corrupted or erased. Restore these files from your most recent backup.

Blanks in Name

Symptoms

Sometimes an object is there, sometimes it is not. Some NIS+ or AIX commands report that an NIS+ object does not exist or cannot be found, while other NIS+ or AIX commands do find that same object.

Diagnoses

Use **nisls** to display the object's name. Look carefully at the object's name to see if the name actually begins with a blank space. (If you accidentally enter two spaces after the flag when creating NIS+ objects from the command line with NIS+ commands, some NIS+ commands will interpret the second space as the beginning of the object's name.)

Solution

If an NIS+ object name begins with a blank space, you must either rename it without the space or remove it and then recreate it from scratch.

Cannot Use Automounter

Symptoms

You cannot change to a directory on another host.

Possible Cause

Under NIS+, automounter names must be renamed to meet NIS+ requirements. NIS+ cannot access /**etc**/**auto*** tables that contain a period in the name. For example, NIS+ cannot access a file named **auto.direct**.

Diagnosis

Use **nisls** and **niscat** to determine if the automounter tables are properly constructed.

Solution

Change the periods to underscores. For example, change **auto.direct** to **auto_direct**. (Be sure to change other maps that might reference these.)

Ownership and Permission Problems

This section describes problems related to user ownership and permissions.

Symptoms

Error messages with operative clauses such as:

”Unable to stat name”

”Unable to stat NIS+ directory name”

”Security exception on LOCAL system”

”Unable to make request”

”Insufficient permission to . . .”

”You name do not have secure RPC credentials”

Another Symptom:

User or root unable to perform any namespace task.

No Permission

The most common permission problem is the simplest: you have not been granted permission to perform some task that you try to do. Use **niscat –o** on the object in question to determine what permissions you have. If you need additional permission, you, the owner of the object, or the system administrator can either change the permission requirements of the object (as described in Administering NIS+ Access Rights,) or add you to a group that does have the required permissions (as described in Administering NIS+ Groups).

No Credentials

Without proper credentials for you and your machine, many operations will fail. Use **nismatch** on your home domain’s cred table to make sure you have the right credentials. See Corrupted Credentials, on page 8-19 for more on credentials–related problems.

Server Running at Security Level 0

A server running at security level 0 does not create or maintain credentials for NIS+ principals.

If you try to use **nispasswd** on a server that is running at security level 0, you will get the error message: **You** name **do not have secure RPC credentials in** NIS+ **domain** name.

Security level 0 is only to be used by administrators for initial namespace setup and testing purposes. Level 0 should not be used in any environment where ordinary users are active.

User Login Same as Machine Name

A user cannot have the same login ID as a machine name. When a machine is given the same name as a user (or vice versa), the first principal can no longer perform operations requiring secure permissions because the second principal’s key has overwritten the first principal’s key in the cred table. In addition, the second principal now has whatever permissions were granted to the first principal.

For example, suppose a user with the login name of **pine** is granted namespace read–only permissions. Then a machine named **pine** is added to the domain. The user **pine** will no longer be able to perform any namespace operations requiring any sort of permission, and the root user of the machine **pine** will only have read–only permission in the namespace.

Symptoms

- The user or machine gets ”permission denied” error messages.
- Either the user or root for that machine cannot successfully run **keylogin**.
- **Security exception on LOCAL system. UNABLE TO MAKE REQUEST.** error message.
- If the first principal did not have read access, the second principal might not be able to view otherwise visible objects.

  When running **nisclient** or **nisaddcred**, if the message **Changing Key** is displayed rather than **Adding Key**, there is a duplicate user or host name already in existence in that domain.

Diagnosis

Run **nismatch** to find the host and user in the hosts and passwd tables to see if there are identical host names and user names in the respective tables:

```
nismatch username passwd.org_dir
```

Then run **nismatch** on the domain’s cred table to see what type of credentials are provided for the duplicate host or user name. If there are both LOCAL and DES credentials, the cred table entry is for the user; if there is only a DES credential, the entry is for the machine.

Solution

Change the machine name. (It is better to change the machine name than to change the user name.) Then delete the machine’s entry from the **cred** table and use **nisclient** to reinitialize the machine as an NIS+ client. (If you wish, you can use **nistbladm** to create an alias for the machine’s old name in the hosts tables.) If necessary, replace the user’s credentials in the cred table.

Bad Credentials

See Corrupted Credentials, on page 8-19.

Troubleshooting Security Problems

This section describes common password, credential, encryption, and other security–related problems.

Symptoms

Error messages with operative clauses such as:

”Authentication error”

”Authentication denied”

”Cannot get public key”

”Chkey failed”

”Insufficient permission to”

”Login incorrect”

”Keyserv fails to encrypt”

”No public key”

”Permission denied”

”Password [problems]”

User or root unable to perform any namespace operations or tasks. (See also Ownership and Permission Problems, on page 8-14.)

”Login Incorrect” Message

The most common cause of a ”login incorrect” message is the user mistyping the password. Have the user try it again. Make sure the user knows the correct password and understands that passwords are case–sensitive and that the letter ”o” is not interchangeable with the numeral ”0,” nor is the letter ”l” the same as the numeral ”1.”

Other possible causes of the ”login incorrect” message are:

- The password has been locked by an administrator.
- The password has been locked because the user has exceeded an inactivity maximum.
- The password has expired.

See Administering Passwords, on page 6-36 for more information.

Password Locked, Expired, or Terminated

A common cause of a ”Permission denied, password expired,” type message is that the user’s password has passed its age limit or the user’s password privileges have expired. See Administering Passwords, on page 6-36 for more information on passwords.

Stale and Outdated Credential Information

Occasionally, you may find that even though you have created the proper credentials and assigned the proper access rights, some client requests still get denied. This may be due to out–of–date information residing somewhere in the namespace.

Storing and Updating Credential Information

Credential–related information, such as public keys, is stored in many locations throughout the namespace. NIS+ updates this information periodically, depending on the time–to–live values of the objects that store it, but sometimes, between updates, it gets out of sync. As a result, you may find that operations that should work, don’t work. The following table lists all the objects, tables, and files that store credential–related information and how to reset it.

Table 65. Where Credential–Related Information is Stored

Item	Stores	To Reset or Change
cred table	NIS+ principal's secret key and public key. These are the master copies of these keys.	Use **nisaddcred** to create new credentials; it updates existing credentials. An alternative is **chkey**.
Directory object	A copy of the public key of each server that supports it.	Run the **nisupdkeys** command on the directory object.
Keyserver	The secret key of the NIS+ principal that is currently logged in.	Run **keylogin** for a principal user or **keylogin –r** for a principal workstation.
NIS+ daemon	Copies of directory objects, which in turn contain copies of their servers' public keys.	Kill the daemon and the cache manager. Then restart both.
Directory cache	A copy of directory objects, which in turn contain copies of their servers' public keys.	Kill the NIS+ cache manager and restart it with the **stopsrc –s nis_cachemgr –a ”–i”** command. The **–i** option resets the directory cache from the cold–start file and restarts the cache manager.
Cold–start file	A copy of a directory object, which in turn contains copies of its servers' public keys.	On the root master, kill the NIS+ daemon and restart it. The daemon reloads new information into the existing **NIS_COLD_START** file. For a client, first remove the cold–start and shared directory files from /**var**/**nis**, and kill the cache manager. Then re–initialize the principal with **nisinit –c**. The principal's trusted server reloads new information into the principal's existing cold–start file.
passwd table	A user's password or a workstation's superuser password.	Use the **passwd** command. It changes the password in the NIS+ passwd table and updates it in the cred table.
passwd file	A user's password or a workstation's superuser password.	Use the **passwd** command, whether logged in as superuser or as yourself, whichever is appropriate.

Updating Stale Cached Keys

The most commonly encountered out–of–date information is the existence of stale objects with old versions of a server's public key. You can usually correct this problem by running **nisupdkeys** on the domain you are trying to access. (See Administering NIS+ Credentials, on page 6-2, for information on using the **nisupdkeys** command.)

Because some keys are stored in files or caches, **nisupdkeys** cannot always correct the problem. At times you might need to update the keys manually. To do that, you must understand how a server's public key, once created, is propagated through namespace objects. The process usually has five stages of propagation:

- Stage 1: Server's Public Key Is Generated, on page 8-18

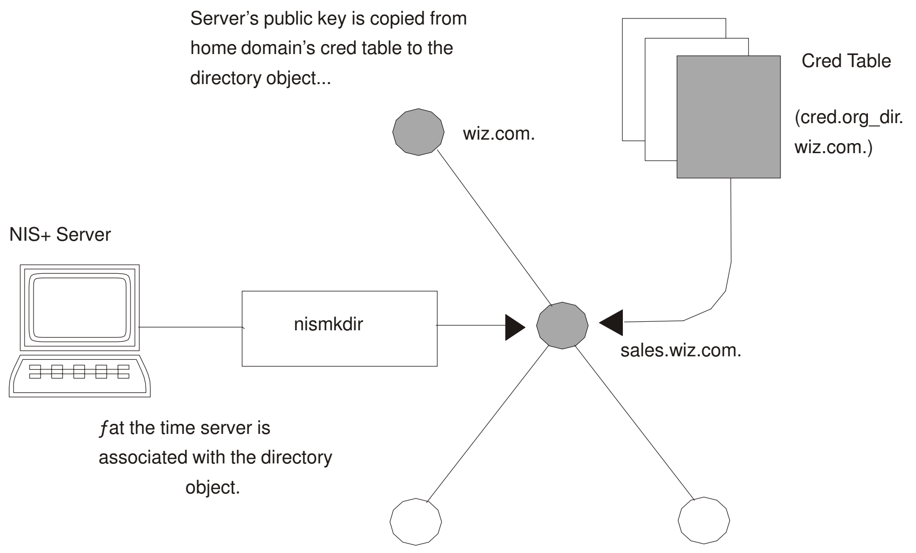
- Stage 2: Public Key Is Propagated to Directory Objects, on page 8-18
- Stage 3: Directory Objects Are Propagated Into Client Files, on page 8-18
- Stage 4: When a Replica is Added to the Domain, on page 8-19
- Stage 5: When the Server's Public Key Is Changed, on page 8-19

Stage 1: Server's Public Key Is Generated

An NIS+ server is first an NIS+ client. So, its public key is generated in the same way as any other NIS+ client's public key: with the **nisaddcred** command. The public key is then stored in the cred table of the server's home domain, not of the domain the server will eventually support.

Stage 2: Public Key Is Propagated to Directory Objects

Once you have set up an NIS+ domain and an NIS+ server, you can associate the server with a domain. This association is performed by the **nismkdir** command. When the **nismkdir** command associates the server with the directory, it also copies the server's public key from the cred table to the domain's directory object. For example, assume the server is a client of the wiz.com. root domain, and is made the master server of the sales.wiz.com. domain. For more information, see the diagram.


Server's public key is copied from home domain's cred table to the directory object...
wiz.com.
Cred Table
(cred.org_dir.
wiz.com.)
NIS+ Server
nismkdir
sales.wiz.com.
ƒat the time server is associated with the directory object.

Its public key is copied from the cred.org_dir.wiz.com. domain and placed in the sales.wiz.com. directory object. This can be verified with the **niscat –o Sales.wiz.com**. command.

Stage 3: Directory Objects Are Propagated Into Client Files

All NIS+ clients are initialized with the **nisinit** utility or with the **nisclient** script.

Among other things, **nisinit** (or **nisclient**) creates a cold–start file /**var**/**nis**/**NIS_COLDSTART**. The cold–start file is used to initialize the client's directory cache /**var**/**nis**/**NIS_SHARED_DIRCACHE**. The cold–start file contains a copy of the directory object of the client's domain. Since the directory object already contains a copy of the server's public key, the key is now propagated into the cold–start file of the client.

In addition when a client makes a request to a server outside its home domain, a copy of the remote domains directory object is stored in the client's **NIS_SHARED_DIRCACHE** file. You can examine the contents of the client's cache by using the **nisshowcache** command.

This is the extent of the propagation until a replica is added to the domain or the server's key changes.

Stage 4: When a Replica is Added to the Domain

When a replica server is added to a domain, the **nisping** command is used to download the NIS+ tables, including the cred table, to the new replica. Therefore, the original server's public key is now also stored in the replica server's cred table.

Stage 5: When the Server's Public Key Is Changed

If you decide to change DES credentials for the server (that is, for the root identity on the server), its public key will change. As a result, the public key stored for that server in the cred table will be different from those stored in:

- The cred table of replica servers (for a few minutes only)
- The main directory object of the domain supported by the server (until its time–to–live expires)
- The **NIS_COLDSTART** and **NIS_SHARED_DIRCACHE** files of every client of the domain supported by server (until their time–to–live expires, usually 12 hours)
- The **NIS_SHARED_DIRCACHE** file of clients who have made requests to the domain supported by the server (until their time–to–live expires)

Most of these locations will be updated automatically within a time ranging from a few minutes to 12 hours. To update the server's keys in these locations immediately, use the commands:

Table 66. Updating a Server's Keys

Location	Command
cred table of replica servers (instead of using **nisping**, you can wait a few minutes until the table is updated automatically)	**nisping**
Directory object of domain supported by server	**nisupdkeys**
NIS_COLDSTART file of clients	**nisinit –c**
NIS_SHARED_DIRCACHE file of clients	**nis_cachemgr**

Note: You must first kill the existing **nis_cachemgr** before restarting **nis_cachemgr**.

Corrupted Credentials

When a principal (user or machine) has a corrupt credential, that principal is unable to perform any namespace operations or tasks, not even **nisls**. This is because a corrupt credential provides no permissions at all, not even the permissions granted to the nobody class.

Symptoms

User or root cannot perform any namespace tasks or operations. All namespace operations fail with a ”permission denied” type of error message. The user or root cannot even perform a **nisls** operation.

Possible Cause

Corrupted keys or a corrupt, out–of–date, or otherwise incorrect /**etc**/**.rootkey** file.

Diagnosis

Use **iptrace** to identify the bad credential.

Or, if the principal is listed, log in as the principal and try to run an NIS+ command on an object for which you are sure that the principal has proper authorization. For example, in most cases an object grants read authorization to the nobody class. Thus, the **nisls** object command should work for any principal listed in the cred table. If the command fails with a ”permission denied” error, then the principal's credential is likely corrupted.

Solution

- **Ordinary user**. Perform a **keylogout** and then a **keylogin** for that principal.
- **Root/superuser**. Run **keylogout –f** followed by **keylogin –r**.

Keyserv Failure

The **keyserv** is unable to encrypt a session. There are several possible causes for this type of problem:

Possible Causes and Solutions

- The client has not keylogged in. Make sure that the client is keylogged in. To determine if a client is properly keylogged in, have the client run **nisdefaults** –v (or run it yourself as the client). If **(not authenticated)** is returned on the **Principal Name** line, the client is not properly keylogged in.
- The client (host) does not have appropriate LOCAL or DES credentials. Run **niscat** on the client's cred table to verify that the client has appropriate credentials. If necessary, add credentials as explained in Creating Credential Information for NIS+ Principals, on page 6-12.
- The **keyserv** daemon is not running. Use the **ps** command to see if **keyserv** is running. If it is not running, restart it and then do a **keylogin**.
- While **keyserv** is running, other long running processes that make secure RPC or NIS+ calls are not. For example, **automountd**, **rpc.nisd**, and **sendmail**. Verify that these processes are running correctly. If they are not, restart them.

Machine Previously Was an NIS+ Client

If this machine has been initialized before as an NIS+ client of the same domain, try **keylogin –r** and enter the root login password at the Secure RPC password prompt.

No Entry in the cred Table

To make sure that an NIS+ password for the principal (user or host) exists in the cred table, run the following command in the principal's home domain

```
nisgrep –A cname= principal cred.org_dir.domainname
```

If you are running **nisgrep** from another domain, the **domainname** must be fully qualified.

Changed Domain Name

Do not change a domain name.

If you change the name of an existing domain you will create authentication problems because the fully qualified original domain name is embedded in objects throughout your network.

If you have **already** changed a domain name and are experiencing authentication problems, or error messages containing terms like "malformed" or "illegal" in relation to a domain name, change the domain name back to its original name. The recommended procedure for renaming your domains is to create a **new** domain with the **new** name, set up your machines as servers and clients of the new domain, make sure they are performing correctly, and then remove the old domain.

When Changing a Machine to a Different Domain

If this machine is an NIS+ client and you are trying to change it to a client of a different domain, remove the **/etc/.rootkey** file, and then rerun the **nisclient** script using the network password supplied by your network administrator or taken from the **nispopulate** script.

NIS+ Password and Login Password in /etc/passwd File

Your NIS+ password is stored in the NIS+ passwd table. Your user login password may be stored in NIS+ passwd table or in your /**etc**/**passwd** file. (Your user password and NIS+ password can be the same or different.) To change a password in an /**etc**/**passwd** file, you must run the **passwd** command.

The /**etc**/**irs.conf** file specifies which password is used for which purpose. If the /**etc**/**irs.conf** file is directing system queries to the wrong location, you will get password and permission errors.

Secure RPC Password and Login Passwords Are Different

When a principal's login password is different from his or her Secure RPC password, **keylogin** cannot decrypt it at login time because **keylogin** defaults to using the principal's login password, and the private key was encrypted using the principal's Secure RPC password.

When this occurs the principal can log in to the system, but for NIS+ purposes is placed in the authorization class of nobody because the keyserver does not have a decrypted private key for that user. Since most NIS+ environments are set up to deny the nobody class create, destroy, and modify rights to most NIS+ objects this results in "permission denied" types errors when the user tries to access NIS+ objects.

To be placed in one of the other authorization classes, a user in this situation must explicitly run the **keylogin** program and give the principal's Secure RPC password when **keylogin** prompts for password. (See Keylogin, on page 6-16.) (In this context, *net work password* is sometimes used as a synonym for Secure RPC password. When prompted for your network password, enter your Secure RPC password.)

But an explicit **keylogin** provides only a temporary solution that is good only for the current login session. The keyserver now has a decrypted private key for the user, but the private key in the user's cred table is still encrypted using the user's Secure RPC password, which is different than the user's login password. The next time the user logs in, the same problem reoccurs. To permanently solve the problem the user needs to change the private key in the cred table to one based on the user's login ID rather than the user's Secure RPC password. To do this, the user need to run the **chkey** program as described in Changing Keys for a NIS+ Principal, on page 6-16.

Thus, to permanently solve a Secure RPC password different than login password problems, the user (or an administrator acting for the user) must perform the following steps:

1. Log in using the login password.
2. Run the **keylogin** program to temporarily get a decrypted private key stored in the keyserver and thus gain temporary NIS+ access privileges.
3. Run **chkey –p** to permanently change the encrypted private key in the cred table to one based on the user's login password.

Preexisting /etc/.rootkey File

Symptoms

Various "insufficient permission to" and "permission denied" error messages.

Possible Cause

An /**etc**/**.rootkey** file already existed when you set up or initialized a server or client. This could occur if NIS+ had been previously installed on the machine and the **.rootkey** file was not erased when NIS+ was removed or the machine returned to using NIS or /**etc** files.

Diagnosis

Run **ls –l** on the /**etc** directory and **nisls –l org_dir** and compare the date of the /**etc**/**.rootkey** to the date of the cred table. If the /**etc**/**.rootkey** date is clearly earlier than that of the cred table, it may be a preexisting file.

Solution

Run **keylogin –r** as root on the problem machine and then set up the machine as a client again.

Root Password Change Causes Problem

Symptoms

You change the root password on a machine, and the change either fails to take effect or you are unable to log in as superuser.

Possible Cause

You changed the root password, but root's key was not properly updated. Either because you forgot to run **chkey –p** for root or some problem came up. (Note that you should not have UserID 0 in the password table.)

Solution

Log in as a user with administration privileges (that is, a user who is a member of a group with administration privileges) and use **passwd** to restore the old password. Make sure that old password works. Now use **passwd** to change root's password to the new one, and then run **chkey –p**.

Note: Once your NIS+ namespace is set up and running, you can change the root password on the root master machine. But do not change the root master keys, as these are embedded in all directory objects on all clients, replicas, and servers of subdomains. To avoid changing the root master keys, always use the **–p** option when running **chkey** as root.

Troubleshooting Slow Performance and System Hang Problems

This section describes common slow performance and system hang problems.

Symptoms

Error messages with operative clauses such as:

”Busy try again later”

”Not responding”

Other common symptoms:

- You issue a command and nothing seems to happen for far too long.
- Your system, or shell, no longer responds to keyboard or mouse commands.
- NIS+ operations seem to run slower than they should or slower than they did before.

Checkpointing

Someone has issued an **nisping** or **nisping –C** command. Or the **rpc.nisd** daemon is performing a checkpoint operation.

Attention: Do not reboot! Do not issue any more **nisping** commands. You may have corrupted information in the NIS+ domain.

When performing a **nisping** or checkpoint, the server will be sluggish or may not immediately respond to other commands. Depending on the size of your namespace, these commands may take a noticeable amount of time to complete. Delays caused by checkpoint or ping commands are multiplied if you, or someone else, enter several such commands at one time. Do not reboot. This kind of problem will solve itself. Just wait until the server finishes performing the **nisping** or checkpoint command.

During a full master–replica resync, the involved replica server will be taken out of service until the resync is complete. Do not reboot—just wait.

Variable NIS_PATH

Make sure that your **NIS_PATH** variable is set to something clean and simple. For example, the default: **org_dir.$:$**. A complex **NIS_PATH**, particularly one that itself contains a variable, will slow your system and may cause some operations to fail. (See NIS_PATH Environment Variable, on page 4-17 for more information.)

Do not use **nistbladm** to set nondefault table paths. Nondefault table paths will slow performance.

Table Paths

Do not use table paths because they will slow performance.

Too Many Replicas

Too many replicas for a domain degrade system performance during replication. There should be no more than 10 replicas in a given domain or subdomain. If you have more than five replicas in a domain, try removing some of them to see if that improves performance.

Recursive Groups

A recursive group is a group that contains the name of some other group. While including other groups in a group reduces your work as system administrator, doing so slows down the system. You should not use recursive groups.

Large NIS+ Database Logs at Start–up

When **rpc.nisd** starts up it goes through each log. If the logs are long, this process could take a long time. If your logs are long, you may want to checkpoint them using **nisping –C** before starting **rpc.nisd**.

The Master rpc.nisd Daemon Died

Symptoms

If you used the **–M** option to specify that your request be sent to the master server, and the **rpc.nisd** daemon has died on that machine, you will get a ”server not responding” type error message and no updates will be permitted. (If you did not use the **–M** option, your request will be automatically routed to a functioning replica server.)

Possible Cause

Using uppercase letters in the name of a home directory or host can sometimes cause **rpc.nisd** to die.

Diagnosis

First make sure that the server itself is up and running. If it is, run **ps –ef | grep rpc.nisd** to see if the daemon is still running.

Solution

If the daemon has died, restart it. If **rpc.nisd** frequently dies, contact your service provider.

No nis_cachemgr

Symptoms

It takes too long for a machine to locate namespace objects in other domains.

Possible Cause

You do not have **nis_cachemgr** running.

Diagnosis

Run **ps –ef | grep nis_cachemgr** to see if it is still running.

Solution

Start **nis_cachemgr** on that machine.

Server Very Slow at Start–up After NIS+ Installation

Symptoms

A server performs slowly and sluggishly after using the NIS+ scripts to install NIS+ on it.

Possible Cause

You forgot to run **nisping –C –a** after running the **nispopulate** script.

Solution

Run **nisping –C –a** to checkpoint the system as soon as you are able to do so.

niscat Returns: Server busy. Try Again

Symptoms

You run **niscat** and get an error message indicating that the server is busy.

Possible Cause

The server is busy with a heavy load, such as when doing a resync.

The server is out of swap space.

Diagnosis

Run **lsps –a** to check your server's swap space.

Solution

You must have adequate swap and disk space to run NIS+. If necessary, increase your space.

NIS+ Queries Hang After Changing Host Name

Symptoms

Setting the host name for an NIS+ server to be fully qualified is not recommended. If you do so, and NIS+ queries then just hang with no error messages, check the following possibilities:

Possible Cause

Fully qualified host names must meet the following criteria:

- The domain part of the host name must be the same as the name returned by the **domainname** command.
- The host name must end in a period.

Solution

Kill the NIS+ processes that are hanging and then kill **rpc.nisd** on that host or server. Rename the host to match the requirements listed above (use SMIT or the **hostname** command). Then reinitialize the server with **nisinit**. (If queries still hang after you are sure that the host is correctly named, check other problem possibilities in this section.)

Note: If you started rpc.nisd with the –B option, you must also kill the rpc.nisd_resolv daemon.

Troubleshooting System Resource Problems

This section describes problems having to do with lack of system resources such as memory, disk space, and so forth.

Symptoms

Error messages with operative clauses such as:

”No memory”

”Out of disk space”

”Cannot [do something] with log”

”Unable to fork”

Insufficient Memory

Lack of sufficient memory or swap space on the system you are working with will cause a wide variety of NIS+ problems and error messages. As a short–term, temporary solution, try to free additional memory by killing unneeded windows and processes. If necessary, exit your windowing system and work from the terminal command line. If you still get messages indicating inadequate memory, you will have to install additional swap space or memory, or switch to a different system that has enough swap space or memory.

Under some circumstances, applications and processes may develop memory leaks and grow too large. you can check the current size of an application or process by running:

```
ps –el
```

The **sz** (size) column shows the current memory size of each process. If necessary, compare the sizes with comparable processes and applications on a machine that is not having memory problems to see if any have grown too large.

Insufficient Disk Space

Lack of disk space will cause a variety of error messages. A common cause of insufficient disk space is failure to regularly remove **tmp** files and truncate log files. Log and **tmp** files grow steadily larger unless truncated. The speed at which these files grow varies from system to system and with the system state. Log files on a system that is working inefficiently or having namespace problems will grow very fast.

Note: If you are doing a lot of troubleshooting, check your log and /**tmp** files frequently. Truncate log files and remove /**tmp** files before lack of disk space creates additional problems. Also check the root directory and home directories for core files and delete them.

The way to truncate log files is to regularly checkpoint your system (Keep in mind that a checkpoint process may take some time and will slow down your system while it is being performed, checkpointing also requires enough disk space to create a complete copy of the files before they are truncated.)

To checkpoint a system, run **nisping –C**.

Insufficient Processes

On a heavily loaded machine it is possible that you could reach the maximum number of simultaneous processes that the machine is configured to handle. This causes messages with clauses like ”unable to fork”. The recommended method of handling this problem is to kill any unnecessary processes. If the problem persists, you can reconfigure the machine to handle more processes as described in your system administration documentation.

Troubleshooting User Problems

This section describes NIS+ problems that a typical user might encounter.

Symptoms

- User cannot log in.
- User cannot **rlogin** to other domain

User Cannot Log In

There are many possible reasons for a user being unable to log in:

- **User forgot password**. To set up a new password for a user who has forgotten the previous one, run **nispasswd** for that user on another machine (naturally, you have to be the NIS+ administrator to do this).
- **Mistyping password**. Make sure the user knows the correct password and understands that passwords are case–sensitive and that the letter **o** is not interchangeable with the numeral **0**, nor is the letter **l** the same as the numeral **1**.
- **”Login incorrect”–type message**. For causes other than simply mistyping the password, see ”Login Incorrect” Message, on page 8-16.
- The user’s password privileges have expired.
- An inactivity maximum has been set for this user, and the user has passed it.

Solution

Edit the **/etc/security/login.cfg**, **/etc/security/user**, and **/usr/lib/security/methods.cfg** configuration files. (Add them, if they do not exist.) The following lines must exist (and not be commented) in both files:

```
NISPLUS
  program=/usr/lib/security/NISPLUS
```

User Cannot Log In Using New Password

Symptoms

Users who recently changed their password are unable to log in at all, or are able to log in on some machines but not on others.

Possible Causes

- It may take some time for the new password to propagate through the network. Have users try to log in with the old password.
- The password was changed on a machine that was not running NIS+.

User Cannot Remote Log In to Remote Domain

Symptoms

User tries to **rlogin** to a machine in some other domain and is refused with a ”Permission denied” type error message.

Possible Cause

To **rlogin** to a machine in another domain, a user must have LOCAL credentials in that domain.

Diagnosis

Run **nismatch***username.domainname*.**cred.org_dir** in the other domain to see if the user has a LOCAL credential in that domain.

Solutions

- Go to the remote domain and use **nisaddcred** to create a LOCAL credential for the user in the that domain.
- Edit the **/etc/security/login.cfg**, **etc/security/user**, and **/usr/lib/security/methods.cfg** files. (Add them, if they do not exist.) The following lines must exist (and not be commented) in these files:

```
NISPLUS
  program=/usr/lib/security/NISPLUS
```

User Cannot Change Password

The most common cause of a user being unable to change passwords is that the user is mistyping (or has forgotten) the old password.

Other possible causes:

- The password Min value has been set to be greater than the password Max value.
- The password is locked or expired. See "Login Incorrect" Message, on page 8-16 and Password Locked, Expired, or Terminated, on page 8-16.

Troubleshooting Other NIS+ Problems

This section describes problems that do not fit any of the previous categories.

How to Tell if NIS+ Is Running

You may need to know whether a given host is running NIS+. A script may also need to determine whether NIS+ is running. You can check if NIS+ is running by entering:

```
ps –ef | grep nis_cachemgr
```

The host has a /**var**/**nis**/**NIS_COLD_START** file.

nisls succeeds.

Replica Update Failure

Symptoms

Error messages indicating that the update was not successfully complete. (Note that the message: **replica_update:** number **updates** number **errors** indicates a successful update.)

Possible Causes

Any of the following error messages indicate that the server was busy and that the update should be rescheduled:

- **Master server busy, full dump rescheduled**
- **replica_update: error result was Master server busy, full dump rescheduled**
- **replica_update: master server busy, rescheduling the resync**
- **replica_update: master server busy, will try later**
- **replica_update: nis dump result Master server busy, full dump rescheduled**
- **nis_dump_svc: one replica is already resyncing**

(These messages are generated by, or in conjunction with, the NIS+ error code constant: **NIS_DUMPLATER**.)

The following messages indicate that there was some other problem:

- **replica_update: error result was ...**
- **replica_update: nis dump result nis_perror** error string
- **root_replica_update: update failed string–variable: could not fetch object from master**

(If **rpc.nisd** is being run with the **–v** (open diagnostic channel) option, additional information may be entered in either the master server or replica server's system log.)

These messages indicate possible problems such as:

- The server is out of child processes that can be allocated.
- A read–only child process was requested to dump.
- Another replica is currently resynching.

Diagnosis

Check both the replica and server's system log for additional information. How much, if any, additional information is recorded in the system logs depends on your system's error reporting level, and whether or not you are running **rpc.nisd** with the **–v** option (diagnostics).

Solution

In most cases, these messages indicate minor software problems which the system is capable of correcting. If the message was the result of a command, simply wait for a while and then try the command again. If these messages appear often, you can change the threshold level in your **/etc/syslog.conf** file.

Appendix A. Information in NIS+ Tables

This appendix summarizes the information stored in the following NIS+ tables:

Without a name service, most network information would be stored in /**etc** files and almost all NIS+ tables have corresponding /**etc** files. With the NIS service, you stored network information in NIS maps that also mostly corresponded with /**etc** files.

If you are creating input files for any of these tables, most tables share two formatting requirements:

- You must use one line per entry
- You must separate columns with one or more spaces or Tabs.

If a particular table has different or additional format requirements, they are described under a heading named ”Input File Format.”

Auto_Home Table

The auto_home table is an indirect automounter map that enables an NIS+ client to mount the home directory of any user in the domain. It does this by specifying a mount point for each user's home directory, the location of each home directory, and mount options, if any. Because it is an indirect map, the first part of the mount point is specified in the auto_master table, which is, by default, /**home**. The second part of the mount point (that is, the subdirectory under /**home**) is specified by the entries in the auto_home map, and is different for each user.

The auto_home table has two columns:

Table 67. Auto_Home Table

Column	Content	Description
Key	Mount point	The login name of every user in the domain
Value	Options & location	The mount options for every user, if any, and the location of the user's home directory

For example:

```
costas  barcelona:/export/partition2/costas
```

The home directory of the user **costas**, which is located on the server **barcelona**, in the directory /**export**/**partition2**/**costas**, would be mounted under a client's /**home**/**costas** directory. No mount options were provided in the entry.

Auto_Master Table

The auto_master table lists all the automounter maps in a domain. For direct maps, the auto_master table provides a map name. For indirect maps, it provides both a map name and the top directory of its mount point. The auto_master table has two columns:

Table 68. Auto_Master Table

Column	Content	Description
Key	Mount point	The top directory into which the map will be mounted. If the map is a direct map, this is a dummy directory, represented with /**—.**
Value	Map name	The name of the automounter map

For example, assume these entries in the auto_master table:

```
/home  auto_home
/-auto_man
/programs auto_programs
```

The first entry names the auto_home map. It specifies the top directory of the mount point for all entries in the auto_home map: /**home**. (The auto_home map is an indirect map.) The second entry names the auto_man map. Because that map is a direct map, the entry provides only the map name. The auto_man map will itself provide the topmost directory, as well as the full pathname, of the mount points for each of its entries. The third entry names the auto_programs map and, since it provides the top directory of the mount point, the auto_programs map is an indirect map.

All automounter maps are stored as NIS+ tables. By default, the AIX environment provides the auto_master map, which is mandatory, and the auto_home map, which is a great convenience. You can create more automounter maps for a domain, but be sure to store them as NIS+ tables and list them in the auto_master table. For more information about the automounter consult books about the automounter or books that describe the NFS file system.

Bootparams Table

The bootparams table stores configuration information about every diskless workstation in a domain. A diskless workstation is a workstation that is connected to a network, but has no hard disk. Since it has no internal storage capacity, a diskless workstation stores its files and programs in the file system of a server on the network. It also stores its configuration information—or **boot parameters** —on a server.

Because of this arrangement, every diskless workstation has an initialization program that knows where this information is stored. If the network has no name service, the program looks for this information in the server's /**etc**/**bootparams** file. If the network uses the NIS+ name service, the program looks for it in the bootparams table, instead.

The bootparams table can store any configuration information about diskless workstations. It has two columns: one for the configuration key, another for its value. By default, it is set up to store the location of each workstation's root, swap, and dump partitions.

The default bootparams table has only two columns that provide the following items of information:

Table 69. Bootparams Table

Column	Content	Description
Key	Hostname	The diskless workstation's official host name, as specified in the hosts table
Value	Configuration	Root partition: the location (server name and path) of the workstation's root partition
		Swap partition: the location (server name and path) of the workstation's swap partition
		Dump partition: the location (server name and path) of the workstation's dump partition
		Install partition.
		Domain.

Input File Format

The columns are separated with a TAB character. Backslashes (\) are used to break a line within an entry. The entries for root, swap, and dump partitions have the following format:

```
client-name root=server:path \
      swap=server:path \
        dump=server:path \
        install=server:path \
        domain=domainname
```

Here is an example:

```
buckarooroot=bigriver:/export/root1/buckaroo \
  swap=bigriver:/export/swap1/buckaroo \
  dump=bigriver:/export/dump/buckaroo \
  install=bigriver:/export/install/buckaroo \
  domain=sales.wiz.com
```

Client_info Table

The client_info table stores each client's server discovery information, such as preferred servers and preferred options. The client_info table resides on each NIS+ server so two clients from the same server can look each other up through their own server.

The client_info table has four columns:

client	Client's name
attr	Client's attributes
info	Information about the client
flags	Flag information about the client

Note: **nispopulate** does not populate this table.

Cred Table

The cred table stores credential information about NIS+ principals. Each domain has one cred table, which stores the credential information of client workstations that belong to that domain and client users who are allowed to log into them. (In other words, the principals of that domain.) The cred tables are located in their domains' **org_dir** subdirectory.

Attention: Do not link a cred table. Each **org_dir** directory should have its own cred table. Do not use a link to some other **org_dir** cred table.

The cred table has five columns:

Table 70. Cred Table

NIS+ Principal Name	Authentication Type	Authentication Name	Public Data	Private Data
Principal name of a principal user	LOCAL	UID	GID list	
Principal name of a principal user or workstation	DES	Secure RPC netname	Public key	Encrypted private key

The second column, authentication type, determines the types of values found in the other four columns.

- **LOCAL**. If the authentication type is LOCAL, the other columns contain a principal user's name, UID, and GID; the last column is empty.
- **DES**. If the authentication type is DES, the other columns contain a principal's name, Secure RPC netname, public key, and encrypted private key. These keys are used in conjunction with other information to encrypt and decrypt a DES credential.

See Administering NIS+ Credentials, on page 6-2, for additional information on credentials and the cred table.

Ethers Table

The ethers table stores information about the 48–bit Ethernet addresses of workstations on the Internet. It has three columns:

Table 71. Ethers Table

Column	Content	Description
Addr	Ethernet–address	The 48–bit Ethernet address of the workstation
Name	Official–host–name	The name of the workstation, as specified in the hosts table
Comment	Comment	An optional comment about the entry

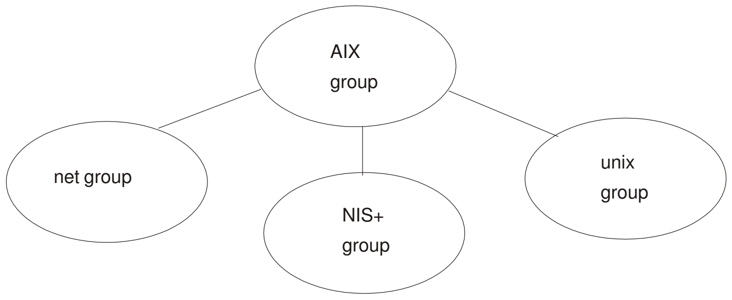
An Ethernet address has the form:

n:n:n:n:n:n hostname

where *n* is a hexadecimal number between 0 and FF, representing one byte. The address bytes are always in network order (most significant byte first).

Group Table

The group table stores information about workstation user groups. In the AIX environment, it stores three kinds of groups: net groups, NIS+ groups, and UNIX groups. For more information, see group table.



A net group is a group of workstations and users that have permission to perform remote operations on other workstations in the group. An NIS+ group is a set of NIS+ users that can be assigned access rights to an NIS+ object. They are described in Security. A UNIX group is simply a collection of users who are given additional UNIX access permissions.

UNIX groups allow a set of users on the network to access a set of files on several workstations or servers without making those files available to everyone. For example, the engineering and marketing staff working on a particular project could form a workstation user group.

The group table has four columns:

Table 72. Group Table

Column	Description
Name	The group’s name
Passwd	The group’s password
GID	The group’s numerical ID
Members	The names of the group members, separated by commas

Hosts Table

The hosts table associates the names of all the workstations in a domain with their IP addresses. The workstations are usually also NIS+ clients, but they don't have to be. Other tables, such as bootparams, group, and netgroup, rely on the network names stored in this table. They use them to assign other attributes, such as home directories and group memberships, to individual workstations. The hosts table has four columns:

Table 73. Hosts Table

Column	**Description**
Addr	The workstation's IP address (network number plus workstation ID number)
Cname	The workstation's official name
Name	A name used in place of the host name to identify the workstation
Comment	An optional comment about the entry

Mail_aliases Table

The mail_aliases table lists the domain's mail aliases recognized by **sendmail**. It has four columns:

Table 74. Mail_aliases Table

Column	Description
Alias	The name of the alias
Expansion	A list containing the members that receive mail sent to this alias; members can be users, workstations, or other aliases
Comment	An optional comment about the entry
Options	(See **sendmail** command description for options)

Input File Format

Each entry has the following format:

```
alias-name:member[,member]...
```

To extend an entry over several lines, use a backslash.

Netgroup Table

The netgroup table defines network wide groups used to check permissions for remote mounts, logins, and shells. The members of net groups used for remote mounts are workstations; for remote logins and shells, they are users.

Note: Users working on a client machine being served by a NIS+ server running in compatibility mode cannot run **ypcat** on the netgroup table. Doing so will give you results as if the table were empty even if it has entries.

The netgroup table has six columns:

Table 75. Netgroup Table

Column	Content	Description
Name	groupname	The name of the network group
Group	groupname	Another group that is part of this group
Host	hostname	The name of a host
User	username	A user's login name
Domain	domainname	A domain name
Comment	Comment	An optional comment about the entry

Input File Format

The input file consists of a group name and any number of members:

```
groupname member-list...
```

The member list can contain the names of other net groups or an ordered member list with three fields or both:

```
member-list::=groupname | (hostname, username, domainname)
```

The first field of the member list specifies the name of a workstation that belongs to the group. The second field specifies the name of a user that belongs to the group. The third field specifies the domain in which the member specification is valid.

A missing field indicates a wildcard. For example, this net group includes all workstations and users in all domains:

```
everybody (,,)
```

A dash in a field is the opposite of a wildcard; it indicates that no workstations or users belong to the group. Here are two examples:

```
(host1, -,wiz.com.)
(-,joe,wiz.com.)
```

The first specification includes one workstation, **host1**, in the wiz.com. domain, but excludes all users. The second specification includes one user in the wiz.com. domain, but excludes all workstations.

Netmasks Table

The netmasks table contains the network masks used to implement standard Internet subnetting. The table has three columns:

Table 76. Netmasks Table

Column	Description
Addr	The IP number of the network
Mask	The network mask to use on the network
Comment	An optional comment about the entry

For network numbers, you can use the conventional IP dot notation used by workstation addresses, but leave zeroes in place of the workstation addresses. For example, this entry

```
128.32.0.0          255.255.255.0
```

means that class B network 128.32.0.0 should have 24 bits in its subnet field, and 8 bits in its host field.

Networks Table

The networks table lists the networks of the Internet. This table is normally created from the official network table maintained at the Network Information Control Center (NIC), though you may need to add your local networks to it. It has four columns:

Table 77. Networks Table

Column	**Description**
Cname	The official name of the network, supplied by the Internet
Addr	The official IP number of the network
Name	An unofficial name for the network
Comment	An optional comment about the entry

Passwd Table

The passwd table contains information about the accounts of users in a domain. These users generally are, but do not have to be, NIS+ principals. Remember though, that if they are NIS+ principals, their credentials are not stored here, but in the domain's cred table. The passwd table usually grants read permission to the world (or to nobody).

Note: There should not be any entry in this table for the user root (user ID 0). Root's password information should be stored and maintained in the machine's /**etc** files.

The information in the passwd table is added when users' accounts are created.

The passwd table contains the following columns:

Table 78. Passwd Table

Column	Description
Name	The user's login name, which is assigned when the user's account is created; the name can contain no uppercase characters and can have a maximum of eight characters
Passwd	The user's encrypted password
UID	The user's numerical ID, assigned when the user's account is created
GID	The numerical ID of the user's default group
GCOS	The user's real name plus information that the user wishes to include in the From: field of a mail–message heading; an "&" in this column simply uses the user's login name
Home	The path name of the user's home directory.
Shell	The user's initial shell program.
Shadow	(See Passwd Table Shadow Column (the following table).)

The passwd table shadow column stores restricted information about user accounts. It includes the following information:

Table 79. Passwd Table Shadow Column

Item	Description
Lastchg	The number of days between January 1, 1970, and the date the password was last modified
Min	The minimum number of days recommended between password changes
Max	The maximum number of days that the password is valid
Warn	The number of days that a user receives warning before being notified that his or her password has expired
Inactive	The number of days of inactivity allowed for the user
Expire	An absolute date past which the user's account is no longer valid
Flag	Reserved for future use: currently set to 0.

Protocols Table

The protocols table lists the protocols used by the Internet. It has four columns:

Table 80. Protocols Table

Column	Description
Cname	The protocol name
Name	An unofficial alias used to identify the protocol
Number	The number of the protocol
Comments	Comments about the protocol

RPC Table

The RPC table lists the names of RPC programs. It has four columns:

Table 81. RPC Table

Column	Description
Cname	The name of the program
Number	The program number
Name	Other names that can be used to invoke the program
Comments	Comments about the RPC program

Here is an example of an input file for the RPC table:

```
#
# rpc file
#
rpcbind     100000     portmap     portmapper
rusersd     100002     rusers
nfs         100003     nfsprog
mountd      100005     mount       showmount
walld       100008     rwall       shutdown
sprayd      100012     spray
llockmgr    100020
nlockmgr    100021
status      100024
bootparam   100026
keyserv     100029     keyserver
nisd        100300     rpc.nisd
#
```

Services Table

The services table stores information about the Internet services available on the Internet. It has five columns:

Table 82. Services Table

Column	Description
Cname	The official Internet name of the service
Name	The list of alternate names by which the service can be requested
Proto	The protocol through which the service is provided (for instance, 512/tcp)
Port	The port number
Comment	Comments about the service

Timezone Table

The timezone table lists the default timezone of every workstation in the domain. The default time zone is used during installation but can be overridden by the installer. The table has three columns:

Table 83. Timezone Table

Field	Description
Name	The name of the domain
Tzone	The name of the time zone (for example, US/Pacific)
Comment	Comments about the time zone

Index

Vos remarques sur ce document / Technical publication remark form

Titre / Title : Bull AIX 4.3 Network Information Services (NIS and NIS+) Guide

Nº Reférence / Reference Nº : 86 A2 22KX 00

Daté / Dated : November 1999

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement.
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : ______________________ Date : __________

SOCIETE / COMPANY : ______________________

ADRESSE / ADDRESS : ______________________

____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

Technical Publications Ordering Form

Bon de Commande de Documents Techniques

To order additional publications, please fill up a copy of this form and send it via mail to:
Pour commander des documents techniques, remplissez une copie de ce formulaire et envoyez-la à :

BULL ELECTRONICS ANGERS	**Managers** / Gestionnaires :		
CEDOC	**Mrs.** / Mme :	**C. DUMOULIN**	+33 (0) 2 41 73 76 65
ATTN / MME DUMOULIN	**Mr.** / M :	**L. CHERUBIN**	+33 (0) 2 41 73 63 96
34 Rue du Nid de Pie – BP 428			
49004 ANGERS CEDEX 01	**FAX :**		+33 (0) 2 41 73 60 19
FRANCE	**E–Mail** / Courrier Electronique :		srv.Cedoc@franp.bull.fr

Or visit our web site at: / Ou visitez notre site web à:
`http://www-frec.bull.com` (Press Room, Technical Literature, Ordering Publications)

CEDOC Reference # Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	

[_ _] : **no revision number means latest revision** / pas de numéro de révision signifie révision la plus récente

NOM / NAME : ______________________________ Date : __________

SOCIETE / COMPANY : ______________________________

ADRESSE / ADDRESS : ______________________________

____

PHONE / TELEPHONE : ______________________ FAX : ______________________

E–MAIL : ______________________________

For Bull Subsidiaries / Pour les Filiales Bull :
Identification: ______________________________

For Bull Affiliated Customers / Pour les Clients Affiliés Bull :
Customer Code / Code Client : ______________________________

For Bull Internal Customers / Pour les Clients Internes Bull :
Budgetary Section / Section Budgétaire : ______________________________

For Others / Pour les Autres :
Please ask your Bull representative. / Merci de demander à votre contact Bull.

PLACE BAR CODE IN LOWER
LEFT CORNER

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 22KX 00

Bull

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

AIX

AIX 4.3 Network
Information
Services
(NIS and NIS+)
Guide

86 A2 22KX 00

AIX

AIX 4.3 Network
Information
Services
(NIS and NIS+)
Guide

86 A2 22KX 00

AIX

AIX 4.3 Network
Information
Services
(NIS and NIS+)
Guide

86 A2 22KX 00