System Management Guide

iPlanet™ Unified Development Server

Version 5.0

August 2001

Contents

List of Figures

List of Procedures

Preface

The *iPlanet UDS System Management Guide* provides information regarding the setting up and maintaining of iPlanet UDS environments as well as the deployment and management of iPlanet UDS distributed applications.

It also covers how you perform these tasks using iPlanet UDS system management tools.

This preface contains the following sections:

Product Name Change

Forte 4GL has been renamed the iPlanet Unified Development Server. You will see full references to this name, as well as the abbreviations iPlanet UDS and UDS.

Audience for This Guide

This manual is intended for system managers rather than application developers. We assume that you:

- are familiar with all the machine architectures and operating systems in your environment
- are familiar with iPlanet UDS concepts and terminology

If you plan to use the iPlanet UDS Escript utility or the iPlanet UDS system management agents, you should have a copy of the *Escript and System Agent Reference Guide* available.

Organization of This Guide

The following table briefly describes the contents of each chapter:

Chapter	Description
Chapter 1, "Introduction to iPlanet UDS System Management"	Explains the basic concepts of iPlanet UDS system management and provides an overview of its features.
Chapter 2, "The iPlanet UDS Environment Console"	Introduces the Environment Console graphical user interface and provides a reference to the Environment Console commands.
Chapter 3, "Setting up and Maintaining an iPlanet UDS Environment"	Describes how to set up your iPlanet UDS environment and how to start and stop iPlanet UDS system management services.
Chapter 4, "Creating and Modifying Environment Definitions"	Explains how to create and modify environment definitions and explains how to specify node properties.
Chapter 5, "Deploying iPlanet UDS Applications"	Describes iPlanet UDS application distributions and explains how to deploy them in a deployment environments.
Chapter 6, "Managing iPlanet UDS Applications"	Explains how to start up, monitor, and reconfigure iPlanet UDS applications.
Chapter 7, "Troubleshooting"	Covers a number of topics useful in uncovering and diagnosing problems in an iPlanet UDS environment or application.

Chapter	Description
Chapter 8, "Managing iPlanet UDS Development Repositories"	Describes how to create and manage central development repositories for iPlanet UDS application developers.
Chapter 9, "Launching iPlanet UDS Applications and Applets"	Explains how to use the Launch Server, Launcher application, and Ftcmd utility to manage the deployment and usage of client partitions.
Appendix A, "Special Setup for Development Environments"	Describes additional setup needed for development environments, including auto-compile services and special library support.
Appendix B, "iPlanet UDS Environment Variables"	Provides a complete listing of iPlanet UDS environment variables (logical names on OpenVMS platforms).
Appendix C, "iPlanet UDS Command Summary"	Provides a complete list, in alphabetical order, of all iPlanet UDS command-line utilities.

Text Conventions

This section provides information about the conventions used in this document.

Format	Description
italics	Italicized text is used to designate a document title, for emphasis, or for a word or phrase being introduced.
`monospace`	Monospace text represents example code, commands that you enter on the command line, directory, file, or path names, error message text, class names, method names (including all elements in the signature), package names, reserved words, and URLs.
ALL CAPS	Text in all capitals represents environment variables (FORTE_ROOT) or acronyms (UDS, JSP, iMQ). Uppercase text can also represent a constant. Type uppercase text exactly as shown.
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+A means press both keys simultaneously.
Key-Key	Consecutive keystrokes are joined with a hyphen: Esc-S means press the Esc key, release it, then press the S key.

Other Documentation Resources

In addition to this guide, there are additional documentation resources, which are listed in the following sections. The documentation for all iPlanet UDS products (including Express, WebEnterprise, and WebEnterprise Designer) can be found on the iPlanet UDS Documentation CD. Be sure to read "Viewing and Searching PDF Files" on page 26 to learn how to view and search the documentation on the iPlanet UDS Documentation CD.

iPlanet UDS documentation can also be found online at http://docs.iplanet.com/docs/manuals/uds.html.

The titles of the iPlanet UDS documentation are listed in the following sections.

iPlanet UDS Documentation

- *A Guide to the iPlanet UDS Workshops*
- *Accessing Databases*
- *Building International Applications*
- *Escript and System Agent Reference Guide*
- *Fscript Reference Guide*
- *Getting Started With iPlanet UDS*
- *Integrating with External Systems*
- *iPlanet UDS Java Interoperability Guide*
- *iPlanet UDS Programming Guide*
- *iPlanet UDS System Installation Guide*
- *iPlanet UDS System Management Guide*
- *Programming with System Agents*
- *TOOL Reference Guide*
- *Using iPlanet UDS for OS/390*

Express Documentation

- *A Guide to Express*
- *Customizing Express Applications*
- *Express Installation Guide*

WebEnterprise and WebEnterprise Designer Documentation

- *A Guide to WebEnterprise*
- *Customizing WebEnterprise Designer Applications*
- *Getting Started with WebEnterprise Designer*
- *WebEnterprise Installation Guide*

Online Help

When you are using an iPlanet UDS development application, press the F1 key or use the Help menu to display online help. The help files are also available at the following location in your iPlanet UDS distribution: `FORTE_ROOT/userapp/forte/cln/*.hlp`.

When you are using a script utility, such as Fscript or Escript, type help from the script shell for a description of all commands, or help <*command*> for help on a specific command.

iPlanet UDS Example Programs

A set of example programs is shipped with the iPlanet UDS product. The examples are located in subdirectories under `$FORTE_ROOT/install/examples`. The files containing the examples have a `.pex` suffix. You can search for TOOL commands or anything of special interest with operating system commands. The `.pex` files are text files, so it is safe to edit them, though you should only change private copies of the files.

Viewing and Searching PDF Files

You can view and search iPlanet UDS documentation PDF files directly from the documentation CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

NOTE You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from http://www.adobe.com. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

➤ **To copy the documentation to a client or server**

1. Copy the `doc` directory and its contents from the CD-ROM to the client or server hard disk.

   You can specify any convenient location for the `doc` directory; the location is not dependent on the iPlanet UDS distribution.

2. Set up a directory structure that keeps the `udsdoc.pdf` and the `uds` directory in the same relative location.

   The directory structure must be preserved to use the Acrobat search feature.

NOTE To uninstall the documentation, delete the `doc` directory.

➤ **To view and search the documentation**

1. Open the file `udsdoc.pdf`, located in the `doc` directory.
2. Click the Search button at the bottom of the page or select Edit > Search > Query.

3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click Search.

   A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

   NOTE For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

   All occurrences of the word or phrase on a page are highlighted.

5. Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]
Previous Highlight	Ctrl+[
Next Document	Ctrl+Shift+]

   To return to the `udsdoc.pdf` file, click the Homepage bookmark at the top of the bookmarks list.

6. To revisit the query results, click the Results button at the bottom of the `udsdoc.pdf` home page or select Edit > Search > Results.

Chapter 1

Introduction to iPlanet UDS System Management

This chapter provides the background you need to understand and manage an iPlanet UDS system. Topics covered in this chapter include the following:

- system overview
- iPlanet UDS runtime system and distributed applications
- iPlanet UDS system management architecture and services
- iPlanet UDS system management tasks
- iPlanet UDS system management tools

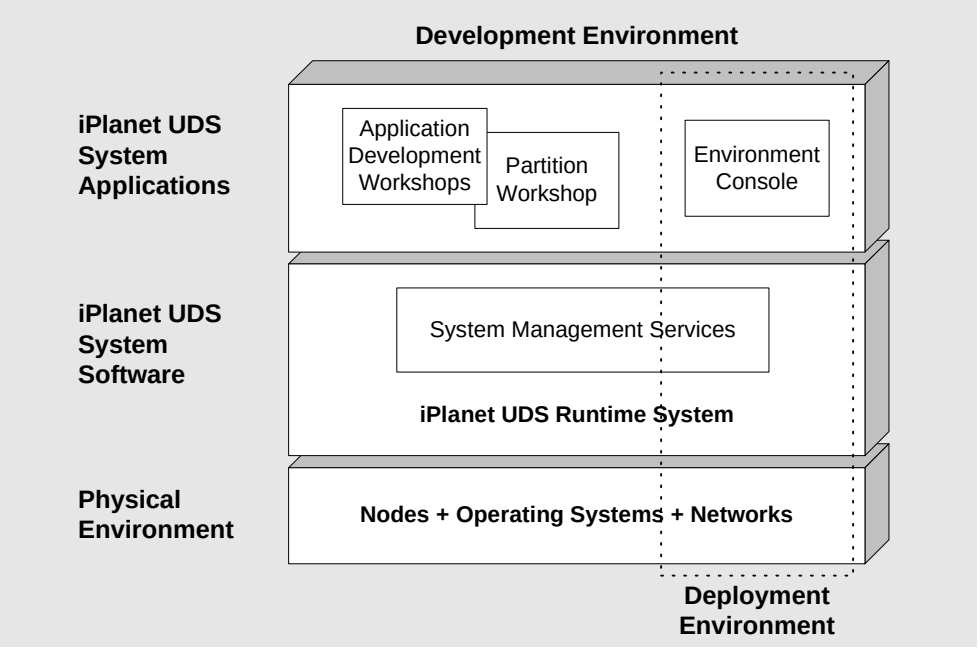
System Overview

iPlanet UDS is a software environment for developing, deploying, and managing distributed client/server applications. It supports the full life cycle of an iPlanet UDS application, from development to the management of the application as it runs on a number of platforms.

To support this breadth of function, an iPlanet UDS system includes software that operates at a number of different levels, as illustrated in Figure 1-1. The different levels of iPlanet UDS software are as follows:

iPlanet UDS Runtime System The runtime system is a common set of services that allows an iPlanet UDS application, or a part of such an application (an application partition), to run on a number of platforms. This set of services is implemented on each platform supported by iPlanet UDS. The runtime system supports access to the local operating system, as well as communication and synchronization between remote partitions.

Figure 1-1 iPlanet UDS Applications and Runtime System



iPlanet UDS System Management Services iPlanet UDS provides a number of system management services that deploy, start, and manage distributed applications. The system management architecture is discussed in detail in "iPlanet UDS System Management Services and Architecture" on page 44. Understanding these system management services is critical to performing system management tasks.

iPlanet UDS Development and Management Applications iPlanet UDS provides a number of applications, most with graphical user interfaces, that support various stages of the iPlanet UDS application life cycle. These applications include the following:

- application development workshops, which are used by developers to create logical applications. These workshops include the Partition Workshop, which is used by developers for to produce installable applications. You can also use the `Fscript` utility to create and produce applications.
- Environment Console or `Escript` utility, which are used by system managers to manage iPlanet UDS environments and applications

The iPlanet UDS development and management applications, like iPlanet UDS user applications, are true iPlanet UDS applications—developed using iPlanet UDS development tools—and can only run in an iPlanet UDS runtime environment.

iPlanet UDS Environments

An iPlanet UDS environment consists of a collection of networked nodes running the iPlanet UDS runtime system. There are two types of environments: deployment environments and development environments. Both deployment and development environments use the same runtime system and system management services, as shown in Figure 1-1 on page 30.

Deployment environments A deployment environment is one in which you can deploy, run, and manage an iPlanet UDS application. A deployment environment has the iPlanet UDS runtime system and system management services installed and running, and includes the Environment Console application (and its command line equivalent, the `Escript` utility) used by system managers to perform management tasks.

Development environments A development environment is one in which you can develop and test an iPlanet UDS application. A development environment has all the software components of a deployment environment, plus the iPlanet UDS applications needed to develop an iPlanet UDS application—the iPlanet UDS Workshops and their command line equivalent, the `Fscript` utility.

The deployment and development environments discussed above both require the installation of the iPlanet UDS runtime system (with the application development software, if necessary) in a physical environment. The physical environment generally consists of a number of computers running different operating systems and linked together by one or more networking systems.

Environment definitions Any iPlanet UDS environment has an environment definition. An environment definition specifies (among other things) all the nodes in a given iPlanet UDS environment. You can create or modify an environment definition using the Environment Console or `Escript` utility. Deployment environments typically only have the definition for the active environment, which is the actual physical environment. Development environments often have other environment definitions as well, which are used to let the application developer simulate other environments and generate applications for these other environments.

Nodes Each computer in this physical environment is a node in the active environment. To function as part of an iPlanet UDS environment, however, two conditions must be met. First, iPlanet UDS software must be installed on the node, and second, any special information about the node must added to the environment definition for the active environment.

Among the properties specified for a node in an environment definition are:

- node name
- operating system
- installed network protocols
- available database resource managers

In addition, there are a number of properties that depend on how you intend to use the node in an iPlanet UDS environment.

Clients and servers For example, depending on its role in an iPlanet UDS environment, a node might be a server node, a client node, or both. Generally, server nodes provide application services and client nodes use those services. A client node runs on a windowing system—Windows 95 or Motif. A server node must run on a multi-tasking operating system, such as Windows NT, UNIX, or VMS.

In an environment that has many client nodes with identical properties, you can define one node as a model node with those properties. This model node can represent any number of client nodes, so you don't have to define them each individually.

An environment definition with its full node specifications is used by iPlanet UDS in partitioning a logical application, as discussed in "iPlanet UDS Runtime System and Distributed Applications."

iPlanet UDS Runtime System and Distributed Applications

This section discusses the iPlanet UDS runtime system in the context of iPlanet UDS distributed applications. It covers aspects of an iPlanet UDS application you must understand to perform iPlanet UDS system management tasks.

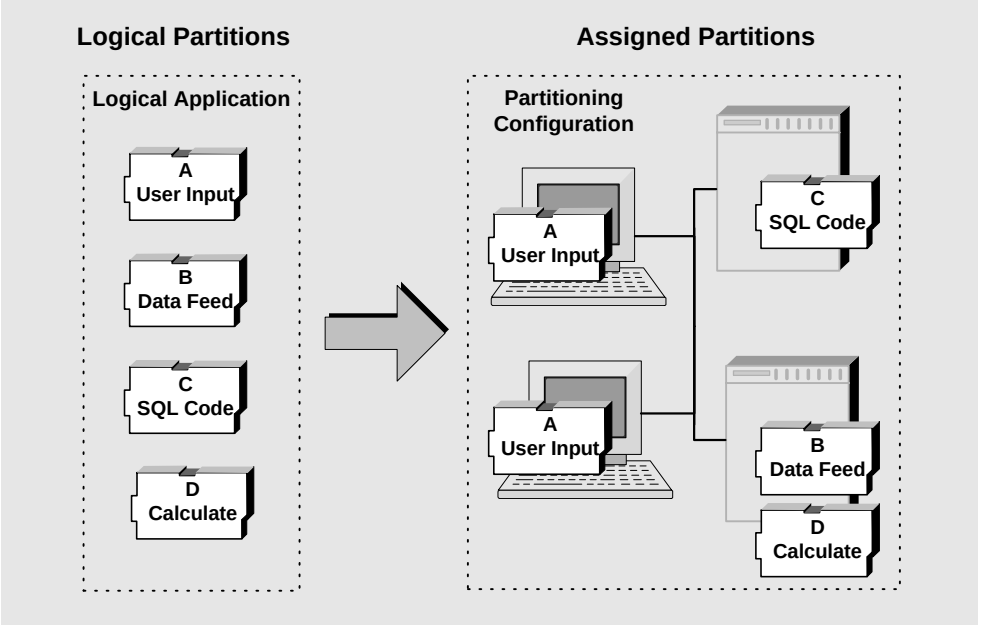
Overview

Partitioning One of the most powerful features of iPlanet UDS is its partitioning capability. Partitioning allows developers to create a distributed application without worrying about the details of its deployment environment.

To create an iPlanet UDS distributed application, developers create a logical application, using the iPlanet UDS Workshops to define classes, create user windows, and write and debug code. After creating this logical application, developers partition the logical application for one or more specific deployment environments.

In the partitioning process, the logical application is considered in the context of a specific deployment environment. When developers partition the application, it is divided into separate logical sections called *logical partitions*. iPlanet UDS creates a default partitioning *configuration*, in which the logical partitions are assigned to various nodes in the target environment, based on the properties of each node. Each such assignment is called an *assigned partition*.

Figure 1-2 Partitioning a Logical Application

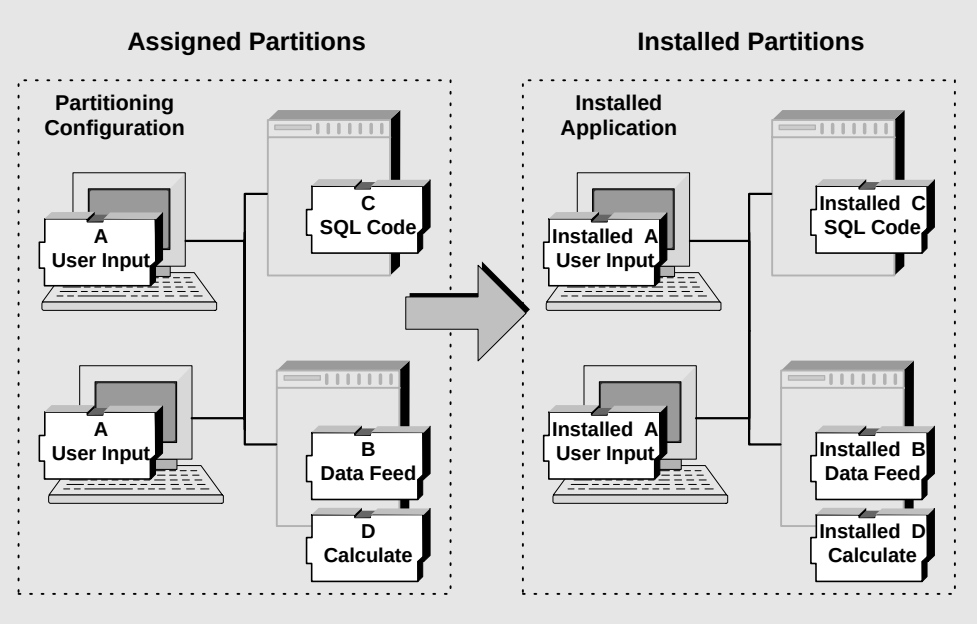


Developers can change the default partitioning configuration by moving assigned partitions to different nodes or by altering the logical partitioning. For more information on partitioning, see *A Guide to the iPlanet UDS Workshops*.

After adjusting the partitioning scheme, developers can run the application configuration in a test mode. In this test mode, iPlanet UDS actually deploys the application into the development environment and runs it as a distributed application.

Application distribution After testing, a developer makes an application distribution. An application distribution consists of all the files needed to deploy the application into a deployment environment. After a developer makes this distribution, you can transfer it to the deployment environment and install it. During installation, code for each partition is installed in a location on its assigned node that is defined by iPlanet UDS. At this point, the assigned partitions become *installed partitions*.

Figure 1-3 Deploying an Application

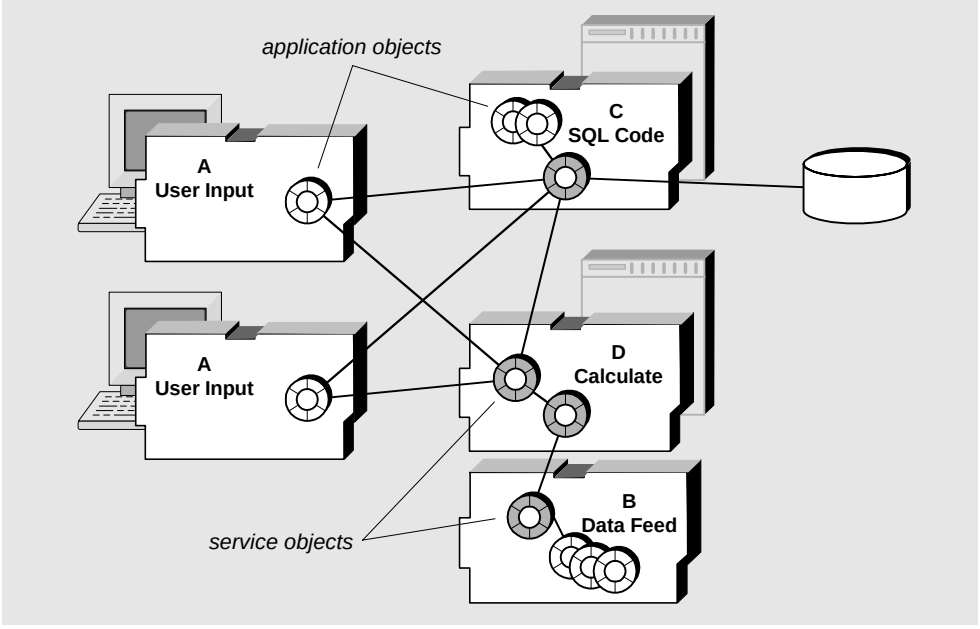


A typical iPlanet UDS application includes a client partition and a set of server partitions. The client and server partitions execute those portions of the application for which each is best suited. The client partition typically performs a graphical user interface function, while the server partitions provide access to databases,

communication channels, data feeds, imaging systems, and other shared application services. An iPlanet UDS application can also be a server-only application, which provide services using one or more server partitions but includes no client partition.

The iPlanet UDS runtime system considers an application to be a collection of objects, each of which represent an element of the application, as illustrated in Figure 1-4. Such objects might include display objects, database objects, and other objects defined by application developers. A running iPlanet UDS application, in essence, is a collection of distributed objects interacting to perform the functions required of the application.

Figure 1-4 Object View of a Distributed Application



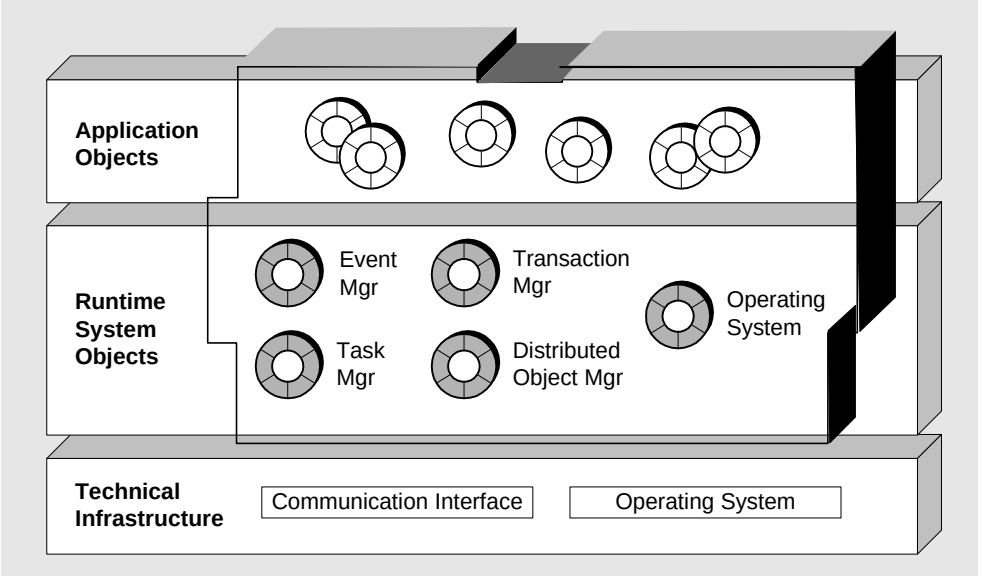
As illustrated in Figure 1-4, each object lives within an active, running partition. An *active partition* is created when you start an installed partition. Each active partition corresponds to a running server process.

Service objects One type of object—the *service object*—is particularly important in iPlanet UDS applications. A service object is a named object that can be shared by a number of users and applications. A service object provides a particular service for the application. Each server partition contains at least one such service object; in fact, you can think of a server partition as the set of objects that support the work of one or more service objects.

iPlanet UDS also creates a number of runtime system objects. Runtime system objects support an application by providing the functions necessary to run each partition on its respective platform and to make the collection of distributed objects function as one application. iPlanet UDS runtime system objects transparently transmit messages and responses between remotely located objects, coordinate the sharing of remote objects, and manage transactions spanning remote application objects.

These runtime system objects support the application objects within each partition and interact with the operating system of each node, as illustrated in Figure 1-5.

Figure 1-5 Application Partition and iPlanet UDS Runtime System Objects



Each running application partition uses the following iPlanet UDS runtime system objects to interact with the operating system:

Distributed Object Manager Manages the distributed object services for an active (running) partition. The distributed object service provides access to logical communications into and out of a partition.

Event Manager Manages receiving and delivering events for the active (running) partition.

Operating System Manages the local operating system services for an active (running) partition. The operating system service provides memory management and other utility functions.

Task Manager Monitors the creation and destruction of tasks, both for the iPlanet UDS executor and for multithreaded servers. It also manages the shared locks and mutexes for an active partition.

Transaction Manager Monitors the state of transactions across partitions.

These iPlanet UDS runtime system objects are represented by system agents, which you can access through the Environment Console and `Escript`, as explained in Chapter 2, "The iPlanet UDS Environment Console" and *Escript and System Agent Reference Guide*. You can also access these agents programmatically, as explained in *Programming with System Agents*.

For more information about the agents themselves, see *Escript and System Agent Reference Guide*

In general, when you manage an iPlanet UDS application, you are most concerned about nodes, installed partitions, and active partitions. For the most part, the interaction of application objects and runtime system objects is invisible to you. However, iPlanet UDS's system management architecture lets you access data maintained by the application and runtime objects, when you need it to monitor performance or troubleshoot.

A Scenario

To better understand what the iPlanet UDS system management processes do and how they work together, consider a typical iPlanet UDS system management operation: autostarting a deployed application.

Starting a distributed application involves starting all the server partitions needed by the client partition. One way to do this is to start a client and let the iPlanet UDS runtime system do the rest.

➤ **Autostarting an application consists of the following basic operations**

1. Starting a client partition.

   The executing client code invokes a method on a remote service object. The Local Object Manager asks the Distributed Object Manager, the iPlanet UDS runtime system object responsible for managing remote objects, to resolve the service object reference.

2. The Distributed Object Manager in the client partition queries the Name Service for the address of the partition containing the service object.

   Because the server partition has not yet been started, the Name Service database does not contain the address of that service object. The Name Service informs the distributed object manager that the address of the service object is not known

3. The Distributed Object Manager requests that the Environment Manager service (Environment agent) start an instance of the partition in which the service object resides.

   The Environment Manager consults the environment repository for the name of a node on which the required installed partition resides. The Environment Manager (Environment agent) then requests that the appropriate Node Manager (Node subagent) start the required installed partition.

4. The Node Manager starts the required installed partition by requesting that the corresponding Installed Partition agent start the partition.

   When the active partition comes up, the service object registers with the Name Service, so its address is now known.

5. The Node Manager notifies the Distributed Object Manager in the client partition that the server partition is now running and the Distributed Object Manager can now resolve the service object reference through the Name Service, as it had originally attempted to do in step 1.

Partitions

As mentioned in "Overview" on page 33, partitions are the basic executable components that make up an iPlanet UDS distributed application. Each partition, when running, corresponds to a single process executing on a node. As a system manager, you manage applications by:

- deploying partitions
- starting and stopping installed partitions
- monitoring active (executing) partitions

This section explains a number of additional concepts regarding partitions: execution type, replication, and reference partitions.

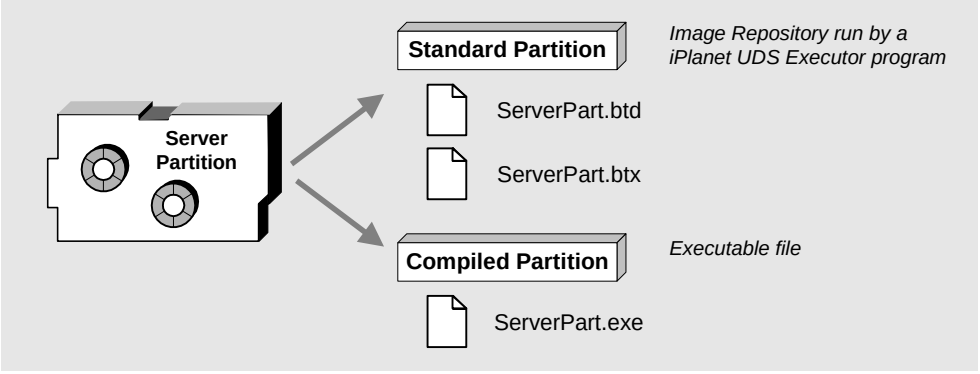
Execution Type

There are two types of application partitions—standard and compiled.

Standard partition Standard partitions consist of an *image repository*, which contains interpretive code that must be run by the iPlanet UDS executor program. A standard partition can run on any iPlanet UDS platform.

Compiled partition Compiled partitions are executable code that has been compiled and linked for a particular server platform. A separate compiled partition executable needs to be created for each target platform on which the partition needs to run.

Figure 1-6 Partition Execution Types



A developer is responsible for indicating which partitions in an application will be compiled partitions. The developer is also generally responsible for compiling these partitions and including them in an application distribution.

Replication

iPlanet UDS lets you replicate service objects and their respective server partitions to enhance application reliability using failover and improve application performance using load balancing.

For example, you can implement failover by providing backup replicates of a server partition that takes over processing when the primary server partition fails. Similarly, you can implement load balancing by distributing demand for a service among a number of replicates of the partition that provides that service. You can use replicates of a server partition for both failover and load balancing at the same time.

Managing replicated server partitions is one of the more demanding challenges that you face as a system manager. You need to provide the proper level of load balancing and failover protection for a given application in a given environment with a given usage pattern. This section provides a conceptual overview of this management task.

With some restrictions, a service object can be designated as:

- non-replicated
- replicated for failover
- replicated for load balancing
- replicated for both failover and load balancing

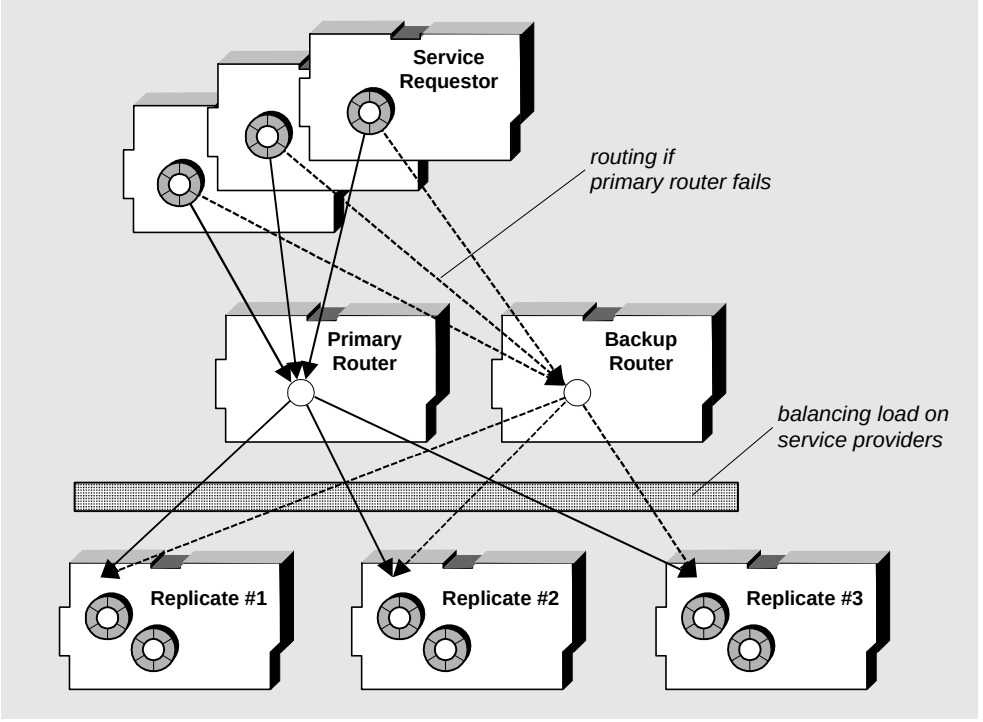
Replication for failover If a service object is marked for failover, then you can replicate the server partition and install it on a number of nodes in an environment. If the primary server partition fails, then iPlanet UDS directs any requests for that service to one of the other running replicates of the server partition.

Replication for load balancing If a service object is marked for load balancing, you can replicate the server partition and install it on a number of nodes in an environment. In addition, iPlanet UDS automatically creates a *router partition* for that service object. The purpose of the router partition is to route requests for a service among the running replicates of the server partition that performs that service.

Replication for failover and load balancing If a service object is marked for both failover and load balancing, then you can replicate the server partition as in the load balancing case. However the router partition that iPlanet UDS automatically creates for the service object can also be replicated. You can replicate the router partition and install it on a number of nodes in an environment in the same way you can replicate the server partition. iPlanet UDS designates one of these router partitions as the primary router. This router routes requests among the running replicates of a server partition. If the primary router fails, another running replicate of the router partition steps in to handle the routing. The scheme is illustrated in Figure 1-7 on page 41.

When you install any replicated partition on a node in an environment, be it a server partition or a router partition, you can specify how many replicates of that partition are "enabled," or started up on that node, when the application starts. Installed partitions that are designated as "disabled" can be started manually to provide special backup under unusual conditions.

Figure 1-7 Replicated Partitions: Load Balancing and Failover



The use of replicated partitions for failover and load balancing will be discussed in more detail in Chapter 6, "Managing iPlanet UDS Applications."

Reference Partitions

It is not uncommon for one application to use a service that is provided in another application. This is facilitated through the use of an iPlanet UDS *reference partition*—a partition within an application that points to a partition within another application. In this way, a service object can be shared between two applications.

When using reference partitions, however, it is important to make sure that reference partitions are already running when you start an application that uses them. This means you must start the applications to which the reference partitions point, or start the individual partitions being referenced before starting the referencing application. (One minor exception to this rule is when your referencing application is started using auto-start. For more information see Chapter 6, "Managing iPlanet UDS Applications.")

Libraries

Most iPlanet UDS application partitions require access to some number of shared executable modules to execute properly. Depending on the platform, these modules are referred to as object libraries or archives, shared images, shared libraries, or dynamic linked libraries (DLLs). In this guide, these shared modules will all be referred to as *libraries*.

As a system manager, you might need to package library distributions, often along with the application distributions that require them. You might also need to deploy library distributions in both deployment and development environments.

Libraries, which can be compiled and linked for each platform (though also available in standard format), are loaded into memory by a node's operating system and used by any number of iPlanet UDS partitions executing on the node. An application requires a particular library if the application's developer has referenced the library code in creating the application.

Libraries, like partitions, must be installed in a standard location on each node, so they can be loaded and used at runtime. As a system manager, you are responsible for deploying library distributions into your iPlanet UDS environment.

iPlanet UDS supports two general types of libraries: system libraries and user libraries.

System Libraries

System libraries are libraries supplied with the iPlanet UDS product, which are automatically installed on each node by the iPlanet UDS installation program. In a development environment, the object classes represented by each library are automatically included in every development repository (for example, the Framework, Display, and GenericDBMS libraries). A number of additional system libraries, used to integrate iPlanet UDS applications with external applications and services, are also installed with the iPlanet UDS product.

As a system manager, you generally don't have to concern yourself with system libraries; however there are a few setup considerations for development environments using the OLE system libraries (see "Support For OLE" on page 325).

User Libraries

Library distribution Unlike system libraries, user libraries are created by developers. When a developer has written code suitable for broader distribution, the projects (one or more) comprising that code can be made into a library distribution—much like an application distribution—that can be deployed and used in other iPlanet UDS environments. A library distribution contains one library for each included project. Each library is compiled and linked for each supported platform.

The libraries included within a library distribution can be of several kinds: the most common are TOOL libraries and C libraries.

TOOL libraries These are created by developers from TOOL (iPlanet UDS's Transactional Object-Oriented Language) projects. They represent TOOL code that might be of more general use to other application developers. For more information on TOOL libraries, see *A Guide to the iPlanet UDS Workshops*.

C libraries These are very similar to TOOL libraries, except they are made from iPlanet UDS C projects rather than TOOL projects. C projects are used by a developer to encapsulate external C routines. These libraries are more likely to be restricted to certain platforms or nodes with certain resources than TOOL libraries, which are more likely to be non-restricted. For more information on C libraries, see *Integrating with External Systems*.

When developers want to use libraries that are part of a library distribution, the library distribution must be deployed in their development environment. Each library within the distribution is installed in a standard location on its assigned node. The library distribution includes a file that can be imported into development repositories so that developers can use the library without having the source code in their repositories.

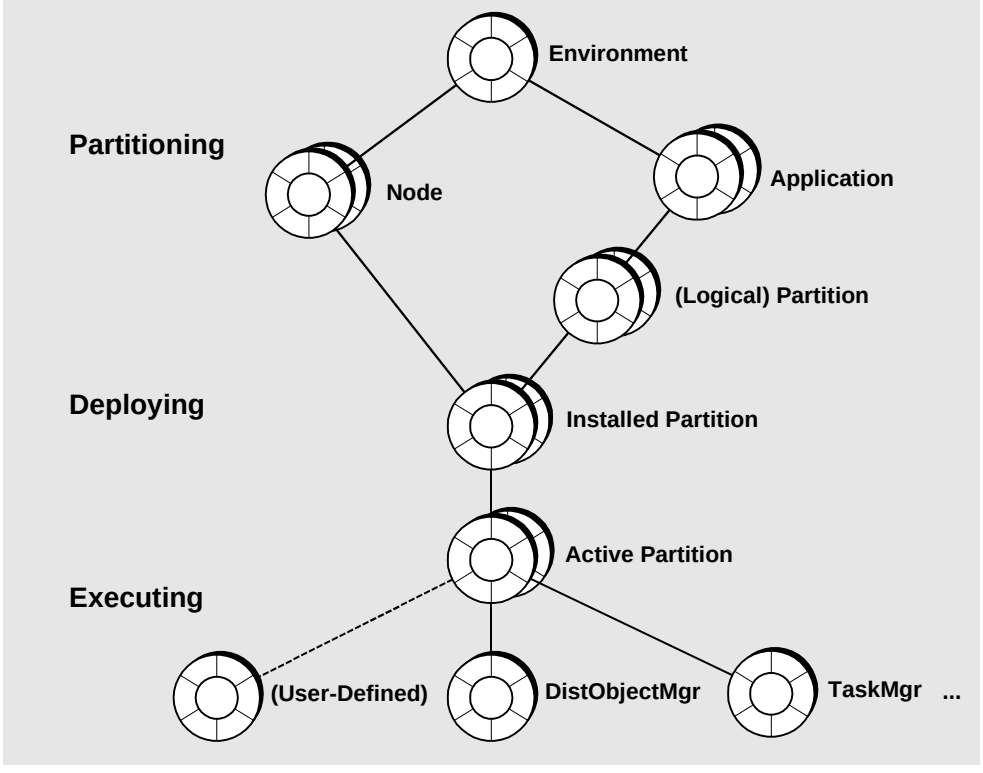
If an application references code in a library, then that library must be installed, along with the application, in any deployment environment where the application will be run.

iPlanet UDS System Management Services and Architecture

At the most general level, system management consists of monitoring and controlling a number of entities within your iPlanet UDS environment. For example, you might want to install a partition on a particular node, start a particular installed partition, or write performance data for a particular active partition to a log file.

In an object-oriented system, such as iPlanet UDS, each entity within a system is represented by an object. The objects of most interest for system management are the objects involved in partitioning, deploying, and executing a distributed application. These objects are shown schematically in Figure 1-8.

Figure 1-8 Some Key Objects in the System Management Domain



The relationships shown in the illustration are functional (they do *not* represent inheritance relationships between iPlanet UDS classes). During partitioning, for example, the developer divides a logical application into a set of logical partitions and assigns these partitions to the nodes specified for a given environment. Application, logical partition, node, and environment objects are all important system management entities.

During deployment, the partitioning configuration—the logical assignment of partitions to a set of nodes—becomes the actual installed configuration. When you install the partitions on their respective nodes in the target environment, these partitions become installed partition objects.

During execution, you or the application start up installed partitions, which creates active partition objects. Starting an installed partition also creates a number of runtime system objects, such as the Distributed Object Manager and Task Manager. These runtime objects can provide important performance monitoring information.

In addition to the system management objects discussed above, a distributed application can have other objects, such as service objects, that you can monitor or control as part of your management of the distributed application. iPlanet UDS system management architecture provides the flexibility for you to incorporate such user-defined objects into your system management domain.

iPlanet UDS's system management architecture is designed to provide you with centralized management of a decentralized system. You can manage an entire iPlanet UDS environment, including all its distributed applications, from a single node in the environment.

For example, you can perform the following basic management tasks:

- install the various components of an application or library on the various nodes in your environment
- start installed partitions on various nodes and shut them down
- monitor the performance or status of an active partition, a node and its components, or an active environment and its components

Furthermore, the system management architecture gives you the flexibility to customize and extend iPlanet UDS's system management capabilities to meet your individual system management needs.

System Management Agents

iPlanet UDS's provides a set of system management *agents* to monitor and control a corresponding set of managed objects. For each object to be managed, there is a system management agent to monitor and control that object. For example, as iPlanet UDS creates each installed partition object, it also creates a corresponding Installed Partition agent. The managed object performs its normal functions without having any knowledge of its agent. iPlanet UDS also deletes these agents when the managed objects are deleted.

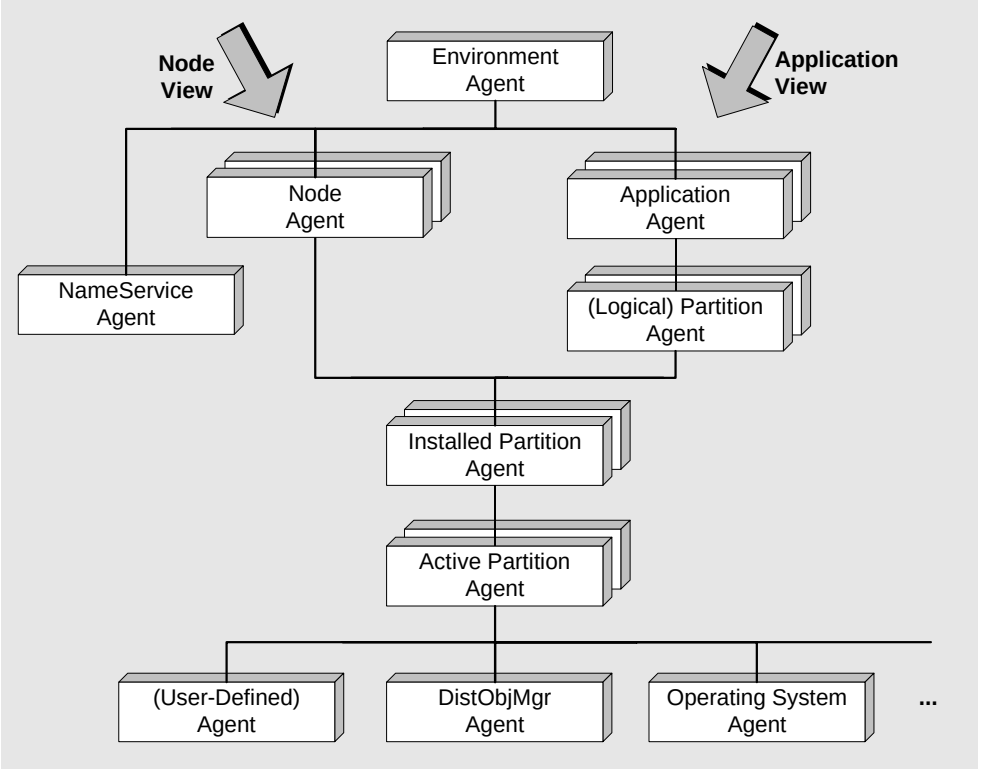
Agent commands Each agent has a set of commands or operations it can perform on its managed object. For example, an Installed Partition agent can start an installed partition. Likewise, an Active Partition agent can shut down an active partition.

Agent instruments Each agent also has a set of instruments, each representing a type of data that can be obtained from or set on the managed object. An instrument can represent a property or attribute of the managed object or it can represent more dynamic information. For example, an Active Partition agent has instruments representing that partition's process ID and the name of the log file to which it logs information. A Distributed Object Manager agent has an instrument representing the number of messages sent by the distributed object manager to remote partitions during a specified period of time.

The instruments defined for each agent generally represent data that is useful for monitoring or control purposes.

Agent hierarchy iPlanet UDS's system management agents are organized in a hierarchical structure of parent agents and subagents, as depicted in Figure 1-9.

Figure 1-9 System Management Agent Hierarchy



In this hierarchical structure, for example, Partition agents are subagents of Application agents, and Installed Partition agents are subagents of both Partition agents and Node agents.

In this hierarchy of agents, commands on the parent agents are passed down through the structure, from parent agents to the subagents that actually perform the task. For example, if you start up an application, the Application agent passes the command down to the Partition subagents, which, in turn, passes the command down to the Installed Partition subagents, which actually start the installed partitions composing the application.

System management tools can easily navigate from any agent to its parent agent or one of its subagents. As you can see from Figure 1-9, there are two pathways, or views, from top to bottom: one through nodes (each of which might be running many applications' partitions) and one through applications (each of which might have partitions running on many nodes).

You that can be readily customize and extend this agent hierarchy. You can define external agents that can communicate with each other (using a well-defined protocol) independent of the objects being managed.

You can add new commands and new instruments to existing agents, or create new agents, without any modification to the objects being managed. You can create your own agents and your own system management applications, providing the monitoring and control you want, for whatever objects you want, within any iPlanet UDS distributed application.

Escript and System Agent Reference Guide explains the iPlanet UDS system management agents and explains how to use them. For information about programming your own agents and your own system management applications, see the *Programming with System Agents*.

The following table summarizes the roles performed by the principal system management agents. For a complete description of each agent's commands and instruments, see *Escript and System Agent Reference Guide*.

Agent	Manages
Active Partition	Running partition of an application.
Ad hoc partition	Acts as the installed partition agent for a running partition that has not been installed using the iPlanet UDS installation process.
Application	Application that has been loaded into the environment.
BtreeCache	Cache used with the B-tree central repository to improve performance.
BtreeRepository	B-tree central repository.
CommMgr	Communications into and out of an active partition.
DBSession	Database session for an active partition that accesses a database using a DBSession object.
DistObjectMgr	Distributed object services for an active partition.
Environment	iPlanet UDS environment (represented by the Environment Manager).
EventMgr	Services for receiving and delivering events.
Installed Partition	Partition of an application that has been installed on a particular machine.
LoadBalancing Router	Router for a load-balanced service object.
Machine	Physical machine on which a Node Manager is running.

Agent	Manages
Model Node	Definition for a model node.
NameService	iPlanet UDS Name Service, which manages how objects, services, partitions, and so forth are known within and across environments.
NativeLangMgr	Services for multinational and multilingual functions.
Node	Node in the environment.
ObjectCache	Object cache used by a client repository session.
OperatingSystem	Operating system services for an active partition, including memory management and other utility functions.
Partition	Logical partition of an application. This partition is not assigned to any particular node.
Process	Operating system process on which an active partition runs.
Repository	Repository managed by the repository server or another iPlanet UDS application.
RepositoryServer	Running repository server, which manages a central repository.
RepositoryServerInfo	Provides information about a repository server running in the current environment.
RepositorySession	Client repository session running on a node that is running a Node Manager.
TaskMgr	Task management services for an active partition.
TransactionMgr	Transaction management services, which monitors the state of transactions across partitions.
Volume	Storage device on a particular machine.

System Management Services

iPlanet UDS has system management services that let you work with the system management agents discussed in the previous section:

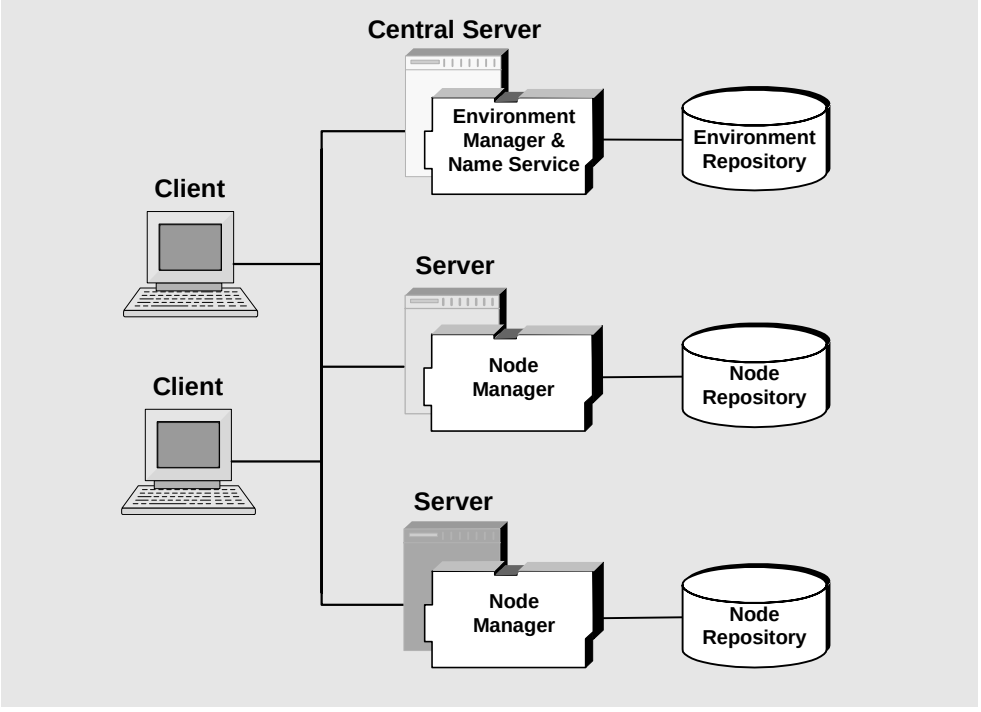
Environment Manager runs only on one server node in the environment (the *central server*) and provides system management functions on an environment-wide basis

Name Service provides address information for iPlanet UDS distributed applications and runs in the same partition as the Environment Manager service

Node Manager runs on each server node (and in certain cases on client nodes) and provides system management functions on a node-wide basis

Each of these services has an associated repository or database for storing information vital to its function. The scheme is illustrated in Figure 1-10.

Figure 1-10 iPlanet UDS System Management Services



Environment Manager

Each active iPlanet UDS environment must have an active Environment Manager service. The Environment agent represents the Environment Manager. The Environment Manager has access to a complete picture of the entire environment and is connected through a Node Manager to every active node. Because the Environment agent is the parent agent of all the other agents, the Environment agent also has some control over all nodes and applications in the environment.

The Environment Manager performs the following functions:

- lets you deploy an application or library distribution, and start or shut down installed partitions throughout the environment
- maintains environment-wide status information
- receives notification of all system management and instrument logging events throughout the environment
- provides Node Manager (Node agent) service for the central server node

Environment repository The Environment Manager maintains an environment repository, which contains configuration information for the environment as a whole, including:

- environment definitions, which contain node specifications for all nodes in the environment
- the application partitioning configuration for all application distributions that are loaded or installed in the environment. Loading applications distributions makes them known to the Environment Manager.
- the library configuration for all library distributions that are loaded or installed in the environment

The environment repository contains the combined information from all the individual node repositories in the environment. As individual node managers perform work, such as installing partitions and starting them up, the node managers maintain information on the status of these partitions in both the individual node repositories and the environment repository. Therefore, you should back up the environment repository regularly.

Simulated environments In a development environment, the environment repository can also contain environment definitions for simulated deployment environments. You can use simulated environment definitions to model a deployment environment for partitioning and testing an application.

Name Service

Each environment requires a Name Service to manage, all communications for distributed applications in an iPlanet UDS environment. A Name Service manages communications for the applications provided by iPlanet UDS as well as for all iPlanet UDS user applications.

Whenever a server partition starts, it registers its service objects with the Name Service. Then, any application process that needs to access a service object on this partition consults the Name Service for the network address of that service object.

The Name Service, which runs in the Environment Manager partition, has to be the first system management service running in an iPlanet UDS environment. (iPlanet UDS processes cannot start if the Name Service is offline.)

When you install iPlanet UDS on any node, you provide the Name Service address. This allows any iPlanet UDS process on that node to communicate with the Name Service.

Name Service database The Name Service maintains a database of the names and network addresses of the service objects available in the current environment *name space*, which contains the names of objects that are accessible from the current environment. This database is stored in the environment repository. When partitions shut down, the Name Service deletes the corresponding entries from the database.

Connected environments You can connect several different iPlanet UDS environments so their name spaces are merged, providing a single name space shared by all of the connected environments. This allows partitions in one environment to access services available in another. This is most useful in failover scenarios where a backup service is in another environment, or in scenarios where a reference partition is pointing to a service in another environment. For more information on connecting environments, see "Connecting Environments" on page 115.

Node Manager

Each node that participates in an iPlanet UDS environment must have an active Node Manager service. The Node agent for a particular machine represents the Node Manager running on that machine. Working through a number of subagents, the Node agent controls all partitions on a given server node.

The Node Manager:

- enables remote management of a node

  A server node runs a Node Manager service to register for remote management by the Environment Console and the `Escript` utility, which are the iPlanet UDS system management applications.

- lets you retrieve iPlanet UDS distributions from the node and install application partitions and libraries on the node

- starts and shuts down execution of partitions on a node

  The Node Manager makes it possible for partitions installed on the Node to be started up as needed by the iPlanet UDS runtime system when an application has started.

Client nodes also provide Node Manager services when they run iPlanet UDS system management applications like the Environment Console or when they run the iPlanet UDS Launch Server.

In general, you do not need the Node Manager when a distributed application is in a steady state, when all partitions are up and running. However, you do need the Node Manager service when you install applications, start up applications, and manage active partitions.

Node repository Every Node Manager service maintains its own node repository to store all relevant information about the node. This information includes:

- the node's properties, including external resource managers
- the node's installed partitions

iPlanet UDS System Management Tasks

iPlanet UDS system management tasks can be grouped into two broad categories:

- setting up and maintaining iPlanet UDS environments (deployment and development)
- deploying and managing distributed applications and libraries

In both development and deployment environments, you need to get iPlanet UDS system software up and running—both the iPlanet UDS runtime system and iPlanet UDS system management services.

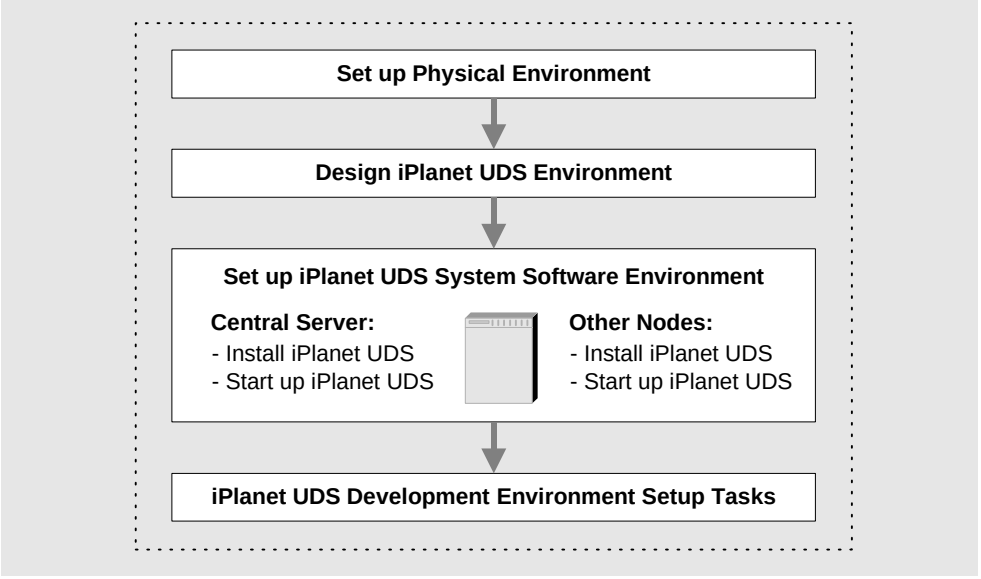
In a deployment environment, you also need to deploy distributed applications and libraries, and manage distributed applications.

In a development environment you are responsible for starting and maintaining the central repositories used for team development, including the possible installation of library distributions. You also have to deploy two server applications used to automate the making of application or library distributions containing compiled components.

Setting Up and Maintaining iPlanet UDS Environments

Setting up a new environment—whether development or deployment—involves different stages, as illustrated in Figure 1-11. These stages are introduced briefly in this section.

Figure 1-11 Typical Environment Setup Tasks



Setting up and Maintaining a Physical Environment

The iPlanet UDS runtime system is designed to be environment independent. Nevertheless, an iPlanet UDS environment has critical dependencies on operating systems, window systems, networking systems, runtime libraries, and database management systems. One of the most important aspects of properly setting up an iPlanet UDS environment is to make sure that the physical environment (hardware and software) meets the requirements for the iPlanet UDS runtime system.

For more information on these requirements, see the *iPlanet UDS System Installation Guide* and *Release Notes*.

Maintaining the physical environment normally involves adding or removing nodes and making changes in operating system versions and networking software. It can also involve diagnosing and resolving performance problems caused by network bottlenecks or hardware failures.

Designing an iPlanet UDS Environment

While the iPlanet UDS runtime system and any iPlanet UDS development and management applications are usually installed on an existing network configuration, you should consider how to best use the available resources based on the type of environment you are setting up.

If an environment is to be a development environment, then the locations of your central development repositories (Repository Servers) and Environment Manager service are key considerations.

If the environment is to be a deployment environment, then the availability and locations of vital application resources, such as database management systems and C program libraries, and the speed and reliability of servers are some key considerations. These considerations might also influence where you place the Environment Manager service.

For more information on designing an iPlanet UDS environment, see Chapter 3, "Setting up and Maintaining an iPlanet UDS Environment."

Setting up and Maintaining an iPlanet UDS Environment

In general, to set up an iPlanet UDS environment, you first install iPlanet UDS on a node that will play the role of a central server. Typically, this node hosts the Environment Manager service and often serves as a central distribution node for installing iPlanet UDS on other nodes in your environment. After you install iPlanet UDS on your central server, you install iPlanet UDS on the other nodes in the environment.

The installation program starts the required system management services on each node, and iPlanet UDS creates a default environment definition, which specifies all the nodes in the environment. Normally, you have to specify additional node properties in the environment definition.

Maintaining an environment might involve tasks such as installing a new version of iPlanet UDS or changing the location of the Environment Manager. You should also routinely back up the environment repository. Also, you need to understand how to start up the Environment Manager and Node Manager in the proper sequence and how to shut them down.

For more information on setting up and maintaining an iPlanet UDS environment, see Chapter 3, "Setting up and Maintaining an iPlanet UDS Environment."

Setting up and Maintaining Development Environments

In a development environment, along with the tasks required for any iPlanet UDS environment, there are normally a few special tasks.

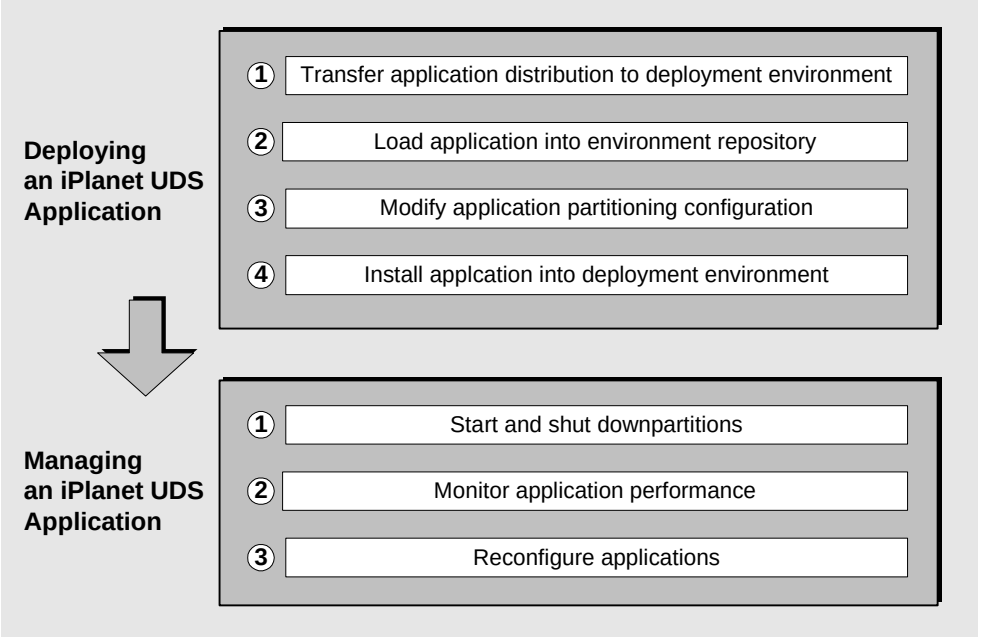
One of these additional tasks is to manage the environment's development repositories, which involves creating central repositories and starting a Repository Server for each. You must compact central repositories regularly, back them up periodically, and implement usage patterns among your development staff that will improve performance. For more information on managing central development repositories, see Chapter 8, "Managing iPlanet UDS Development Repositories."

Other tasks involve reconfiguring two server applications used by developers to make distributions that contain compiled components. You might also need to install library distributions needed by your development team. For more information on these special development environment maintenance tasks, see Appendix A, "Special Setup for Development Environments."

Deploying and Managing iPlanet UDS Applications

Deploying and managing iPlanet UDS applications involves a number of different stages, as illustrated in Figure 1-12. These stages are discussed briefly in this section. For more information about deploying an iPlanet UDS application, see Chapter 5, "Deploying iPlanet UDS Applications." For more information about managing an iPlanet UDS application, see Chapter 6, "Managing iPlanet UDS Applications."

Figure 1-12 Typical Application Deployment and Management Tasks



Application Deployment Tasks

This section describes the tasks shown as "Deploying an iPlanet UDS Application" in Figure 1-12.

1. Transfer application distribution to deployment environment

   The first task in deploying an iPlanet UDS application is to place the application distribution and any library distributions it requires on one of the nodes in the deployment environment. Normally you copy the distributions from a tape or other media and place them in a specified location in the iPlanet UDS directory structure on the selected node, typically the central server node. You might also have to package a distribution for transferring it to a deployment environment.

2. Load application into environment repository

   Once an application distribution is resident on a node in the environment, you load it into the environment repository. The partitions comprising the application and the partitioning configuration can now be accessed by the Environment Manager service, which will coordinate the installation of the application on the various nodes in the environment.

3. Modify application partitioning configuration

   An application distribution represents an application as it was configured by developers—the assignment of logical partitions was to various nodes in some simulated deployment environment. Your deployment environment, however, might differ substantially from the simulated environment definition used by a developer in making an application distribution. For example, an independent software vendor (ISV) or value added reseller (VAR) might have developed an application without any knowledge of your deployment environment. As a result, you might need to reassign partitions within your environment, and might also need to specify failover and load balancing properties for any replicated partitions.

4. Install application into deployment environment

   The final stage in deploying an application is to install the modified application configuration into your deployment environment. Each partition is installed in a known location on its assigned nodes. The Environment Manager can automatically perform this installation on all server nodes that are up and running. The installation of client partitions is a bit less automated, but iPlanet UDS provides you with utilities that streamline installation on client nodes.

Another system management task is deploying new versions of an application. This might require you to delete older versions, or perform a phased changeover to the new version.

Application Management Tasks

This section describes the tasks shown as "Managing an iPlanet UDS Application" in Figure 1-12.

- Start and shut down partitions

  After you successfully deploy the application, you can start it up. Although there are different ways to start the application, the recommended method is to use iPlanet UDS system management tools, such as the Environment Console or `Escript`, which give you the most control over the start and management of the application. You can also shut down one or more running (active) partitions using these tools.

- Monitor application performance

A typical task in managing a running application is to gather performance data. iPlanet UDS system management agents can collect runtime data that you can use to diagnose performance problems. You can maximize application performance by manually starting and stopping server partitions in response to user load patterns, hardware or software problems, or performance bottlenecks.

- Reconfigure an application

  However, to solve performance problems, you might have to make changes in the physical environment or move and replicate application partitions. You might also need to have the developers redesign parts of the application.

Deploying Library Distributions

Deploying a library distribution is similar to deploying an application distribution. In a deployment environment, you normally deploy library distributions as part of the deployment of the applications that reference the libraries. In a development environment, you can deploy a library distribution needed by developers to create and test the applications they are coding. In this situation, the library projects must also be imported into the central development repositories.

Libraries, unlike applications, are not executed and therefore do not need to be managed once they are installed.

For more information on deploying a library distribution, see "Deploying a Library Distribution" on page 177.

iPlanet UDS System Management Tools

iPlanet UDS provides a number of system management tools, the most important of which are the Environment Console and its command line counterpart, `Escript`. iPlanet UDS provides a separate set of command-line tools for managing development repositories.

This section briefly describes all the system management tools.

Environment Console

The Environment Console application provides a graphical user interface for performing most of the system management tasks discussed in "iPlanet UDS System Management Tasks" on page 53. You use the Environment Console to create and modify environment definitions and to deploy an application or library distribution into an environment. You also use the Environment Console to monitor the status of applications and partitions, start and shut down applications and partitions, and to gather performance statistics for running applications.

Most system management procedures explained in this guide assume that you are using the Environment Console. For an introduction to the Environment Console, see Chapter 2, "The iPlanet UDS Environment Console."

Escript Utility

The `Escript` utility is the functional equivalent of the Environment Console application, but with a command-line interface. You can perform any of the functions using Escript commands that you can perform in the Environment Console, but you can also incorporate these functions into scripts for execution at specified times.

For more information on the `Escript` utility, see the *Escript and System Agent Reference Guide*.

Launch Server

The *Launch Server* is an iPlanet UDS service that runs on client nodes and starts iPlanet UDS applets and applications. The Launch Server can run several iPlanet UDS applications under the same operating system process, which enables the applications to start faster and use less memory. It can also ensure that the most recent copy of an application is installed on the client; if it is not, the Launch Server automatically downloads the newer copy.

For more information about the Launch Server, see Chapter 9, "Launching iPlanet UDS Applications and Applets."

Repository Management Tools

iPlanet UDS provides several command-line tools for creating, copying, starting, stopping, and compacting central development repositories and Repository Servers.

For more information on repository management tools, see Chapter 8, "Managing iPlanet UDS Development Repositories."

Chapter 2

The iPlanet UDS Environment Console

This chapter describes the Environment Console, iPlanet UDS's window-based tool for managing iPlanet UDS environments and applications.

This chapter assumes that you have set up an active environment by installing iPlanet UDS system software on all the nodes in your environment. If you have not yet set up your environment, see Chapter 3, "Setting up and Maintaining an iPlanet UDS Environment" for overview information and the *iPlanet UDS System Installation Guide* for detailed installation instructions.

This chapter covers the following topics:

- overview of the Environment Console
- starting the Environment Console
- the Active Environment window
- other Environment Console windows
- summary of menu commands
- using iPlanet UDS windows

Overview

The Environment Console is the main iPlanet UDS system application that you will probably use to manage iPlanet UDS environments and applications.

You can use the Environment Console to perform environment tasks, such as creating and modifying environment definitions, and application tasks, such as deploying and managing iPlanet UDS applications.

More specifically, you use the Environment Console to:

- create and modify environment definitions

  The Environment Console provides a set of node templates and a node property dialog that you use to specify iPlanet UDS nodes, which are the building blocks of iPlanet UDS environment definitions. You can also export environment definitions. For information on creating and modifying environment definitions, see Chapter 4, "Creating and Modifying Environment Definitions."

- deploy iPlanet UDS applications

  You can load an application distribution into your active environment, modify the application partitioning configuration to match your active environment, and install the application into your environment. For information about deploying iPlanet UDS applications, see Chapter 5, "Deploying iPlanet UDS Applications."

- start and shut down applications and their components

  After an application has been installed, you can start the entire application (all its partitions) or start installed partitions one at a time. For information about managing iPlanet UDS application execution, see Chapter 6, "Managing iPlanet UDS Applications."

- monitor status and performance

  You can view the status of system management agents, and monitor instrumentation data to assess the performance of your applications. For information about monitoring iPlanet UDS applications, see Chapter 6, "Managing iPlanet UDS Applications."

The Environment Console connects to and communicates with the executing Environment Manager, and any active Node Managers in your environment. You perform iPlanet UDS system management tasks by accessing your environment repository and the full hierarchy of iPlanet UDS system management agents.

iPlanet UDS system management agents, the agent hierarchy, agent commands, and agent instruments are fully described in *Escript and System Agent Reference Guide*.

In development environments, you usually create and modify environment definitions, which are then used by developers when they configure their applications. When you use the Environment Console to perform environment tasks, you change information stored in the environment repository.

In deployment environments, you spend more time managing applications. When you perform application deployment and management tasks, you use the Environment Console to navigate the hierarchy of system management agents, open windows representing particular agents, invoke commands of those agents, and display or set instrument data of those agents. User-defined agents, if any, also appear in the agent hierarchy along with the system management agents provided by iPlanet UDS.

The `Escript` command-line utility is functionally equivalent to the Environment Console. However, all procedural instructions in this guide assume you are using the Environment Console. For information on Escript procedures, see *Escript and System Agent Reference Guide*.

Starting the Environment Console

You can start the Environment Console on any node in your iPlanet UDS environment.

➤ **To start the Environment Console on Windows or Windows NT**

1. Double-click the Environment Console icon.

➤ **To start the Environment Console on UNIX, OpenVMS, or Windows NT**

1. Type the `econsole` command (see "Using the `econsole` Command" below for information).

   When the Environment Console starts, it opens the Active Environment window shown in Figure 2-1 on page 65.

Using the econsole Command

As mentioned above, you start the Environment Console on command line-based operating systems by executing the `econsole` command.

The syntax of the `econsole` command in portable syntax is:

weconsole [**-fns** *name_server_address*] [**-fl** *logger_flags*][**-fm** *memory_flags*]

The OpenVMS syntax for `econsole` is:

VFORTE ECONSOLE
[/**NAMESERVER=***name_server_address*]
[/**LOGGER=***logger_flags*]
[/**MEMORY=***memory_flags*]

The following table describes the command line flags for the `econsole` command.

Flag	Description
-fns *name_server_address* /**NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-fl *logger_flags* **/LOGGER=***logger_flags*	Specifies the logger flags to use for the Environment Console session. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. Overrides the FORTE_LOGGER_SETUP environment variable setting. On UNIX, you must specify the logger flags in double quotes.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the memory flags to use for the Environment Console session. See "-fm Flag (Memory Manager)" on page 375 for syntax information. Overrides defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.

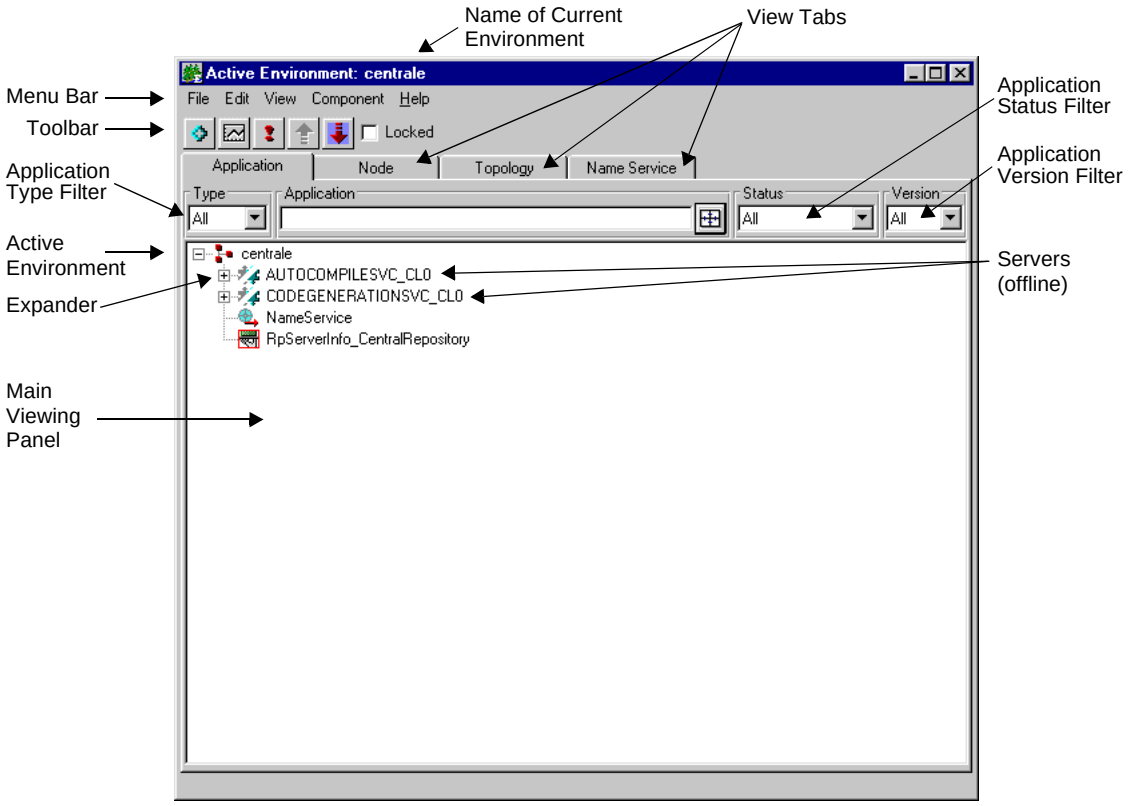
The Active Environment Window

The Active Environment window is the first window you see when you start the Environment Console. This window is a view of your active environment—the environment that is up and running at your site, to which your node belongs. In the Active Environment window, you can view all the components of your environment—nodes and applications—and perform all the tasks that involve them. You can use the Active Environment window to modify the environment definition for your active environment, and to deploy and manage applications.

Like many other windows in the Environment Console, the Active Environment window actually represents a system management agent (in this case the Environment agent), giving you access to that agent's commands and instruments.

The Active Environment window consists of four areas: the main viewing panel, the menu bar, the toolbar, and the status bar.

Figure 2-1 Active Environment Window



Menu Bar

The Environment Console menu bar provides all the commands you can execute from the currently active window. The menus for the Active Environment window are summarized below, and a full list of the commands is provided at the end of this chapter.

The Active Environment window menus are:

File menu Provides commands at the environment level. Used for creating and modifying environment definitions: opening property dialogs, and opening, locking, unlocking, exporting, importing, and saving environment definitions. Also used to load application or library distributions into the environment repository.

Edit menu Provides the standard editing commands, which you can use while editing environment information.

View menu Provides commands to modify the view displayed in the main viewing window.

Component menu Provides commands for starting, shutting down, and installing nodes; modifying log flags and viewing logs; dumping environment status; exporting the environment to an Escript file; showing partitions and removing lost partitions; listing and changing directory; connecting, disconnecting, and showing environments; showing administration details.

On certain platforms, a Help menu is also available. If you select an active node, other menus might be available.

Toolbar

The Environment Console toolbar is a set of icons that represent commonly used menu commands. The following buttons are on the Active Environment window toolbar:

Icon	Name	Description
	Node Template	The Node Template button opens the Node Template window, from which you can drag and drop node templates. For a full description of node templates and how to use them to create environment definitions, see the New… command description in "Creating a New Simulated Environment Definition" on page 129.

Icon	Name	Description
	Charts	The Charts button opens a window that contains a timeline for a specific instrument.
	Environment Alert	The Environment Alert button opens a Send Alert window, which you can use to send an alert event to all applications that are registered for the event.
	Start Up	The Start Up button executes the startup command of the selected agent. You can use it to start an application, logical partition, or installed partition.
	Shut Down	The Shut Down button executes the shutdown command of the selected agent. You can use it to shut down a Node Manager, application, logical partition, or active partition.
	Locked toggle field	The Locked toggle field provides an easy way to lock and unlock an active environment definition, and to determine its lock status.

Main Viewing Panel

The main viewing panel displays the active environment as a hierarchy of nodes. Nodes containing other nodes nested within them can be expanded by clicking the box with the plus symbol (“+”) to the immediate left of the icon.

You can view the system agent hierarchy in any of four ways:

- Application outline view (See Figure 2-2 on page 68), which displays the active environment’s applications, partitions, and service objects.
- Node outline view (See Figure 2-3 on page 69), which displays the nodes on which the active environment’s applications and service objects are deployed.
- Topology outline view (See Figure 2-4 on page 70)
- Name Service view (See Figure 2-5 on page 71)

In each of these four views the active environment is displayed as a hierarchy of elements, each with an icon representing its type and current status. The menus available in each view enable you to perform administrative tasks on the displayed elements.

Figure 2-2 Application Outline View

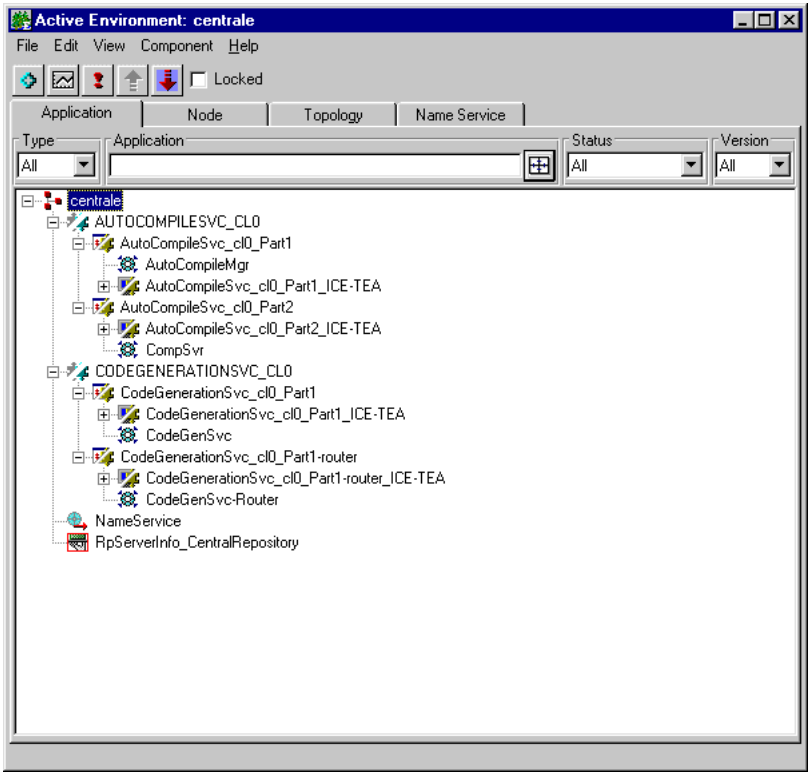


Figure 2-3 Node Outline View

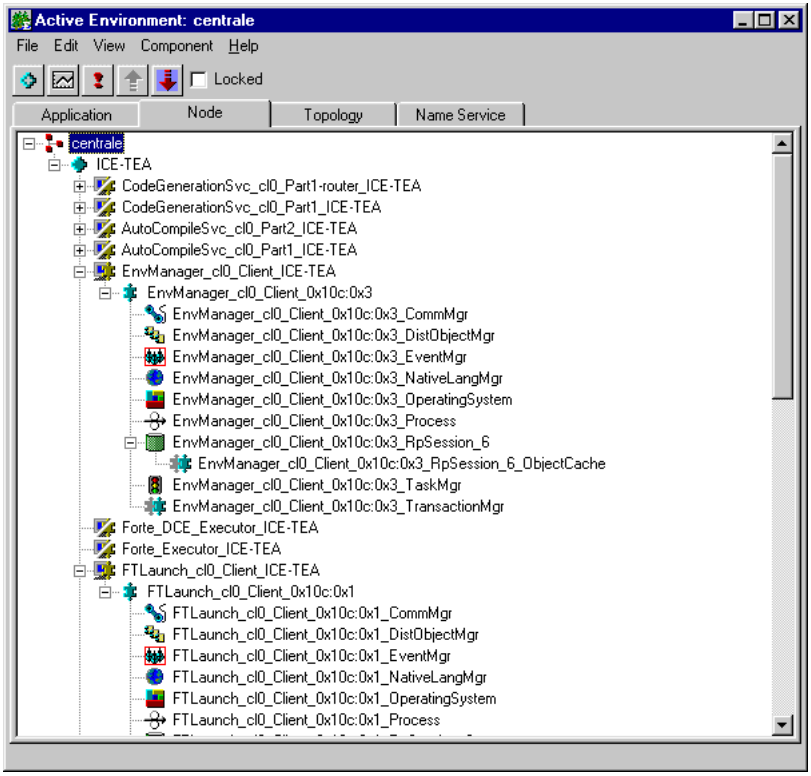


Figure 2-4 Topology Outline View

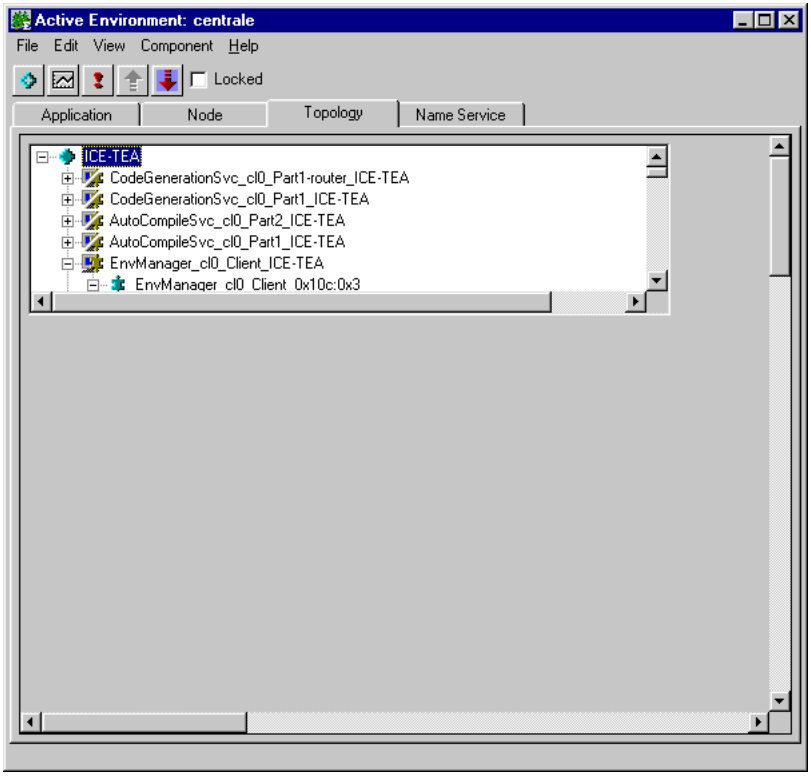
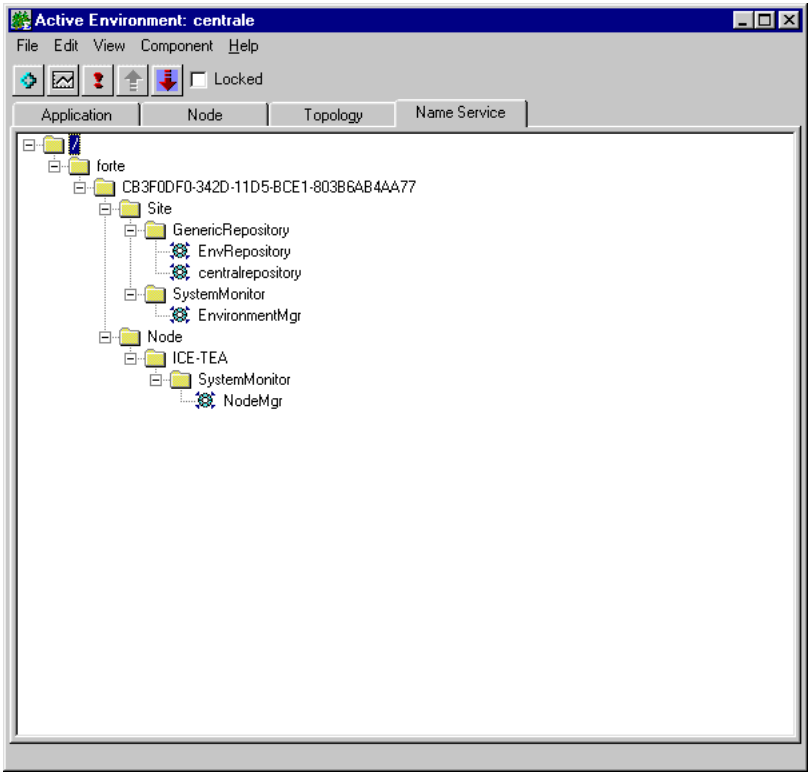


Figure 2-5 Name Service view



For more information on the agent hierarchy, see "System Management Agents" on page 46 and *Escript and System Agent Reference Guide*.

Name, Type, and Status in outline views In the application and node outline views each agent is represented by an icon that indicates both the agent's type and its status, and its name. For example, in Figure 2-2 on page 68 the node CODEGENERATIONSVC_CL0 is represented by an icon that indicates that the agent is a server, and that its status is "offline." (This means that the server has been loaded into the current environment but that it is currently inactive.) For information about the various agent types and their possible status, see *Escript and System Agent Reference Guide*.

There are two basic modes for using the Active Environment window: Environment Edit mode (for modifying your active environment definition) and Agent mode (for deploying and managing applications).

Environment Edit mode From the main viewing panel, you can modify the environment definition by opening the properties dialog for the environment or for individual selected nodes, or by adding or deleting nodes. Refer to Chapter 4, "Creating and Modifying Environment Definitions," for information about modifying your environment definition.

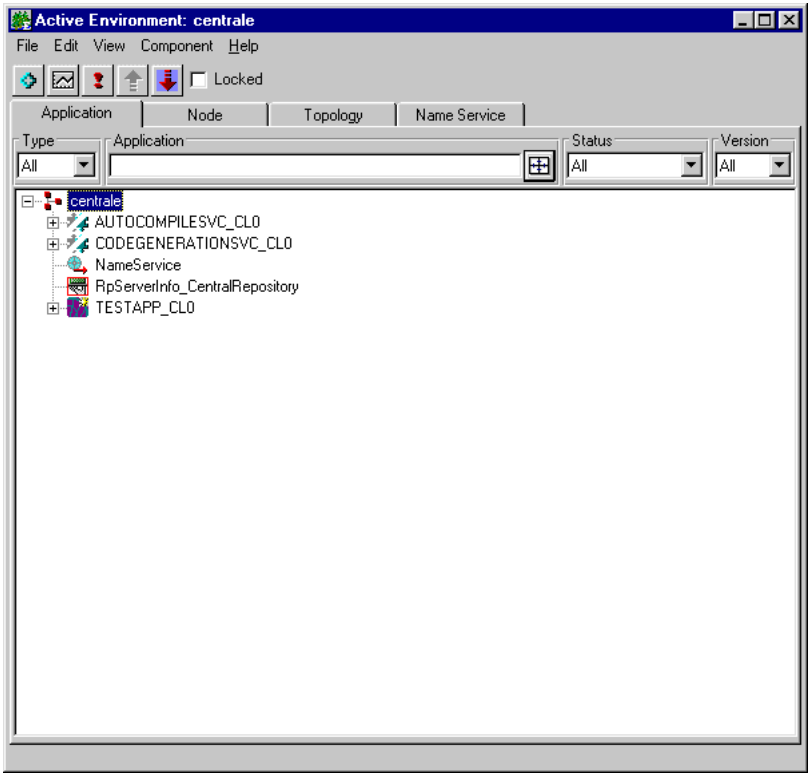
Agent mode From the main viewing panel, you can navigate the subagent hierarchy below the Environment agent by expanding the node or application outline views and locating any component (or agent) of interest.

➤ **To navigate the agent hierarchy**

1. Select the tab for Application or Node view.

   Suppose, for example, you selected the application outline view. A list of applications loaded or installed in the environment would be displayed, as shown in Figure 2-6.

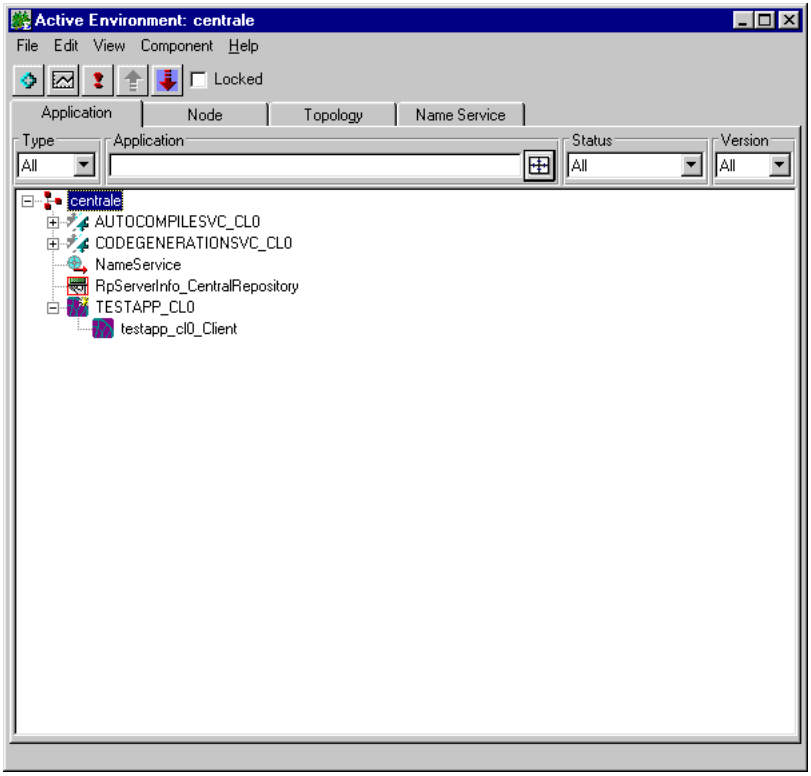
Figure 2-6 Application View with Installed Application



2. Click the expansion arrow next to an application of interest.

   Suppose, for example, you selected an installed application. The application expands to show its logical partitions (Partition agents), as shown in Figure 2-7.

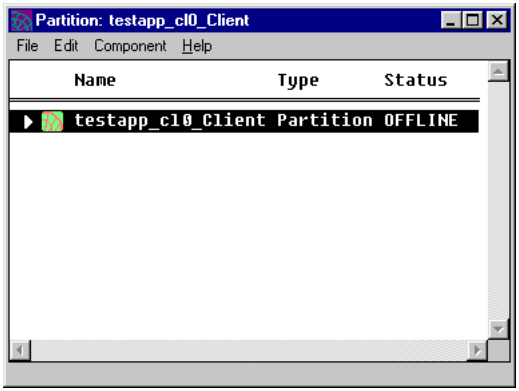
Figure 2-7 Application View with Expanded Installed Application



3. Double-click a Partition agent.

   The Agent window for the Partition agent opens, as shown in Figure 2-8.

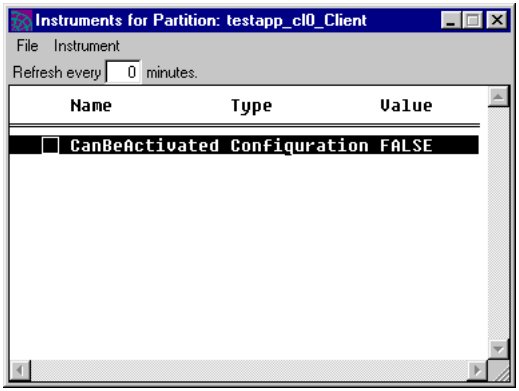
Figure 2-8 Agent Partition Window



4. Select Instruments from the File menu of the Agent window.

   The Instruments window for the Partition agent appears, displaying the instruments of the Partition agent, as shown in Figure 2-9

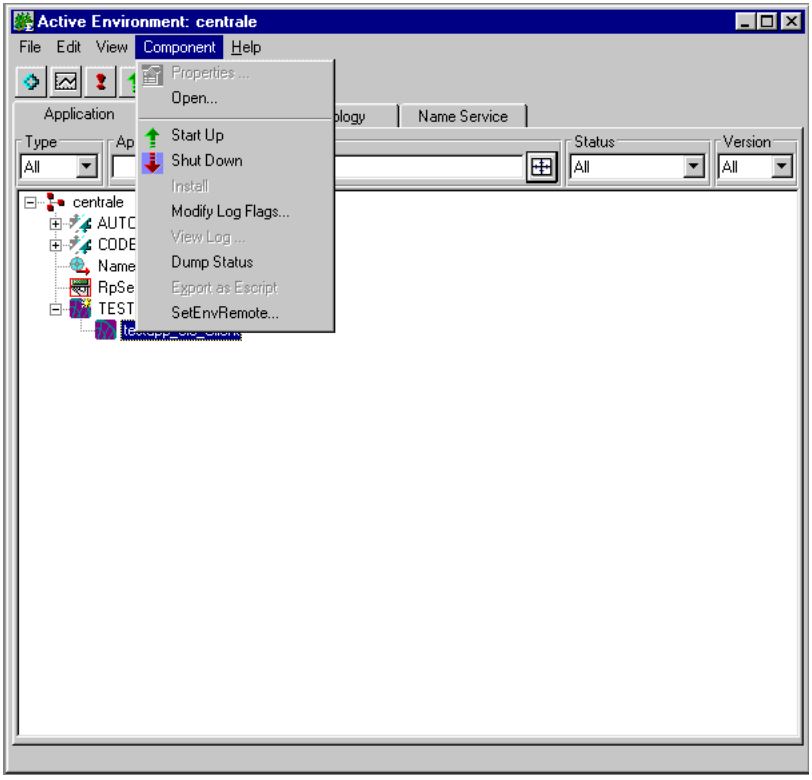
Figure 2-9 Instrument Window for a Partition Agent



5. Select the partition in either its Agent window or in the Active Environment window.

   The commands for the Partition agent are either in a menu called Component, or in custom menus defined by the agent. For example, in Figure 2-10, the Component menu for an Installed Partition agent provides the Start Up command.

Figure 2-10 Commands for an Installed Partition Agent



Status Bar

The status bar displays status updates for some tasks you perform in the Active Environment window. When you open, save, or export an environment definition, or when you deploy iPlanet UDS applications, the status bar displays the status of these processes as they progress.

Status information also includes the source and destination directories for exporting environment definitions and for loading application distributions.

Exiting the Environment Console

To exit the Environment Console, you select the Exit (or Quit) command from the File menu. The Environment Console prompts you to save any changes you have made to the active environment definition or to other environment definitions before exiting.

Other Environment Console Windows

In addition to the Active Environment window, there are a number of other windows used for performing various system management tasks. These are described briefly below. The windows are grouped according to whether they are used for environment tasks (creating and modifying environment definitions) or application tasks (deploying and managing iPlanet UDS applications).

Environment Tasks: Environment Edit Mode

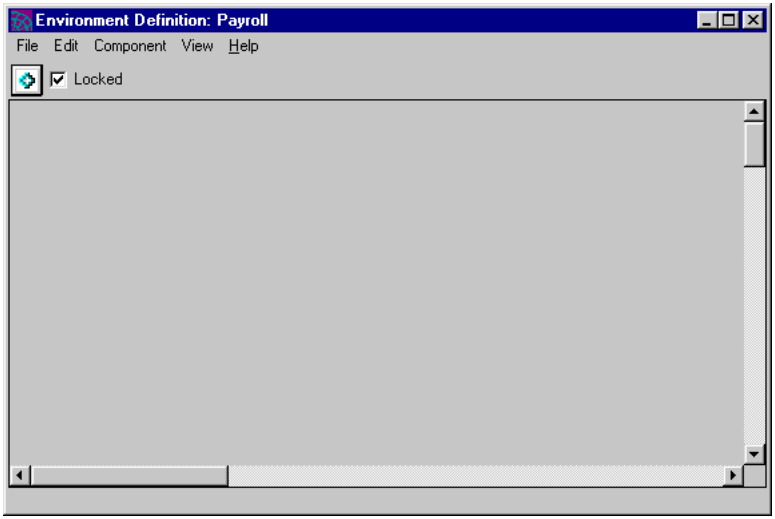
There are two windows, in addition to the Active Environment window, used in Environment Edit mode to create and modify environment definitions. For more information on using these windows, see Chapter 4, "Creating and Modifying Environment Definitions."

Environment Definition Window

The Environment Definition window, shown in Figure 2-11, is used mainly in development environments, and represents the environment definition of an environment other than the active environment. It is used to specify the nodes in a simulated deployment environment—that is, an environment used to simulate a real deployment environment for the purpose of testing and partitioning an application.

The Environment Definition window is opened when you create a new environment definition or when you open a non-active environment definition.

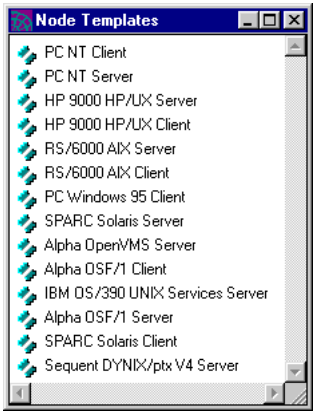
Figure 2-11 Environment Definition Window



Node Template Window

The Node Template window, shown in Figure 2-12, provides node templates used in creating or modifying environment definitions. You can drag a node template from the Node Template window to an Environment Definition window or the Active Environment window.

Figure 2-12 Node Template Window



For a description of each of the node templates, see "Node Templates" on page 132.

Application Tasks: Agent Mode

You will use a number of windows, in addition to the Active Environment window, to deploy and manage applications:

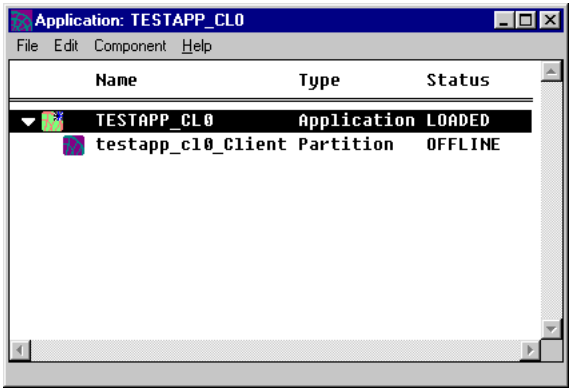
- Agent windows
- Instruments windows
- Agent Information windows
- Chart windows
- Component Log window

Agent Window

You can open a window representing each agent that appears in the subagent hierarchy displayed in the Active Environment window. To open an Agent window, double-click on the agent in the Active Environment window.

Figure 2-13 shows an Agent window for the TESTAPP_CL0 Partition agent. This agent represents a loaded instance of the server partition of the TESTAPP application.

Figure 2-13 Typical Agent Window



The viewing panel of an Agent window represents the current agent and its status. You can expand the current agent to view its subagents, in the same way you expand agents within the Active Environment window. You cannot open more than one Agent window or Agent Information window for a given agent.

Within an Agent window, you can view the instruments defined for the agent and execute the commands performed by the agent. You can display the instruments available for the agent represented by this window by choosing File > Instruments. The commands that are available in the menus for this window depend on what kind of agent is selected.

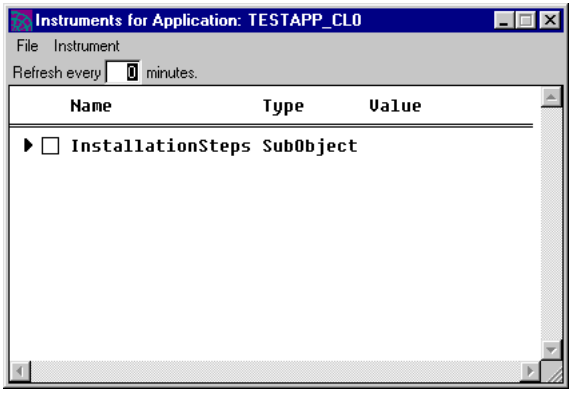
For more information on using the Agent window, see Chapter 6, "Managing iPlanet UDS Applications." For information about a particular agent, its commands and instruments, and its statuses, see *Escript and System Agent Reference Guide*.

Instruments Window

You can open the Instruments window for the current agent in the Agent window by choosing the File > Instruments command in the Agent window. This window, as shown in Figure 2-14, displays the instruments defined for the current agent and the value of each instrument. If you want to view the instruments for another agent, you need to open an Agent window for that agent, then open the Instruments window for that agent.

From this window you can refresh instrument values, set the values of instruments that are not read-only, and turn logging of the instruments on and off by clicking the check box.

Figure 2-14 Instruments Window



The type of an instrument determines how it works:

Average Read only. Contains an average value.

Compound Contains more than one instrument. These instruments can be of different types.

Configuration Read/write or read only. Contains a simple value, such as an integer value or a TextData object.

Counter Read only. Contains a value based on counting something.

SubObject Contains more than one instrument. All instruments are of the same type.

Timer Read/write. Timer that prompts the agent to do something after a certain interval or set of intervals.

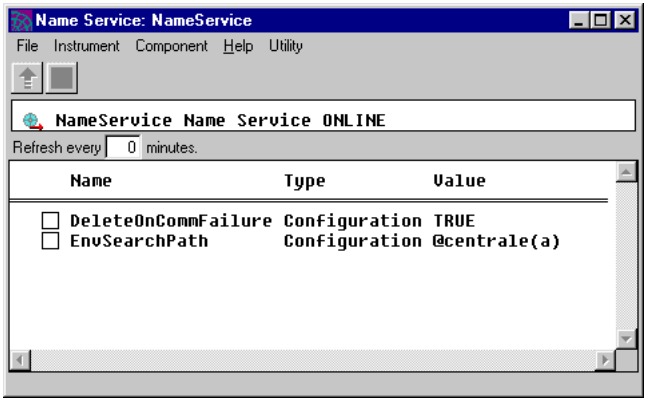
For more information on using the Instruments window to monitor and manage applications, see "Monitoring iPlanet UDS Applications" on page 194. For information about what instruments are available for each agent, see *Escript and System Agent Reference Guide*.

Agent Information Window

The Agent Information window, opened within the Active Environment window, is available only for agents that do not have subagents. This window, shown in Figure 2-15, displays the current agent and a list of the agent's instruments.

You can open this window by double-clicking an agent that has no subagents (no expansion arrow).

Figure 2-15 Agent Information Window



Within the Agent Information window, you can refresh instrument values, set the values of instruments that are not read only, and turn logging of the instruments on and off. You can also execute the commands performed by the agent.

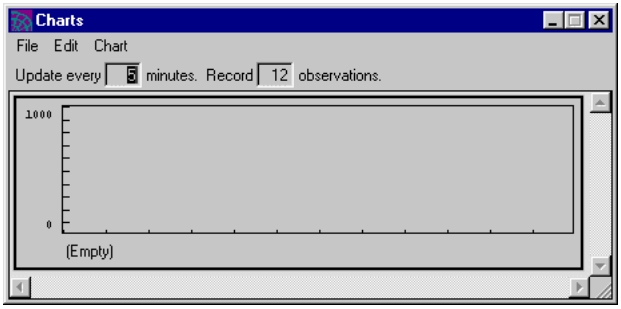
You cannot open more than one Agent or Agent Information window for a given agent.

Charts Window

The Charts window, shown in Figure 2-16, is opened only from within the Active Environment window. It displays histograms—charts displaying series of values—for any Average or Counter instruments you want to chart. You simply drag the instrument you want to track from its respective instrument window, and drop it on an open chart in the Charts window.

From within the Charts window, you can set display properties for each graph. For more information on using the Charts window, see "Tracking Instrument Data: Charts Window" on page 201.

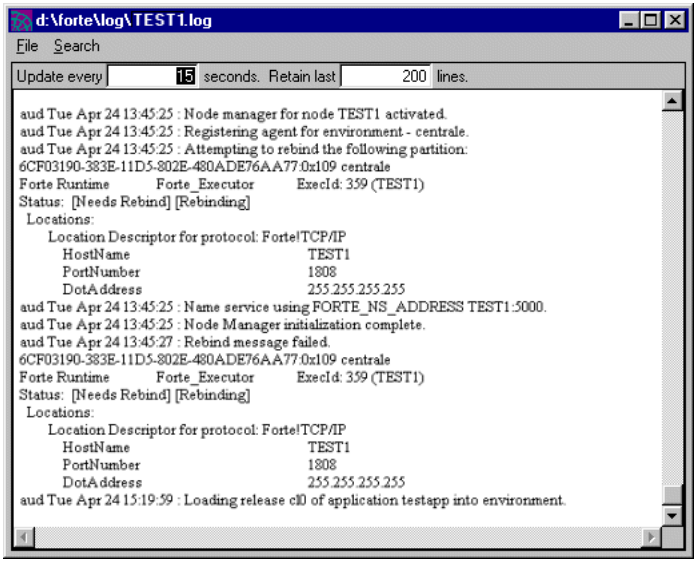
Figure 2-16 Charts Window



Component Log Window

You can open a window that displays the log file for the component selected in the Active Environment window. The Component Log Window is shown in Figure 2-17. To open this window from the Active Environment window, select a Node agent, then click Component > View Log.

Figure 2-17 Component Log Window



This window lets you view the log for the component.

To change how often the contents of this window is updated, change the number of seconds in the field at the top of the window: “Update every ____ seconds.”

To change how many lines of the log file are displayed in this window, change the number in the field at the top of the window: “Retain last ____ lines.”

For information about the commands available for this window, see the iPlanet UDS Online Help.

Using iPlanet UDS Windows

The Environment Console is a graphically-oriented system application developed in iPlanet UDS. As such, windows in the Environment Console have behavior dependent on your host window system, but provide a few iPlanet UDS-specific enhancements.

Using the Mouse

The Environment Console behaves like any standard applications in your window system—mouse clicks select objects, double-clicks open objects, and click-and-drag operations move or copy objects.

To work in iPlanet UDS, you need only one mouse button. If your mouse has more than one button, you use the left-most button, or, if your mouse is configured specifically for left-handed use, the right-most button.

To activate some commands, you use mouse clicks in conjunction with qualifier keys. For example, to select multiple nodes, you first click one node, then, while depressing the Shift key, click successive nodes to add to or delete from the selection.

Using the Keyboard

The Environment Console mirrors its host window system in the flexibility of keyboard control it provides.

In the Windows and Motif window systems, you can use the keyboard exclusively to complete any Environment Console task (even those designed for mouse interaction), because these window systems have been designed to allow mouse-independent operation.

The Environment Console offers keyboard equivalents for many menu commands on every window system. When available, the keyboard equivalent is displayed on the menu next to the command.

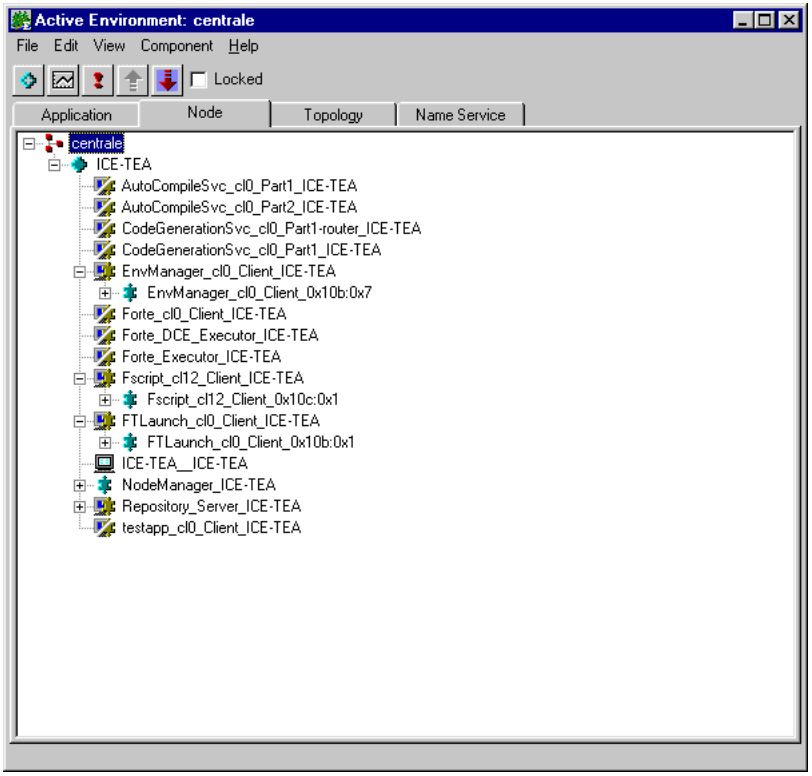
Using the Hierarchical Browser

The Environment Console uses iPlanet UDS outline fields to provide browsing of agent hierarchies. For example, the Environment Console in Node Outline View, shown in Figure 2-18, provides a hierarchical display of Node subagents.

You may be familiar with browsers as the file directory display mechanisms in your host window system. iPlanet UDS hierarchy displays behave most like the File Manager application in Microsoft Windows. These displays are in the form of a tree, with each successive child level indented to the right of its parent level. To select an item from a browser, you click it. To open an item, you either double-click it, press Return while it is selected, or click the plus sign ("+") to expand it.

Expanding and Collapsing Items Items with a plus sign ("+") to their immediate left have sublevels nested within them. To display the sublevels click the plus sign. To hide the sublevels click the negative sign ("-") to the immediate left of the item.

Figure 2-18 Hierarchical Display



Using the Clipboard

You can use your host window system clipboard to cut, copy, and paste textual and image information within the Environment Console, and to exchange such information between the Environment Console and other applications in your host window system.

NOTE Under the Motif window system, the Environment Console offers limited support for the Motif clipboard. Nevertheless, under any circumstance, you can paste text from any Motif application into iPlanet UDS by using the technique you use to select and paste text from one xterm to another: select the text, and press the middle mouse button in the target window. This technique works because it does not employ the Motif clipboard.

Using Multiple Windows

You can open any number of Environment Console windows. iPlanet UDS lets you work with any number of concurrent windows, but any single agent window can only be opened once.

Chapter 3

Setting up and Maintaining an iPlanet UDS Environment

The first task you must perform as an iPlanet UDS system manager is to set up an iPlanet UDS environment.

Although the design and physical composition of development environments and deployment environments might differ somewhat, the general approach to getting the iPlanet UDS runtime system installed and running is essentially the same for both.

Once you have set up your iPlanet UDS environment, you may have to shut down or restart nodes in that environment and possibly restore system management repositories.

This chapter explains how to set up and maintain an iPlanet UDS environment, and covers the following topics:

- overview of environment design
- summary of environment setup process
- setting up a central distribution node
- modifying the environment definition
- connecting iPlanet UDS environments
- manually starting and stopping system management services
- maintaining system management repositories

Setting up an iPlanet UDS Environment

Getting an iPlanet UDS environment up and running involves more than simply installing iPlanet UDS system software on each node in your physical environment. It also includes ensuring that iPlanet UDS system management services are started up in the correct sequence.

In addition, if you are setting up a development environment, you normally create a central development repository that supports some number of developers working together to develop and test distributed applications (see Chapter 8, "Managing iPlanet UDS Development Repositories"). You may also need to reconfigure two server applications, CodeGenerationSvc and AutoCompileSvc, used to make distributions that contain compiled components (see "Auto-Compile Services" on page 315).

Designing Your Environment

Before you set up your environment, you should consider the design issues discussed in this section.

Deployment environment A deployment environment usually contains the iPlanet UDS runtime system on all nodes and the iPlanet UDS system management applications on a central server node, as well as any user applications. There might also be other software, such as databases, libraries, and so forth, that need to be available in the environment.

If you are setting up a deployment environment, consider:

- the availability and location of vital resources (such as database management systems) that will be needed by your distributed application
- the speed and reliability of servers
- the characteristics of your client nodes
- the networking protocols

Development environment A development environment has the same elements as a deployment environment, but also has iPlanet UDS repository services, development repositories, and application development software, such as the iPlanet UDS Workshops and `Fscript`.

If you are setting up a development environment, you should also consider:

- the location of your central development repositories (Repository Servers).
- the presence of application resources, such as databases and external libraries

  Because your environment will be used not only to develop, but to test distributed applications (by simulating deployment environments), vital application resources must also be present.
- the resources available on your development workstations

  While performance requirements for your servers may be less demanding than in deployment environments, the requirements for your development workstations (usually client nodes) may be more demanding than in a deployment environment.

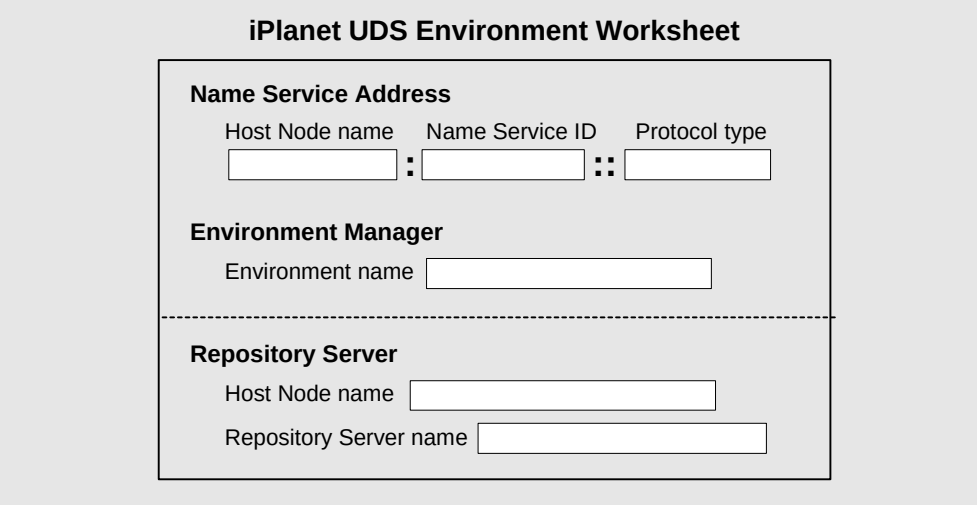
Location of system management processes In either case, the main iPlanet UDS environment design consideration is where to locate iPlanet UDS system management services, such as the Environment Manager. Generally speaking, you should designate one server node as a central server—a node that hosts the Environment Manager service (and Name Service, since it runs in the same server partition). Normally, this node will also serve as a central distribution node used for installing iPlanet UDS system software on other nodes in your environment.

Since each node in your environment will have to access the Name Service to communicate with remote partitions, you must supply the Name Service address to each node at installation time. (In a development environment you must also supply each development node with the name of your central development repository.)

Therefore, before beginning to set up your environment, you should decide on the Name Service address, as well as a few other items contained in the Environment Worksheet shown in Figure 3-1.

The worksheet assumes that all nodes in your physical environment are networked using TCP/IP, DECnet, or both.

Figure 3-1 Worksheet for Setting up an iPlanet UDS Environment



The information needed in the worksheet is the following:

Name Service Address This address is the value that other nodes will use to connect to this environment using their FORTE_NS_ADDRESS environment variable or the **-fns** flag on most iPlanet UDS commands. The following three parts of the Name Service address each depend on the network protocols being supported:

- Host Node name: This is the name of your central server node—the node on which the Environment Manager service and Name Service will run. The name must be a valid TCP/IP or DECnet node name (or a name valid for both).
- Name Service ID: This is an ID whose form depends upon the network transport protocol used to access the Name Service. For TCP/IP, it is a port socket number not currently in use by other processes—you can use a value between 1025 and 65535. For DECnet, it is a unique, case-sensitive alpha-numeric DECnet object name (“$” and “_” also allowed), less than 8 characters in length. An example is “forte_ns”.
- Protocol type: This is a case-sensitive value of “TCP/IP” for TCP/IP and “DECnet” for DECnet. (Leaving the value blank will default it to TCP/IP.)

Typical Name Service addresses would be: “MyNode:5000” (for TCP/IP) and “YourNode:forte_ns::DECnet” (for DECnet).

Environment Manager This is the environment name the Environment Manager service will give to your active environment. The name must consist of alpha-numeric characters and can be of any length. (The default name is "CentralEnv" or "centrale".) If you plan to connect environments, you should ensure that each environment name is unique so that developers and system managers can easily specify different environments in their search paths.

Repository Server (development environment only) The following two properties are needed:

- Host Node name: This is the name of the node in your environment on which the Repository Server will run, generally a node with a sizeable amount of disk space.
- Repository Server name: This is the name by which the Repository Server will be advertised in your development environment. The Repository Server name will be used by all development workstations that wish to access the central repository. (You can actually have a number of central repositories.) The Repository Server name must consist of no more than 32 alphanumeric characters.

Summary of the Environment Setup Process

This chapter assumes your environment consists of at least one server and at least one client workstation. Your environment, however, can include any number of iPlanet UDS–supported server and client platforms.

➤ **To install and start an iPlanet UDS environment**

1. Make sure your network is working properly.

   Use netcopy utilities or "ping" nodenames (on TCP/IP) to make sure servers are running and communicating.

2. Set up a central distribution node (Windows NT, UNIX, or OpenVMS) at your site.

   This node will be the source for installing iPlanet UDS on all nodes in your environment. The iPlanet UDS distribution media consists of multiple CD–ROMs. This step is described in detail in *iPlanet UDS System Installation Guide*.

3. Install iPlanet UDS on your central server node, using the appropriate platform-specific installation program. This step is described in detail for each platform in *iPlanet UDS System Installation Guide*.

   The central server node hosts two important iPlanet UDS system management services needed by other nodes: the Name Service and Environment Manager.

   The installation program copies iPlanet UDS software into an iPlanet UDS directory structure on your target node, sets a number of environment variables (logical names on OpenVMS systems), and starts the Name Service and Environment Manager.

4. Install iPlanet UDS on all remaining nodes in your environment, using the appropriate platform-specific installation programs. This step is explained in *iPlanet UDS System Installation Guide*.

   The installation program copies iPlanet UDS software into an iPlanet UDS directory structure on each target node and sets a number of environment variables (logical names on OpenVMS systems). On server nodes, the installation program also starts the Node Manager.

5. Modify the default environment definition created for your environment.

   An environment definition, which specifies all the nodes in your environment, is created automatically as you install each node, in Step 3 and Step 4. However, you may need to modify this definition. This step is explained in Chapter 4, "Creating and Modifying Environment Definitions."

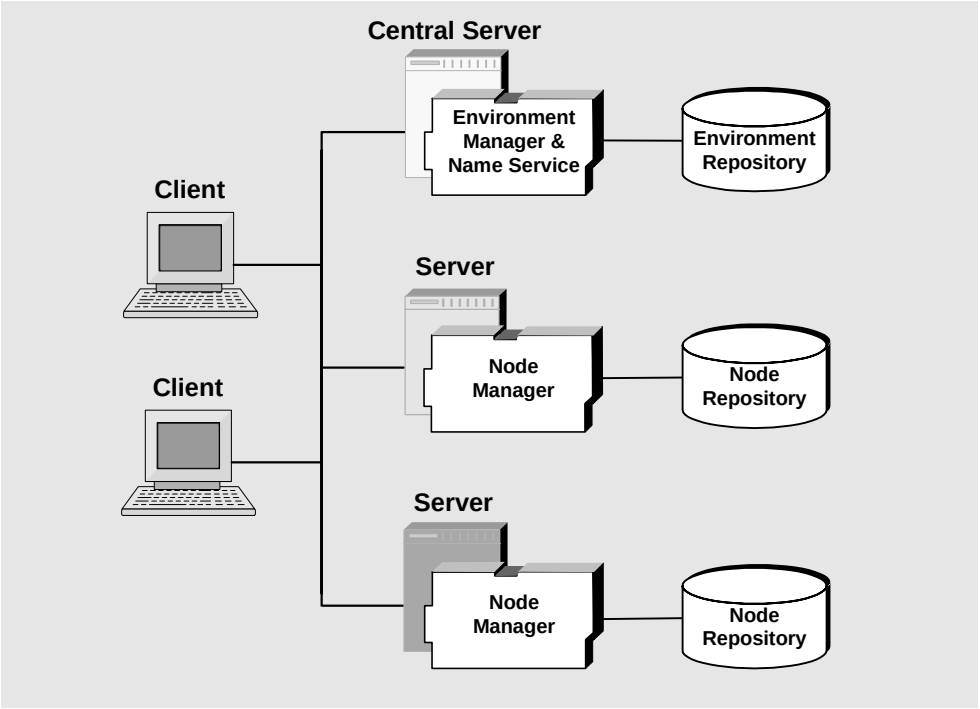
For detailed information about installing iPlanet UDS system software, see *iPlanet UDS System Installation Guide*.

Nodes In Your Environment

This section provides an overview of the different types of iPlanet UDS nodes that you can set up in your development or deployment environment. For information about how to install each of these types of nodes, see *iPlanet UDS System Installation Guide*.

Figure 3-2 illustrates a simple iPlanet UDS environment. This environment could be either a development or a deployment environment.

Figure 3-2 An iPlanet UDS Environment



Central Server Node

A central server node is the brain of the environment. This node runs the Environment Manager (and its Name Service), which manage and control the environment and applications running in the environment.

Environment repository This node maintains information about the entire environment in an *environment repository*. This environment repository contains information about the nodes and applications available in the environment. If you connect multiple environments together, then the environment repository also contains information about other environments, as described in "Connecting Environments" on page 115.

To start your environment, you start the Environment Manager (and, therefore, the Name Service). When the Node Managers for other nodes start, they connect to this environment using the name server address defined by this Environment Manager. "Startup Commands (nodemgr and start_nodemgr)" on page 104 explains how to start the Environment Manager.

An iPlanet UDS central server node has the following iPlanet UDS components installed:

- iPlanet UDS runtime system
- Environment Manager
- iPlanet UDS repository services
- Launch Server
- iPlanet UDS system management software
- iPlanet UDS application development software

As you install the central server node, you are prompted for the values you defined in the worksheet described in "Designing Your Environment" on page 88.

Server Nodes

A server node is a node capable of running iPlanet UDS server applications and server partitions of distributed iPlanet UDS applications. A server node has a unique identity in the environment, and connects to the environment by running a Node Manager and identifying the name server address for the Environment Manager it wishes to connect to.

You have two options for installing a server node: with iPlanet UDS system management and application software components, or without (just the iPlanet UDS runtime system).

As you install the server node, you are prompted for the name server address for the Environment Manager, as well as other values specific to your node. After you finish installing the iPlanet UDS components, you can redefine the environment variables using the iPlanet UDS Control Panel (if it is available) or directly in the operating system, as described in "Using the iPlanet UDS Control Panel" on page 345 and "Setting Environment Variables Without the iPlanet UDS Control Panel" on page 353.

Development Server

For a development environment, you can choose to install a Server Node as a custom installation of the iPlanet UDS application development software. This server node has the following components installed:

- iPlanet UDS runtime system
- iPlanet UDS repository services
- Node Manager
- Launch Server
- iPlanet UDS system management software
- iPlanet UDS application development software

If you want to run a server for test purposes, but not run any iPlanet UDS application development or system management software on the server, you can also use a deployment server.

Deployment Server (iPlanet UDS Runtime System)

For a deployment environment, you can use a node that has been set up as for a development environment. However, you can also choose to install a smaller set of iPlanet UDS components, which provide only the features needed to participate in the environment and run iPlanet UDS applications. You can install this configuration using the iPlanet UDS Runtime System installer. In this case, a server node has the following components installed:

- iPlanet UDS runtime system
- Node Manager
- Launch Server

When you install the iPlanet UDS runtime system on a platform that can run either a client or server node, such as UNIX and Windows NT, you get the components for both a runtime client and a runtime server.

Client Nodes

A client node is intended to run only client partitions of iPlanet UDS applications. This client node can be uniquely defined as part of the environment, or might be known as a member of a model node group, which is discussed in "Specifying Node Properties" on page 134. A client node participates in an environment when it starts a Node Manager by running the Launch Server, the Environment Console, or `Escript`.

Like the server nodes, you can install a client node either with iPlanet UDS system management and application software components, or without.

As you install the client node, you are prompted for the name server address for the Environment Manager (unless the client is standalone).

Development Client

For a development environment, you can choose to install a Client Only installation as a custom installation of the iPlanet UDS application development software. You can also choose whether to have the client installed as standalone, so that it does not participate in an environment after you install it unless you later redefine some of the settings for the client node. This client node has the following components installed:

- iPlanet UDS runtime system
- Launch Server
- iPlanet UDS system management software
- iPlanet UDS application development software

When you install the iPlanet UDS development system on a platform that can run either a client or server node, such as UNIX and Windows NT, you get the components for both a development client and a development server.

Deployment Client (iPlanet UDS Runtime System)

For a deployment environment, you can use a node that has been set up as for a development environment. However, you can also choose to install a smaller set of iPlanet UDS components, which provide only the features needed to participate in the environment and run iPlanet UDS applications. You can install this configuration using the iPlanet UDS Runtime System installer. In this case, a server node has the following components installed:

- iPlanet UDS runtime system
- Launch Server

When you install the iPlanet UDS runtime system on a platform that can run either a client or server node, such as UNIX and Windows NT, you get the components for both a runtime client and a runtime server.

Setting up Client Nodes after Installation

This section briefly describes common setup tasks you need to perform after you install iPlanet UDS on a client node.

Defining a Client Node Name

After you finish installing the iPlanet UDS components, you need to define a unique name for the client node and, if the node is a member of a model node group, specify the name of the defined model node. You can define the environment variables for these names using the iPlanet UDS Control Panel (if it is available) or directly in the operating system, as described in "Using the iPlanet UDS Control Panel" on page 345 and "Setting Environment Variables Without the iPlanet UDS Control Panel" on page 353.

Setting up the Launch Server

Client nodes (except those on OpenVMS platforms) have a utility called the Launch Server, which acts as a Node Manager for client nodes. The Launch Server enables your client nodes to automatically download deployed iPlanet UDS applications and lets several iPlanet UDS applications run their client partitions in a single iPlanet UDS process. The Launch Server is also used by several icons to start iPlanet UDS applications using the Ftcmd utility, as described in "Using the Ftcmd Utility" on page 307.

Ideally, your end users are not aware that the Launch Server exists when they start your iPlanet UDS applications, but they still enjoy the benefits that this utility provides. On UNIX, Windows NT, and Windows 95, icons that use the `Ftcmd` utility to start iPlanet UDS applications automatically start the Launch Server, as does the Launcher application.

On UNIX nodes, you need to set the FORTE_FTLAUNCH_PORT as a user-specific environment variable on each client node before you can start the Launch Server on that node.

The Launch Server utility and the `ftlaunch` command are fully described in Chapter 9, "Launching iPlanet UDS Applications and Applets."

Launcher Application

All iPlanet UDS installations (except those on OpenVMS platforms) provide the Launcher application. This application is an end user application intended to let an end user start up and shut down all their iPlanet UDS applications from a central control panel on their desktop. This application uses the Launch Server to start applications, shut down applications, and download more recent releases of deployed applications. As the system manager, you can choose to provide this application to your end users, or to have your end users use icons or commands to start iPlanet UDS applications. Chapter 9, "Launching iPlanet UDS Applications and Applets" discusses the Launcher application and how to set up application icons on client nodes.

For more information about the Launcher application see "About the Launcher Application" on page 294.

Structure of Installed iPlanet UDS Software

This section briefly describes the directory structure, environment variables, and default environment definition that are set up when you install iPlanet UDS software. *iPlanet UDS System Installation Guide* describes this structure and the environment variables for each platform in much more detail.

While it is important to first install iPlanet UDS software on the node you designate as your central server, installation of iPlanet UDS on all platforms is about the same. Basically it consists of copying the iPlanet UDS directory structure and source files to the target node, setting a number of environment variables used by iPlanet UDS at startup time, and automatically starting and shutting down the appropriate system management services for your target node.

You can install the software required for a central server node, the software required for a node in an application development environment, or just the iPlanet UDS runtime system. For information about how to install any of these options, see *iPlanet UDS System Installation Guide*.

iPlanet UDS System Software Directory Structure

When you install iPlanet UDS software on any platform, the installation program creates the directory structure shown in Figure 3-3 on page 99.

Figure 3-3 iPlanet UDS System Directory Structure

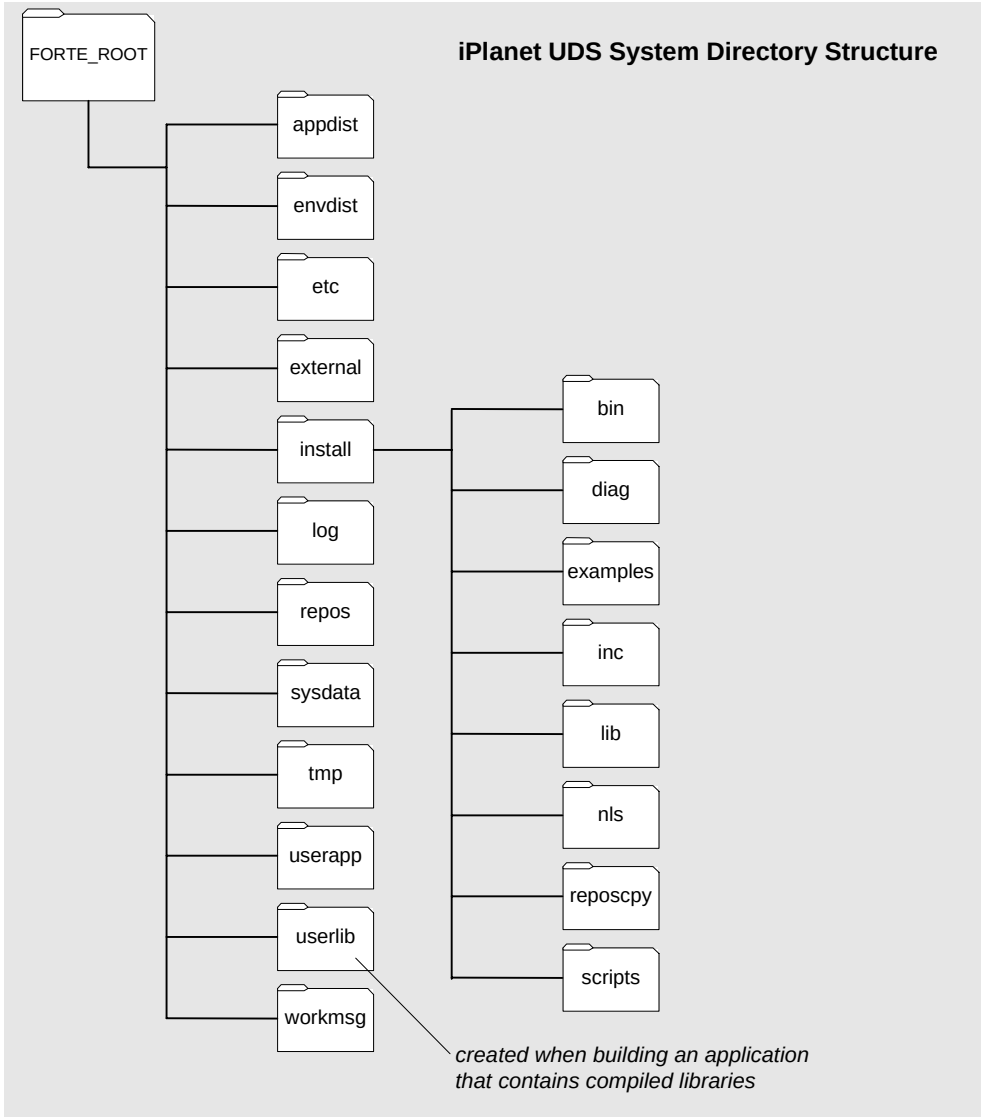


Table 3-1 lists the contents of the iPlanet UDS root directory. Many of the directories within the iPlanet UDS root directory are empty upon installation. These empty directories are used for the repository, application distributions, installed user applications, and other information used by the iPlanet UDS runtime system.

Table 3-1 Contents of the FORTE_ROOT directory

Directory	Content
appdist	Application and library distributions are created here when a developer makes a distribution, or placed here when you copy a distribution from a tape or other media in order to deploy the distribution.
envdist	Environment definitions are placed here when they are exported from the environment repository.
external	This directory is used as a place to put external libraries.
install	This directory contains installed iPlanet UDS software, as described in Table 3-2.
log	System and application log files are written here.
repos	Development repositories are created and stored here. This is the most critical directory to back up.
sysdata	This data contains information important to the iPlanet UDS runtime system and iPlanet UDS system management applications.
tmp	This directory is used as a temporary holding directory when making an application or library distribution that contains compiled components.
userapp	Application partitions and libraries are installed here by iPlanet UDS during the deployment process. Only application partitions that run on a particular node are installed in the userapp directory of that node.
userlib	When building an application, this directory is created to hold compiled libraries for the application. This directory does not exist when you first install iPlanet UDS.
workmsg	This directory is used for working copies of message files for international language support.

Table 3-2 lists the contents of the install directory, which contains iPlanet UDS system files, scripts to run iPlanet UDS components, and iPlanet UDS examples.

Table 3-2 Contents of the install directory

Directory	Content
bin	iPlanet UDS system executables.
diag	Diagnostic tools used by iPlanet UDS technical support.
examples	Sample projects and examples provided by iPlanet UDS.
hyperhlp	(UNIX only) Help files used by iPlanet UDS development and management software.
inc	(Development only) Header and template files used for C++ code generation and 3GL integration.
lib	C and C++ shared libraries used by iPlanet UDS system executables.
nls	Internationalization files.
reposcpy	iPlanet UDS development system seed repository used by the system when creating new repositories and the iPlanet UDS system repository.
scripts	Miscellaneous scripts and files provided as iPlanet UDS utilities.

Environment Variables

The iPlanet UDS installation program sets a number of environment variables (logical names in OpenVMS) using information you provide—or default values—for the host node. These settings are read by iPlanet UDS processes at startup unless you specifically override them.

The following table briefly describes the environment variables set by the installation program:

Environment Variable	Purpose
FORTE_ROOT	The root of the iPlanet UDS directory structure on your node.
FORTE_REPOSNAME	In a development environment, the name of the Repository Server accessed by your node.
FORTE_NS_ADDRESS	The network address of the Name Service in your environment.
FORTE_LOGGER_SETUP	Specifies log file names on your node, and the type and level of messages being logged by iPlanet UDS in these log files.

In addition to environment variables set by the installation program, there are a number of others you can set by hand. For more information on environment variables, see Appendix B, "iPlanet UDS Environment Variables."

iPlanet UDS stores environment variables in a different location on each platform (see "Setting Environment Variables Without the iPlanet UDS Control Panel" on page 353 or the *iPlanet UDS System Installation Guide* for details.)

The Default Environment Definition

If you install iPlanet UDS first on your central server node and then on other server nodes, your system management services are started in the correct sequence to create an environment definition for your active environment automatically. An environment definition specifies (among other things) all the nodes in your environment.

As each Node Manager starts in your iPlanet UDS environment, the Environment Manager dynamically adds a specification for that new node to the environment definition created when you first started up the Environment Manager. The environment definition is stored in the environment repository on your central server node.

In many situations, after installing iPlanet UDS on each node in your environment, you will need to add more detail to your node specifications. For example, you may need to specify resource managers available on a node.

For instructions on how to modify an environment definition, see Chapter 4, "Creating and Modifying Environment Definitions."

Environment definitions are important in both deployment and development environments. In a deployment environment, an environment definition contains application partitioning configuration information for all applications you load and install in your environment. In a development environment, the simulated environment definitions are used for partitioning and testing applications, and in making an application distributions.

Starting System Management Services

If you are installing the iPlanet UDS runtime system, iPlanet UDS automatically installs the components needed to run a Node Manager and run deployed iPlanet UDS applications.

If you are installing the iPlanet UDS application development software, you specify whether your target node will function as a central server, a regular server, or a client-only node. Depending on the option you select, the installation program will install the appropriate system management service, as shown in the following table.

Installation Option	System Management Service Installed
Central Server node	Environment Manager (with Name Service)
	Repository Server (development environments only)
Server node	Node Manager
Client–only node	Launch Server

The installation program stores the startup commands for these system management services. You can use these commands in your system startup files. iPlanet UDS places the startup commands in a different location on each platform. (See *iPlanet UDS System Installation Guide* for details.)

Client–only nodes Unlike servers, client-only nodes do not normally run a Node Manager service. However, if the Environment Console, Escript utility, or Launch Server is running on a client, these programs will provide a Node Manager service. In this case, the client node can be managed, just like a server node running a Node Manager, by iPlanet UDS system management tools.

Startup Sequence

As a rule, you install iPlanet UDS first on your central server node, then on other server nodes to ensure that your system management services are started in the correct sequence.

For your system management services to communicate properly, it is important that your system management services start in the following order:

1. Environment Manager, which automatically starts the Name Service.
2. Node Managers on all server nodes and Launch Servers on all client nodes on which iPlanet UDS has been installed.

If you wish to restart a single failed or stopped service, you can use the appropriate startup script to restore the service. The startup sequence does not matter.

If you wish to restart more than a single service, be aware of the sequence. For example, the Name Service (that is, the Environment Manager partition) must be running before other Node Manager services can be started.

For information about starting the Environment Manager, Node Managers, and Launch Servers, see "Starting System Management Services" on page 103 and "Starting the Launch Server" on page 304.

If you are setting up a development environment, you normally will start one or more Repository servers after all your server nodes are started. For information about starting Repository servers, see "Starting Central Repository Servers" on page 262.

Startup Commands (nodemgr and start_nodemgr)

Each of the system management services are started by starting the Node Manager partition on the host node. Since the Environment Manager service is really a more global Node Manager service, its startup command is the same as for a Node Manager, except it references an environment name. And since the Name Service resides in the same partition as the Environment Manager, it starts automatically when you start the Environment Manager.

iPlanet UDS provides scripts for starting the system management services. It is preferable to use the script, which calls the startup command, rather than simply executing the startup command because the script checks that the startup command executes successfully and, if so, directs output to a standard log file in a standard location. (You can also use the startup script within other scripts.)

The startup commands and the corresponding startup scripts are summarized below:

Portable Syntax

Service	Startup Command	Startup Script
Node Manager	nodemgr	start_nodemgr
Environment Manager (and Name Service)	nodemgr -e *environment_name*	start_nodemgr -e *environment_name*

OpenVMS Only

Service	Startup Command	Startup Script
Node Manager	VFORTE NODEMGR	NODEMGRSTART
Environment Manager (and Name Service)	VFORTE NODEMGR /ENVIRONMENT=*environment_name*	ENVMGRSTART /ENVIRONMENT=*environment_name*

Windows NT Only

Service	Startup Command	Startup Script
Node Manager	Node Manager icon	No scripts for the Windows NT platform
Environment Manager (and Name Service)	Environment Manager icon	

Each startup command (or script) reads a number of iPlanet UDS environment variables. You can set flags to override these environment variable settings. For example, you can override the default settings for the kind of messages logged by the runtime system and the amount of object memory allocated to a startup partition. The commands (and scripts) use the same syntax and flags, as shown below.

If you set environment variables that are used by the Node Manager for a particular node, you need to set the environment variables before you start the Node Manager; otherwise, the Node Manager does not pick up the environment variable values. Because the Environment Manager is a special kind of Node Manager, this fact is also true for the Environment Manager.

Portable (All Platforms)

{nodemgr | start_nodemgr}
[**-e** *environment_name* [**-b** *environment_definition_file*]]
[**-fns** *name_server_address*] [**-p** *master_password*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-fnd** *node_name*] [**-i** *initialization_file*]

OpenVMS

{VFORTE NODEMGR | NODEMGRSTART | ENVMGRSTART}
[/ENVIRONMENT=*environment_name*
[**/BOOT=***environment_definition_file*]]
[**/PASSWORD=***master_password*]
[**/NAMESERVER=***name_server_address*]
[**/LOGGER=***logger_flags*]
[**/MEMORY=***memory_flags*]
[**/STACK**=*integer]*
[**/NODE=***node_name*]
[**/INITIALIZATION_FILE=***initialization_file*]
[**/DETACH**]
[**/ERROR=***error_file*]
[**/OUTPUT=***output_file*]
[**/PROCESS_NAME=**"*process_name*"]
[**/UIC=***uic*]

The following tables explain each of the command line flags.

This Flag	Specifies
-e *environment_name* **/ENVIRONMENT=** *environment_name*	The node acts as the Environment Manager for the environment as well as the Node Manager for this node. The *environment_name* specifies the name of the environment that the Environment Manager will manage. The first time the Environment Manager starts, it creates an environment repository of the specified name, thereafter it accesses the repository.

This Flag	Specifies
-b *environment_definition_file* **/BOOT=** *environment_definition_file*	(Used only with **-e** or **/ENVIRONMENT** flag) The *environment_definition_file* specifies the environment definition file to be used with the Environment Manager. Usually, you only specify this option to recover an environment definition when the environment repository has been lost or corrupted.
-fns *name_server_address* **/NAMESERVER=***name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-p *master_password* **/PASSWORD=***master_password*	The new password for the environment repository. If no password existed before, this password is set for the repository. If a password did exist before, then this password replaces the previous password. This password is the password for the environment repository until the password is again changed using this flag.
-fl *logger_flags* **/LOGGER=***logger_flags*	The startup logger flags for the command. See "-fl Flag (Log Manager)" on page 371 for syntax information. This specification overrides the FORTE_LOGGER_SETUP environment variable setting. On UNIX, you must specify the setting in double quotes.
-fm *memory_flags* **/MEMORY=***memory_flags*	The startup object memory flags for the command. See "-fm Flag (Memory Manager)" on page 375 for syntax information. This specification overrides default platform-specific object memory allocation. On UNIX, you must specify the setting in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fnd *node_name* **/NODE=***node_name*	Simulates a non-existent node (for testing purposes only). Specifies a node name under whose identity to run the service. This specification overrides FORTE_NODENAME environment variable setting. Default is host name.

This Flag	Specifies
-i *initialization_file* **/INITIALIZATION_FILE=** *initialization_file*	An Escript initialization script to be executed by the Node Manager upon startup. See *Escript and System Agent Reference Guide*. The Escript script can only contain environment editing commands, and is used principally to specify resource managers for a node in the active environment definition. You should specify the initialization file only the first time you start the Node Manager. If you do not specify an Escript initialization file, the node is entered in the environment definition with only the node name and architecture specified. If a particular node is already specified in the environment definition, the information in the initialization file is ignored.

OpenVMS Flags

This Flag	Specifies
/DETACH	Starts the Node Manager as a detached process. By default, the Node Manager starts as a subprocess of the current process.
/ERROR	The error output file for the Node Manager when you start it as a detached process using the /DETACH qualifier. If you do not specify this qualifier, the default is FORTE_ROOT:[LOG]NODEMGR_*node*.ERR, where *node* is the local node name.
/OUTPUT	The output file for the Node Manager when you start it as a detached process using the /DETACH qualifier. If you do not specify this qualifier, the default is: FORTE_ROOT:[LOG]NODEMGR_*node*.LOG *node* is the local node name.
/PROCESS_NAME	The process name for the detached process in which the Node Manager runs when started using the /DETACH qualifier. If you do not specify this qualifier, the default is: "Forte_NODEMGR" (or "Forte_ENVMGR" for the Environment Manager).
/UIC	The UIC under which the Node Manager should run when it is started as a detached process using the /DETACH qualifier. UIC may be specified in either octal [nnn,mmm] or identifier format.

Log files for UNIX The UNIX `start_nodemgr` script copies the previous log file for the Node Manager as `a_`*nodeid*`.log` in the `FORTE_ROOT/log` directory before it starts a new log file named *nodeid*`.log` in the same directory so that the previous log is not immediately overwritten. If a `a_`*nodeid*`.log` file already exists, it is overwritten.

Process Names

When iPlanet UDS system management services start, they are given process names that correspond to their startup commands. These names are shown in the table below.

Service	UNIX Process Name	Windows NT Process Name	OpenVMS Process Name
Node Manager	nodemgr	nodemgr	Forte_NODEMGR
Environment Manager	nodemgr -e	envmgr	Forte_ENVMGR

You can check on a system management service by using operating system commands to look for the corresponding process name.

Platform	Commands for checking for running processes
UNIX	`# ps -auxww | grep nodemgr`
OpenVMS	`$ show system/output=` *filename* `$ search` *filename* `"forte_nodemgr"`
Windows NT	`ps | grep nodemgr`

On UNIX platforms, you use some variant of the `ps` command. Be sure you use a flag that will avoid truncating output before showing the process name.

On OpenVMS platforms, you use the show system command.

On Windows NT you can also use the Windows NT Task List window.

Startup Batch Files

You can use the iPlanet UDS system startup commands created by the iPlanet UDS installation program to start system management services on any node. These startup batch files include the appropriate system management service startup script for the node and are stored in the files shown below.

Platform	iPlanet UDS System Startup Command
UNIX	FORTE_ROOT/forteboot.sh (and.csh)
OpenVMS	SYS$STARTUP:FORTE_STARTUP_Vversion.COM
Windows NT	No batch files. Drag the Environment Manager icon into the Startup program folder. These services can also be started automatically as an NT service, as described in "Using Windows NT Services."

You usually place these iPlanet UDS startup commands in your system startup file so the Node Manager (or Environment Manager) is started up whenever you reboot your system. You can add additional startup commands to these batch files, for example, to start application partitions once the Node Manager has started up.

Using Windows NT Services

This section describes how you can run the Node Manager and Repository Server as services on the Windows NT platform.

Why Use Windows NT Services?

For Windows NT, a service is an executable object defined in the registry database maintained by the Service Control Manager. The executable file associated with a service can be started at boot time by a boot program or by the system, or it can be started on demand by the Service Control Manager. iPlanet UDS supports running Win32 services.

By default, when you install iPlanet UDS on a machine running Windows NT, the Node Manager or Environment Manager and the Repository Server can run only while the user who started these iPlanet UDS services is logged in. These processes shut down when this user logs off.

If a user with Administrator access installs iPlanet UDS so that you can start these iPlanet UDS services as a Windows NT service processes, these service processes can continue even after you log off. When a service is invoked by the Service Control Manager, the service process can be logged on as the default System account.

These NT services use environment variables that are set in the Windows NT registry for the Local Machine.

Controlling the Node Manager or Environment Manager and Repository Server Services

After you have installed the Node Manager or Environment Manager and Repository Server as Windows NT services, you can start and shut down the services using the Windows NT Service Control Panel.

The Node Manager is registered in the registry database as iPlanet UDS Node Manager. The Environment Manager is registered in the registry database as iPlanet UDS Environment Manager. The Repository Server is registered in the registry database as iPlanet UDS Repository Manager.

Because the Environment Manager and Node Manager run as Windows NT services, any `ftexec` processes started by the Environment Manager and Node Manager behave as though they are Windows NT services.

NOTE Any user whose user ID belongs to the User user group can start a service.

➤ **To get to the Services control panel**

1. In the Program Manager, double-click on the Control Panel icon in the Main program group.
2. In the Control Panel, double-click on the Services icon (gears). The Services control panel appears.

➤ **To start a service**

1. In the Services control panel, select the desired service.
2. If you want to override the default startup parameters, you can type the desired values in the Startup Parameter field.
3. Click the Start button. A dialog box with a clock appears, indicating that the service is starting.

➤ **To change the service configuration**

1. In the Services control panel, select the desired service.
2. Click the Startup button. A new window appears.
3. In the new window, you can set the following options:

Options	Descriptions
Startup Type	This field specifies how the service can be started. Select one of the following options: • Automatic—starts when the system is booted up • Manual—is started by the administrator • Disabled—cannot be started
Log On As	This field specifies the user ID that you want the service to use when it logs on. Leave this value set as "System Account", so that the service starts up using the user ID "System".

Only users in the Administrator user group can change the service configuration.

For more information about using the Windows NT Services control panel, see Windows NT online help.

➤ **To stop an NT service, do one of the following**

- Use the Environment Console or `Escript` to shut down these processes, as described in "Shutting Down System Management Services" on page 113 and "Stopping Central Repository Servers" on page 265.
- In the Services control panel in Windows NT, select the desired service and click the Stop button.

  Shutting down the Node Manager NT service directly using Windows NT utilities also shuts down all processes running on that node. If you shut down the Environment Manager NT service, you also shut down the entire environment.

Shutting Down System Management Services

When shutting down services in an iPlanet UDS environment, you can generally shut them down in any order.

➤ **To shut down all iPlanet UDS processes in the iPlanet UDS environment**

1. In the Environment Console, open the Node Outline view, by choosing the View > Node Outline command.
2. Select the Environment agent.
3. Choose the Component > Shut down command.
4. Choose OK.

➤ **To shut down a Node Manager and all iPlanet UDS processes running on that node**

1. In the Environment Console, open the Node view, by choosing the View > Node Outline command.
2. Select the Node agent for the node you want to shut down.
3. Choose the Component > Shutdown command.

Maintaining System Management Repositories

An environment repository maintains, among other things, information contained in the node repositories of all the individual nodes in an environment. As you modify application configuration information or as individual Node Managers perform work, such as installing partitions, information on these objects is not only maintained in the individual node repositories, but in the environment repository as well. In fact, iPlanet UDS attempts to keep the environment repository and node repositories synchronized at all times.

The benefit of this redundancy is that Node Managers do not have to interrogate a remote environment repository to perform local tasks. It also means that a node repository can be rebuilt automatically. For example, if a node repository becomes corrupted, you could shut down the Node Manager, delete the node repository, and then restart the Node Manager. A new node repository would be built by synchronizing with the environment repository.

However, if the environment repository becomes corrupted, or the node hosting the Environment Manager fails, the environment repository cannot be fully rebuilt from information in the node repositories. The node specifications can be rebuilt, but application partitioning information is lost.

For this reason, it is very important that you back up your environment repository, especially in a deployment environment, where a great deal of application partitioning configuration information may reside in your environment definition.

Backing up and Restoring Environment Repositories

There are two approaches to backing up your environment repository: directly backing up the environment repository files, or exporting the active environment definition. When you export the active environment definition, the Name Service database stored in the environment repository is not exported. Therefore, if you have static information in the Name Service database (for example, in the case of connected environments), you should always back up your environment repository by backing up the repository files, rather than by exporting the environment definition.

Backing up Repository Files

The environment repository is simply the node repository of the central server node (which hosts the Environment Manager). To back up the repository, back up the following two B-tree files:

- FORTE_ROOT/sysdata/envrepos/*node_name*.btd
- FORTE_ROOT/sysdata/envrepos/*node_name*.btx

If your environment repository gets corrupted, you can restore these files and restart the Environment Manager. See "Starting System Management Services" on page 103.

Exporting Environment Definitions

To export the active environment definition, you can use the Environment Console Export command under the File menu (or the Environment agent's `ExportEnv` Escript command). The environment definition is written to the file `FORTE_ROOT/envdist/environment_ID.edf`.

You can then back up the .edf file. If your environment repository is lost or corrupted, you restore this file and restart the Environment Manager using the **-b** flag followed by the name of the .edf file:

nodemgr -e *environment_name* **-b** *environment_ID*.**edf**

Connecting Environments

Up to this point, your iPlanet UDS environment—characterized by a single Name Service with a single Name Service address—has been regarded as an isolated group of nodes. The Name Service brokers communication between partitions executing on different nodes in the environment. As each partition (including each Node Manager partition) starts, it registers its service objects and network address with the Name Service, so that other partitions can find remote services and communicate with them. The Name Service defines a unique name space for the environment.

It is possible, however, to expand the name space of your environment to encompass other iPlanet UDS environments as well. Partitions within your environment can then find services in other iPlanet UDS environments and access them. Similarly, partitions within other environments can find and access services in your environment. This is most useful in failover scenarios where a backup service may be in another environment, or in scenarios where a reference partition is pointing to a service in another environment.

You can connect iPlanet UDS environments together into an expanded name space and perform a number of related tasks using the system management agent that manages the iPlanet UDS Name Service—the NameService agent.

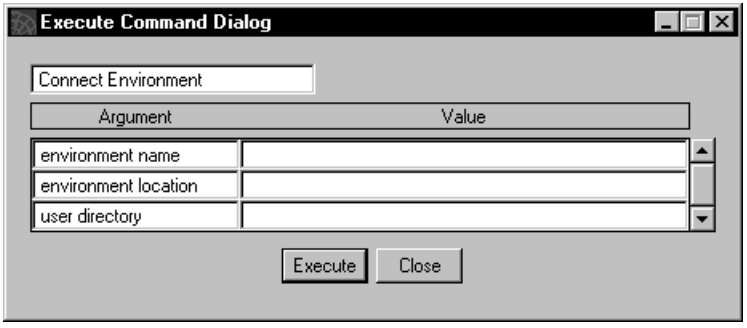
If you plan to connect environments, you should ensure that each environment name is unique so that developers and system managers can easily specify different environments in their search paths.

➤ **To connect an environment to your current iPlanet UDS environment**

1. Start the Environment Console.
2. Select the Application Outline view.
3. Select the NameService agent.
4. Select the Connect Environment command on the Utility menu.

   A Connect Environment dialog box appears.

5. Provide the name or UUID of the target environment to be connected (environment name) and its Name Service address (environment location) in the Execute Command dialog.



The user directory field is only required if your target environment contains explicitly registered services in its name space, as opposed to implicitly registered service objects. For information on explicit registration using the NameService agent, see *Escript and System Agent Reference Guide*. In this case, you need to provide a user directory name for the root of the directory tree containing the target environment's explicitly registered services.

Viewing Connected Environments

If you wish to see if some other environment is already connected to your environment, you can use the NameService agent's Show Administration command.

➤ **To see if an environment is connected to your environment**

1. Select the NameService agent in the Application Outline View.
2. Select the Show Administration command on the Utility menu.
3. Provide the name of the environment.

   If you do not provide the name of an environment, the command shows all environments directly connected to your environment, but not all environments indirectly connected to your environment.

This information is displayed in the trace window (if you do not have a trace window displayed, start the Environment Console with the **-fcons** flag).

Setting an Environment Search Path

When working with connected environments, you can specify an environment search path that specifies the order in which the Name Service will search the name space (the connected environments) to find a requested service. By default, this environment search path is set to "@(a)", which means the Name Service searches only in the current environment for a service, and tries to autostart the service if it is not already running.

Usually, your application services are located in your local environment. However, if a service is not available, or has failed, you might want to search the connected environments in a particular order for a backup service to use. You would then set a default environment search path accordingly.

Individual service object search paths The default environment search path can be overridden by an environment search path specified for any individual service object. A developer who has sufficient knowledge of the iPlanet UDS environment(s) in which a service object will be used can specify an environment search path property for the service object. For example, if a developer knows that a primary service will reside in one environment and a secondary backup will reside in another, then she can specify the appropriate search path for the service object. A service object's explicit environment search path takes precedence over the default environment search path. For information on specifying environment search paths for a service object, see *A Guide to the iPlanet UDS Workshops*.

➤ **To set a default environment search path**

1. Start the Environment Console.
2. Select the Application Outline view.
3. Select the NameService agent and open the NameService agent information window by double-clicking on the agent.
4. Double-click the EnvSearchPath instrument.

5. Specify a search path. Because most services are in your local environment, you should specify the current environment first, with the value "@(a)".

   Enter a string that includes one or more environment paths. The syntax of the search list string is:

   path [(a)] [: *path* [(a)]…

   where *path* is:

   @ | *@environment_name*

Variable	Definition
@	Indicates your local environment. This is normally specified as the first environment in the search path, since you would normally look first in your local environment for a service.
@environment_name	Indicates the name of an environment connected to your local environment. If two or more connected environments share the same environment name, and these environments are specified in the environment search path, then you must use the environment UUID to specify each environment. Specify this environment UUID in place of the *@environment_name*, for example B763E430-22FF-11D0-A5AA-5BC569EDAA77. For more information about the environment UUID, see the NameService agent in *Escript and System Agent Reference Guide*.
(a)	Indicates that a service identified by a specific path should be started up automatically, if necessary. For information on auto-start, see "Auto–Startup" on page 189.

   The following example illustrates a search list that looks first in the current environment, second in the "la" environment, and last in the "sf" environment:

   ```
   @(a):@la:@sf
   ```

For information on additional NameService agent commands, see *Escript and System Agent Reference Guide*.

How the Environment Search Path Can Affect Your Applications

You should use the environment search list very carefully, because it can have a large impact on the behavior of deployed applications.

Failover on service objects The default environment search list takes effect for *all* service objects in the environment that do not have their own environment search lists. If a service object in the current environment fails and it does not have its own search list, it will failover to the environment specified in the environment's default search list.

NOTE This failover occurs even if the service object is not defined as having failover.

Search path for reference partitions If a reference partition does not have its own environment search list when iPlanet UDS searches for the service that is referenced by the partition, iPlanet UDS searches the environment specified in the default environment search list.

For more information about service objects, reference partitions, and failover in connected environments, see *A Guide to the iPlanet UDS Workshops*.

Fault Tolerance for Multiple Connected Environments

This section discusses how the iPlanet UDS runtime system enables iPlanet UDS applications to be highly available and fault tolerant.

This section refers to the combined Name Server and Environment Manager for a given environment as the Environment Manager.

Name space iPlanet UDS enables you to define multiple environments, each of which defines a *name space*. The name space contains addressing information for the named objects, such as service objects and named anchored objects, which are managed by the Environment Manager for a given environment.

Global name space You can connect several environments to create a *global name space* in which applications and the Environment Manager in one environment know about named objects in other environments that the Environment Manager is connected to. Each Environment Manager manages the part of this global name space corresponding to its environment. Any iPlanet UDS partition in these connected environments can access service objects in any part of the global name space. Each Environment Manager manages all named objects created by the servers directly connected to it.

Home environment A partition can connect to one Environment Manager as part of an environment, then disconnect and connect to another Environment Manager as part of another environment, and so forth. The environment to which a partition initially connects is considered the partition's *home environment*.

Environment Manager Failover

When you use connected environments, you can make your applications more available by ensuring that an Environment Manager for one environment can locate and access all the named objects in the other connected environments. With this capability, if the Environment Manager for a named object's environment goes down, a partition that tries to access that named object can ask one of the other Environment Managers to locate the named object.

When a partition first accesses a named object, the partition stores the addressing information for all currently running instances of the named object that it has located using the environment search path. The Environment Manager for the partition's environment also stores this addressing information in its name service database.

By copying this addressing information between environments, iPlanet UDS ensures that applications in one environment can access named objects in another connected environment, even when the Environment Manager is not available in the named object's environment.

Therefore, to make sure that all Environment Managers in your connected environments have addressing information about all named objects available throughout the global name space, you should "warm up" your environment, as described in "Preparing Environment Managers to Access Named Objects in Other Environments" on page 121.

Preparing Environment Managers to Access Named Objects in Other Environments

To make sure that all Environment Managers in your connected environments have addressing information about all named objects available throughout the global name space, you should "warm up" your environment.

NOTE If the service objects in your environment reference other service objects, you need to be aware of issues described in iPlanet UDS Technical Note 10660. This section describes a very simple case in which service objects do not reference other service objects.

To "warm up" the Environment Managers of the connected environments, write a routine to run in each environment. Each routine performs the following steps:

1. Start a partition that contacts all of the named objects in the other environments in the global name space.
2. Wait for the Environment Manager to commit the copied object addressing information to the name server database at the end of the routine.

   The Environment Manager commits the copied addressing information to the disk only at a fixed interval—1 second by default. The "warm-up" routine must wait this interval to ensure that all of the object addressing information will be persistently stored and available if the Environment Manager fails and is restarted.

➤ **To run these routines**

1. Start all of the servers in all environments. You can perform this step using the Environment Console or with an Escript script.
2. Start the routine in each environment.

Environment Manager Failover for Partitions

iPlanet UDS lets partitions specify multiple name service addresses, so that they can failover to other Environment Managers in connected environments if the Environment Managers in their home environments fail.

Partitions can specify multiple Environment Manager locations in the FORTE_NS_ADDRESS environment variable or on the **-fns** flag of most iPlanet UDS commands.

NOTE To make sure that all Environment Managers in your connected environments have addressing information about all named objects available throughout the global name space, you should "warm up" your environment, as described in "Preparing Environment Managers to Access Named Objects in Other Environments" on page 121.

The syntax for specifying multiple Environment Manager locations is:

address[**::***protocol_name*][**;** *address*[**::***protocol_name*] . . .]

The syntax of *address* is protocol dependent.

Protocol	Address Syntax
TCP/IP	machine_name:port_number
DECnet	*machine_name*:*object_name*
Unix Domain	path_name

The optional *protocol_name* specifies a protocol other than the default for that platform. The default protocol for OpenVMS is DECnet and for all other platforms is TCP/IP.

The *protocol_name* is one of the values:

```
TCP/IP
DECnet
Unix Domain
```

The following is an example of how you could specify multiple Environment Manager locations in UNIX:

```
setenv FORTE_NS_ADDRESS 'Test1:5000;BackupUNIX:6000;LocalTest:1000'
```

When a partition initially starts, it connects to the first active Environment Manager on the list. This Environment Manager becomes the home Environment Manager.

If the partition loses its connection to its home Environment Manager, iPlanet UDS automatically connects the partition to the next active Environment Manager on the list. When the home Environment Manager again becomes available, it automatically connects to the partition and the partition drops its connection to the backup Environment Manager. While the partition is not connected to its home Environment Manager, it cannot create named objects using the ObjectLocationMgr RegisterObject method.

For more information about the **-fns** flag on the iPlanet UDS `ftexec` command, see "Manual Startup" on page 190.

Environment Manager and Lost Partition Information

When an Environment Manager unexpectedly can no longer access a partition, it considers the partition to be lost. To deal with these lost partitions, the Environment Manager can either:

- automatically delete information about the lost partition and its named objects
- retain information about the lost partition and named objects until the system manager explicitly requests that all information about all lost partitions and their named objects be deleted

Deleting information about lost partitions requires no manual maintenance, however, it might provide lower availability when certain types of communication failures occur. Retaining information about lost partitions provides higher availability, but requires the system manager to manually remove obsolete information from the name service database.

You can select whether the Environment Manager automatically deletes information about lost partitions and their named objects by setting the value of the DeleteOnCommFailure instrument of the NameService agent to `TRUE` or `FALSE`. The default value is `TRUE`.

If the Environment Manager is shut down normally within the Environment Console or `Escript` by using the Shutdown command on the Environment Manager, all information related to object names and partitions is removed.

The system manager can use the `RemoveLostParts` command on the NameService agent to delete information about lost partitions from the name service database.

For more information about the DeleteOnCommFailure instrument and `RemoveLostParts` command of the NameService agent, see *Escript and System Agent Reference Guide*.

Chapter 4

Creating and Modifying Environment Definitions

An environment definition specifies all the nodes in an iPlanet UDS environment as well as other environment properties.

In many situations, following the setup of your environment, you will need to add more detail to the active environment definition created automatically by iPlanet UDS's system management services.

Simulated environment definitions have special significance in a development environment, where they are used by developers to partition, test and make an application distribution for a deployment environment other than the active environment definition.

This chapter explains how to create and modify environment definitions using the Environment Console, and covers the following topics:

- creating a new environment definition
- saving and exporting an environment definition
- modifying an environment definition

All procedures in this chapter will assume you are using the Environment Console. You can use the equivalent Escript commands to perform the same tasks. See *Escript and System Agent Reference Guide*.

Introductory Concepts

An environment definition is a key component of both deployment and development environments. The environment definition specifies all the objects in an environment's system management domain and is stored in the environment repository on your central server node.

In particular, an environment definition specifies all the nodes in an iPlanet UDS environment. When you set up an iPlanet UDS environment, as described in Chapter 3, "Setting up and Maintaining an iPlanet UDS Environment," your system management services automatically create an environment definition for your active environment. As each Node Manager starts up, the Environment Manager dynamically adds a specification for that new node to the environment definition created when you first started up the Environment Manager.

In many situations, following the installation of iPlanet UDS on each node in your environment, you will need to add more detail to your node specifications. For example, you may need to specify resource managers available on a node. You may also wish to represent large numbers of similar client nodes with a single *model node*.

In addition, if you have loaded or installed applications (or library distributions) in the environment, the environment definition also includes specifications of the application (or library distribution), logical partition, and installed partition objects, as well.

In a development environment, environment definitions have special significance. An environment definition is used when developers partition, test and make an application distribution for some deployment environment. In performing such tasks, developers need to be able to *simulate* the target deployment environment.

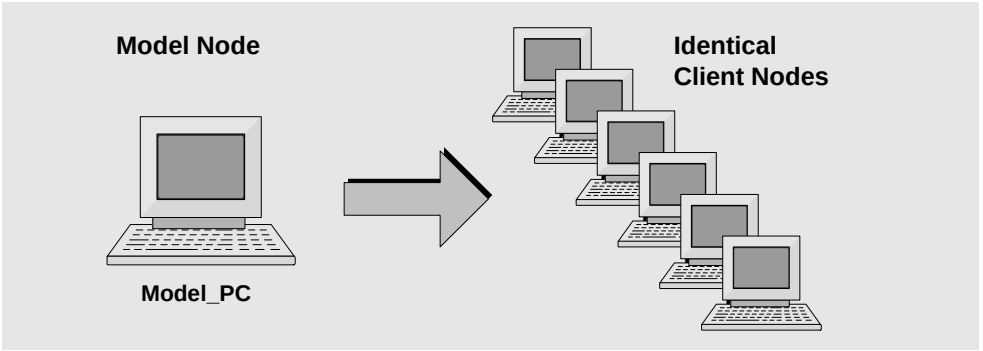
Model nodes and *simulated deployment environments* are discussed in a bit more detail in the following sections.

Model Nodes

In any environment definition, you can represent a large number of identical client nodes as a single model node. Any partition that is assigned to the model node can then be installed and executed on any of the individual nodes represented by the model node.

You normally use model nodes to represent large numbers of client nodes with the same architecture. Instead of specifying each of the client nodes separately in your environment definition, you can use one specification to represent them all. Each of the nodes must run the same operating system.

Figure 4-1 Model Nodes

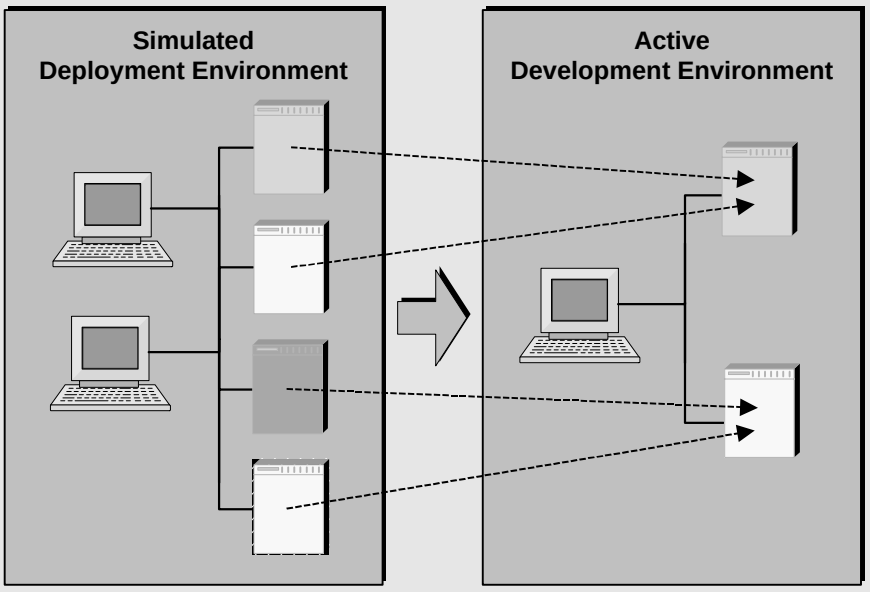


One problem with using model nodes is that iPlanet UDS system management functions cannot distinguish between the nodes represented by the model node. For example, when an application is being installed in an environment with model nodes, iPlanet UDS has no way of knowing which of the individual nodes have been successfully installed. This disadvantage to using model nodes is largely mitigated by using the `Launch Server` utility, which automatically downloads needed applications onto client nodes. For more information, see "Installing an Application" on page 170 and "Deploying Applications to Client Nodes" on page 298.

Simulated Deployment Environments

In iPlanet UDS, you can simulate a deployment environment by using nodes in your development environment to simulate the nodes in the deployment environment. Developers can use this simulated environment definition in the Partition Workshop or Fscript to configure an application and make a distribution for a particular environment definition. To simulate an environment, you create a simulated deployment environment definition and assign every node in the simulated deployment environment to a testing node in your active (development) environment, as shown in Figure 4-2.

Figure 4-2 Relationship between Simulated and Active Environments



Each node in the simulated deployment environment is mapped to a testing node in the development environment.

Each node in the active development environment can serve as a testing node for one or more nodes in the simulated deployment environment.

You can use the Environment Console or `Escript` utility to create a simulated deployment environment definition.

➤ **To create a simulated deployment environment definition**

1. Create a new environment definition, as described in "Creating a New Simulated Environment Definition" on page 129.
2. Specify the active (development) environment as a test environment for the simulated deployment environment.

   See "Specifying New Environment Properties" on page 129.
3. Specify the nodes in your simulated deployment environment—each node should correspond to a node in the actual deployment environment.

4. Map each node in your simulated environment to a test node in the active (development) environment.

   See "Specifying Node Properties" on page 134.

These steps assume you are creating your environment definition from scratch. However, you can eliminate steps 1 and 3 by importing an environment definition that has been exported from a deployment environment (see "Exporting an Environment Definition" on page 141) or by running an Escript script created in the deployment environment that specifies all the nodes in the deployment environment, as described in *Escript and System Agent Reference Guide*. If you use these shortcuts, you still need to perform the mapping in steps 2 and 4, so that developers can test the applications they've partitioned using the deployment environment definition.

Procedures for steps 1–4 are found in the following sections of this chapter.

Creating a New Simulated Environment Definition

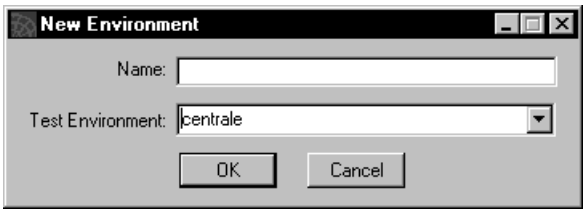
To create a simulated deployment environment definition, you create a new environment definition and specify the nodes for that environment.

Specifying New Environment Properties

To create a new environment definition, start up the Environment Console as described in Chapter 4, "Creating and Modifying Environment Definitions." Choose the File > New command. The Environment Console displays the New Environment Properties dialog, where you specify the properties of your new environment.

The figure below shows the New Environment Properties dialog.

Figure 4-3 New Environment Dialog



Fill in the two properties of the New Environment dialog as follows:

Property	Description
Name	You use the Name field to name the new environment definition. The name should match that of your target deployment environment. After you have created the new environment definition, you cannot modify its name.
Test Environment	You use the test environment drop list to select your active environment as the test environment for the new environment. Your active environment must contain adequate node resources to simulate your new environment. For example, if your simulated environment definition contains an OpenVMS server node running Oracle DBMS, your active environment must also contain a server node running a database. If you want to test platform-specific configurations, such as particular databases or compiled partitions, your active environment should contain the same types of nodes as your simulated environment.

When you complete the fields in the New Environment dialog, and click OK, the Environment Console opens two windows: an Environment Definition window and the Node Template window.

Adding a Node to an Environment Definition

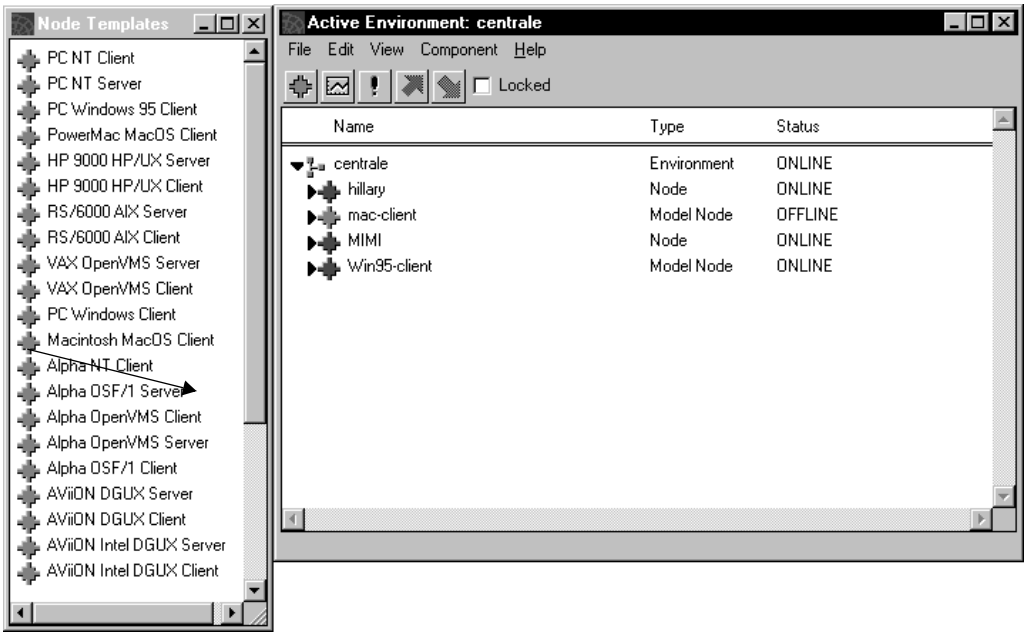
After you have entered the appropriate information and closed the New Environment Properties dialog, the Environment Console opens an Environment Definition window and a Node Template window. (Both the Node Template window and Environment Definition window are also displayed when you open an existing environment definition to modify it.)

The Environment Definition window is a blank slate where you can specify new nodes. The Node Template window provides templates representing the node architectures that iPlanet UDS supports.

To add a new node to a simulated environment definition, you drag a template from the Node Template window into the main viewing panel of the Environment Definition window, as shown in Figure 4-4.

You can add a new node to your active environment definition, as well, by selecting the File > Node Template command. The Environment Console displays the Node Template window and you can drag node templates into the main viewing panel of the active environment window. (Normally, you do not add a new node in this way, since it is automatically added when the Node Manager for the new node is started up.)

Figure 4-4 Specifying a Node in an Environment Definition



You can open as many environment definition windows as you wish, and exchange node definitions among them by dragging node template icons from one environment definition window to another.

Node Templates

Node templates are what you use to specify new nodes. They represent the node architectures that iPlanet UDS supports, as defined in the following table.

Template Node Name	Description
Alpha NT Client	Alpha client node running Windows NT.
Alpha NT Server	Alpha server node running Windows NT.
Alpha OpenVMS Client	Alpha client node running OpenVMS.
Alpha OpenVMS Server	Alpha server node running OpenVMS.
Alpha OSF/1 Client	Alpha client node running Digital UNIX.
Alpha OSF/1 Server	Alpha server node running Digital UNIX.
Aviion Intel DGUX Client	Aviion Intel client node running DG/UX.
Aviion Intel DGUX Server	Aviion Intel server node running DG/UX.
HP 9000 HP/UX Client	HP 9000 PA-RISC client node running HP/UX.
HP 9000 HP/UX Server	HP 9000 PA-RISC server node running HP/UX.
Mips SINIX Client	Mips client node running SINIX.
Mips SINIX Server	Mips server node running SINIX.
PC Windows 95 Client	Intel PC running Windows 95.
PC NT Client	Intel PC client running Windows NT.
PC NT Server	Intel PC server running Windows NT.
RS/6000 AIX Client	RS/6000 client node running AIX.
RS/6000 AIX Server	RS/6000 server node running AIX.
Sequent DYNIX/ptx V4 Client	Sequent client node running DYNIX/ptx Version 4.
Sequent DYNIX/ptx V4 Server	Sequent server node running DYNIX/ptx Version 4.
SPARC Solaris Client	SPARC client node running Solaris.
SPARC Solaris Server	SPARC server node running Solaris.
VAX OpenVMS Client	VAX client node running OpenVMS.
VAX OpenVMS Server	VAX server node running OpenVMS.

Upgrading existing environment definitions The following templates are no longer supported, and you should upgrade these templates to the new templates. You should not define new nodes that use these templates.

iPlanet UDS Release 2 Template Node Name	Description	New Template Name
AViiON DGUX Client	Aviion Motorola client node running DG/UX.	AViiON Intel DGUX Client
AViiON DGUX Server	Aviion Motorola server node running DG/UX.	AViiON Intel DGUX Server
Sequent DYNIX/ptx Client	Sequent client node running DYNIX/ptx (Version prior to Version 4).	Sequent DYNIX/ptx V4 Client
Sequent DYNIX/ptx Server	Sequent server node running DYNIX/ptx (Version prior to Version 4).	Sequent DYNIX/ptx V4 Server

A word about locking environment definitions After you create a new simulated environment definition, the definition is open to public access by other users in the active environment. For this reason, when you begin specifying a new environment in an Environment Definition window, the Environment Console locks the definition, giving you exclusive access to it. If you unlock the definition for any reason, you must lock it again before attempting any changes.

When you create a new simulated environment definition, you have a lock on that definition. You cannot obtain a lock for and change any other environment definition until you release the lock on the new environment definition.

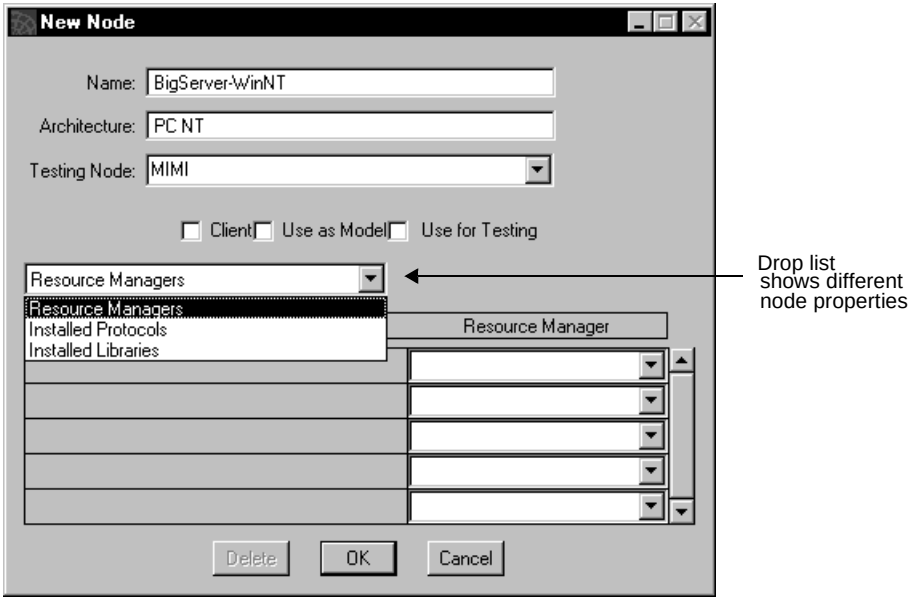
For more information on acquiring, holding, and releasing locks on active environments and on environment definitions, and on the consequences of locking them, see "Locking Environment Definitions" on page 143.

Specifying Node Properties

To specify the properties of a node in an environment definition, you modify the properties dialog for that node. When you add a new node to an environment definition or when you wish to modify the properties of an existing node, the Environment Console opens a Node Properties dialog for the node, Figure 4-5.

To add a node to an environment definition, see "Adding a Node to an Environment Definition" on page 130. To modify an existing node specification, select the node in the Environment Definition window (or Active Environment window) and choose File > Properties.

Figure 4-5 Node Properties Dialog



The properties specified in the top portion of the Node Properties dialog are as follows:

Property	Description
Name	You must enter a node name (for example,“eliot”) in the Name field. The name should represent either the name of an existing node or, in the case of a model node, the name of the model group. In this case, the node name can be of any length. The names of nodes should be unique for the first 8 characters. After you have defined the name of the node and selected the OK button, you cannot change the name of the node. To rename a node, you need to copy the existing node and give the new node a different name, then delete the original node.
Architecture	A read-only field that displays the architecture of the node that you are defining (for example, RS/6000 AIX).
Testing Node	Specifies the node in the active (test) environment that will be used to simulate this node in a simulated deployment environment. This field is relevant only when you are specifying a node in a simulated deployment environment. You can designate a node in your active (development) environment to serve as a testing node for a node in a simulated environment. Select a testing node in your active (development) environment from the drop list or simply enter the name of the node (for example,“coach”). To ensure reliable tests, the testing node should match the architecture of the node that you are specifying.
Client	If the Client toggle field is checked, the node is a client node. If the Client toggle field is not checked, the node is designated as a server-only node. Client architectures (Windows 95 and Windows NT nodes) are automatically specified as client nodes (Client field is checked and cannot be changed). You can designate server architectures as client nodes if you wish. This setting only affects iPlanet UDS’s default partitioning—in addition to server partitions, a client partition will also be assigned to the node. You can manually repartition the application to add or remove the client partition from the node. (In other words, you can change this property from server-only, to server and client, and the other way around, without resorting to using a new node template.)

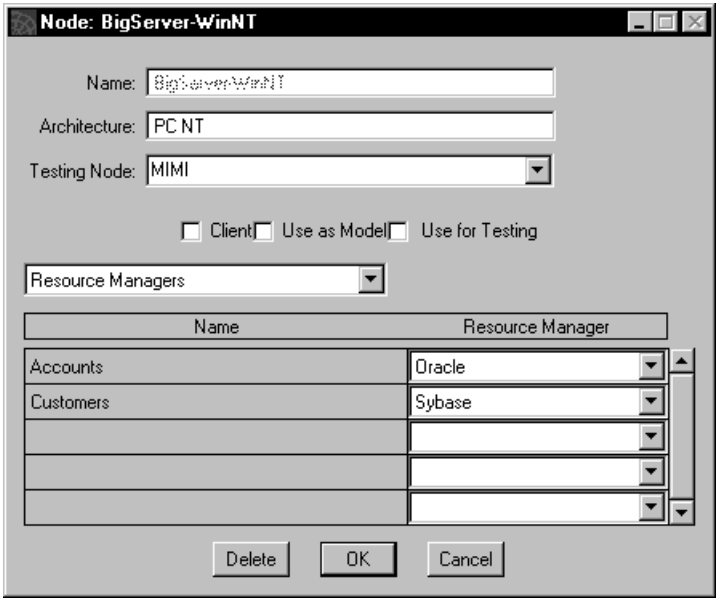
Property	Description
Use as Model	Check the Use as Model toggle field if you want the current node to be a model node that represents any number of client nodes. For example, instead of specifying 100 PC nodes, you can specify one model node to represent all of them. To set a client node to use a model node definition, you set the client node's FORTE_MODELNODE environment variable to the name of the model node. See Appendix B, "iPlanet UDS Environment Variables," for information on environment variables.
Use for Testing	In an active (development) environment, check the Use for Testing toggle field if you want the current node to be used as a testing node for simulated environments. The node name will then appear in the drop list of the Testing Node field when specifying a node in a simulated deployment environment.

In the bottom portion of the Node Properties dialog, you use the drop-down list to control the settings for each node property—available resource managers, communication protocols, and installed libraries. When you select a property from the list, the appropriate array field appears for you to complete.

Resource Managers

You use the Resource Managers array field in the Node Properties dialog to register resource managers into a node's definition. A resource manager is a database management system available on the computer for which you are defining a node.

Figure 4-6 Resource Manager Property on the Node Properties Dialog



The iPlanet UDS development system uses this information to properly partition iPlanet UDS applications, placing DBResourceMgr and DBSession service objects onto nodes that can support them.

When you specify a database resource manager, the name you associate with the database manager is the name you will use to reference this resource manager in your TOOL code. You can assign this resource manager a name meaningful for your environment.

Valid values are:

Resource Manager Type	Description
DB2	DB2/6000 database system.
Informix	Informix database system.
ODBC	ODBC access to any supported database system.
Oracle	Oracle Version 8 database system.
Rdb	Rdb database system.
Sybase	Sybase database system.

The resource manager that you specify must be supported for the architecture of the current node. See the platform matrix at `bhttp://www.forte.com/support/platforms.html` for a current list of supported resource managers for each architecture type.

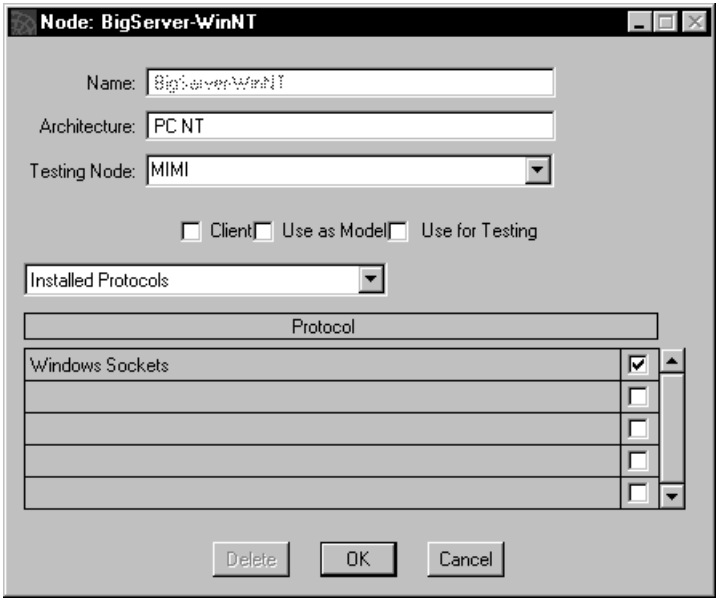
For more information about setting up database resource managers, see *Accessing Databases*.

Installed Protocols

You use the Installed Protocols array field to determine what communications protocols the node can employ to communicate with other iPlanet UDS nodes, and thus participate in iPlanet UDS environments.

As shown in Figure 4-7, the Installed Protocols array field provides a list of iPlanet UDS-supported communications protocols appropriate to the architecture of the node you are defining.

Figure 4-7 Installed Protocols Property on the Node Properties Dialog



iPlanet UDS supports the following communications protocols:

Protocol Name	Description
Berkeley Sockets	Standard Berkeley socket library. This is the standard TCP/IP interface on most UNIX systems.
Digital DECnet	Digital DECnet protocol for VMS.
Digital UCX	Digital TCP/IP protocol for VMS.
Pathworks DECnet	Digital Pathworks DECnet protocol.
Pathworks TCP/IP	Digital Pathworks TCP/IP protocol.
PC-NFS	PC/NFS protocol for MS/Windows.
TLI	TCP/IP TLI protocol for UNIX systems.
UNIX Domain Sockets	Berkeley socket library for interprocess communication on a single node. This is available on most UNIX systems.
Windows Sockets	Protocols on MS/Windows that support the Windows Sockets interface.

The protocol that you specify must be supported for the architecture of the current node. See the *Release Notes* for a current list of supported protocols for each architecture type.

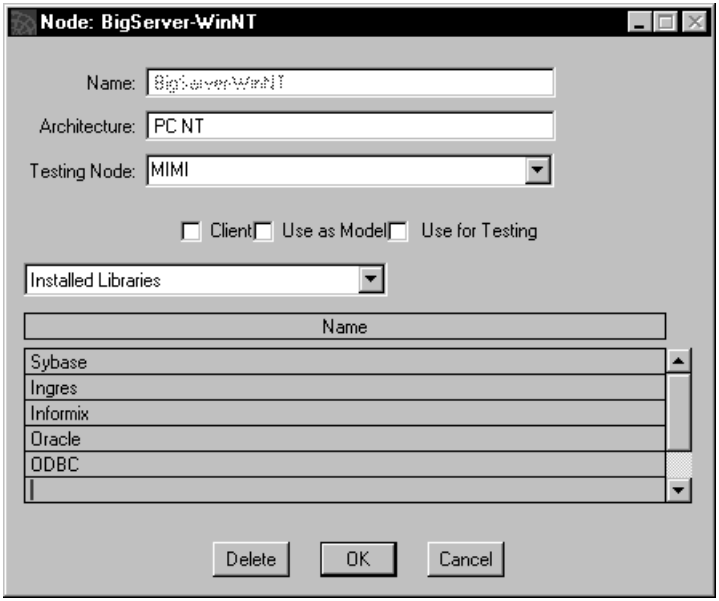
The default node definition includes support for all the protocols iPlanet UDS can use on the node's architecture. The more protocols a node supports, the greater its processing burden in listening for iPlanet UDS environment and application communications. As you modify a node definition, you can designate support for only so many communications protocols as you wish the node to use.

Installed Libraries

You can add libraries that are installed on each node in this array.

The iPlanet UDS application development system uses information in the Installed Libraries array field to partition iPlanet UDS applications referencing restricted libraries.

Figure 4-8 Installed Libraries Property on the Node Properties Dialog



In the case of libraries for C projects, the array field specifies the C project libraries that are installed on the node. For more information on making library distributions, see *A Guide to the iPlanet UDS Workshops*. For information on installing library distributions, see "Deploying a Library Distribution" on page 177.

In the case of OLE libraries, you have to explicitly register the name of the appropriate interface project in the array field. (For more information, see "Support For OLE" on page 325.)

Saving and Exporting an Environment Definition

After you specify the nodes in an environment definition, you can save the definition by choosing the File > Save Environment command. Alternatively, you can close the Environment Definition (or Active Environment) window, and the Environment Console prompts you to save the environment definition.

The Environment Console saves the environment definition to the active environment's environment repository.

Exporting an Environment Definition

You would normally export a simulated environment definition or your active (development) environment definition for backup purposes.

In a deployment environment, you might export your active environment to send to developers to use in partitioning and testing an application being developed for your site. In this situation, the system manager at the development site would import the environment definition into the development environment repository. The environment definition would then be modified to make it a simulated environment definition; the Test Environment property would be set to the active (development) environment, and the Test Node property of each node would be mapped to a node in the active environment.

➤ **To export an environment definition from the environment repository**

1. Select the File > Export Environment command.

   iPlanet UDS exports the environment definition into a file (.edf) and stores it in the "`envdist`" directory in the iPlanet UDS system software directory.

For example, the following example shows the UNIX path syntax for an environment definition "ClassEnv" that is stored in the file "`classenv.edf`":

```
FORTE_ROOT/envdist/classenv.edf
```

Importing an Environment Definition

In the Environment Console, you can import environment definition files in the Active Environment window using the File > Import Environment command.

Modifying an Environment Definition

You can make modifications to any environment definition, whether it be your active environment or a simulated environment.

You can make any of the following changes to your environment definition:

- modify environment properties
- add nodes or change the properties of an existing node
- copy a node specification
- delete a node specification

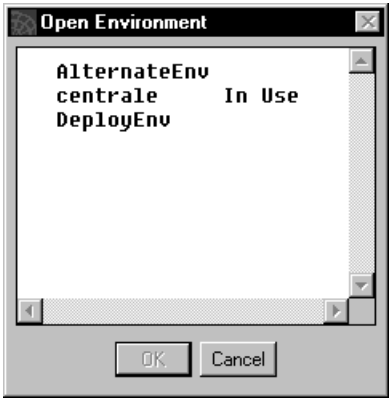
To modify an environment definition, you have to open the environment definition and lock it to prevent the possibility of others from trying to make modifications at the same time.

Opening an Environment Definition

To open an environment definition that you want to modify, choose the File > Open command. The Environment Console displays an Environment Definition Selection window that contains a list of environment definitions stored in the environment repository.

The list shows the existing environment definitions and indicates which are currently in use (locked). Environment definitions can be in use by either a developer who is configuring an application using that environment definition, or by another system manager who is changing the environment definition. Select a definition and click the OK button.

Figure 4-9 The Environment Definition Selection Window



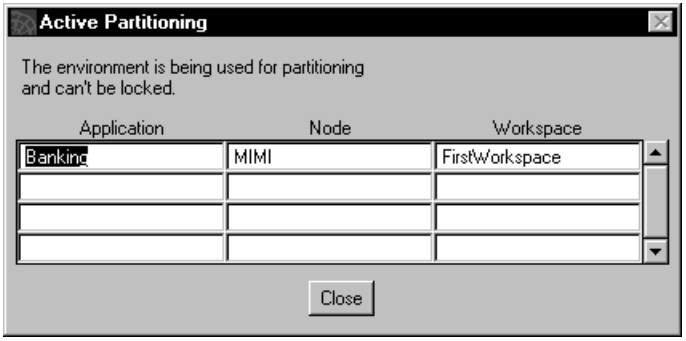
Locking Environment Definitions

You must place an exclusive lock on an environment definition (including the active environment definition) before you can make modifications to it. When you lock an environment definition, it is unavailable for changes by other users; they can only open and view the definition while you have it locked. In addition, in the case of an active environment, no one can partition applications, or load or install them while you have the environment definition locked. Therefore you should release the lock as soon as possible.

This lock affects only your ability to change the properties of the environment and nodes in the environment. This lock does not affect your ability or the ability of another user to load, install, uninstall, or delete an application.

Should you attempt to lock a definition already locked by another user, the Environment Console posts a notice, shown in Figure 4-10, that the environment is locked, and also posts a notice to the user holding the lock that another user wishes to obtain it.

Figure 4-10 Environment Locking Notice



To lock and unlock an environment definition, use the File > Lock Environment and File > Unlock Environment commands (or the toggle in the Tool Bar).

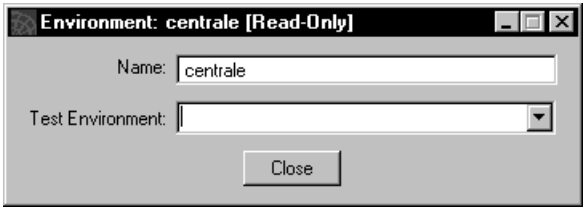
The Lock Environment command explicitly locks an environment definition (including the active environment definition). If an environment definition is locked, the Lock Environment command is dimmed, and the Unlock Environment command is enabled.

Modifying Environment Properties

The File > Properties command opens the properties dialog for an active environment, or for an environment definition. The properties dialog shows the properties of the environment object: its name, test environment (if a simulated environment definition), log filename, and logging properties (if an active environment definition).

The figure below shows an Environment Properties dialog for an active environment called “DocEnv.”

Figure 4-11 Environment Properties Dialog



In this example, the properties dialog offers only a Close button, meaning that you can only inspect the environment’s properties because it was locked by another user when the dialog was opened.

If you were to open the properties dialog for an environment definition, you could change any of the following fields:

Field	Description
Test Environment	For simulated environment, shows what environment is used as a test environment.

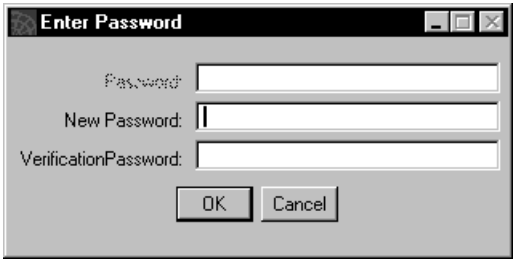
NOTE You cannot change the name of an environment or environment definition after the environment or environment definition has been created.

Setting and Using Passwords for an Environment

This section explains how you can set a password on any environment and require that password when a user starts the Environment Console or `Escript`.

➤ **To set a password on an environment**

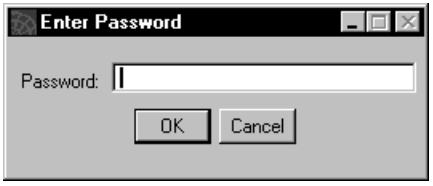
1. Lock the environment by selecting the Locked toggle on the tool bar.
2. Select the File > Set Password command.
3. Enter the current password, then the new password twice to verify the password.



4. Click the OK button.
5. Unlock the environment by selecting the Locked toggle on the tool bar.

➤ **To specify the password for an environment**

1. When the Environment Console starts, it displays the Enter Password dialog, where you enter the password for the environment.



2. Enter the password and click the OK button.

Replacing the Password for an Environment Repository

If the password for an environment repository is lost, iPlanet UDS provides a **-p** flag on the `nodemgr` command that lets you reset the password for the environment repository.

The syntax for the `nodemgr` command is described in "Startup Commands (nodemgr and start_nodemgr)" on page 104.

Modifying Node Properties

The Properties… command (on the Component menu) opens the properties dialog for a selected node. In addition to modifying individual node properties on this properties dialog, you can copy a node specification or delete a node specification, as described below.

Copying a Node Specification

You can copy a node definition between any two environment definitions. To copy a node specification, select the node icon you wish to copy and drag it into the Environment Definition window of the target environment definition.

Make sure that the environment to which you are copying is locked.

Deleting a Node from an Environment Definition

➤ **To delete a node specification from an environment definition**

1. Lock the environment definition.
2. Select the node you wish to delete
3. Choose the Edit > Delete command.

   The Environment Console posts a warning that deleting the node causes the repartitioning of application configurations that are dependent on that node.
4. Click the OK button to delete the node definition.

In an active environment, you cannot delete a node that is running a Node Manager. You have to shut down the Node Manager first.

Deleting an Environment Definition

➤ **To delete an environment definition**

1. Choose the File > Delete… command to display an Environment Definition Selection window containing the list of environment definitions stored in the active environment's environment repository.

2. Select a definition from the list.
3. Click the window's OK button.

You cannot delete the active environment definition, or an environment definition that is held in a lock by another user.

Deploying iPlanet UDS Applications

One of the most common system management tasks is deploying application (or library) distributions in your environment. This chapter provides background on the structure of distributions and on the procedures for deploying them.

The chapter covers the following:

- background on application distributions
- deploying an application distribution
- deploying a library distribution
- upgrading an application

For information on creating an application distribution, see *A Guide to the iPlanet UDS Workshops*.

All procedures in this chapter will assume you are using the Environment Console. You can use the equivalent Escript commands to perform the same tasks. See *Escript and System Agent Reference Guide* for information about using `Escript` and system agent commands.

About Application and Library Distributions

An *application distribution* is a collection of files that are needed to deploy a partitioned application in an iPlanet UDS environment. An application distribution contains the components of an application as well as files describing the node assignments and structure of the application.

While a developer generally makes an application distribution after partitioning and testing a logical application, you, as a system manager, need to understand enough about making a distribution to successfully deploy one (or possibly package one).

A *library distribution* is a collection of files that are needed to deploy a library configuration in an iPlanet UDS environment. A library distribution contains the component libraries as well as files describing the node assignments for any restricted libraries.

Making an Application Distribution

A developer makes an application distribution using the File > Make Distribution command of the Partition Workshop or the MakeAppDistrib command of the `Fscript` utility. The Make Distribution command uses a two-stage process for making an application distribution.

In the first stage, the Make Distribution command creates a number of files and places them in a standard location in the iPlanet UDS directory structure. The command creates a directory path, as shown in Figure 5-1 on page 152, using the iPlanet UDS environment name, application name, and compatibility level number, respectively.

environment_ID/*distribution_ID*/**cl#**

(Note that for the OpenVMS operating system, levels in the directory structure are collapsed, creating a single directory: *environment_ID_distribution_ID*_**cl#**.)

The Make Distribution command also creates the .adf file and the contents of the `codegen` and `generic` directories (see the table below).

Name	Purpose
—.ace	File which maps iPlanet UDS application component names to unique identifier names
—.adf	Application distribution file. Contains information about the application (partitioning) configuration—what partitions are assigned to what nodes—for use in the deployment process. (The .adf extension is added to the application's unique identifier name.)
generic	Directory that contains all portable files. The partition directories below the generic directory contain all the standard partitions.
codegen	Directory that contains all source files used for generating C++ compiled partitions.
platform1…	Directories that contain non-portable compiled partitions for each supported platform.

Name	Purpose
appgbl	Directory in which you or iPlanet UDS can place files that would be installed along with any partition (generic directory) or set of partitions assigned to a node (platform directories). For example, this file is used for message files that are used for internationalization.
—.btd, —.btx	Standard partition image repositories. (The .btd and .btx extensions are added to the partition's unique identifier name.)
—.pgf	Partition generation file. Contains the source code used to make compiled partitions for a single logical partition. (The .pgf extension is added to the partition's unique identifier name.)
—.exe, …	Compiled partition executable file.(The name is platform dependent and based on partition's unique identifier name.)

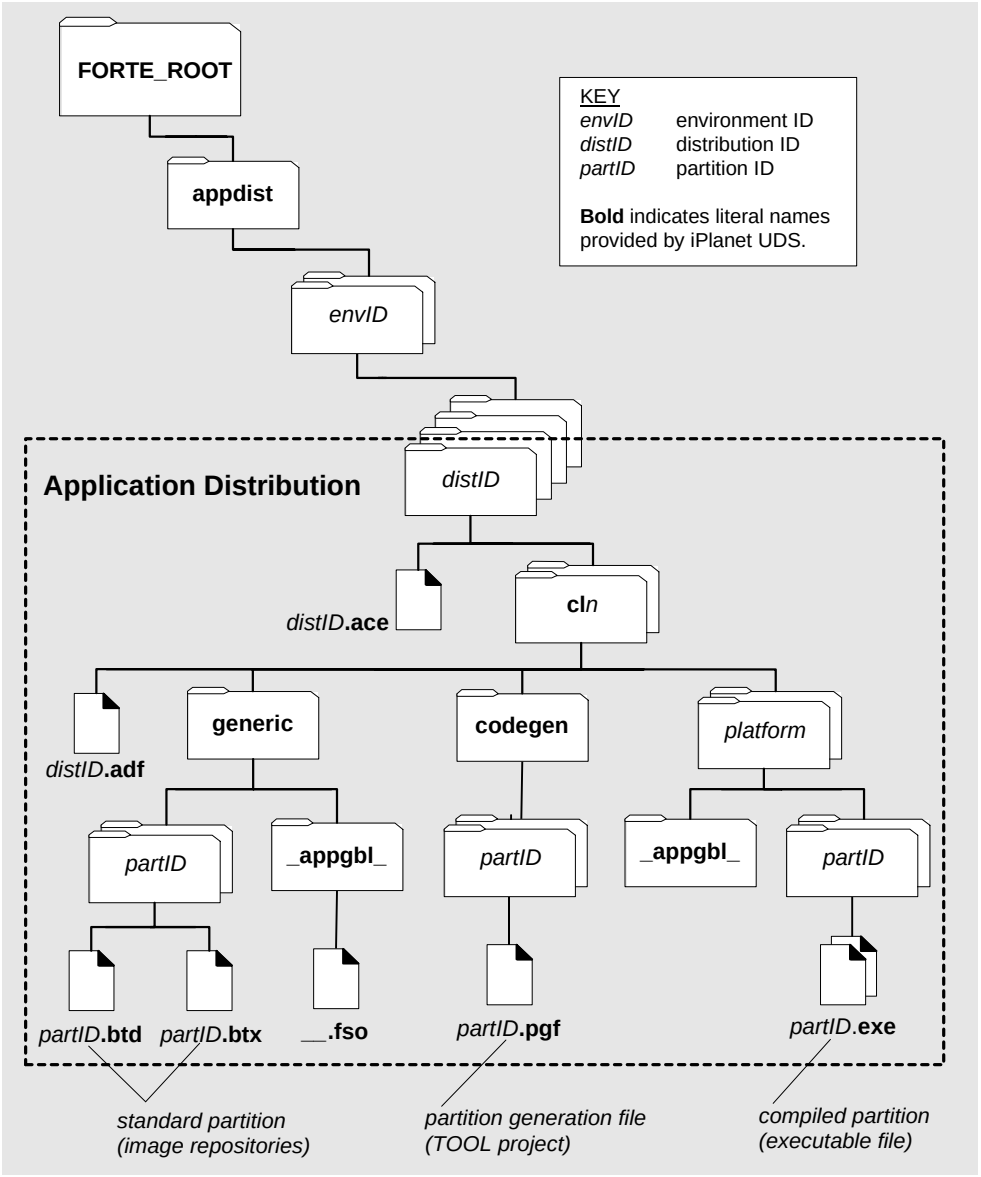
If an application partitioning configuration contains only standard partitions, then no.pgf files are created. If the configuration contains partitions marked as compiled, there is a .pgf file created for each compiled partition.

In the second stage, the Make Distribution command automates the steps needed to make compiled partitions and place them in the application distribution directory structure. These steps involve generating portable C++ source code for each .pgf file, transferring this source code to the node architectures for which the partition needs to be compiled, compiling and linking the source on those nodes, and then transferring the compiled partitions back to the appropriate location, shown in Figure 5-1, in the distribution directory structure.

Code generation This process involves two server applications: CodeGenerationSvc, which generates the C++ source code, and CompilerSvc, which performs the compiling and linking. As a system manager in a development environment, one of your responsibilities is to make sure these two server applications are properly configured and installed to perform these tasks, given the node architectures and user load in your development environment. For more information, see "Auto-Compile Services" on page 315.

The second stage of Make Distribution is optional—a developer can choose to perform all these steps by hand, using some command line utilities provided by iPlanet UDS. For more information, see *A Guide to the iPlanet UDS Workshops*.

Figure 5-1 Application Distribution Directory Structure



Making a Library Distribution

The Make Distribution command is used to make library distributions in the same way application distributions are made. Developers will typically make library distributions when their applications require a particular set of libraries for execution. A developer may also make a library distribution for sale or for use by other developers in some remote development site.

As with application distributions, The Make Distribution command uses a two-stage process for making a library distribution. In the first stage, the Make Distribution command creates a number of files and places them in a standard location in the iPlanet UDS directory structure. The command creates a directory path, as shown in Figure 5-2 on page 155, using the iPlanet UDS environment name, library configuration name, and compatibility level number, respectively.

environment_ID/*distribution_ID*/**cl#**

(Note that for the OpenVMS operating system, levels in the directory structure are collapsed, creating a single directory: *environment_ID_distribution_ID*_**cl#**.)

The Make Distribution command also creates the .adf file and the contents of the `generic` and `codegen` directories (see the table below).

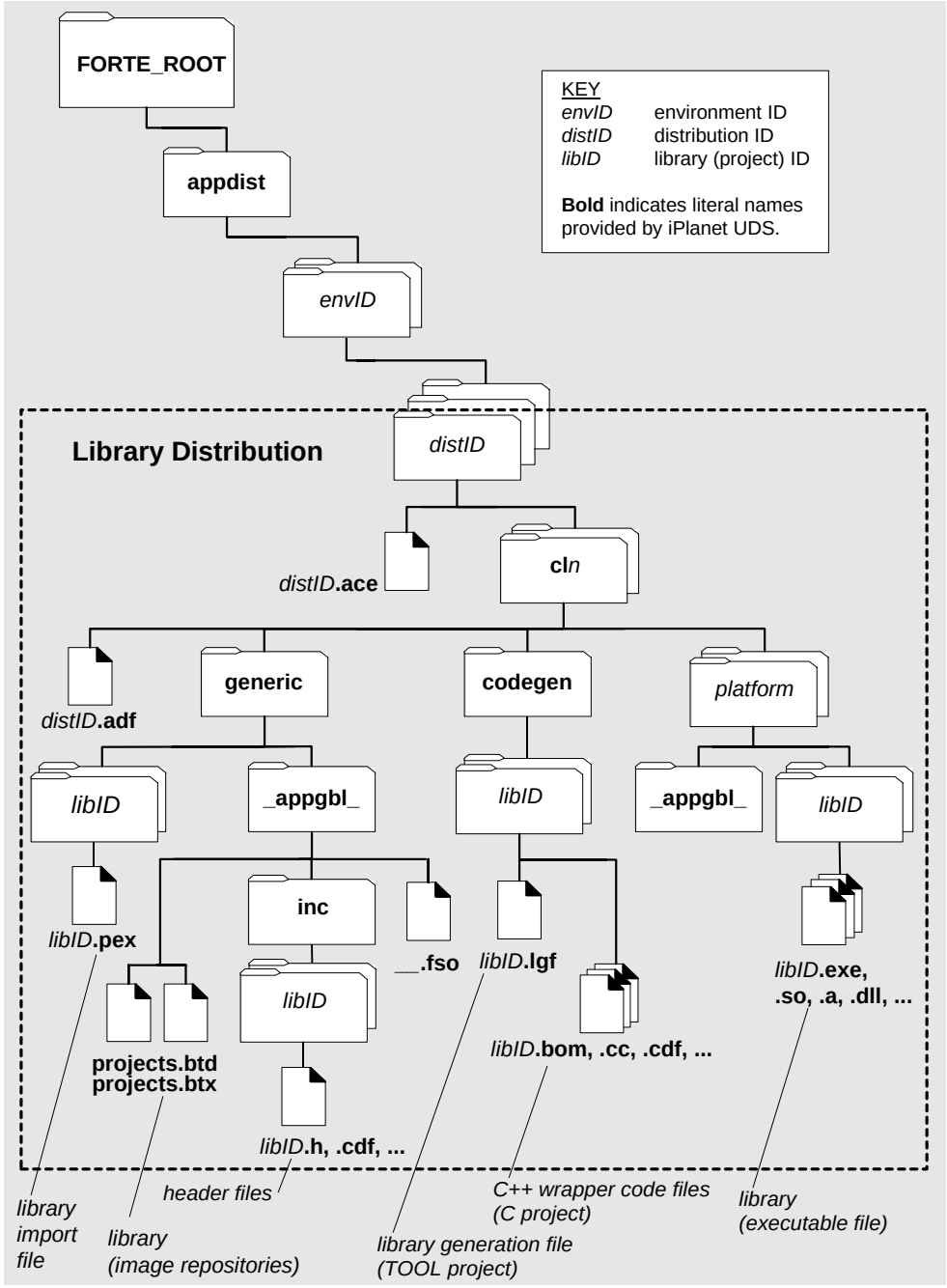
Name	Purpose
—.ace	File which maps iPlanet UDS library configuration component names to unique identifier names.
—.adf	Application distribution file. Contains information about the library (partitioning) configuration—what libraries are assigned to what nodes—for use in the deployment process. (The .adf extension is added to the library configuration's unique identifier name.)
generic	Directory that contains all portable files. The partition directories below the generic directory contain all the standard partitions.
codegen	Directory that contains all source files used for generating C++ compiled partitions.
platform1…	Directories that contain non-portable compiled partitions for each supported platform.
appgbl	Directory in which you or iPlanet UDS can place files that would be installed along with any library or set of libraries assigned to a node.

Name	Purpose
projects.btd, projects.btx	Standard image repository to be used on platforms for which no compiled libraries have been made.
—.fso	File that maps method names to ids, which might be used for some external interfaces or compiled partitions.
—.h, —.cdf, …	Header files
—.pex	Export file used to import the library into development repositories in a target development environment.
—.lgf	Library generation file. Contains the source code used to make compiled libraries for a single project. (The .lgf extension is added to the library's unique identifier name.)
—.bom, —.cc, —.cdf, …	C++ wrapper files used as source code for making compiled libraries for a single C project.
—.so, —.exe, —.a, —.dll…	Compiled library file. (The name is platform dependent and based on library's unique identifier name.)

If a library configuration contains TOOL projects (which may reference any number of system libraries), then there is a .lgf file created for each such TOOL project. If the configuration contains C projects, then a .cc and .cdf and possibly a number of other files are created for each such C project (the .bom file lists them all).

In the second stage, the Make Distribution command automates the steps needed to make compiled libraries and place them in the library distribution directory structure, in the same way as it does for partitions in an application distribution. For more information, see "Auto-Compile Services" on page 315.

Figure 5-2 Library Distribution Directory Structure



Naming Conventions

In creating distribution directories and files, iPlanet UDS uses unique identifier names derived from iPlanet UDS component names. These identifier names preserve uniqueness while conforming to the most limiting of platform naming conventions.

Environment's unique identifier The first 8 characters of its iPlanet UDS environment name. If an environment of the same eight-character identifier name already exists, iPlanet UDS replaces the eighth character with an integer, starting at 1, and increments the number to create additional like identifiers.

Distribution's unique identifier The first 8 characters of its iPlanet UDS main project name. If a distribution of the same eight-character identifier name already exists in the same environment, iPlanet UDS replaces the eighth character with an integer, starting at 1, and increments the number to create additional like identifiers. A distribution's unique identifier is used to build the distribution directory name, .adf file name, and .pex file name (if any).

Partition's unique identifier The first 6 characters of its iPlanet UDS partition name plus the iPlanet UDS-generated partition number. A partition's unique identifier is used to build the .pgf file name, image repository name, and compiled partition name.

Library's unique identifier By default, this is the first 8 characters of its iPlanet UDS main project name (or library name for a C project). A library's unique identifier is used to build the .lgf file name, C++ wrapper file names, and compiled library name. You can override the default naming convention and create your own library names using the Fscript command, `SetAppId`.

Naming Conventions Example

An example is provided below for an application and/or library distribution whose main iPlanet UDS project is called Security_Trading and whose compatibility level is "#."

Component	iPlanet UDS Name	Unique Identifier Name
Environment (**i**th similar name)	deployment_environment_**i**	deploym**i**
Distribution (**j**th similar name)	security_trading_**j**_cl#	securit**j**
Partition (**m**th partition)	security_trading_**j**_cl#_part**m**	securi**m**
Library	libraryname (C project) projectname (TOOL project)	libraryn projectn

Packaging an Application Distribution

Once an application distribution is complete, it is ready to be packaged, along with any library distributions that must accompany it, for deployment.

A distribution for a compatibility level "#" is packaged for deployment by copying the distribution_ID directory—and any files it contains—plus the `cl#` directory, and everything contained below it, to a distribution medium such as a magnetic tape.

You can copy this branch of the directory structure using the `tar` command on UNIX platforms and the `backup` command on OpenVMS.

Any library distributions needed for the application to execute must also be packaged and included with the application distribution.

Installing Additional Files with Your Application Distribution

When you prepare to install an application or library, you can include additional files, such as help files or release notes. You need to put the additional files in specific subdirectories of the `FORTE_ROOT/appdist` directory. The exact location of these files depends on your answers to the following questions:

- What platforms will these additional files be installed on?
- What components of the distribution (partitions or libraries) do these additional files belong with?

After you have added the files to the appropriate location in the application distribution directory, iPlanet UDS's installation program can automatically install these files in the appropriate location along with the application or library distribution.

The following table explains where to place files that you want to add to the application or library distribution for installation, depending on where you want the additional files installed. The directory locations are all under the following directory:

`FORTE_ROOT/appdist/`*`environment_ID/distribution_ID`*`/cl#`

	All Components	**Specific Component**
All Platforms	/generic/_appgbl_	**/generic/***component_ID*
Specific Platform	**/***platform_ID***/_appgbl_**	**/***platform_ID***/***component_ID*

platform_ID represents the iPlanet UDS identifier for a platform, for example, IBM_AIX represents AIX.

component_ID represents one of the following:

- a partition ID for a partition if the distribution is an application distribution
- a library ID if the distribution is a library distribution

For more information about the directory structures of application and library distributions, see "About Application and Library Distributions" on page 149.

For example, suppose you have an application in the MyEnv environment whose distribution ID is myapplic. If you have release notes that you want to have automatically installed with all partitions on all platforms, you should place the file in the following directory:

```
FORTE_ROOT/appdist/MyEnv/myapplic/cl0/generic/_appgbl_
```

Documenting a Distribution

Every application distribution should be accompanied by sufficient documentation to enable you, as an iPlanet UDS system manager, to deploy the application in your environment and to troubleshoot runtime problems.

The following list covers the most important information that should be documented by developers for an application distribution:

- the library distributions required for the application to execute (if any)
- the function each partition performs
- the node architecture(s) each compiled partition requires
- the external resources each partition requires, by exact name
- the restricted libraries each partition requires (if any)
- the external object system (OLE, for example) each partition requires (if any)
- the object memory allocation each partition requires
- the logger flags that should be used in troubleshooting the application
- the names and purpose of any files the developer chose to place in any _appgbl_ directory
- any environment variables that must be defined for partitions of the application
- environment search paths that have been set for any service objects in the application
- whether the application or library is compatible with any previous releases of the application or library

Deploying an Application Distribution

The process of deploying an iPlanet UDS application in your target deployment environment consists of four basic steps, which are described in this section.

➤ **To deploy an application distribution**

1. Transfer the application distribution to a server node in your deployment environment.
2. Load the application distribution into your deployment environment's environment repository.

   You can use the Environment Console or `Escript` utility to do this.
3. Modify the application's partitioning configuration, if you wish.

   You can assign partitions to additional nodes and modify a number of assigned partition properties.
4. Install the application in your deployment environment.

   iPlanet UDS automatically downloads each partition to the node (or nodes) in your environment on which it is supposed to run.

Before you try to run your newly installed application, make sure that any library distributions required by the application have also been deployed (see "Deploying a Library Distribution" on page 177) and that any reference partitions required by the application have been started.

The four deployment steps will be discussed, in turn, in the following sections.

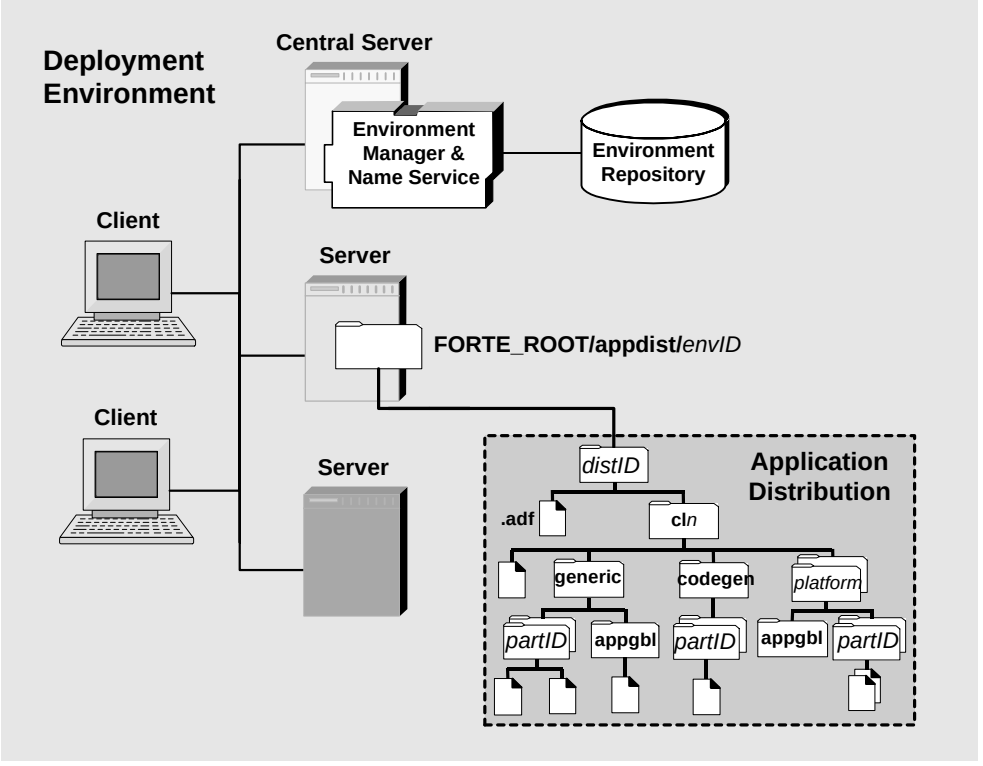
Transferring a Distribution to a Deployment Environment

Transferring a distribution to an iPlanet UDS deployment environment consists of copying the contents of the *distribution_ID* directory, and the required `cl#` directory structure beneath it, to any iPlanet UDS server node you wish in your deployment environment. Do not copy the distribution to a client node and try to load the distribution; the Environment Manager cannot locate a distribution on a client node. You can use whatever copying technique you prefer (such as network copying utilities or copying onto tape or diskette).

Copy the distribution, as shown in Figure 5-3, to the following iPlanet UDS directory structure on the selected server node:

`FORTE_ROOT/appdist/`*`environment_ID`*

Figure 5-3 Transferring Distribution to Deployment Environment



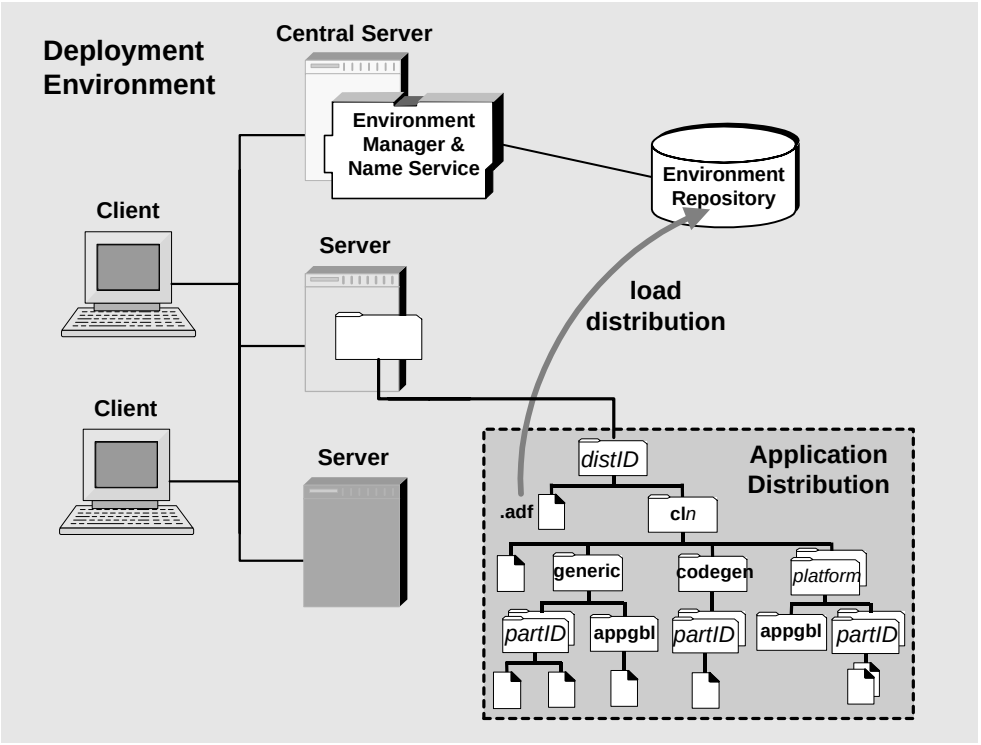
You may have to first create the *environment_ID* directory, using the first eight characters of your deployment environment name, before you can copy the application distribution. On OpenVMS platforms, use the directory structure shown in Figure 5-1 on page 152.

Loading a Distribution into an Environment Repository

When you load a distribution, you are loading the information contained in the distribution's .adf file into your deployment environment's environment repository, as illustrated in Figure 5-4.

The .adf file contains all the information needed by the Environment Manager in order to install all partitions onto the appropriate nodes in your deployment environment. For example, the .adf file contains instructions on where both standard and compiled partitions should be installed.

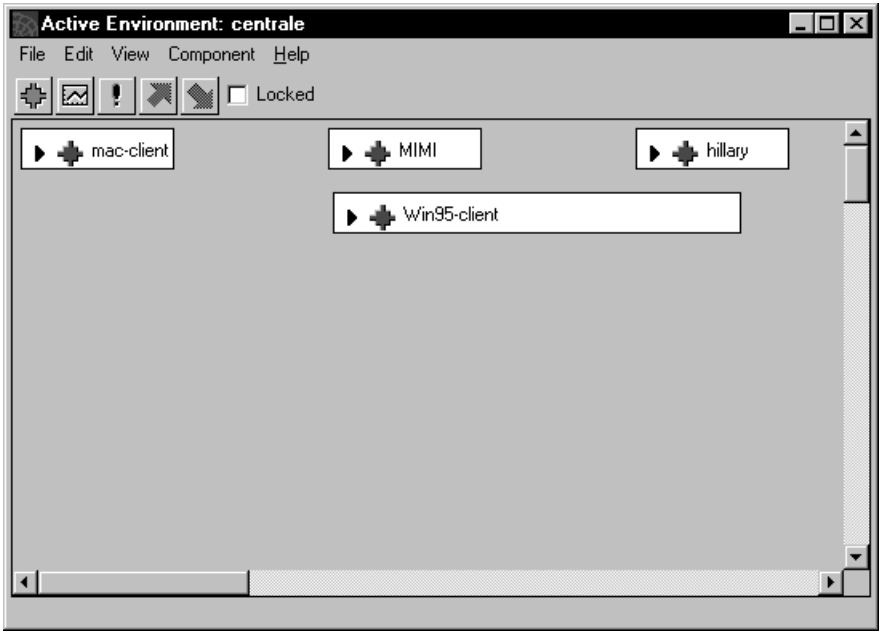
Figure 5-4 Loading Distribution into Environment Repository



➤ **To load a distribution**

1. Make sure the distribution has been transferred to the deployment environment.
2. Open the Environment Console.

   The Active Environment window appears.



In our example, the active environment, “centrale,” consists of four nodes: a model node for Windows 95 clients (Win95-client), a model node for Macintosh (mac-client), a Windows NT server node (MIMI), and a UNIX server node (hillary).

3. Select the File > Load Distribution command.

   The Load Distribution dialog appears.

   In the case of a local distribution, the window shows the `FORTE_ROOT/appdist` directory for your local node. If you select a remote node from the droplist, the window shows the `FORTE_ROOT/appdist` directory for the remote node. You can select a distribution to load from the displayed list.

   In our example, the distribution resides on "hillary," the central server node for the "DocEnv" environment. In the figure above, the TimeIt distribution is the only distribution on the node "hillary." Within the `TimeIt` directory represented in the window, there is a single release of the TimeIt application distribution, "cl0," indicating that this is the first release of the application (compatibility level is 0).

4. Select the distribution in the Load Distribution window, and click Load.

You can also load a distribution using `Escript`, as described in *Escript and System Agent Reference Guide*.

When a Distribution Conflicts with an Installed Application

If you are loading a new distribution for an application that is already installed in the environment, you might receive an error message that says that the application distribution you are loading conflicts with the installed application of the same name and release number. This message means that the configuration of the new distribution is different from the configuration of the installed application, which makes them incompatible.

Before you can load and reinstall the new distribution of the application, you need to uninstall the application currently installed in the environment.

iPlanet UDS determines whether two configurations of an application are compatible based on the following rules:

- For client applications and server applications, the new configuration must have the same number of partitions, and each partition must contain exactly the same service objects.
- For library distributions, the new release must have the same number of libraries and the names and UUIDs (universally unique identifiers) of each library must match exactly. The UUID of a library is taken from the UUID of its associated project. This UUID is assigned when the project is created in your repository.

  To ensure that a given project has the same UUID, even when it is exported and imported, the application developer must export the project with their UUIDs. To export projects with UUIDs, use the Fscript `ExportPlan` command with the `ids` argument. For information about the `ExportPlan` command, see *Fscript Reference Guide*.

Modifying a Partitioning Configuration

After loading an application distribution, you can view and modify its partitioning configuration using the Environment Console or `Escript` utility. You would normally modify a configuration for either of two reasons:

- The simulated environment used by developers for partitioning the application may differ from your current deployment environment—a very likely case if you have made recent changes to your environment or the application was developed by an independent software vendor (ISV) or value added reseller (VAR) who had no knowledge of your particular environment.
- You anticipate performance or other problems in the partitioning configuration created by the application developers and therefore want to modify it.

You can modify any of the partitioning configuration properties normally specified in the right-hand panel of the iPlanet UDS Partition Workshop. (You *cannot* modify the logical partitioning scheme or change the replication properties of service objects.)

You can modify partition assignments and a number of assigned (installed) partition properties as described in the following sections.

Partition Assignments

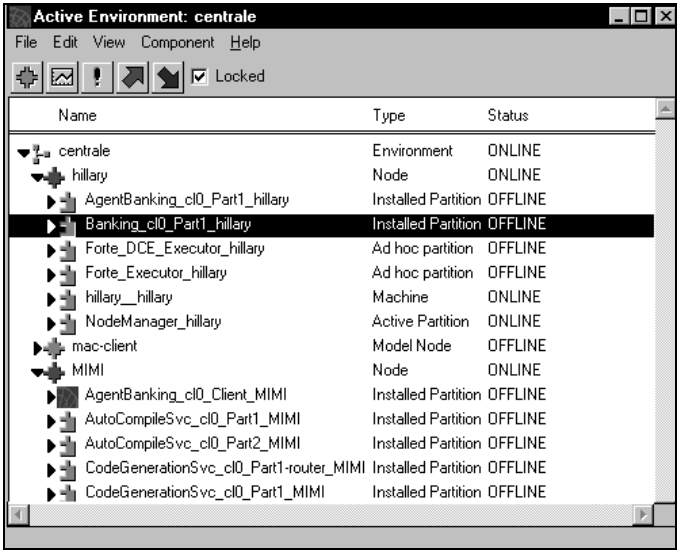
You can reassign partitions to different nodes, make additional assignments of replicated partitions, or remove assignments. You can only reassign compiled partitions, however, to nodes corresponding to the architectures for which the partitions were compiled.

When an application distribution is loaded into your environment repository, iPlanet UDS checks the partitioning assignments. If any partition is assigned to a node not found in your environment, that assignment will be dropped. If all assignments for a logical partition are dropped, iPlanet UDS will perform a default configuration (just as it does in the Partition Workshop), assigning the partition to a node in the environment. In this case, you may have to manually assign (or re-assign) the partition to a node (or nodes) in your environment. In assigning the partition, make sure that the target node has the platform architecture and resources required for the partition to run.

You cannot completely install an application unless every logical partition has been assigned to at least one node in your environment.

➤ **To reassign a partition**

1. Select the Node Outline view in the Active Environment window.
2. Expand the node to which the partition is currently assigned.



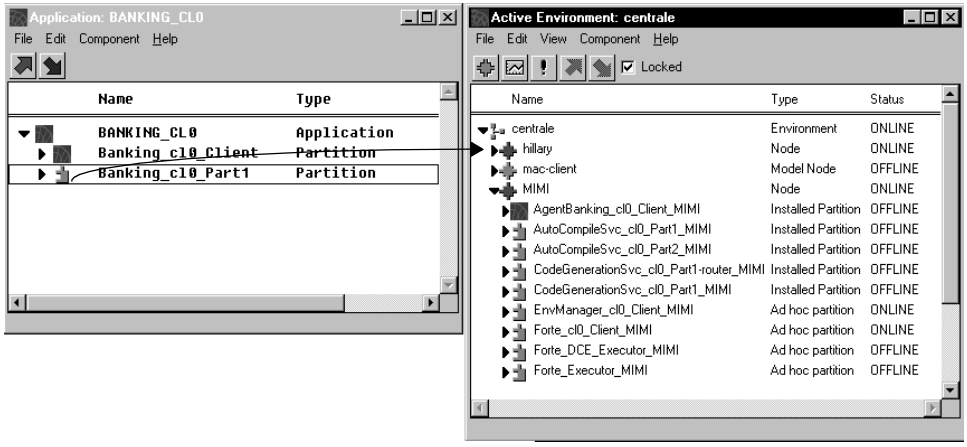
3. Drag the partition you wish to reassign and drop it on the node to which you wish to assign it.

➤ **To copy a partition assignment**

1. Select the Node Outline view in the Active Environment window.
2. Expand the node to which the partition is currently assigned.
3. Select the partition you wish to copy and copy it to the clipboard by selecting the Edit > Copy command.
4. Select the node to which you want to assign the partition and paste the partition from the clipboard by selecting the Edit > Paste command.

➤ **To assign an unassigned partition**

1. Select the Application Outline view in the Active Environment window.
2. Open the Application Agent window and expand the application in that window.



3. Select the Node Outline view in the Active Environment window.
4. Drag the logical partition you wish to assign from the Application Agent window and drop it on the node to which you wish to assign it.

Installed or Assigned Partition Properties

You can change some of the properties of partitions (described below) either:

- after you have loaded the application distribution, but before you have installed it

  You can change these partition properties before you install the application.

- after you have installed the application

  You can change these properties of the partition after you have installed the application. However, in this case, you need to reinstall the application after you change the partition's properties.

You can set the following properties for installed partitions:

Compiled You can deselect this checkbox if you decide to run a partition in interpreted mode rather than as a compiled partition. You can also select this checkbox to run a partition in compiled rather than interpreted mode, but only if the partition has already been compiled for the target node's architecture.

Disabled For replicated partitions, you can modify whether or not a partition is automatically started (enabled) on each node to which it is assigned by selecting or deselecting this checkbox. Every partition must be enabled on at least one node. If you disable a partition assigned to a node, it is not automatically started on that node.

Thread Package For server partitions only, you can specify whether the partition should run using DCE/POSIX threads or iPlanet UDS threads.

Replication Count For replicated partitions only, you can modify the number of replicates that are available for load balancing or failover.

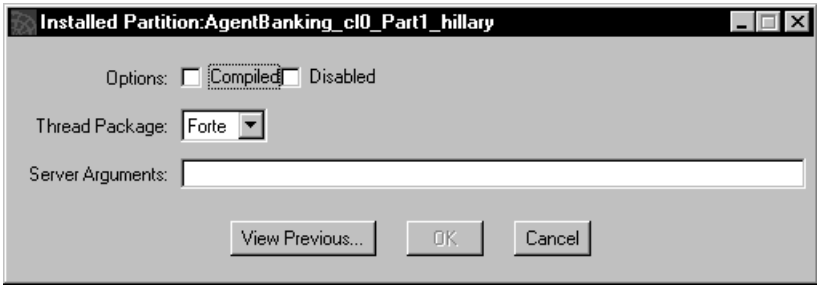
Server Arguments You can set flags which alter the default object memory space and logger settings set for a partition on startup.

The Installed Partition Properties dialog also displays the type of thread package that this partition uses. However, you cannot change this value in the Environment Console. You can only change the thread package specification for a partition in the Partition Workshop or in `Fscript`. For more information about thread packages, see *A Guide to the iPlanet UDS Workshops*.

➤ **To set properties of an assigned or installed partition**

1. In the Environment Console, choose the View > Node Outline command in the Active Environment window.
2. Expand the node to which the partition is currently assigned.
3. Select the partition and then choose the Component > Properties… command.

   The Installed Partition Properties dialog opens.



4. Change any of the fields described above and click OK.
5. Install the loaded application or reinstall the installed application that contains the partitions with the changed properties.

Before you reinstall an installed application to implement changes in its partition properties, you can compare the new property values for a partition with those for the currently installed partition. To see the properties for the currently installed partition, select the View Previous... button on the Installed Partition Properties dialog.

Installing an Application

Once you have loaded your application distribution into the environment repository and made any changes in the partitioning configuration that you want, you are ready to install your application into your deployment environment.

Installation consists of downloading each partition to the node (or nodes) in your environment on which it is supposed to run. Installation is fully automated. The Environment Manager oversees and coordinates the downloading of partitions onto all nodes that have a Node Manager service running at the time of installation.

Throughout the application installation process, iPlanet UDS tracks the installation status of each node. It records the installation result for each targeted node in an environment, and posts events to announce the success or failure of installation on all targeted nodes.

If some target nodes are not available for installation—for example, server nodes that are not up and running or client nodes that are not currently running a Node Manager process—iPlanet UDS will perform a partial installation. Since iPlanet UDS keeps track of the installation status of each node, you can complete an installation incrementally as conditions permit, that is, as nodes become available for installation.

The Installation Procedure

You can perform installations using either the Environment Console or `Escript` utility.

(In a development environment in which you wish to install an application *developed in that same environment*, you can install the application by selecting an option in the Make Distribution command, when making the application distribution.)

➤ **To install an application**

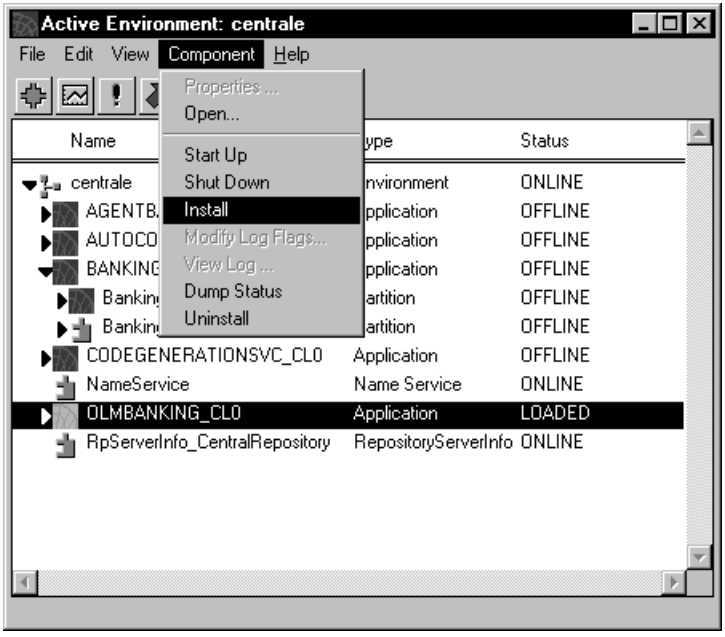
1. In the Active Environment window, select the View > Application Outline command.

   In the Application Outline view, the top level is the list of applications, the second level is the set of logical partitions for the application, and the third level is the set of assigned nodes for the partition.

2. Select the application you want to install.

   Notice that the Environment Console shows the status of the application as "Loaded."

3. Select the Component > Install… command.



   If your application successfully installs, the application status in the application outline view changes from "loaded" to "offline."

   If the application status changes to "partially installed," the Environment Manager installed the application on as many nodes as were available, but not on all targeted nodes.

4. If necessary, complete a partial installation by starting Node Manager processes on the previously unavailable nodes and issue the Install... command.

   On client-only nodes (Windows), use the Launch Server (or the Environment Console or `Escript` utility) to install the client partitions.

Installing on Server Nodes

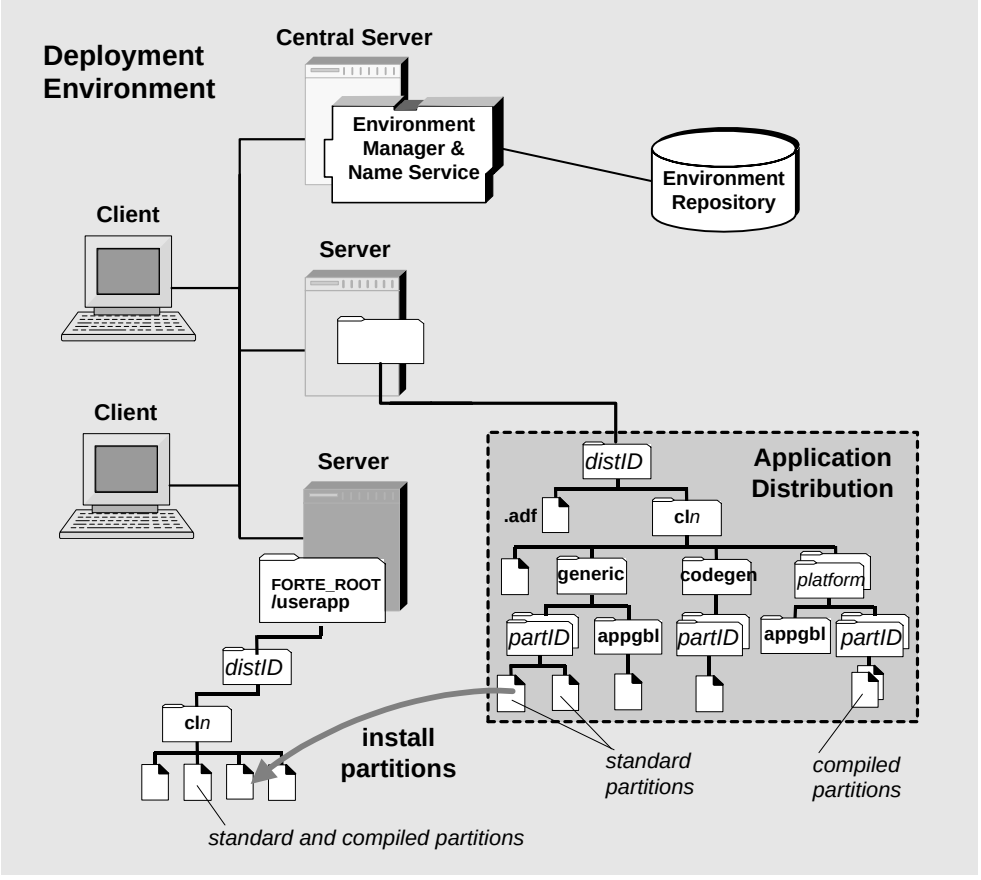
For installation on server nodes, the Environment Manager informs each Node Manager of the application partitions that need to be installed on its node.

In the case of standard partitions, the Node Manager downloads the appropriate image repositories from the node on which the application distribution resides (if it does not reside locally). The image repository for a standard partition consists of two files named after the partition—one with an .btx and one with a .btd extension (distributions created before Release 3.0 are .idx and .dat files).

The Node Manager places each image repository in the following directory, as shown in Figure 5-5:

FORTE_ROOT/userapp/*distribution_ID***/cl#**

Figure 5-5 Installing Application on Server Node



In the case of compiled partitions, the Node Manager downloads the appropriate executable partition files from the node on which the application distribution resides (if it does not reside locally) and also places them in the same directory as the standard partitions.

For the Node Manager to download partition files from a remote node, the Node Manager for the remote node must be up and running.

Installing on Client Nodes

Installation on client nodes is complicated by the fact that a client node, unlike a server nodes, does not usually run a Node Manager unless the client node normally starts a Launch Server.

Windows 95 and Windows NT clients are only receptive to remote management by the Environment Manager when they are running one of these utilities.

If you want your application to be installed on client nodes as well as server nodes in your environment, start the Environment Console, the `Escript` utility, or the Launch Server on each client before you begin installation.

There are two ways of handling installations onto client nodes: explicitly and as-needed.

Explicitly installing client partitions You can explicitly assign client partitions to specific client nodes or model node groups, as described in "Partition Assignments" on page 166. You then start a Node Manager on the client node (by running the Environment Console, the `Escript` utility or the Launch Server). When you install the application using the Environment agent's Install command, the Node Manager on the client node automatically installs the needed components on the client node.

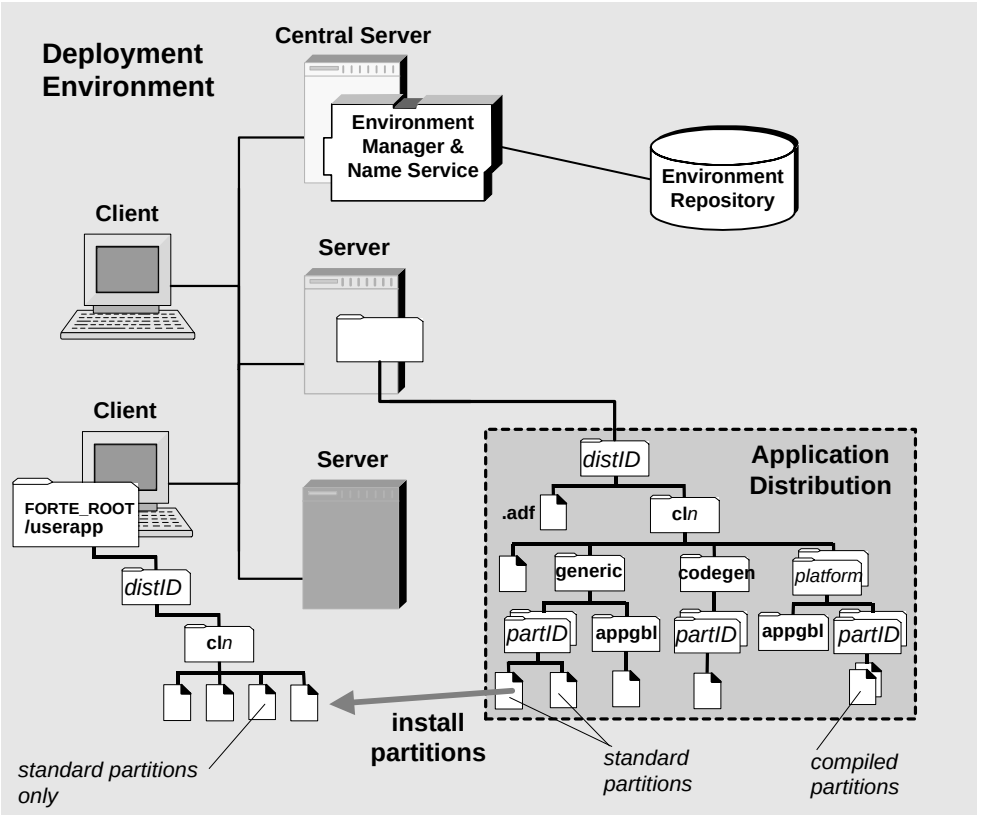
Launch Server installation of client partitions You can assign your client partitions or define them as publicly-available applications, and then let end users download the applications as-needed, when they want to run the application. This type of client installation works only for standard client partitions, not compiled client partitions. End users can also download updated releases of standard client partitions at runtime, if they use the Launcher application or if you set up their icons to do so. The Launch Server, along with the Launcher application and `Ftcmd` utility provide this download feature. This approach to installing client partitions is discussed in "Deploying Applications to Client Nodes" on page 298.

In situations where it may not be practical to install your application on all client nodes simultaneously, you can perform the installation at a later time on a node by node basis.

The Node Manager service provided by these utilities downloads the files for the client partition from the application distribution, and places them in the following directory, as shown in Figure 5-6:

FORTE_ROOT/userapp/*distributionID***/cl#**

Figure 5-6 Installing Application on Client Node



Generating Icons for Standard Client Partitions

In previous releases, iPlanet UDS generated icons for standard client partitions that start the application as a separate process in the client node's operating system.

iPlanet UDS now generates icons for standard client partitions that use the `ftcmd run` utility to start the applications. On Windows and Windows NT platforms, iPlanet UDS places command icons in the iPlanet UDS program group.

Windows platforms iPlanet UDS, generates icons on the Windows platforms that start the installed client partition using the Launch Server. These icons specify `ftcmd run` commands that have the Launch Server run the specified application.

NOTE On the UNIX platforms, you can define scripts or aliases that start client partitions using the `ftcmd run` command.

The `Ftcmd` utility is not available for OpenVMS platforms (VAX VMS or Alpha VMS).

For example, if you assign the client partition of the TimeIt example to a Windows client node and install the application on this node, the Command field of the Properties dialog for the icon would contain the following line:

```
ftcmd run TimeIt
```

The generated icons contain the `ftcmd run` command with the application name without the compatibility level.

You can also define icons that have the Launch Server start a client partition by using the following command syntax with the icons:

`ftcmd run` *application_name* [*release*] [*arguments*] [*update*]

For information about using this command, see "run" on page 311.

Generating icons with ftexec commands To have iPlanet UDS generate icons that start applications using the `ftexec` command), set the following configuration flag on each client node where you want this type of icon generated:

cfg:em:2

Set this configuration flag in either the Log Flags tab page of the iPlanet UDS Control Panel or the FORTE_LOGGER_SETUP environment variable. This configuration flag must be set before a Node Manager or a `Launcher Server` acting as Node Manager is started on the client node.

Generating Icons for Compiled Client Partitions

If you are installing a compiled client partition, On Windows 95 and Windows NT, iPlanet UDS creates an icon that starts the compiled client executable.

Creating Icons by Hand

For those situations in which you may need to create an icon yourself to run an installed client partition, you can do so, as described below, by copying and modifying one of the existing iPlanet UDS system application icons (such as the Environment Console icon).

You can create icons that use a compiled executable, the `ftexec` command (described in "Standard Partitions (Using the ftexec Command)" on page 190), or the `ftcmd run` command (described in "Using the Ftcmd Utility" on page 307).

➤ **To create a Windows 95 or NT client icon**

1. Make a copy of the existing icon.
2. Select the icon copy, then open the Properties dialog using the Properties command on the popup menu.
3. On the Shortcut tab page, edit the Target field to provide the command-line argument to run your client partition.

Installing Applications with Reference Partitions

Before you can install applications with reference partitions, you need to makes sure that the application that reference partition points to is also installed.

If you are installing an application with a reference partition that points to a service that is only available in another environment, you need to make sure that the developers have followed the correct steps for creating the replicated partition and setting the environment search path for this case. These steps are explained in detail in *A Guide to the iPlanet UDS Workshops*. You also need to make sure that this other environment is connected to your current environment. For information about connecting environments, see "Connecting Environments" on page 115 and *Escript and System Agent Reference Guide*.

Completing Partial Installations

If any targeted nodes are unavailable at the time of installation, the installation of an application in your deployment environment will be only partially successful. A particular server node may be offline or a number of client nodes may not become available for installation until a later time. You might also add nodes to your environment and need to install partitions on these new nodes.

In such situations, the Environment Manager completes only a partial installation and reports which partitions await installation, and on which nodes. Using this information you can complete partial installations as conditions permit. You can save the partial installation report by clicking the Export Report button.

You can complete a partial installation in a couple of ways:

- Re-install the application when the needed nodes or Node Managers are up and running—the Environment Manager will attempt to complete what remains to be installed and issue a report.

Deploying a Library Distribution

If your application requires access to libraries (shared libraries, object libraries, shared images, DLLs, or TOOL libraries), these libraries should be included in a library distribution packaged with your application distribution. The documentation accompanying your application distribution should include information about which libraries are needed by which partitions, so you can make sure that the appropriate libraries are installed on each node in your deployment environment.

iPlanet UDS automates the deployment of library distributions using its system management services. The process is essentially the same as for deploying application distributions: you load the library distribution into your environment repository, modify the library configuration to match your deployment environment and application partitioning configuration, and then install the library distribution into your deployment environment.

You can perform these tasks using either the Environment Console or the `Escript` utility. The steps for deploying library distributions using the Environment Console are summarized below. For information about using the `Escript` utility, see *Escript and System Agent Reference Guide*. Any differences between library distributions and application distributions are noted.

➤ **To deploy a library distribution**

1. Transfer the library distribution to a server node in your deployment environment.
2. Load the library distribution into your deployment environment's environment repository using the File > Load Distribution command.
3. Modify the library's partitioning configuration, if you wish.

   A non-restricted library is assigned, by default, to every server node in an environment. If your environment differs from the environment definition used to create the library configuration, you may have to assign the non-restricted libraries to nodes in your environment.

A restricted library requires special resources to support the library. You can assign a restricted library to nodes in your environment, or re-assign it, providing the target nodes have the resources to support the library.

Unlike partitions, libraries are not started by iPlanet UDS. Therefore, there are no properties analogous to the start options, server type, or server argument of assigned partitions. You can, however, choose whether to install each library as compiled or standard, if the distribution includes both the compiled executable and the image repository for the library.

4. Install the library configuration into your deployment environment.

   iPlanet UDS automatically downloads each library to a standard location on the node (or nodes) in your environment to which it is assigned. Installed libraries, because they never become executing iPlanet UDS processes, have no corresponding system management agents—they are merely part of the definitional information stored in an environment definition.

   Each node specification in an environment definition includes information regarding the restricted libraries installed on the node. In development environments, restricted library information is used in application partitioning.

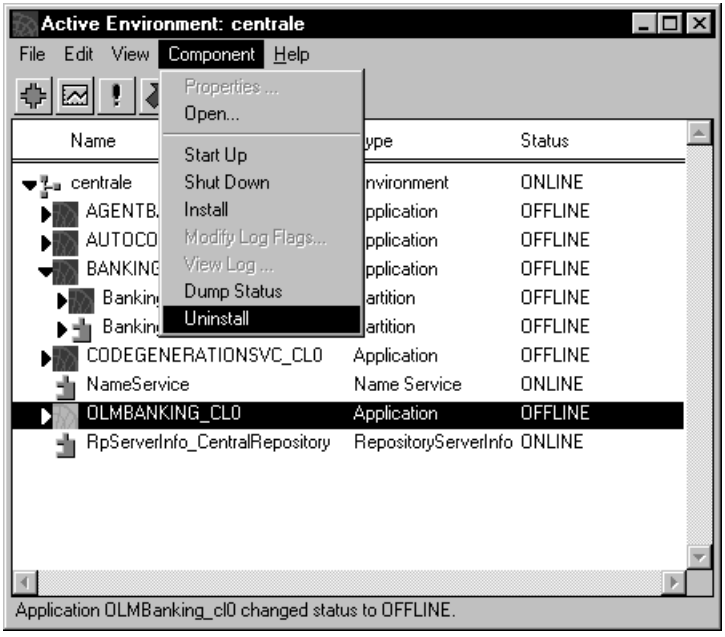
   In development environments, where developers need to access libraries to code and test their applications, you not only have to install the library configuration into the development environment, but you must also import the individual libraries into the development environment. Each library distribution includes a .pex file for each library that can be imported into development repositories, as shown in Figure 5-2 on page 155.

Removing an Application or Library

When you remove, or uninstall, an iPlanet UDS application or library, the Environment Manager deletes the information about the application from the environment repository and node repositories. After you uninstall an iPlanet UDS application or library, you can no longer manage the application or library using the Environment Console or `Escript`. However, the files for the application or library distributions still exist in the `FORTE_ROOT/userapp` subdirectories where they were copied when the application or library was installed. If you want to delete these files completely, you can do so after you uninstall the application or library.

➤ **To uninstall an application or library distribution**

1. In the Environment Console, choose the View > Application Outline command to display a list of installed applications and libraries.
2. Select the Application agent for the application or library you want to uninstall. (Application agents also represent libraries.)
3. Choose the Component > Uninstall command to uninstall the application or library.



You can also use the Edit > Delete command to uninstall the application or library.

Upgrading Applications

This section provides a brief explanation of how to perform the simplest upgrades of libraries and applications. For a more thorough discussion of the options you can consider when upgrading applications, including rolling upgrades, see *iPlanet UDS Programming Guide*.

Although iPlanet UDS does not support versioning control, iPlanet UDS lets application developers differentiate between incompatible releases of an application by assigning each release a different compatibility level. Releases of an application with different compatibility levels are treated by the iPlanet UDS runtime system and by iPlanet UDS system management services as completely different applications.

For information about compatibility between releases, see *A Guide to the iPlanet UDS Workshops* and *iPlanet UDS Programming Guide*.

Because these different applications can coexist in your deployment environment, you can deploy a new release of your application while the old one is still in use.

Upgrading Installed Applications

By default, iPlanet UDS assumes that, after you have installed an application, the configuration of the installed application is the correct configuration for the application. This assumption can affect whether you can install a new copy of the application over the installed application, or whether you need to uninstall the currently installed application first.

Incompatible configurations If you change the contents of any logical partition, iPlanet UDS cannot reinstall the application because the new configuration is incompatible with the configuration of the installed application. You need to uninstall the application before you can install a new distribution that uses a new configuration.

Changed partition assignments If you have changed only the partition assignments for an application that is already installed in the current environment, you can reinstall the application directly over the installed application distribution. However, the configuration of the application will remain the same as that for the previous installation of the application. The elements of the configuration that stay the same are the partition startup arguments, whether the partitions are compiled or interpreted, the number of replicates, where the partitions are assigned, and so forth.

If you want to change the configuration, you need to either change the configuration of the previous installation of the application before you install the newer application distribution, or you need to uninstall the application completely before installing the newer application distribution.

➤ **To change the configuration of an installed application without changing the contents of any logical partitions**

1. Uninstall the application using the Environment Console or `Escript`. You might need to ask your system administrator to perform this task, depending on who is permitted to use the Environment Console or `Escript` in your environment. For information about how to uninstall an application, see "Removing an Application or Library" on page 178.
2. In the Partition Workshop, configure the application the way you want, then select the File > Make Distribution command.
3. In the Make Distribution dialog, select the Full Make and auto-install options. You can also select the Auto Compile option, if appropriate. Click the Make button.

➤ **To upgrade an installed application**

1. Install the new release of the application on the clients and servers while the old release is running.
2. After all server nodes are running, move client nodes to the new release incrementally, by starting the new release on, for example, 50 of the clients, then 100, and so on until all clients are running the new release.

   When you install an upgraded release of an application on your client nodes, be sure to upgrade the corresponding command icon, as well. For information about command icons, see "Installing on Client Nodes" on page 173.

   If the application uses standard client partitions, you can have the end users upgrade their applications as needed using the services of the Launch Server, as described in "Deploying Applications to Client Nodes" on page 298.
3. When all client nodes are running the new release, shut down and remove the old release from the server nodes and client nodes.

Upgrading Reference Partitions

When you upgrade an application whose partitions are used as reference partitions for other applications, you need to make sure that the applications with reference partitions still reference the correct release of the partition. If the compatibility level of the application containing the partition has changed, then you need to reconfigure and redeploy the applications whose reference partition uses that partition, so that the reference partition uses the correct release of the application.

You do not need to change the compatibility level of the application containing the reference partition when you change the reference partition definition, and the compatibility levels of the two applications do not need to be the same.

For example, suppose you have an application BankClient, which contains a reference partition that represents a partition in the application AppServices. Both applications are at compatibility level 0. If you install a new release of AppServices at compatibility level 1, be aware that unless you update the reference partition in BankClient, that reference partition still references the partition of the older release of AppServices.

➤ **To make the reference partition reference the partition of the newer release of a changed application**

1. In the Partition Workshop, view the configuration of the application with the reference partition and define a new reference partition that references the partition in the new release of the application. You can delete the old reference partition.
2. Make a new distribution for the application with the reference partition.
3. Reinstall the application with the reference partition in your deployment environment.

The steps for partitioning and making a distribution are described in *A Guide to the iPlanet UDS Workshops*.

Upgrading Libraries

When you upgrade a library to another compatibility level, you need to update and redeploy any applications that use this library. However, the compatibility levels of the library and its associated applications do *not* have to be the same.

If the compatibility level of the library does not change because the changes to the library are very minor, you do not need to upgrade the applications that use the library; you can simply install the revised library.

For example, suppose you have an application called Payroll, which uses classes and methods provided by the library TimeCardFunctions. Both the application and the library are at compatibility level 0.

If you install a new release of TimeCardFunctions at compatibility level 1, be aware that unless you update and redeploy the Payroll application, that application still references the older release of the TimeCardFunctions library.

➤ **To make the application reference the newer release of the library**

1. The application developer imports the .pex file from the library distribution into the development repository where the code for the application resides.
2. The application developer makes a new distribution of the application.
3. You install the updated release of the application in your deployment environment.

If you are upgrading libraries that are used by developers in a central repository, the developers need to check out all components of each library, reimport the .pex file for each upgraded library, then integrate the changed libraries into the system baseline. If the upgraded library is used in a private repository, the developer simply needs to reimport the new .pex file into the repository. For more information about using repositories, see *A Guide to the iPlanet UDS Workshops*.

Partial Upgrades

When developers make an incompatible upgrade in an individual partition (or service object) in an application, they issue an upgraded release of the application—with a new compatibility level—for you to install in your deployment environment.

However, under certain circumstances, a new release of the service object in an application can be fully compatible with the rest of the application. In this situation, a developer can make a partial distribution without increasing the compatibility level of the application (see *iPlanet UDS Programming Guide*). The partial distribution contains only the compatible, revised portions of the application.

You can deploy the partial distribution in your deployment environment just like any other application distribution. In this case, however, the distribution contains only the upgraded application partitions. When you deploy this partial upgrade, the new partitions are substituted for the old partitions. In this situation you must make sure that:

- the partitioning configuration of the upgrade distribution corresponds exactly to that of the old application
- your old application partitions are shut down before you install the new ones

For information about upgrading deployed and running applications, see *iPlanet UDS Programming Guide*.

Managing iPlanet UDS Applications

Having deployed an application in an iPlanet UDS environment, you are now ready to manage its execution. iPlanet UDS lets you start and monitor a distributed application from one central console. If you detect performance problems, you can modify the application's partitioning configuration to resolve them. By properly starting, monitoring, and reconfiguring your distributed application, you can maximize its performance.

This chapter covers the following topics:

- starting iPlanet UDS applications
- monitoring iPlanet UDS applications
- reconfiguring applications
- managing applications with replicated partitions

All procedures in this chapter will assume you are using the Environment Console. You can use the equivalent Escript commands to perform the same tasks, as described in *Escript and System Agent Reference Guide*.

Starting iPlanet UDS Applications

Starting an iPlanet UDS application involves starting the application client and all server partitions needed to achieve the full functionality of your distributed application.

Starting Client Partitions

The end user of an application typically starts client partitions manually when they are needed.

On Windows, client partitions are started using command icons, which are usually installed when you install the client partition of the application.

On Windows and UNIX platforms, you can start client partitions using:

- `ftcmd run` command, described in "Using the Ftcmd Utility" on page 307

  You can use this command to have the Launch Server start a standard (interpreted) client partition. On platforms with icons, you can specify this command for the icon. For example, you can start a client partition for the Banking application on a Windows NT machine using the following command on the command line:

  ```
  ftcmd run Banking
  ```

  The `Ftcmd` utility and the Launch Server are not available on VMS.

- `ftexec` command, described in "Manual Startup" on page 190

  You can use this command to start a standard (interpreted) partition. On platforms with icons, you can specify this command for the icon. For example, you can start a client partition for the Banking application on a UNIX machine using the following command on the command line:

  ```
  ftexec -fi bt:${FORTE_ROOT}/userapp/banking/cl0/bankin0
  ```

- compiled executable (compiled partitions only), described in "Compiled Partitions" on page 192

  If a client partition is compiled, you can start the client partition using the executable for the client partition, as shown in the following example, in which the client partition for the Banking example has been compiled:

  ```
  ${FORTE_ROOT}/userapp/banking/cl0/bankin0.exe
  ```

Starting Server Partitions

You can start a server partition using any of the following approaches:

managed startup Use system management tools to control the startup of all server partitions. This approach is the most effective way to manage your applications.

auto-startup Start a client partition and leave it up to iPlanet UDS system management services to start server partitions as they are needed.

manual startup Start server partitions by hand without using system management tools.

These approaches are described briefly in the following sections. Each has its own advantages and disadvantages; however, the emphasis of this chapter is on the managed startup approach, since this provides you the most control of the execution of your application.

Managed Startup

You can use the iPlanet UDS system management services to start up and shut down applications from a centralized point in the environment. Basically, in the Environment Console, you start all of an application's server partitions before any client partitions are run. You can start all enabled server partitions with a single command or start each server partition one at a time.

When you start an application or partition using the Environment Console or `Escript`, you do not have to know the details of the application to start the servers properly; the iPlanet UDS system management services start the correct number and type of server partitions for an application when you start up an application.

➤ **To start all enabled server partitions in an application**

1. Select the application in the Application Outline view of the Active Environment window or in the Application Agent window.
2. Choose the Component > Start Up command or click on the startup icon in the tool bar.

All enabled installed partitions within the application will be started to their full replication count.

➤ **To start a single installed server partition**

1. Select the installed partition in any Environment Console Agent window or view.

2. Choose the Component > Start Up command or click on the startup button in the tool bar.

   One active partition will be started. Starting up a single installed partition overrides the start option properties specified for the logical partition (enabled/disabled and replication count).

You can shut down all active partitions within an application with a single command or shut down individual active partitions one at a time.

➤ **To shut down an application or a single server partition**

1. Select the application or active partition of interest in any window of the Environment Console.
2. Select the Component > Shut Down command from the Component menu or click on the shutdown icon in the tool bar.

Applications with replicated partitions Managing the startup of an application is especially useful in the case of an application with replicated partitions. In this case it's best to do some advance planning: you try to determine in advance the number of replicates of each partition to start on each node. In making this determination, you try to establish how best to meet the application's normal failover and load balancing requirements.

Generally, you implement this normal startup configuration after loading an application distribution into your environment repository, but before you actually install the application into your environment. You assign replicated partitions to various nodes and set their start option properties (enabled/disabled and replication count). At this stage you can designate some replicates as reserve partitions by assigning them a disabled status on a node.

Having created your normal startup configuration (and installed the application in your environment), you can start your application by simply invoking the Component > Start Up command on the Application agent. The Environment Manager takes over, and uses Node Managers on each node to start enabled partitions up to their designated replication count.

You can subsequently start additional partitions from the Environment Console, or shut them down, as the need arises. You have complete control over which installed partitions to start and how many active partition replicates of each to have running at any time.

For more information on managed startup, see "Managing Applications with Replicated Partitions" on page 209.

Auto–Startup

In this approach to starting an application, a user simply starts the application's client partition. As the client partition requires the services of remote server partitions, one of each of those partitions are automatically started by the iPlanet UDS runtime system, using iPlanet UDS system management services.

If a particular server partition is not running when the client requires the service it provides, the Node Manager for a node on which the partition resides is directed to start that server partition. Once started, the address of the service object(s) the partition supports is registered with the Name Service. Further requests for that service are then made directly to the now-running server partition.

The Node Manager starts only enough copies of a server partition to run the application. Usually, the Node Manager starts only one copy of the server partition, regardless of the replication count specified for any service object in the replicated server partition.

As each new service is needed, an instance of the server partition providing the service is started, until all required server partitions for the application are up and running.

This method of starting an application requires no intervention by a system manager at any stage of the process. Application services just automatically start as needed, using the iPlanet UDS runtime system and system management processes. For information on implementing auto-start in connected environments, see "Setting an Environment Search Path" on page 117.

However, from a system management point of view, auto-startup has a number of important disadvantages:

- Auto-startup does not provide for managed startup nor the shutdown of an application.
- Only a minimum number of partitions are started. One installed partition is started for each service, meaning that no replicate partitions are running on any other node for failover or load balancing purposes.
- Server partitions are started one at a time, as they are needed, rather than in parallel before they are needed. If any server partition takes some length of time to start, this can result in significant delays in the running of the client partition.
- Disabled partitions are not started. If no enabled partitions are available to be started, auto-startup does not start available disabled partitions.

Manual Startup

Server partitions can also be started without using the system management services by starting each partition on its respective node using the techniques described in this section.

You can start up server partitions manually in an environment in which Node Managers are for some reason not running. (The Environment Manager must still be running for the application to run properly.)

The partition startup command depends on whether the partition is a standard partition or a compiled partition. These two cases are described in the sections that follow.

Standard Partitions (Using the ftexec Command)

ftexec command If the partition is a standard client partition running on a server platform or a standard server partition, you usually start the partition using the `ftexec` executable (the interpreter) with the application's image repository on the node where the repository resides. This section explains the syntax for starting one of these partitions using the `ftexec` command.

You can also start client partitions using the ftcmd run command, as described in "run" on page 311.

The syntax is slightly different for client partitions and server partitions.

➤ **To start a standard client partition, enter the following version of the ftexec command**

Portable (All Platforms)

ftexec -fi *image_repository_name* [**-fs**] [**-fns** *name_server_address*] [**-fnd** *node_name*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-fcons**] [**-fnw**] [**-fterm**] [**-fss**] [**-fnomad**]

OpenVMS

VFORTE FTEXEC
/IMAGE_REPOSITORY=*image_repository_name*
[**/STANDALONE**]
[**/NAMESERVER=***name_server_address*]
[**/NODE=***node_name*]
[**/LOGGER=***logger_flags*]
[**/MEMORY=***memory_flags*]
[**/STACK**=*integer]*
[**/FCONS**]
[**/FNW**]
[**/FTERM**]

➤ **To start a standard server partition, enter the following command**

Portable (all platforms)

ftexec -fi *image_repository_name* **-ftsvr 0** [**-fns** *name_server_address*] [**-fnd** *node_name*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*]

OpenVMS

VFORTE FTEXEC
/IMAGE_REPOSITORY=*image_repository_name*
/SERVER_ONLY 0
*[/***NAMESERVER=***name_server_address]*
[**/NODE=***node_name*]
[**/LOGGER=***logger_flags*]
[**/MEMORY=***memory_flags*]
[**/STACK**=*integer*]

➤ **To start a standard server partition that uses POSIX threads, enter the following command**

Portable (all platforms)

ftexecd -fi *image_repository_name* **-ftsvr 0** [**-fns** *name_server_address*]
[**-fnd** *node_name*] [**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*]

If you are starting a server partition that provides database services, you should check whether the database vendor supports POSIX threads before you try to run the partition using the `ftexecd` command. If the database does not support POSIX threads, use the `ftexec` command to start the partition.

See the table in "Startup Flags" on page 192 for an explanation of each of the command line flags.

Compiled Partitions

If the partition is a compiled partition, then it is an independent executable that incorporates all repository information. To start the partition, enter the executable name (using the full path name, where the directory path delineators you use depend on platform):

FORTE_ROOT/userapp/*distribution_ID***/cl#/***partition_ID flags*

Path Component	Description
distribution_ID	The first 8 characters of the application distribution name.
cl#	The compatibility level number.
partition_ID	The first 8 characters in the partition name.

You can specify the same flags when you start compiled partitions that you can specify for starting standard partitions, as described in "Standard Partitions (Using the ftexec Command)" on page 190 and "Startup Flags" below, with the exception of `-fi`.

Startup Flags

The following table explains each of the command line flags.

Flag	Description
-fi *image_repository_name* **/IMAGE_REPOSITORY=** *image_repository_name*	Specifies the image repository which contains the partition to start executing. There is no default, and this must be specified. The *image_repository_name* has the format: bt:FORTE_ROOT/userapp/*distribution_ID*/cl#/*partition_ID*. For example, you could specify bt:c:\forte\userapp\banking\cl0\bankin0 to indicate the image repository for the client partition of the Banking example program on a Windows NT machine. For the exact directory path specifications for each platform, see "About the B-tree Repository Format" on page 249. For applications that were deployed prior to Release 3.0, the image repository starts with "ct:" instead of "bt:".

Flag	Description
-fs /STANDALONE	(Clients only) Starts a client application as a standalone application.
-ftsvr 0 **/SERVER_ONLY 0**	(Standard server partitions only) Run `ftexec` as a multi-threaded server process only.
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-fnd *node_name* **/NODE=***node_name*	Specifies the node name to use for this session. If you do not specify the node name in the command, the default node name depends on the operating system. On Windows, the default node name is set by the FORTE_NODENAME environment variable. On all other platforms, the actual node name is used.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the amount of memory the command should use in the context of the operating system where it runs. See "-fm Flag (Memory Manager)" on page 375 for syntax information. On UNIX, you must specify the setting in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the starting log tracing flags for the command. See "-fl Flag (Log Manager)" on page 371 for syntax information. This specification overrides the FORTE_LOGGER_SETUP environment variable setting. On UNIX, you must specify the setting in double quotes.

Flag	Description
-fcons /FCONS	(Clients only) On Windows platforms, this flag specifies that the background trace window also displays. By default, on these platforms, this trace window is not shown for iPlanet UDS client applications that do not use a command-line interface. On UNIX and OpenVMS platforms, this flag specifies that the client partition tries to run even if it cannot make a required connection to an X windowing system. Without this flag, such a partition would fail without the X windowing system connection.
-fnw /FNW	(UNIX and OpenVMS clients only) Specifies that the client session begins as a multithreaded partition without opening an X windowing system connection.
-fss	(Clients only) Displays the iPlanet UDS splash screen.
-fterm /FTERM	(UNIX clients only) Specifies that the client session run as always attached to the terminal, so that it always responds to terminal commands, such as Control-c.
-fnomad	(Clients only) Starts a client application without initially connecting to a name service or environment (a nomadic application).

Monitoring iPlanet UDS Applications

One important aspect of managing an application is monitoring its resource usage and performance in order to eliminate any problems or bottlenecks that might arise. There can be many kinds of resource usage and performance problems: partitions might be offline or overloaded, network bandwidth might be at its limit, or servers might be overutilized. Any combination of these problems can cause bottlenecks in the resource usage and performance of your application

Agent status You can uncover and diagnose application resource usage and performance problems using a number of application monitoring capabilities provided by iPlanet UDS system management facilities. Most iPlanet UDS system management agents, for example, have a status attribute that allows you to discover when partitions within an application have gone offline.

Agent instruments In addition, you can view specific resource usage and performance indicators for an application using specific instruments defined for specific system management agents. These instruments correspond to resource usage and performance indicators, or data, that agents can obtain from the objects they are managing. For example, the DistObjectMgr agent, which manages the Distributed Object Manager within an active partition, can report on the number of messages sent to and from the active partition since it was started.

In addition to the system management agents and instruments provided by iPlanet UDS, you can create your own agents and instruments to capture specific resource usage and performance indicators for an application or particular partitions within an application.

Since the classes used by the iPlanet UDS system management agents are published (see *Programming with System Agents*), you or the application developers can write an iPlanet UDS application to monitor data related to an application. Such an application would typically view the value of instrument data at regular periods of time, do some data analysis, present the results in graphical form, send warning messages, and perhaps start or shut down partitions as the need arises.

The Environment Console, for example, provides a limited data analysis capability for tracking instrument data over time. The Charts window lets you view any instrument data you choose, over time periods you specify. By tracking instrument data in this way, you can get a feel for how resource usage and performance relates to activity and determine the baseline usage patterns of your applications.

Log files Using instruments provided by the Active Partition agent, you can also log specific instrument data for any active partition at regular intervals. This data is written to the active partition log file (and to the environment log file as well, if you wish).

You can also use log files more generally as another way of monitoring application resource usage and performance. In addition to instrument data, you can request that certain messages generated by your application or by the iPlanet UDS runtime system be written to specific log files. This logging capability is especially useful for troubleshooting problems that might be difficult to uncover in routine monitoring of your application.

The following section will discuss techniques for monitoring the status of your executing application, viewing the value of instrument data and tracking instrument data over time, and logging instrument data. For more general information about iPlanet UDS logging capabilities, see "Logging and Log Files" on page 217.

Monitoring Status

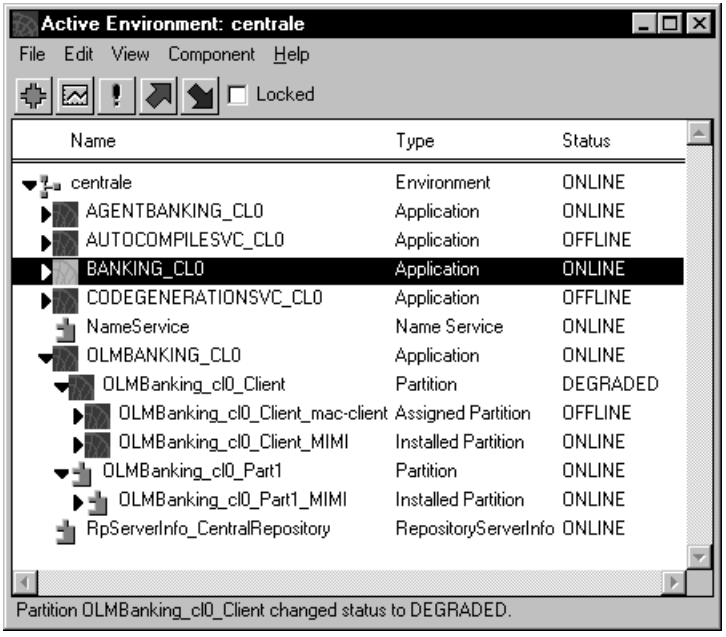
You can use the status attributes of a number of system management agents to monitor the status of an application and its partitions in order to uncover failures or redundant processes.

Status can be viewed in the Environment Console's application outline or node outline views, or in the `Escript` utility. In general, the most important status is whether an object is online or offline.

For information about the possible statuses for each agent, see *Escript and System Agent Reference Guide*.

Application view For any managed object in the application view to be considered ONLINE, all its children must also be online. For example, the full replication count of active partitions for an installed partition must be ONLINE for an installed partition to be ONLINE. Similarly all enabled, installed instances of a logical partition must be ONLINE for the logical partition to be online.

Figure 6-1 Application Outline View Displaying Information about Applications

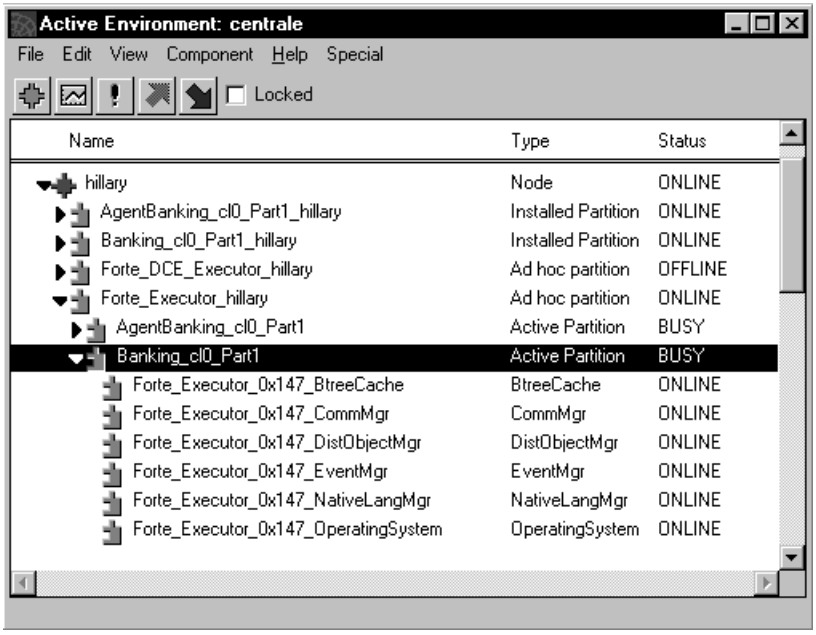


If one or more, but not all, child objects of a parent object are not ONLINE, then the status of the parent object is considered DEGRADED. The DEGRADED status propagates up to the Application agent. If the full replication count of active partitions for an installed partition is not ONLINE, then all objects above active partition in the hierarchy will be DEGRADED. An application has a DEGRADED status when it is not running as specified in its partitioning configuration.

Because the DEGRADED status is propagated up to the highest level, it is possible to monitor the top level of an application and track down problems existing at a lower level. This approach is sometimes referred to as management by exception.

Node view The node view includes an additional partition—the iPlanet UDS executor (ftexec), an installed partition which executes standard iPlanet UDS partitions. An iPlanet UDS executor partition can be BUSY running a standard (active) partition or simply ONLINE—waiting to run a standard partition after one has been shut down. iPlanet UDS executors can proliferate if not specifically shut down, especially in development environments where many developers might be testing applications concurrently.

Figure 6-2 Node Outline View Displaying Information about Applications



Standard partitions and iPlanet UDS executor partitions When you start a standard partition, the iPlanet UDS runtime system looks for a running iPlanet UDS executor partition with sufficient memory to run the standard partition. If such as partition exists, then the runtime system has that partition load the and interpret the standard partition. Otherwise, the runtime system automatically starts a new iPlanet UDS executor partition, which loads and runs the standard partition.

Therefore, to monitor a standard server partition, you need to check the Active Partition agents for both the application partition and the iPlanet UDS executor partition that is running the partition.

Agents for the iPlanet UDS executor partitions for standard clients do not appear as part of the agent hierarchy, so you can only monitor the Active Partition agent for the standard partition.

➤ **To locate a particular iPlanet UDS executor server partition**

1. Locate the Forte_executor_*nodename* agent, which is a subagent of the Node agent for the node where the standard partition is running.
2. Click the expansion arrow next to the Forte_executor_*nodename* agent, then click the Active Partition agent whose name matches the name of the standard partition.

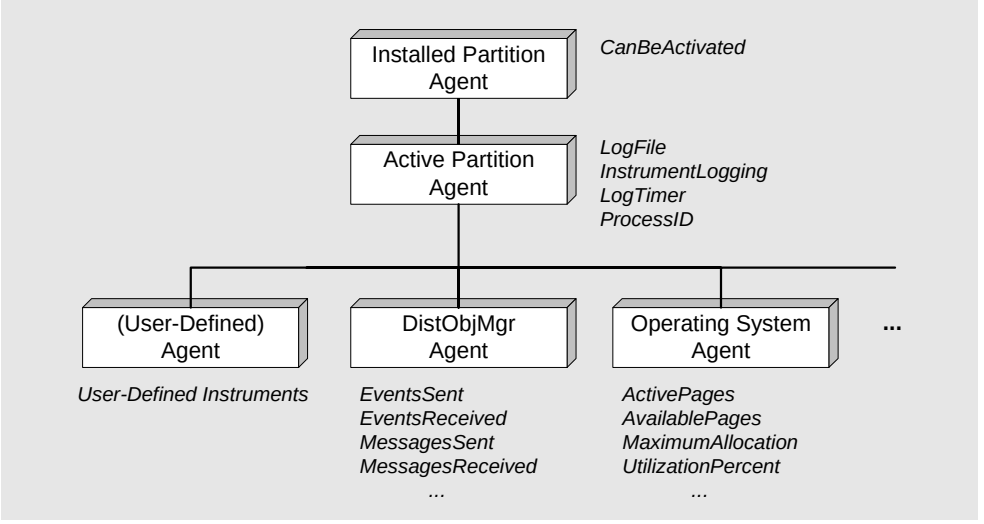
Viewing Instrument Data

You can monitor application performance by viewing instrument data that corresponds to specific application performance indicators. These indicators correspond to data that a system management agent can obtain from the object it is managing.

Each system management agent has a set of instruments; each instrument represents a type of data that can be obtained from or set on the object being managed by the agent. The instruments defined for each agent generally represent data that is useful to expose to the management system for monitoring or control purposes. An instrument might represent a property or attribute of the managed object or it might represent more dynamic information. For example, the OperatingSystem subagent of the Active Partition agent has an instrument representing the current number of active pages of object memory being used by the active partition.

Figure 6-3 shows a few of the instruments defined on the system management agents normally used for monitoring application performance. Most information about the performance of an application would be reported by its Active Partition agents and their subagents. In addition, you can create more application-specific performance indicators by defining your own system management agents—usually for service objects in your application—with instruments corresponding to performance related data maintained by the respective service objects. For information about all the iPlanet UDS-defined instruments, see *Escript and System Agent Reference Guide*.

Figure 6-3 Some Instruments of System Management Agents



Whether your instruments are iPlanet UDS-defined or user-defined, you view instrument values in the same way.

➤ **To view the value of an instrument**

1. Open the Agent window corresponding to the agent for which the instrument is defined.

   You can open the Agent window by selecting the agent in any open Agent window and choosing the File > Open command, or by double-clicking on the agent.

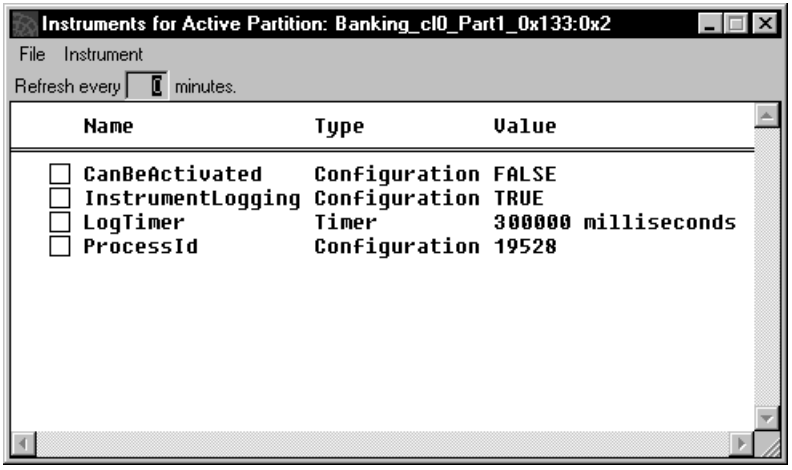
2. In the open Agent window, choose the File > Instruments... command.

   The Instruments window for the current agent opens, showing all instruments defined for that agent, the type of each instrument, and the current value of each instrument, as shown in the figure below.

   To view the value of an instrument at a later time, select the instrument in the Instruments window and select the Instrument > Refresh Instrument command.

   You can refresh all instrument values for the current agent by selecting the Instruments > Refresh All Instruments command from the File menu.

Figure 6-4 Instrument Window



Each instrument has a type associated with it that determines how it works:

Name	Description
Average	Read only. Contains an average value.
Compound	Contains more than one instrument. These instruments can be of different types.
Configuration	Read/write or read only. Contains a simple value, such as an integer value or a TextData object.
Counter	Read only. Contains a value based on counting something.

Name	Description
SubObject	Contains more than one instrument. All instruments are of the same type.
Timer	Read/write. Timer that prompts the agent to do something after a certain interval or set of intervals.

Tracking Instrument Data: Charts Window

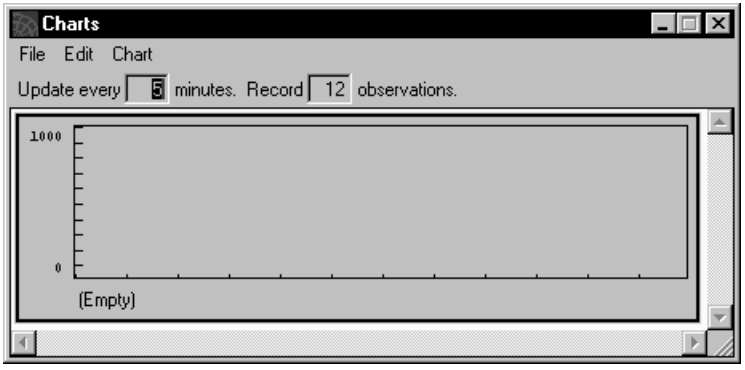
To track instrument data, that is, to view instrument data over time, you can use the Charts window of the Environment Console. (You or your application developers can also write your own performance tracking application using classes in *Programming with System Agents*.)

➤ **To track the value of an instrument**

1. Open the Charts window by choosing the File > Charts command in the Active Environment window.

   The Charts window opens.

2. Create a new, empty chart by selecting the Chart > New Chart command in the Charts window.



3. Open the Agent window corresponding to the agent for which the instrument of interest is defined.

   You can open the Agent window by selecting the agent in any open Agent window and selecting the File > Open command, or by double-clicking on the agent.

4. In the open Agent window, select the File > Instruments command from the File menu.

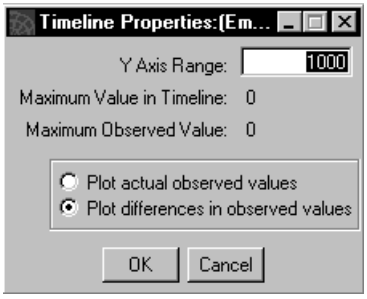
   The Instruments window for the current agent will open, showing all instruments defined for that agent and the current value of each instrument.

5. Drag the instrument of interest from the Instruments window and drop it onto the new, empty chart (created in step 2) in the Charts window.

   The chart will now display the data for the instrument, adding to the chart at the specified update interval.

6. You can change the interval between each update to the chart by changing the Update every ___ minutes field. You can also change the number of entries recorded on the chart by changing the Record ___ observations field.

7. To set the vertical scale parameters for the chart and see the maximum and minimum values recorded by this chart, select the Chart > Chart Properties command. You can also set whether the chart displays the actual values or the difference between the previous value and the current value in the Timeline Properties dialog.



8. To modify the appearance of the chart, you can select the Line Color, Line Weight, or Set Default... commands from the Chart menu.

Tracking Instrument Data with Log Files

Using instruments provided by the Active Partition agent, you can also log specific instrument data for any active server partition at regular time intervals.

NOTE Client partitions do not have log files. However, these partitions can write log information to a trace window.

By default, instrument data is not logged. To have iPlanet UDS log instrument data for an active partition, you need to do the following:

- Specify that instrument data be logged to the Active Partition agent's log file, the Environment Manager's log file, or both. You also need to specify how often the data should be logged.
- Set one or more instruments of the Active Partition or its subagents to be logged.

The details of these steps are described in the following sections.

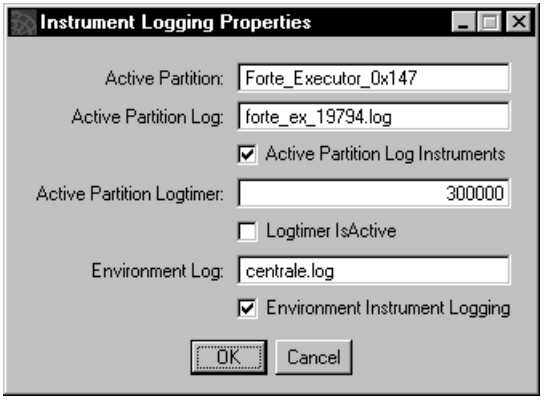
Specifying When and Where to Log Instrument Data

When you log instrument data, you can log it to the Environment Manager's log file, the Active Partition agent's log file, or both. If you are logging data to one of these log files, you must also turn on the LogTimer instrument for the Active Partition agent. You can also adjust how often data is logged.

➤ **To specify when and where to log instrument data**

1. In the Agent window for the Active Partition agent or one of its subagents, open the Instruments window by selecting the Instruments... command from the File menu of the current Agent window.
2. Open the Instrument Logging Properties dialog by selecting the File menu's Instrument Logging... command.

3. Change the instrument logging properties, as appropriate:



The following table briefly describes the Instrument Logging properties:

Instrument Logging Property	How to specify it
Active Partition Log Instruments	Indicates whether to log data for the instruments of this active partition to the log file for this active partition at the intervals defined by the LogTimer instrument for this Active Partition agent. This field corresponds to the Active Partition agent's InstrumentLogging instrument. A check sets this instrument's value to TRUE.
Active Partition Log	(Only available for iPlanet UDS Executor active partitions) Specifies into what file to write the instrument information being logged.
Active Partition Logtimer	Indicates the time interval, in milliseconds, between log entries for this active partition. 60000 is 1 minute. This field corresponds to the *interval_in_msec* setting of the Active Partition agent's LogTimer instrument.
Logtimer IsActive	Indicates whether the LogTimer for this Active Partition agent is triggering log entries at its defined time intervals. This field corresponds to the *is_active* setting of the Active Partition agent's LogTimer instrument. A check sets this setting to TRUE.

Instrument Logging Property	How to specify it
Environment Instrument Logging	Indicates whether instrument data is being logged to the environment log file. This value affects all instrument logging in the environment. This field corresponds to the Environment agent's InstrumentLogging instrument. A check sets this instrument's value to TRUE.

These agents and instruments are described in detail in *Escript and System Agent Reference Guide*.

4. You can change the name of the Environment Manager's log file by entering the new name of the log file in the Environment Log field. On Active Partition agents for iPlanet UDS executor partitions or compiled server applications, you can also change the active partition's log file in the Active Partition Log field. For detailed information about changing log file names, see "Changing Log File Names" on page 219.

The iPlanet UDS system software lets you maintain a variety of log files to monitor system status and application performance. Each active partition, including the Node Manager partitions, writes to a log file, which can be used for tracking instrument data as described above. In addition, using the InstrumentLogging instruments of the active Environment agent, you can set the Environment Manager to log environment-wide instrument data to an environment log file.

For additional information on log files, see "Logging and Log Files" on page 217.

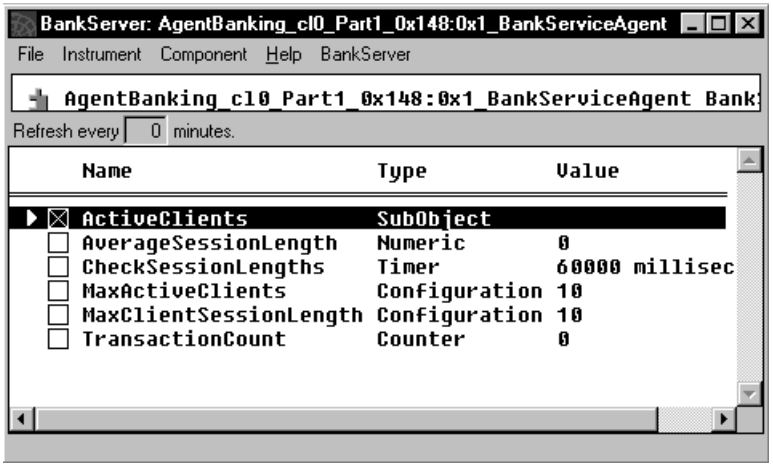
Setting an Instrument for Logging

Along with specifying when and where to log instrument data, you need to set each instrument that you want to log. Each instrument has a setting that defines whether it can be logged or not. By default, an instrument is not logged.

➤ **To set an instrument to be logged**

1. In the Environment Console's Active Environment window, select the View > Node Outline command.
2. Find the agent whose instrument you want to log by expanding the browser outline view.
3. Open the Agent window for that agent by selecting the agent, then selecting the Component > Open command.

4. Open the Instruments window for the agent by selecting the File > Instruments... command in the Agent window.



Set the instrument to be logged by doing one of the following:

- Select the instrument, then select the Instrument > IsLogged command.
- Click the checkbox next to the instrument you want to log.

The checkbox next to the instrument indicates whether an instrument is being logged or not.

Managing Running Applications

To improve the performance of your running applications, you can use the commands provided by subagents of the Active Partition agents for the application. You can also change the instrument values for these subagents.

For information about all the commands and instruments provided by iPlanet UDS-defined agents, see *Escript and System Agent Reference Guide*.

Changing Instrument Values

If an instrument is a Configuration instrument and is not read only, you can change its value.

➤ **To change the value of a changeable instrument**

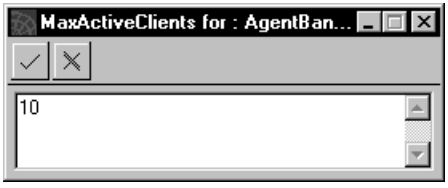
1. Open the Agent window corresponding to the agent for which the instrument is defined.

   You can open the Agent window by selecting the agent in any open Agent window and selecting the File > Open command, or by double-clicking on the agent.

2. In the open Agent window, select the File > Instruments command.

   The Instruments window for the current agent will open, showing all instruments defined for that agent and the current value of each instrument.

3. Open the instrument value window by selecting the Instrument > Modify command.



4. Change the value in the text field.

   To change the value of the instrument, click the icon with the check mark. The value of the instrument in the Instrument window also changes.

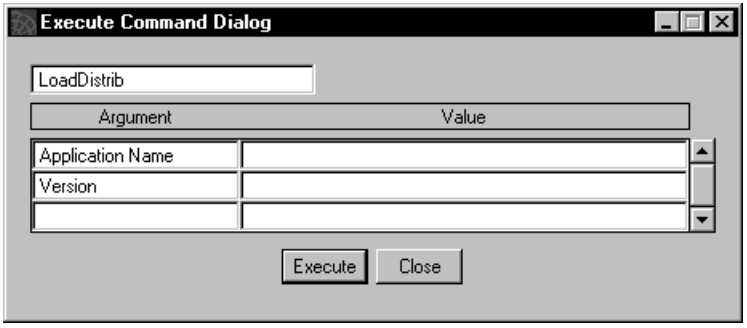
   To cancel the change to the value, click the icon with the x. The value in the text field reverts to the current instrument value.

Using Agent Commands

Most commands for iPlanet UDS-defined agents appear in the Active Environment window under either the Component menu or the Utility menu.

If you select an agent command that requires user input, such as the LoadDistrib command on the Node agent, you see an Execute Command dialog, like this:

Figure 6-5 Execute Command Dialog



Enter the argument values, as necessary for the command, then select the Execute button. For information about command arguments, see *Escript and System Agent Reference Guide*.

If you select an agent command that produces output, such as the ShowAdmin command on the NameService agent, this output is printed to the trace window of the Environment Console.

Reconfiguring Applications

Your efforts to improve application performance might often require you to modify the partitioning configuration of your running application.

For example, you might add a server to your environment in order to accommodate increasing numbers of clients. You might want to off load some functionality from an overloaded server and place it on your new server. This will require modifying your application's partitioning configuration.

You might also need to increase the amount of object memory space for a partition.

➤ **To reconfigure an application**

1. Shut down your application.
2. Lock your active environment.
3. Reconfigure the application in the Environment Console (or using the appropriate Escript commands).

   See "Modifying a Partitioning Configuration" on page 165.
4. Unlock the environment definition.
5. Reinstall the application.

   See "Installing an Application" on page 170.

When you reinstall the application, the system management services perform an incremental installation, installing only those partitions necessary to deploy the new configuration of the application.

Managing Applications with Replicated Partitions

Using iPlanet UDS system management features, you can enhance application reliability and performance by replicating server partitions.

Failover For example, to improve application reliability, you can provide backup replicates of a server partition in case the primary server partition fails. You typically run the replicates on different servers.

Load balancing Similarly, to improve application performance, balance the demand for a service among several replicates of the partition that provides that service. You can run the replicates on the same or different servers.

You might need to do more work to manage applications with replicated partitions for failover and load balancing than in the case of applications with non-replicated partitions. For example, failover and load balancing work properly only when you start an application using the managed startup method. Auto–starting an application generally does not start all the needed replicates of a server partition. In addition, you might need to start additional server partition replicates during times of peak usage.

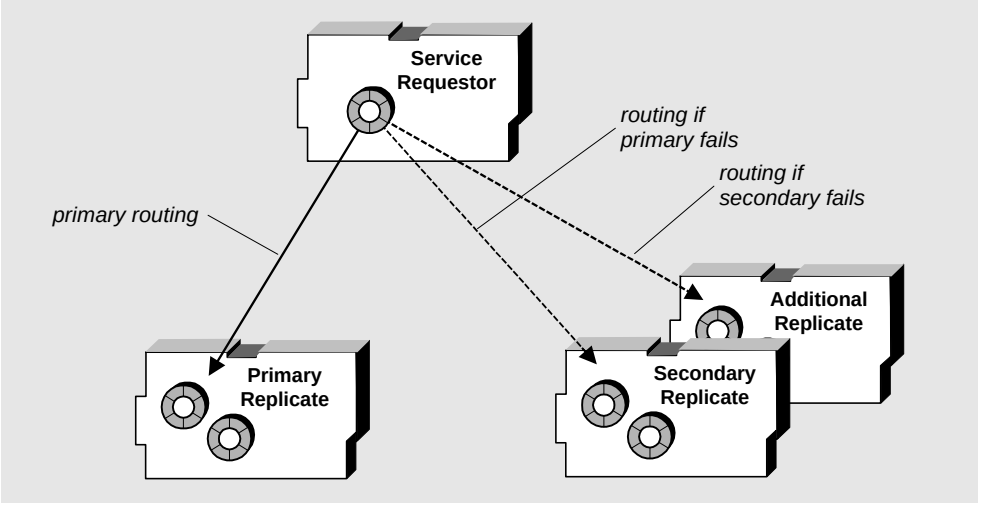
The following sections explain how you can manage replicated partitions to provide for failover and load balancing in iPlanet UDS applications.

Failover

When a developer chooses to replicate a service object for the purpose of failover, you usually assign the replicates of that partition to different nodes.

A typical scheme is illustrated in Figure 6-6. The application's partitioning configuration includes installed replicates of a partition on two additional servers. The secondary replicate on one server is designated as enabled and the additional replicate on the other server is designated as disabled.

Figure 6-6 Failover Scheme



When you start the application by invoking the Startup command of the Application agent, the enabled replicates are started. As requests for the service partition are made in the normal functioning of the application, the requests are directed to the primary replicate.

If the primary replicate fails, or if this server crashes, requests for the service partition are then directed to the secondary replicate. For failover protection, you can then start the additional replicate on the second server using the Startup command of the Installed Partition agent.

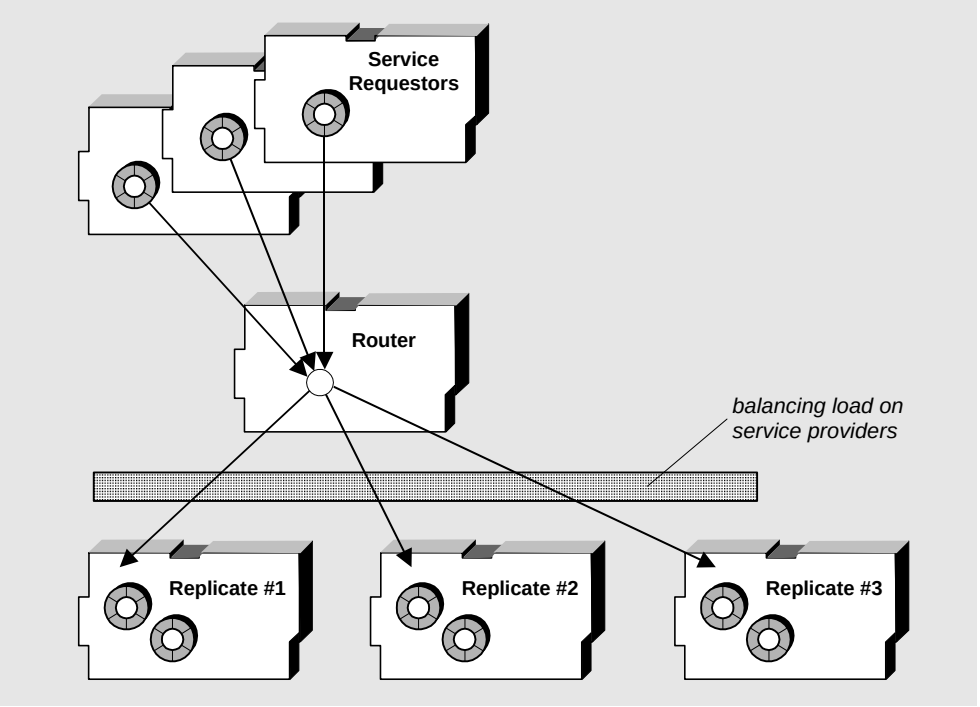
The specific replicates you designate as enabled or disabled depends on what you expect your normal failover requirements to be, taking into account the reliability of each server in your environment and the consequences of failure. The enabled/disabled property specifies the default startup configuration. Once your application is running, you can either start or shut down additional replicates to meet current failover needs.

Load Balancing

When a developer chooses to replicate a service object for the purpose of load balancing, iPlanet UDS automatically creates a router partition in addition to a server partition that can be replicated for load balancing. This router partition routes requests for a service among the replicates of the server partition that performs that service. Although you normally assign the router partition to the same node as one of the server partition replicates it is managing, you can place it on any server node in the environment.

A typical scheme is illustrated in Figure 6-7. The application's partitioning configuration includes partition replicates on three servers. The replicates of on two of the servers (Replicate #1 and Replicate #2) are designated as enabled and the replicate on the third server (Replicate #3) is designated as disabled.

Figure 6-7 Typical Load Balancing Scheme



When you start the application by invoking the Startup command of the Application agent, the enabled replicates are started, as well as the Router partition. As requests for the service partition are received in the normal functioning of the application, the router queues and then dispatches them, in the order received, to the next available running replicate.

Under normal conditions, this arrangement balances the load on the service partition. However, if you find performance lags at certain peak periods because these two active partitions become overloaded, you can start Replicate #3 on the third server using the Startup command of the Installed Partition agent. The fact that this partition was designated disabled only means that it did not get started when the application was originally started.

If startingReplicate #3 still does not resolve the performance bottleneck, you can start another replicate on any one of the three servers, exceeding the replication count originally assigned. The replication count does not limit your ability to start a partition, because it, as well as the enabled/disabled attribute, merely reflects what you set as the default, or normal, state of affairs.

The specific replicates you designate as enabled or disabled, and the number of replicates you designate for each server, depend upon what you expect your normal load balancing requirements to be, taking into account the power and usage of each server in your environment. The enabled/disabled attribute and the number of replicates attribute specify the default, or normal, startup configuration. Once your application is running, you can either start or shut down replicates to suit current performance needs.

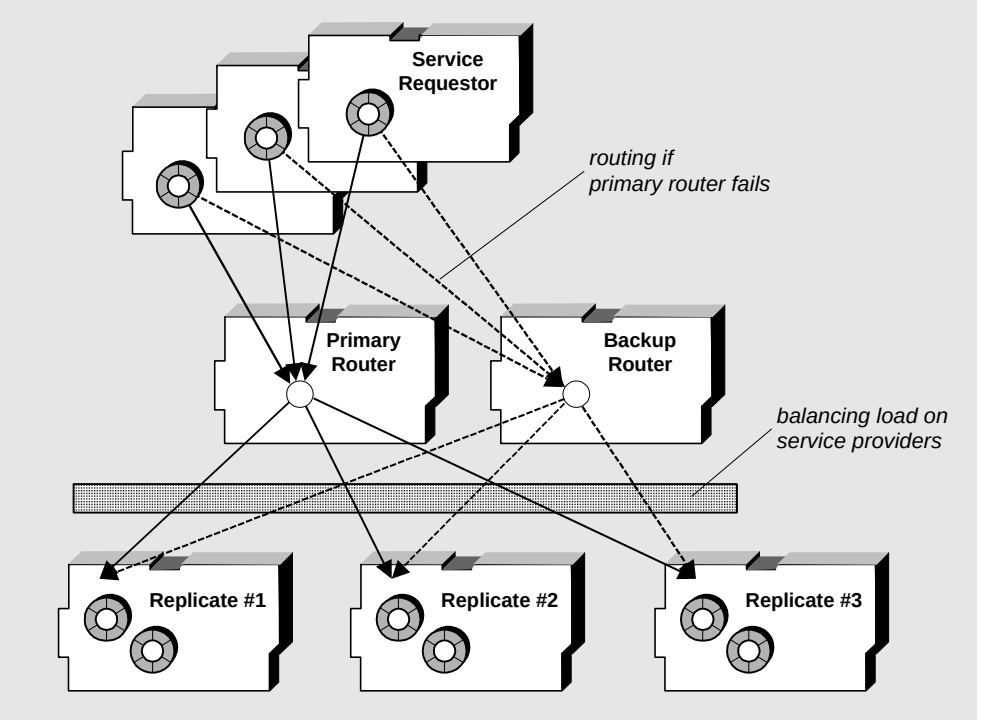
Failover and Load Balancing Combined

When a developer chooses to replicate a service object for the purpose of failover and load balancing, iPlanet UDS automatically creates a server partition that you can replicate for load balancing and a router partition that you can replicate for failover. In this way, if the primary router partition fails, a backup router can take over and manage the server partitions that are sharing the work load. You usually assign the replicated router partitions to different servers. A typical scheme is illustrated in Figure 6-8, in which the Router has two replicates (for failover), and each Router replicates deals with 3 replicates of a service partition for load balancing.

When you start the application by invoking the Startup command of the Application agent, the enabled replicates of the service partition are started, as well as both replicates of the Router partition. As requests for the service partition are received in the normal functioning of the application, the primary router queues and then dispatches them, in the order received, to the next available running service partition replicate.

Under normal conditions this arrangement balances the load on the service partition. If the node with the primary router crashes, however, the backup replicate of the router automatically takes over the task of balancing requests for the service partition.

Figure 6-8 Load Balancing Scheme with Failover



As in the cases of failover and load balancing considered separately in the previous two sections, you can start or stop additional replicates of the service partition or the Router, depending on the current load, server capacity, and the criticality of the situation. The basic principles of load balancing apply to the service partition replicates, while the principles of failover apply to the Router replicates.

Chapter 7

Troubleshooting

This chapter covers a number of topics relating to monitoring and troubleshooting problems in your iPlanet UDS environment and with your distributed applications.

A number of monitoring techniques are important to uncovering and diagnosing problems. This includes using log files, the Environment Console or `Escript`, and operating system commands to monitor your applications and their partitions.

A number of troubleshooting issues are discussed regarding memory, connectivity, and database access.

This chapter covers the following topics:

- backing up iPlanet UDS files
- logging and log files
- routine monitoring
- memory issues
- connectivity and database access issues

All procedures in this chapter will assume you are using the Environment Console. You can use the equivalent Escript commands to perform the same tasks, as described in *Escript and System Agent Reference Guide*.

Backing up iPlanet UDS Files

Backing up iPlanet UDS files on a regular basis can help you resolve problems—or at least recover from them, if you are unable to resolve them.

It is always a good idea to back up files that would be difficult to recreate. In general it is wise to back up your environment repository, application distributions, installed applications, and shared libraries. You might also back up specific log files. In a development environment, central repository files should definitely be backed up regularly.

The iPlanet UDS system directories you might consider backing up and their contents are reviewed below:

repos Development repositories are created and stored here. If a central development repository resides on a node, this repository should be backed up on a regular basis (see "Backing up Repositories" on page 270).

sysdata/envrepos Node repositories are created and stored here.On a central server, you should back up your environment repository before and after all major application deployments. The environment repository is the node repository for this node (*node_name*.btd and *node_name*.btx).

log Active partition, node, and environment log files are written here, and should be backed up if you want historical data on your system's performance.

appdist Application distributions and library distributions are created and placed here. Back up this directory if the distributions are not already on other media.

envdist This directory contains any environment definitions that were exported to.edf files. Back these up if they are important to you, but not already on other media.

userapp Application partitions are installed here. While these can be reinstalled from the distribution, if your environment repository is backed up, you may wish to also back them up. Libraries needed by an iPlanet UDS application are also installed here. Like installed partitions, you may wish to back them up even though they can be reinstalled from the library distribution.

Logging and Log Files

The iPlanet UDS runtime system can provide a broad range of logging information, giving you the flexibility to decide what information you want to log and where or how you want to log it.

You can have iPlanet UDS log various types of information to one or more log files. In general, the information falls into three categories: requested message output, instrument data, and audit traces.

Requested Message Output The iPlanet UDS runtime system and most iPlanet UDS applications generate many types and levels of messages. However, you have to specifically request that these messages be logged, either to a trace window or to specific log files. You can request message logging for any iPlanet UDS process, that is, any iPlanet UDS partition. In general, you can specify the name of one or more log files and set logging filters (or flags) that specify the type and level of messages you want to have logged to each.

Instrument Data You can decide to have instrument data—such as application resource usage or performance statistics—collected by iPlanet UDS for an active partition and written to the active partition log file (and the environment log file) at time intervals you specify. The logging of instrument data is determined by values you set for special logging instruments of the Active Partition agent.

Audit Traces Audit traces for important system events (such as installing and uninstalling applications, starting and stopping applications and partitions, and so forth) are automatically written to the appropriate log file (environment, Node Manager, or active partition), depending on the event. You cannot turn off the logging of this information.

The following table indicates what kind of logging information goes into each log file:

	Environment log	Active Partition log	Node Manager log
Message output		●	●
Instrument data	●	●	
Audit traces	●	●	●

Each iPlanet UDS partition has a default log file, called standard output, whose name depends on the iPlanet UDS partition for which logging is being performed. These names are shown in the table below:

iPlanet UDS Partition	Standard Output Log File in FORTE_ROOT/log/
Client partition	(screen trace window only)
Active standard partition	forte_ex_*process_ID*.log (for example, forte_ex_13456.log)
Active compiled partition	*filename_process_ID*.log
Launch Server	On UNIX, ftlaunch_*port*.log, where port is the socket number for the Launch Server. On Windows, screen trace window only.
Node Manager	*node_name*.log, where *node_name* is the first 8 characters of the node name.
Environment Manager	*node_name*.log, where *node_name* is the first 8 characters of the node name.
	environment_ID.log, where *environment_ID* is the first 8 characters of the environment name.

Because the Environment Manager serves as the Node Manager for its host node, while also performing environment-wide system management functions, two log files are created for the Environment Manager service. One is the standard Node Manager log. The other (*environment_ID*`.log`) is a log to which environment-wide audit traces and instrument data can be written.

Log file buffering Log files use normal operating system file buffering. To change this default behavior and cause log buffers to autoflush, set the logging flag to cfg:os:21. Setting this flag causes all log files to flush at that time, and from then on the files will flush after each write. To flush all log files for a active partition, use the Component > FlushLogFiles command on the Active Partition agent for that partition. You can also use the Flush method in the LogMgr class to flush all log files.

Changing Log File Names

You can change the default log file name, for the Environment Manager or an Active Partition agent. Changing the default log file name makes iPlanet UDS close the current log file and open a new file with the name you specify.

NOTE You can only specify log file names for Active Partitions agents for compiled server partitions and iPlanet UDS executor partitions. Client partitions can only log information to trace windows.

You can change the log file name for an Active Partition agent or the Environment Manager by navigating to the agent for either one and opening the Instruments window. You can then either change the instrument logging options by using the Instrument Logging Properties dialog or by directly changing the values of the instruments that control these options.

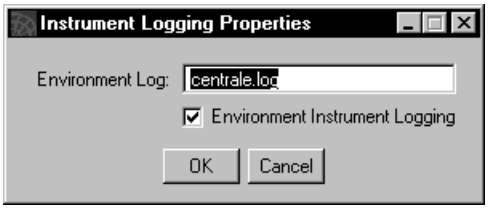
Rules for log file names When you specify the new name of the log file, you need to use a portable file name syntax (UNIX style). If the Log Filename does not start with a `/`, then the file is given relative to the `FORTE_ROOT/log` directory on the node on which the service is executing. If the Log Filename does start with a `/`, then it specifies an absolute path on the node on which the service is running.

The steps described in this section for changing the log file names using the Environment Console correspond to changing the values of the LogFile instrument for the Active Partition agent and changing the EnvironmentLog instrument for the Environment agent. For information about changing instrument values, see "Using Agent Commands" on page 208. For more information about these agents, see *Escript and System Agent Reference Guide*.

The steps for changing the log file name are different depending on whether you are changing the log file name for the Environment Manager, a compiled active partition, or an interpreted active partition. Although all active server partitions can log data to log files, compiled partitions each have their own log files, while interpreted partitions log data to the log files of their instances of the iPlanet UDS interpreter (iPlanet UDS Executor). Therefore, to change the log file name for an interpreted partition, you need to change the log name for the active partition of its iPlanet UDS Executor instance, as described below.

➤ **To change the log file name for the Environment Manager**

1. In the Environment Console's Active Environment window, open the Instruments window by selecting the File > Instruments command.
2. Open the Instrument Logging Properties dialog by selecting the File > Instrument Logging command.



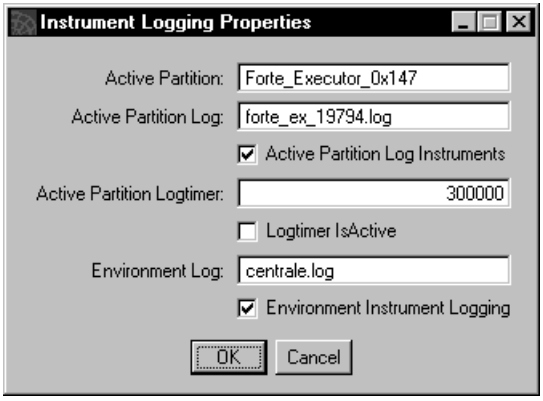
3. In the Instrument Logging Properties dialog, enter the new name of the log file in the Environment Log field.

You can change the log file name for the Environment Manager in any Instrument Logging Properties dialog. You can access an Instrument Logging Properties dialog from an Active Partition agent Instruments window or from the Instruments window of any subagents of an Active Partition agent.

➤ **To change a log file name for a compiled active partition or iPlanet UDS executor partition**

1. In the Environment Console's Active Environment window, select the View > Node Outline command.
2. Find the compiled Active Partition agent whose log file name you want to change by expanding the browser outline view.
3. Open the Agent window for the active partition by selecting the agent, then selecting the Component > Open command.
4. Open the Instruments window for the current agent by selecting the File > Instruments command in the current Agent window.

5. Open the Instrument Logging Properties dialog by selecting the File > Instrument Logging command.



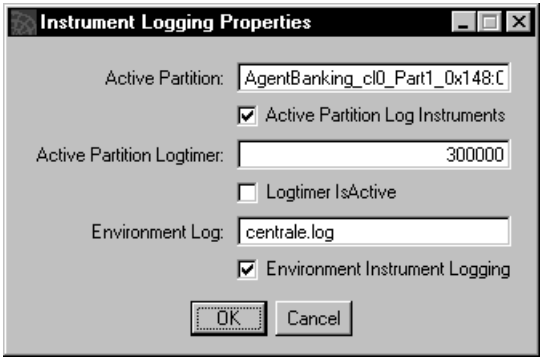
6. In the Instrument Logging Properties dialog, enter the new log file name for the active partition's log file in the Active Partition Log field. You can also change the name of the Environment Manager's log file by entering the new name of the log file in the Environment Log field.

➤ **To change the log file name for an interpreted active server partition**

1. In the Environment Console's Active Environment window, select the View > Node Outline command.
2. Locate the Forte_executor_*nodename* agent, which is a subagent of the Node agent for the node where the standard partition is running.
3. Click the expansion arrow next to the Forte_executor_*nodename* agent, then double-click the Active Partition agent whose name matches the name of the standard partition.

   The Agent window for the Active Partition agent opens.
4. Open the Instruments window for the Active Partition agent by selecting the File > Instruments command of the current Agent window.

5. Open the Instrument Logging Properties dialog by selecting the File > Instrument Logging... command the Instruments window.



6. In the Instrument Logging Properties dialog, enter the new log file name for the active partition and the instance of the iPlanet UDS executor in the Active Partition Log field. You can also change the name of the Environment Manager's log file by entering the new name of the log file in the Environment Log field.

Requested Message Output Logging

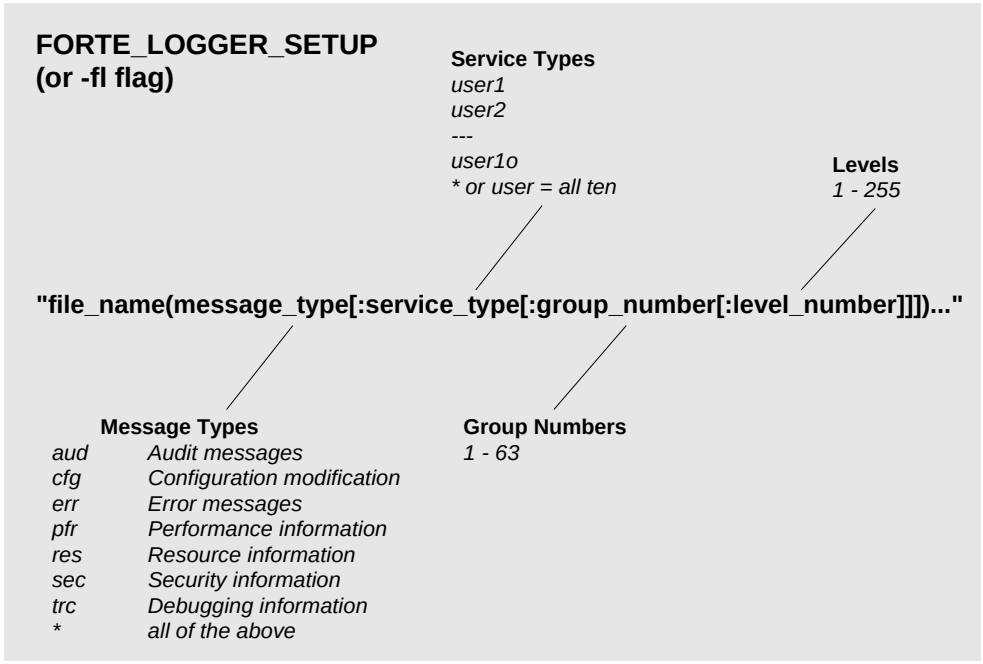
You normally specify requested message logging on a node by node basis, using the FORTE_LOGGER_SETUP environment variable, which is read when each iPlanet UDS partition starts on the node. You can override this environment variable value by setting message filters (`-fl`) in the startup command for any particular partition.

Many types of messages may be generated by the iPlanet UDS runtime system or by an application. Only those messages—or categories of message—you specify using the FORTE_LOGGER_SETUP environment variable or `-fl` flag, however, will be written to log files. For information about using the `-fl` flag, see "-fl Flag (Log Manager)" on page 371. For information about setting environment variables, see "Using the iPlanet UDS Control Panel" on page 345 and "Setting Environment Variables Without the iPlanet UDS Control Panel" on page 353.

Specifying Message Filters

The syntax for specifying your logging message filters is shown in Figure 7-1. Often you specify a standard output log file, "%stdout," as the log file name (see "Logging and Log Files" on page 217). You can specify more than one log file if you want different information written to different files. For each file you can specify any number of filters, each separated by a space.

Figure 7-1 Specifying iPlanet UDS Message Output Filters



For each log file you can specify four levels of filter, as follows:

Message type (mandatory) You can include more than one message type separated by spaces—the message types available are shown above.

Service type Each message type can be divided into a maximum of ten user-defined service types, which typically map to important application services. iPlanet UDS also has a number of service types, most of which are not documented.

Group number Each service type can be divided in turn into a maximum of 63 group numbers, which typically map to a group of facilities.

Level number Each group number can be subdivided into up to 255 additional levels.

The filtering hierarchy for messages generated by an application, or at least the most important logging message filters, should be documented by an application's developer. These filters are defined by application developers when they use the `task.Part.LogMgr.Put` or `PutLine` methods, defined on the Partition class in the Framework Library online Help.

An example message filter is as follows:

```
FORTE_LOGGER_SETUP: "%stdout(err:user trc:os:1:1) userlog(prf:user)"
```

This filter specifies two output files: standard output (which gets redirected by iPlanet UDS) and "userlog." Standard output will log two message types (Error and Debug), while userlog will log performance information.

Client nodes On a client node, to direct logging to a log file (in addition to a trace window on the screen), specify a log filename (in addition to %stdout) with the desired message filters in the FORTE_LOGGER_SETUP environment variable for the node.

iPlanet UDS, by default, sets a FORTE_LOGGER_SETUP environment variable:

```
FORTE_LOGGER_SETUP: "%stdout(err:sh)"
```

The err:sh setting has no particular significance other than to initialize the logging mechanism and open up standard output—you normally replace it with a setting of your own choice.

Setting the Logger Flag for a Partition

As mentioned earlier, you can override the FORTE_LOGGER_SETUP environment variable by setting a logger flag (`-fl`) in the startup command for any iPlanet UDS partition.

For an iPlanet UDS system management service or iPlanet UDS system application, such as the Environment Console, Escript, and ftexec, the flag is placed as a command option in the startup command.

For a partition in an application, you can set the logger flag using the appropriate properties dialog in the Environment Console (which utilizes the `SetArgs` command of the Partition agent).

➤ **To specify the logger flag for a partition**

1. Lock the active environment definition.
2. Select the Application Outline view in the Active Environment window.
3. Expand the Application to view its logical partitions.
4. Expand the logical partition of interest to view its assigned or installed partitions.
5. Select the assigned or installed partition of interest, then select the Component > Properties… command.
6. Enter the `-fl` logger flag arguments in the Server Arguments field.

Any partition when started will overwrite the existing log file for that partition. If you want to preserve the old log file, either back it up, or change the name of the file to which the process will log messages. (Active partitions rarely start with the same process ID number.)

For more information about using the `-fl` flag, see "-fl Flag (Log Manager)" on page 371. For information about setting the logger flag using the iPlanet UDS Control Panel, see "Log Flags Tab Page" on page 351.

Dynamically Modifying Message Filters

It is possible to dynamically change the current message filters for an active application or Node Manager partition. The modified message filters apply to the first log file specified in either the FORTE_LOGGER_SETUP environment variable or in the `-fl` logger flag in the startup command for the given partition.

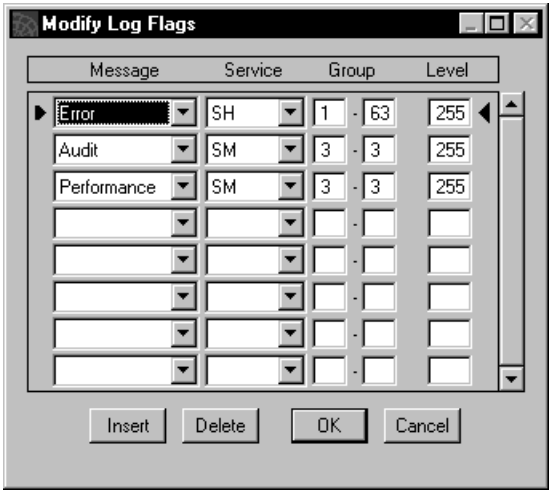
You can change the message types and levels using the Environment Console (or using the `Escript ModLoggerRemote` commands of the Active Partition and Node agents).

➤ **To dynamically modify the message filters for an active partition**

1. Select the Application Outline view in the Active Environment window.
2. Expand the Application to view its logical partitions.
3. Expand the logical partition of interest to view its installed partitions.

4. Expand the installed partition of interest to view its active partitions.

   Select the active partition of interest, then select the Component > Modify Log Flags command.



5. Enter whatever message types, service types, group numbers, and level numbers you wish as your new settings, and click OK.

   In this window, the Message field values represent the following message types:

Message Field values	Meaning	Type Used for -fl and FORTE_LOGGER_SETUP	Put or PutLine Constant
Audit	Audit messages	aud	SP_MT_AUDIT
Configuration	Configuration modification	cfg	SP_MT_CONFIGURATION
Error	Error messages	err	SP_MT_ERROR
Performance	Performance information	prf	SP_MT_PERFORMANCE
Resource	Resource information	res	SP_MT_RESOURCE

Message Field values	Meaning	Type Used for -fl and FORTE_LOGGER_SETUP	Put or PutLine Constant
Security	Security messages	sec	SP_MT_SECURITY
Debug	Debugging Information	trc	SP_MT_DEBUG

Useful Message Filters

A number of iPlanet UDS runtime system message filters you might find useful in diagnosing system management problems are described in the following table:

Filter	Function
trc:os:10	When this filter is set, the value of environment variables will be logged when any iPlanet UDS process is started.
trc:lo:25	Always set this filter when tracking down problems, otherwise key exceptions may not be displayed. But most users should not have this filter set because they may be alarmed at the number of harmless exceptions logged.
trc:cm:*:4	This filter is most useful if there are communication problems when first setting up your environment.
trc:os:1:1	Mostly used by developers to track object memory requirements. Logs automatic memory management activities.
trc:os:5:5	Mostly used by developers to track object memory requirements. Used with trc:os:1:1 to show objects.
trc:db:1–8	Used to diagnose problems in accessing a database.
trc:os:14	Used to show dynamically loaded libraries.
trc:rp:2:50	Used to show user operations on the local client that can affect the repository. trc:rp:2:75 provides more detailed information.

NOTE The trc message type corresponds to the Debug Message type available in the iPlanet UDS Control Panel.

For information about the meaning of the group and level numbers, see "Group Number Option" on page 373 and "Level Number Option" on page 374.

Instrument Data Logging

Instrument logging usually monitors an application's performance and resource usage, and can be a rich source of information about the underlying causes of performance bottlenecks or other types of problems.

You can write instrument data for an Active Partition agent, any of its subagents, or any other agents residing in the active partition to the active partition log file. You do this by designating those instruments you want to have logged. This can be instrument data of any agent in an active partition, including user-defined instruments of user-defined agents.

You also have to set a log timer interval at which designated instrument data is collected and then turn on the logging of that collected data. These operations are performed using instruments of the Active Partition agent, as described in some detail in "Tracking Instrument Data with Log Files" on page 203.

In addition, if you want this instrument data to be logged to the environment log file as well, you use the InstrumentLogging instrument of the Environment agent to turn this logging on, as described in *Escript and System Agent Reference Guide*.

Audit Trace Logging

Audit traces for important system events (such as installing and removing applications, starting and stopping applications and partitions, and so forth) are automatically gathered by iPlanet UDS and written to Node Manager log files and the Environment Manager log file.

The audit traces provide a source of information about the sequence of important system management events, and are therefore a useful source of information for discovering the state of your application when particular problems arise.

You cannot turn off the logging of audit trace information.

Routine Monitoring

This section describes how you can perform routine iPlanet UDS system monitoring.

Monitoring System Management Services

One of the first steps in approaching any system management problem is to make sure you have a functioning iPlanet UDS environment.

To function properly, an iPlanet UDS environment must have all its iPlanet UDS system management services up and running. There are two approaches to checking these services:

- going to the node on which the service should be running and use operating system commands to verify that the service is online
- using iPlanet UDS tools or utilities from any node to check the status of various system management services in the environment

Using the Operating System

To use operating system commands, you look for a process whose name corresponds to the system management service you are trying to check. The process names for iPlanet UDS system management services correspond to their startup commands and are shown in "Process Names" on page 109. For example, the Node Manager is identified by "nodemgr."

On UNIX platforms, you use some variant of the `ps` command. Be sure you use a flag that will avoid truncating output before showing the process name.

On OpenVMS platforms, you use the `show system` command.

On the Windows NT platform, you look at the Windows NT Task list.

Using iPlanet UDS System Management Tools

The other approach to checking which system management services are up and running is to use iPlanet UDS facilities.

Name Service

For example, you can use the Application View of the Active Environment window and check if the NameService agent is active.

If you cannot find the Name Service then you have three possible problems:

- Name Service is offline
- network link to the central server node is not functioning
- FORTE_NS_ADDRESS environment variable is not set correctly

Node Manager

There are a number of ways of checking whether a Node Manager is online:

- start the Environment Console and see if the node is online
- use `Escript` to look for the node with commands like `FindEnv`, `ShowAgent`, and `FindSubAgent`, as described in *Escript and System Agent Reference Guide*
- look for the "attached to manager..." message when starting any iPlanet UDS process on the node

Monitoring Application Partitions

There are a number of ways of monitoring an application for problems:

- monitoring the status of the application and its partitions through the Environment Console and `Escript`
- using operation system commands to directly look at the application's server partitions
- periodically checking log files to see if exceptions have occurred

Using iPlanet UDS System Management Tools

The Environment Console and Escript both report the status of applications and their partitions. For any object in the application view to be considered online, all its child objects must also be online.

When an application has a degraded status, it is not running as specified in its partitioning configuration. To find the source, or sources, of the degradation, you move down through the object hierarchy looking for the offline partitions. For more detailed information, see "Monitoring Status" on page 196 or *Escript and System Agent Reference Guide*.

Using the Operating System

You can also monitor server partitions by looking at the processes running them. If a process is not running, or is abnormally large or abnormally small (compared to others or compared to its normal size), it may indicate a problem.

To check the process size for standard partitions, look for instances of the "ftexec." This command is used to start the iPlanet UDS executor that runs the image repositories for a standard partition. For compiled partitions, look for the partition process by executable file name.

When you shut down an application, the iPlanet UDS executors that ran standard partitions remain online until you specifically shut them down. As a result, you might often find a number of iPlanet UDS_Executor partitions which remain "online," but are not "busy." You should shut down these orphaned ftexec processes if they impact the performance of their host node, otherwise you can leave them running to be used by an application you may soon start.

Monitoring Log Files

You can also check for problems in the running of an application by looking at log files. There is a log file for each active partition, each node, and for the environment as a whole, as described in "Logging and Log Files" on page 217.

These log files are normally written to the `FORTE_ROOT/log` directory, so you can scan the log files looking for exceptions that would indicate problems in the running of your application. (You can generally search log files while they are open.)

Using the iPlanet UDS Keepalive Feature

iPlanet UDS provides a keepalive feature that helps you and your applications to quickly detect network failures. This feature is very similar to TCP Keepalive, and it works in the following way:

1. The Communication Manager keeps track of how long a connection has been inactive.
2. If a connection has been inactive longer than a specified amount of time (the *keepalive cycle*), the Communication Manager pings the remote partition on this connection.

3. If the Communication Manager receives a reply from the remote partition, the Communication Manager allows the connection to continue and resets its record of how long the connection has been inactive.

   If the Communication Manager does not receive a reply from the remote partition within the specified amount of time (the *keepalive interval*), the Communication Manager either pings the remote partition again, or closes the connection. The number of times the Communication Manager pings a remote partition before it closes the connection is the *keepalive count*.

4. When the Communication Manager closes a connection, the Distributed Object Manager raises a DistributedAccessException object, which the application should handle to recover state information before failing over to another service.

The keepalive feature is off by default. To turn on the keepalive feature, set the length of time specified for the keepalive cycle to a value greater than 0.

Setting Keepalive Threshold Values with Environment Variables

iPlanet UDS defines the following environment variables for setting Keepalive threshold values:

FORTE_KEEP_COUNT Specifies the number of pings that the keepalive feature attempts before it closes the connection. The default value is `3`.

FORTE_KEEP_CYCLE Specifies the length of time, in seconds, that a connection can be inactive before keepalive processing starts. The default value is `0`. If this value is set to `0`, all keepalive processing is disabled. If a client has this value set to `0`, the client sends a message to each partition it connects to, telling that partition not to check on the client's connection using the keepalive feature.

FORTE_KEEP_INTERVAL Specifies the interval, in seconds, after a ping message is sent, during which a reply is expected. The default value is `10 seconds`.

Setting Keepalive Threshold Values Using the CommMgr Agent

The agent for the Communication Manager is the CommMgr agent. The CommMgr agent is an agent that manages the communications service for an active partition. The communications service provides access to low-level network connections in and out of a partition.

The CommMgr agent has the following instruments that let you monitor and tune the iPlanet UDS keepalive feature:

KeepAliveCloses Represents the total number of connections that have been closed by keepalive processing in a partition.

KeepAliveCount Specifies the number of pings that the keepalive feature attempts before it closes the connection.

KeepAliveCycle Specifies the length of time, in seconds, that a connection can be inactive before performing keepalive processing. Setting this value to `0` turns off the keepalive feature.

KeepAliveInterval Specifies the interval, in seconds, after a ping message is sent, during which a reply is expected.

For more information about these instruments, see the CommMgr agent information in *Escript and System Agent Reference Guide*.

Restrictions

Applications and partitions that block in the operating system can fail to respond to pings, which would cause the Communication Manager to close connections that are in use.

iPlanet UDS attempts to determine situations, such as database processing, graphical user interface applications in non-preemptive situations, and so forth, that are likely to cause an application to block. In these situations, iPlanet UDS disables the keepalive feature, regardless of the setting of the keepalive cycle.

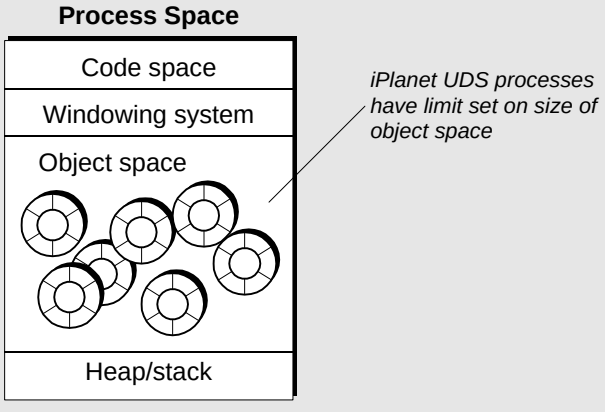
The following describes some of the situations where iPlanet UDS automatically disables the keepalive feature:

- running a Release 2.0 partition that connects to a Release 3.0 partition, or the reverse
- non-preemptive command line programs that can block in the operating system
- C wrappers that block in the operating systems, are in non-preemptive partitions, or are in projects that are marked as not being threadsafe

Memory Issues

iPlanet UDS generates an out of memory exception when a partition runs out of object memory space, shown in Figure 7-2. This can happen even though there may be sufficient process space to run the partition.

Figure 7-2 Object Memory Space



iPlanet UDS—not the operating system—controls the object memory space. By default, iPlanet UDS sets a maximum limit on the size of the object memory space. Within this limit, the iPlanet UDS memory manager attempts to optimize a partition's object memory space by deleting objects which are no longer referenced by an application.

When the memory manager reclaims memory, it looks at the objects in memory and determines which are still being referenced. If an object is referenced (there is a pointer to the object accessible to the application) then the object is retained. If there is no such pointer, the object is discarded.

The memory manager performs two types of memory reclamations:

Generational In this type of reclamation, the memory manager collects only among objects created since the last reclamation (the newest generation of objects). It performs a generational reclamation when half the memory available after the last reclamation has been allocated. This type of reclamation is relatively quick and efficient.

Stable In this type of reclamation, the memory manager collects among objects of all previous generations. It performs a stable reclamation when the utilization percent after a generational reclamation falls above a target utilization percent. This type of reclamation is relatively more resource intensive than a generational memory reclamation, and generally reclaims more memory.

The memory management scheme is designed so that in most cases, the memory manager performs a generational reclamation. If a generational reclamation does not free enough memory, the memory manager performs a stable collection to try to free more space. And if a stable reclamation does not free enough memory, the memory manager can expand the object space allocated to the partition.

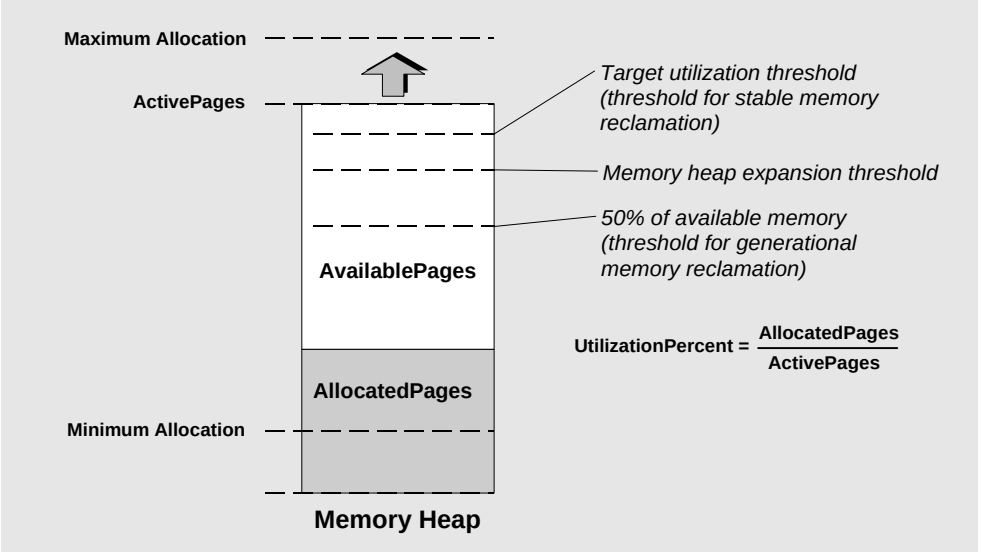
The values that determine when stable reclamation and memory expansion occur are instruments of the OperatingSystem agent. These can be initially set by the `-fm` startup flag for the partition and modified using agent commands. The SetIntProperty and GetIntProperty methods on the OperatingSystem class (described in the Framework Library online Help) also allow a TOOL program to set and get the values.

For example (see Figure 7-3 on page 236), when an iPlanet UDS partition starts up, the memory manager sets the object memory heap to a minimum value (the MinimumAllocation—by default, 1024 pages). Once half of that available memory is allocated to objects, the memory manager performs a generational memory reclamation. It then records the amount of available memory remaining in the memory heap (the AvailablePages). Once half of *that* memory is used, the memory manager performs another generational memory reclamation, and so on.

If, after a generational reclamation, the utilization falls above a target utilization threshold (by default, 85%), the memory manager then performs a stable reclamation. After a stable reclamation, the memory manager records the value of allocated memory (the PeakAllocatedPages). If the corresponding utilization falls above an expansion threshold (ExpandAtPercent—by default, 80%), the memory manager performs a memory expansion. This expands the memory heap (the ActivePages) by a set amount (the ExpandByPercent—by default, 10%).

This process continues until the partition is shut down, the maximum amount of memory (the MaximumAllocation) is reached, or the operating system cannot supply additional memory.

Figure 7-3 Memory Management Instrument Values (OperatingSystem Agent)



Changing Memory Settings

If a partition requires more than the default maximum memory allocation, you have to increase this limit at partition startup time. You can change the minimum and maximum memory allocations (and many other memory management settings) using the `-fm` memory flag when starting an individual service or partition. You can also change these values while a service or partition is running by using the MaximumAllocation and MinimumAllocation instruments of the OperatingSystem agent, as described in *Escript and System Agent Reference Guide*.

Default allocations for two important iPlanet UDS processes are:

Partition	Minimum Object Memory	Maximum Object Memory
ftexec	2000 Kbytes	10000 Kbytes
nodemgr	4000 Kbytes	8000 Kbytes

Specifying Object Memory Flag

A few of the memory flags you can use your object memory allocation are:

(n:*min_size*, x:*max_size*)

Parameter	Description
min_size	Minimum object memory size—the amount first allocated to the partition at startup
max_size	Maximum object memory size—the amount to which object memory can grow

For information about all the values that can be used with the `-fm` flag, see "-fm Flag (Memory Manager)" on page 375.

If a partition requires more than the default object memory, you can use the `-fm` flag in the command line to increase the allocation at startup. Likewise, if a Node Manager or other process requires much less, you can use the `-fm` flag to reduce the allocation at startup.

You might find two logging message filters helpful in monitoring the use of object memory by a partition. The output indicates statistics before and after iPlanet UDS performs automatic memory management—a process in which iPlanet UDS frees up space occupied by unused objects.

Filter	Function
trc:os:1:1	When this filter is set, summary information on number of pages and objects in memory before and after automatic memory management will be logged.
trc:os:5:5	When this filter is set, detailed information on number of pages and objects in memory before and after automatic memory management will be logged.

Setting the -fm Flag for a Partition

You can set a memory flag (`-fm`) in the command you use to start up any iPlanet UDS partition.

For an iPlanet UDS system management service or iPlanet UDS system application, such as the Environment Console, `Escript`, `ftexec`, `ftcmd`, and so forth, you can specify the `-fm` flag as a command option for the command that starts the service or application.

If you specify the `-fm` flag when you start up a partition, the iPlanet UDS system checks whether any idle running instances of the interpreter (ftexec) have enough allocated memory to run this partition. If so, iPlanet UDS runs the partition on that instance of the interpreter. If not, iPlanet UDS starts a new instance of the interpreter that allocates the maximum amount of memory required for the partition, as defined on the `-fm` flag.

For a partition in an application, you can set the `-fm` flag using the appropriate properties dialog in the Environment Console, as described in the following steps.

➤ **To specify-fm flag for a partition**

1. Lock the active environment definition.
2. Select the assigned or installed partition of interest in the Active Environment window, then select the Component > Properties command.
3. Enter the `-fm` memory flag and its arguments in the Server Arguments field.

You can set the memory flag in `Escript` using the `SetArgs` command of the Logical Partition agent, or in `Fscript` using the SetPartArgs command.

For more information about setting the `-fm` flag, see "-fm Flag (Memory Manager)" on page 375.

Thread Stack Size

In some cases you may need to override the default FORTE_STACK_SIZE settings, which sets the thread stack size in bytes for iPlanet UDS and Posix threads. iPlanet UDS applications with deep levels of nested function calls, applications that call-out to C or C++ libraries, or auto-compilation of large applications may require that you override the default FORTE_STACK_SIZE settings.

The size needed for the thread stack depends on the depth of the call graph of the thread running on the stack and the size of the local data for the methods and procedures in the call graph. For example, if an application makes heavy use of recursion, the depth of its call graph can get quite deep, requiring a correspondingly large amount of run-time stack space. Also, if a method or procedure uses a large amount of local data (for example a large local buffer) the amount of run-time stack required grows.

iPlanet UDS has identified some situations where third party software used by iPlanet UDS applications require changing the default stack size. Motif clients, some database products, and several C APIs require that FORTE_STACK_SIZE be set to a larger than default value. Both client and server partitions may require increasing FORTE_STACK_SIZE.

Setting the Thread Stack Size

To override the default FORTE_STACK_SIZE you can either set the FORTE_STACK_SIZE environment variable before running the application or specify the `-fst` flag during startup.

For example, you can specify `-fst 100000` to override the stack size for a partition.

NOTE If both `-fst` and FORTE_STACK_SIZE have been set, then the larger of the two values is used.

If you need to increase the thread stack size, you may want to start with a value of 100000. Some applications may require a larger stack size than this; for example, code generation and auto-compilation of a very large application may require stack sizes over 200000. However, a larger stack size means your applications use more dynamic memory. In addition, a larger stack size may reduce the number of POSIX threads that a partition can create under AIX.

Connectivity Issues

You might encounter connectivity problems when you set up your iPlanet UDS environment or add additional nodes to an existing environment.

Setup

It is important to separate iPlanet UDS related problems from general network malfunctions. The best approach is to establish that your network is up and running, independent of iPlanet UDS. Consider the following:

- Does server have network problems? Try ftp, telnet, netcopy, ping.
- Are you using a Domain Name Server?

If the network appears to be running properly, you can check iPlanet UDS connectivity by trying to open the Environment Console or `Escript`. Check that all FORTE_NS_ADDRESS environment variables are set properly, as described in "Environment Manager Failover for Partitions" on page 122.

If iPlanet UDS connectivity proves difficult to establish, set the communication message filter, trc:cm:*:4, and try to diagnose the problem. Try it on the client side first and, if necessary, the server side.

Sometimes multiple network protocols collide on PC/Windows nodes. In this case it might help to reinstall iPlanet UDS in a "clean" environment.

Ongoing

At times, connectivity issues arise after your environment has been successfully set up. In this case, your troubleshooting should include checking that:

- the network is working properly, independently of iPlanet UDS
- iPlanet UDS system management processes are all up and running
- no two users (usually clients) are colliding by trying to use the same network address
- you are not trying to connect to too many remote partitions from a PC client. The PC supports a limited number of connections at one time.

Database Access Issues

With database access problems, as with connectivity problems, you need to separate iPlanet UDS related problems from general database malfunctions. The best approach is to establish that your database is up and running independent of iPlanet UDS:

- Check that the database is running.
- Use the interactive database monitor to mimic as closely as possible the application database access.
- Check database locks from the database monitor.
- Try the DynamicSQL example iPlanet UDS application to verify that iPlanet UDS can access the database.

NOTE Be careful that an application references database resource managers by names that are the same as those you have set in your environment. Be sure the application developers provide you with any resource manager names referenced in their application code.

Another possible issue, in Oracle environments, is that SQL*Net must be running for access to more than one database to work properly. SQL*Net support is built into the iPlanet UDS product.

If all else fails, use the database message filters to diagnose the problem. Database filters and the type of messages they will log, shown below, must be set in the partition that is running the database session.

Filter	Function
trc:db:1	Cursor operations
trc:db:2	SQL being sent to the database
trc:db:3	Trace SQL Prepare statements
trc:db:4	Not used
trc:db:5	Transaction tracing (begin/commit/abort)
trc:db:6	Database session startup/shutdown
trc:db:7	Trace locking done in DBSession
trc:db:8	High level method tracing

For more information about setting up databases, see *Accessing Databases*.

Chapter 8

Managing iPlanet UDS Development Repositories

This chapter provides background information about iPlanet UDS development repositories and describes the iPlanet UDS repository utilities you use to create and manage them.

With this release, iPlanet UDS provides a new type of repository format, called the B-tree repository. This chapter explains how to create, maintain, and run repositories in the B-tree repository format.

This chapter also explains how to use secure repositories.

About iPlanet UDS Development Repositories

There are two types of development repositories: central repositories and private repositories. *A Guide to the iPlanet UDS Workshops* describes the development repositories that iPlanet UDS application developers use to create and store applications.

NOTE In a mixed node environment, where some nodes are running under an iPlanet UDS system prior to release 5.0, repository files for the two systems may be incompatible. For information on upgrading systems and maintaining compatibility with prior releases, refer to the Technote, "Upgrading to iPlanet UDS 5.0 in Mixed-Node Environments," available from iPlanet technical support.

This section provides background information about central and private repositories.

Central repository A *central repository* is a repository shared by a group of developers in your organization. This repository contains all the iPlanet UDS libraries and all the other plans integrated into it. Most of the time, iPlanet UDS application developers do their work using a central repository. Central repositories are described in detail under "About Central Repositories" on page 244.

Private repository A *private repository* is an independent repository that a developer can use without having to interact with the central repository in a distributed environment. If the developer is running iPlanet UDS in stand-alone mode, he can access a private repository, although he cannot access a central repository. This repository can be in the iPlanet UDS distributed development environment, but does not have to be. Private repositories let a single programmer work completely independently of other developers. Private repositories are described in detail under "About Private Repositories" on page 247.

Standard repository A *standard repository*, by default, requires no passwords. However, you can set a master password, passwords for each workspace, and a baseline password to prevent unauthorized users from changing information in the repository. These passwords are for updating only; any user can still read the information in the repository.

Secure repository A *secure repository* requires passwords for reading information from the repository, changing information in the repository, using workspaces, creating new workspaces, and copying the repository.

About Central Repositories

A central repository is a repository shared by a group of developers in your organization. This repository contains all the iPlanet UDS libraries and all the other plans integrated into it.

A central repository allows for centralized access to plans, which lets multiple developers collaborate. Normally, all plans being shared by a department or development team are stored in a single repository. Central repositories are always on server nodes in the environment, so that all developers who need to access the repositories can use them.

iPlanet UDS's installation program automatically creates a single central repository for the environment (see your *iPlanet UDS System Installation Guide* for information). However, there can be any number of central repositories within a single environment. As the iPlanet UDS system manager, you create the central repositories needed by the iPlanet UDS developers you are supporting.

Repository servers Every central repository is associated with a repository server. The repository server manages the repository and coordinates all access to it. After creating a central repository, you must start the repository server to allow multiple developers to access the repository. You must stop the repository server process when you wish to perform maintenance on the central repository. Each time you start the repository server, you assign a name to it (the repository server name). You and your application developers use the repository server name to refer to the specific process in subsequent commands, for example, to copy the repository or to open the repository for use with the iPlanet UDS Workshops.

The system manager needs to do the following tasks for central repositories:

- create and copy central repositories for iPlanet UDS application developers
- start the repository servers, which enables the developers to access the central repositories
- stop the repository servers to perform maintenance on the central repositories or for other reasons
- maintain central repositories by compacting them, backing them up, recovering them if necessary, and tuning their performance

Repository sessions Any application that accesses a central repository (such as the Repository Workshop) does so by establishing a session with the corresponding repository server. This session is called a *repository session*. The repository server uses repository sessions to manage repository workspaces and control access to the repository.

Shadow repositories Because a central repository is located on a central server in the development environment, iPlanet UDS provides a special feature called a *shadow repository* that lets an application developer significantly improve the performance of the central repository by creating a local copy of part of the repository on her workstation, while still maintaining a repository session. For more information, see "Creating Shadow Repositories" on page 258.

Repository format Central repositories are B-tree repositories. This repository format is identical across platforms, and is used for central repositories as well as private repositories, shadow repositories, and environment repositories.

About Shadow Repositories

A shadow repository is a local copy of part of a central repository. The shadow resides on a developer's workstation but maintains a connection to the central repository. When a developer first uses a shadow repository, only the data he needs is cached locally. When the shadow is first created (usually by the application developer), it is essentially empty. As he begins to work in it, creating new plans or components, modifying existing components, or even just browsing through the components in the plans, the relevant parts of the repository are copied to the shadow.

A shadow repository is either attached to a central repository or detached from it.

Attached shadow An *attached shadow* repository is connected directly to the central repository. Because this is the most reliable way to work, most of the time, programmers do their development work in an attached shadow.

When working with an attached shadow, the developer has the option of saving changes in the local shadow without saving to the central repository. When the developer sets his preferences so that changes are not automatically committed to the central repository, changes he saves to a workspace are not written to the central repository until he gives an explicit Commit to Central command, exits from the iPlanet UDS Workshops, or invokes a command like `IntegrateWorkspace`, which affects the central repository. Although this is not the most reliable way to work, it provides improved performance for the repository.

A developer can detach the shadow from the repository at any time.

Detached shadow A *detached shadow* repository is disconnected from the central repository. Changes that a developer makes in the detached shadow are not made to the central repository until she re-attaches the shadow to the central repository. A detached shadow is useful for allowing a developer to take her work home with her or for working outside the distributed development environment. Also, if many developers are sharing one central repository, some of the developers might want to detach their shadows to reduce the load on the repository server and improve repository performance for all.

When the developer has finished working outside the distributed environment, she can bring the detached shadow back into work, and attach it back to the central repository. Attaching her shadow to the central repository copies all her changes from the shadow to the central repository.

Normally, developers are responsible for creating and maintaining their own shadow repositories (see *A Guide to the iPlanet UDS Workshops* for information about creating and maintaining shadows). However, you can also create shadow repositories from a script or command line using the `rpshadow` command described in "Creating Shadow Repositories" on page 258.

About Private Repositories

A private repository is a single-user, single-access development repository that runs on only one node. Private repositories are designed for independent, high-performance development inside or outside an iPlanet UDS distributed environment. A private repository provides somewhat better performance than an attached shadow; however, it does not provide the collaboration facilities and centralized access provided by a central repository.

A developer can use a private repository to build an entire distributed application on his own. He can also use it to build part of an application, and then export component and plan definitions into a central repository.

Private repository formats iPlanet UDS uses a B-tree format for private repositories on all platforms.

About Repository Security

In terms of security, iPlanet UDS supports two kinds of repositories: standard repositories, and secure repositories.

Standard repository A standard repository, by default, requires no passwords. However, you can set a master password, passwords for each workspace, and a baseline password to prevent unauthorized users from changing information in the repository. However, any user can still read the information in the repository. You can create standard repositories by copying the B-tree seed (btseed) files from the `FORTE_ROOT/install/reposcpy` directory or by using the `rpcreate` or `rpcopy` commands.

Secure repository A secure repository requires passwords for reading information from the repository, changing information in the repository, using workspaces, creating new workspaces, and copying the repository. You can create secure repositories by using the `rpcreate` or `rpcopy` commands with the `-secure` flag.

File permissions You can also change the file permissions to restrict access to the repository by various users. By making the file permissions more restrictive, you can prevent unauthorized users from accessing the repository files directly to read the source code using various utilities.

Security for Standard Repositories

By default, standard iPlanet UDS development repositories do not require passwords. However, you can set the following passwords to provide a level of security for the repository:

master password Provides global access to all password-protected functions. You can set the master password using the `rpstart -p` flag or the Fscript `SetPassword` command.

baseline password Prevents unauthorized users from integrating a workspace into the system baseline in the repository. You can set the baseline password in the Repository Workshop using the File > Set Baseline Password command or in `Fscript` using the `SetPassword` command.

workspace passwords Restricts access to an existing workspace. You can set workspace passwords when you first create new workspaces. You can also set workspace passwords in the Repository Workshop using the File > Set Workspace Password command or in Fscript using the `SetPassword` command.

The Fscript commands are described in *Fscript Reference Guide*.

Security limitations of standard repositories Even if you use the master, workspace, and baseline passwords, you cannot prevent unauthorized users from reading and copying source code from the repositories in the following ways:

- copying the repository using `rpcopy`

  Users can create their own copies of repository files by using `rpcopy` on a running repository server, even if file permissions prevent them from accessing the original repository files.

- creating a new workspace.

  Users can then read any public plans in the repository.

- directly parsing the contents of the repository files

If you are concerned about this level of security, consider using secure repositories, as described in the next section.

Security for Secure Repositories

A secure repository requires passwords for reading information from the repository, changing information in the repository, using workspaces, creating new workspaces, and copying the repository.

When you create a secure repository, you must set all of the following passwords:

administrator password prevents unauthorized users from copying the repository using `rpcopy` or creating new workspaces.

master password provides global access to all password-protected functions.

baseline password prevents unauthorized users from integrating the workspace into the system baseline in the repository.

workspace password restricts access to the FirstWorkspace workspace.

For information about setting these passwords when you create a secure repository, see "rpcreate Command" on page 253.

By default, the files of secure B-tree repositories have their permissions set so that only authorized users can read or change them.

However, on the Windows NT and Alpha NT platforms, these file permissions are not set. To prevent unauthorized users from reading the files directly on NT platforms, you can disable file sharing for the drive containing the repository files or use the File Manager to set the security attributes for the files.

About the B-tree Repository Format

The B-tree repository format consists of the following files:

File Name	Purpose
repository.btd	Contains the data for the repository.
repository.btx	Contains an index for the data.
repository.rop	For central and private repositories, contains historical information about activities performed on the repository since its creation.

B-tree repositories run transparently under iPlanet UDS, using no special processes of their own.

B-tree files are completely portable across all platforms that iPlanet UDS supports. You can move them from machine to machine as you wish using a binary copy. You can also change their directories and file names using standard operating system commands. If you change the file names of the B-tree files, the .btd, .btx, and .rop files must share the same name, and you must be sure that the data file ends in ".btd", and the index file ends in ".btx".

CAUTION Do not run B-tree repositories over NFS mounted disks.

Specifying a B-tree repository In iPlanet UDS command lines, you refer to a B-tree repository by specifying its format, its location, and its name (without the .btd or .btx extension), using the following syntax:

bt: *full_path_name*

The following examples illustrate how to specify a B-tree repository name:

Platform	Command Syntax
UNIX	bt:$FORTE_ROOT/repos/myShadow
VAX/VMS	BT:FORTE_ROOT:[REPOS]MYSHADOW
Windows	bt:c:\forte\repos\myshadow

Recovering a repository to a consistent state If a program that accesses the repository terminates abnormally, iPlanet UDS saves the uncommitted changes in a file called *repository*`.btb`, where *repository* is the same name as that of the affected B-tree repository.

The next time iPlanet UDS opens this B-tree repository, it returns the repository to the consistent state that existed when changes were last committed. Any changes made to the repository since the last commit point are discarded. You cannot recover uncommitted changes.

To copy a B-tree repository that has a .btb file, do one of the following:

- recover the repository by opening the repository, then copy its .btd and .btx files
- copy the .btx, .btd, and .btb files to the new location, then recover the repository by opening it
- use the `rpcopy` command to create a copy of the repository in a new location

If you try to use the repository without letting iPlanet UDS recover the repository with the information in the .btb file, your repository will be in an unknown state.

Repairing damaged index files iPlanet UDS automatically rebuilds missing index files for B-tree repositories when you open a B-tree repository. If you ever get a message that indicates that the B-tree index file has been damaged or that asks you to rebuild the index, delete the index (.btx) file and open the repository using any iPlanet UDS process, including the `rpstart` command or an Fscript `Open` command. iPlanet UDS automatically rebuilds the index.

Creating Repositories

This section explains how to create two types of repositories

- private and central repositories, which store source code
- shadow repositories, which is a local cache containing part of a central repository

The following sections describe how to create each type of repository.

Creating Private and Central Repositories

Private and central repositories are both B-tree repositories and use the same repository format. The only difference between a central repository and a private repository is the way you use them, as described in "About Central Repositories" on page 244 and "About Private Repositories" on page 247.

To create a private or central repository, you can do one of the following:

- Copy B-tree repository seed files to create a new, empty repository. This is the only way to create an empty repository on a node running Windows 95.
- Copy the files for an existing B-tree repository to create a new repository that contains the same source code as the existing B-tree repository. This is a way to create a new repository on a node running Windows 95.
- Use the `rpcreate` command to create a new, empty repository. You cannot use this command on a node running Windows 95.
- Use the `rpcopy` command to make a copy of an existing central or private repository. You cannot use this command on a node running Windows 95.

The following sections describe these procedures.

For a central repository, the new central repository must be on the Environment Manager node or another server node, but never a client node.

Starting the repository server Note that after creating a central repository, you must use the `rpstart` command to start the repository server for the central repository. See "Starting Central Repository Servers" on page 262 for information.

Copying Repository Seed Files

iPlanet UDS provides two B-tree repository seed files that you can copy to create a new, empty repository. Simply use your operating system binary copy commands to copy the seed files into the FORTE_ROOT/repos directory, changing their names in the copy command.

The seed files are:

- `FORTE_ROOT/install/reposcpy/btseed.btd`
- `FORTE_ROOT/install/reposcpy/btseed.btx`

The new B-tree files for the repository are completely portable. You can use them on any platform, and use a binary copy to move them from one platform to another as desired.

You can place repositories in any directory. However, you need to specify the full path when you reference repositories that do not reside in FORTE_ROOT/repos. For example, you can identify a repository in the FORTE_ROOT/repos directory using "bt:MyRepos", whereas you need to identify a repository in another directory with its full path, as in "bt:/Code/Shadow/MyRepos" on a Unix node.

Copying Repository Files

To copy an existing private repository, simply use your operating system copy commands to copy the following B-tree repository files to the appropriate location:

- *repository*.btd
- *repository*.btx

Before copying the B-tree files, make sure that no application developers are using the repository.

The copied B-tree files for the private repository are completely portable. You can use them on any platform, and move them from one platform to another as desired.

If you are copying a B-tree repository that has a .btb file, do one of the following:

- Recover the repository by opening the repository, then copy its .btd and .btx files.
- Copy the .btx, .btd, and .btb files to the new location, then recover the repository by opening it.

CAUTION If you try to use the repository without letting iPlanet UDS recover the repository with the information in the .btb file, your repository will be in an unknown state.

rpcreate Command

The `rpcreate` command creates a new, empty repository. You must use the `rpcreate` command on the node where you want the new repository.

The syntax of the `rpcreate` command is:

Portable (all platforms)

rpcreate -fr *target_repository_name* [**-r**]
[**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*] [**-secure**]

OpenVMS

VFORTE RPCREATE
/**REPOSITORY=***target_repository_name*
[/**REPLACE**]
[/**MEMORY=***memory_flags*]
[**/STACK**=*integer]*
[/**LOGGER=***logger_flags*]
[**/SECURE**|**NOSECURE'**]

The following table describes the `rpcreate` command flags:

Flag	Purpose
-fr *target_repository_name* **/REPOSITORY=** *target_repository_name*	Specifies the name of the new repository. To specify the name, specify "bt:" and a full path as part of the *target_repository_name*, for example, "bt:d:\myreposdir\myrepos" on Windows NT. If you do not specify a full path, then the repository is placed in FORTE_ROOT/repos/.

Flag	Purpose
-r **/REPLACE**	Specifies that the new repository replaces any existing repository with the same name. If you do not specify this flag and a repository with the same name already exists, you will get an error.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags in the `rpcreate` command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.
-secure **/SECURE**	Specifies that the new repository is a secure repository.

Creating a secure repository To create a secure repository, use the `rpcreate` command with the `-secure` flag. For example, to create a secure B-tree repository named SecretProject, you could use the following command:

```
rpcreate -fr bt:SecretProject -secure
```

When you invoke the `rpcreate` command with the `-secure` flag, you are prompted for the following passwords in order:

- administrator
- master
- baseline
- FirstWorkspace workspace password

You must define all these passwords when you create a secure repository.

rpcopy Command

The `rpcopy` command makes a copy of an existing central or private repository. You must use the `rpcopy` command on the node where you want the new repository. The syntax of the `rpcopy` command is:

Portable (all platforms)

rpcopy -s *source_repository_name* **-fr** *target_repository_name* [**-r**]
[**-fns** *name_server_address*] [**-fs**] [**-fm** *memory_flags*]
[**-fl** *logger_flags*] [**-secure**] [**-nonsecure**]

OpenVMS

VFORTE RPCOPY
/SOURCE_REPOSITORY=*source_repository_name*
/**REPOSITORY=***target_repository_name*
[/**REPLACE]**
[/**NAMESERVER=***name_server_address*]
[/STANDALONE]
[/**MEMORY=***memory_flags*]
[/**LOGGER=***logger_flags*]
[**/SECURE |/NOSECURE**]

The following table describes the `rpcopy` command flags:

Flag	Purpose
-s *source_repository_name* **/SOURCE_REPOSITORY=** *source_repository_name*	Specifies the source repository to copy. Specify a repository name, and be sure to use the appropriate syntax for the format of the existing repository. A B-tree repository uses “bt:” as the identifier. Specify a full path as part of the *source_repository_name*, for example, “bt:d:\myreposdir\myrepos” on Windows NT. If you do not specify a full path, then the repository is assumed to be in FORTE_ROOT/repos/.
-fr *target_repository_name* **/REPOSITORY=** *target_repository_name*	Specifies the name of the new repository to create. Specify “bt:” as the identifier and a full path as part of the *target_repository_name*, for example, “bt:d:\myreposdir\myrepos” on Windows NT. If you do not specify a full path, then the repository is placed in FORTE_ROOT/repos/.

Flag	Purpose
-r /REPLACE	Specifies that the new repository replaces any existing repository with the same name. If you do not specify this flag and a repository with the same name already exists, you will get an error.
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-fs **/STANDALONE**	Specifies that this command makes a new repository by copying an existing repository without connecting to an environment. The existing repository must be on the local node, and a repository server must not be using the repository.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags in the `rpcopy` command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.

Flag	Purpose
-secure **/SECURE**	Specifies that the new copy of the repository is a secure repository. You only need to specify this flag when you are making a secure copy of a standard repository. By default, the copy of a secure repository is also a secure repository with the same set of passwords and the copy of a standard repository is also a standard repository with the same set of passwords, if any. You are prompted for the administrator password, as well as for any passwords that are not set in the original standard repository.
-nonsecure **/NOSECURE**	Specifies that the new copy of the repository is a standard repository, with no administrator password. Other passwords that have been set in the original repository are set with the same values in the copy. You only need to specify this flag when you are making a standard copy of a secure repository. By default, the copy of a secure repository is also a secure repository with the same set of passwords and the copy of a standard repository is also a standard repository with the same set of passwords, if any.

For example, to create a copy of a B-tree repository in another directory, you can invoke the `rpcopy` command as shown:

```
rpcopy -s bt:centralstource -fr bt:e:\backups\centralsource
```

You can make a new central repository by copying an existing central or private repository using the `rpcopy` command. The new central repository will contain the exact contents of the repository that you copied, including plans, workspaces, and passwords. Remember, after you create the new central repository, you must use the `rpstart` command to start the repository server associated with it.

If you are copying a central repository that has shadow repositories associated with it, the shadows are not copied to the new central repository. The shadows for the source repository cannot be used with the new repository. To retain work in existing shadow repositories, re-attach these shadows to the central repository before you copy the central repository. Your developers will have to create new shadows for the new copy of the repository as they need them.

When the source repository you are copying is a central repository, and the repository server is running for the source repository (see "Starting Central Repository Servers" on page 262), you must specify the source repository by using its repository server name. In addition, if the repository server is running on the source repository, you can only use `rpcopy` to copy it if there are no application developers making modifications to the repository (they must be using read-only workspaces). Using `rpcopy` guarantees a consistent copy of the repository.

If the repository server is not running for the source repository, you must specify the source repository using the repository name with the appropriate syntax. It is faster to copy a source repository if the repository server is not running, so you might want to shut down the repository server when you copy the source repository.

After you invoke the `rpcopy` command on a secure source repository, you are prompted for the administrator password. If you do not supply this password, you cannot copy the secure repository.

For example, to create a secure copy of a secure B-tree repository called SecretProject and call that copy AnotherProject, use the following command:

```
rpcopy -s bt:SecretProject -fr bt:AnotherProject
```

In this example, AnotherProject is a secure repository, with the same passwords as SecretProject.

Creating Shadow Repositories

Usually application developers create their own shadow repositories using the Repository Workshop. However, you can create a shadow repository from a script or a command line by using the `rpshadow` command.

Before you use the `rpshadow` command, the repository server must be running on the central repository. See "Starting Central Repository Servers" on page 262 for information on starting the repository server.

rpshadow command

Portable (all platforms)

rpshadow -fr *target_shadow_name* **-n** *repository_server_name* [**-r**]
[**-fns** *name_server_address*] [**-fm** *memory_flags*] [**-fl** *logger_flags*]

OpenVMS

VFORTE RPSHADOW
/**REPOSITORY=***target_shadow_name*
/**SERVER_NAME=***repository_server_name*
[**/REPLACE**]
[**/NAMESERVER=***name_server_address*]
[**/MEMORY=***memory_flags*]
[**/LOGGER=***logger_flags*]

The following table describes the `rpshadow` flags:

Flag	Purpose
-fr *target_shadow_name* **/REPOSITORY=** *target_shadow_name*	Specifies the name of the shadow repository to create. This must be a B-tree repository. The default directory for the shadow is FORTE_ROOT/repos. However, you can specify a full path name to place the repository in any directory you want.
-n *repository_server_name* **/SERVER_NAME=** *repository_server_name*	Specifies the repository server corresponding to the central repository from which to make the shadow. The existence of the shadow will be recorded in the central repository.
-r **/REPLACE**	Specifies that the new shadow replaces any B-tree repository with the same name as the new shadow. If you do not specify this flag and a B-tree repository with the same name already exists, you will get an error.
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.

Flag	Purpose
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags in the `rpshadow` command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.

When you use the `rpshadow` command, iPlanet UDS creates an attached shadow in the specified directory. The new shadow will be empty until an application developer starts working with it. The shadow you create with the `rpshadow` command can be used exactly the same way as a shadow created with the Repository Workshop.

You can use the `rpshadow` command on the node where you want the new shadow repository. Alternatively, you can use the `rpshadow` command to create a new shadow repository, then copy the new shadow repository to the desired node.

Additional Information About Using Shadows

Complete information about using shadows in the Repository Workshop is provided in *A Guide to the iPlanet UDS Workshops*.

You can access up to 15 different repository workspaces using one shadow repository. You can find out what workspaces have been cached in the shadow repository by using the Utility > Show Repository Info command in the Repository Workshop or the Fscript `ShowReposInfo` command. You cannot delete a workspace from a shadow repository after you have accessed it. If you want to access a sixteenth workspace using a shadow repository, you need to create a new shadow repository.

A cached workspace synchronizes itself with the central repository by applying the changes that have occurred in the central repository to itself. This synchronization occurs when you open a workspace in an attached shadow.

If you have not synchronized a workspace in the shadow with the central repository in a long time, the workspace might not open in the shadow repository, because it cannot completely synchronize itself with the central repository. If this occurs, you need to create a new shadow for accessing this workspace.

Making a Standard Repository a Secure Repository

To make a standard repository a secure repository, use the `rpcopy` command's `-secure` flag, then set all the required passwords when you are prompted.

For example, to replace a standard B-tree repository called SecretWork with a secure B-tree repository of the same name, you can specify the following command:

```
rpcopy -s bt:SecretWork -fr bt:tempRepos -secure
```

You can then delete the B-tree files for SecretWork, and rename the B-tree files for tempRepos as SecretWork.

When you invoke the `rpcopy` command with the `-secure` flag, you are prompted for all of the following passwords that have not yet been set:

- administrator
- master
- baseline
- passwords for all defined workspaces

You must define all these passwords when you create a secure repository.

Making a Secure Repository a Standard Repository

To make a secure repository a standard repository, use the `rpcopy` command with the `-nonsecure` flag. The new copy of the repository has no administrator password. Other passwords that have been set in the original repository are set with the same values in the copy.

For example, to replace a secure B-tree repository called Internal with a standard B-tree repository of the same name, you can specify the following command:

```
rpcopy -s bt:Internal -fr bt:tempRepos -nonsecure
```

You can then delete the B-tree files for Internal, and rename the B-tree files for tempRepos as Internal.

Starting Central Repository Servers

To provide multiple developers with access to a central repository, you must start the repository server associated with that central repository. The `rpstart` command starts the repository server for the repository you specify and assigns a repository server name to the server being started. This repository server name is the name that developers will use when they want to open the associated central repository, for example, when they give the `forte` command to run the iPlanet UDS Workshops.

When you start a repository server for a central repository, its name is registered by the name server, and distributed clients must use this repository server name to access the central repository.

The `rpstart` command starts one or more processes; which processes depend on the format of the repository. The following section describes how to use the `rpstart` command, and how the command starts repository servers for B-tree repositories.

Repository server as an NT service On Windows NT, you might have a Repository Server NT service set up when you install iPlanet UDS. If you run a repository server as a Windows NT service processes, this service process can continue even after you log off, instead of shutting down when you log off. You can start and shut down this Repository Server NT service using the Windows NT Service Control Panel, as described in "Controlling the Node Manager or Environment Manager and Repository Server Services" on page 111.

rpstart Command

The `rpstart` command starts a repository server for the specified central repository. You specify the central repository whose server you wish to start, and you assign a name to that server. Every time you start the same repository, you must assign the same repository server name to it. If you use a different server name, existing shadow repositories will no longer be able to connect to the central repository.

After the repository server is started, all application developers who want to access the repository must use the repository server name, rather than the repository name. In addition, you must use the repository server name in all subsequent commands on the repository, such as `rpcopy` or `rpstop`.

The syntax of the `rpstart` command is:

Portable (all platforms)

rpstart -n *repository_server_name* **-fr** *repository_name* [-w]
[**-fns** *name_server_address*] [**-p** *master_password*] [**-fm** *memory_flags*]
[**-fst** *integer*] [**-fl** *logger_flags*] [**-scm** *source_code_manager*]

OpenVMS

VFORTE RPSTART
/SERVER_NAME=*repository_server_name*
/REPOSITORY=*repository_name*
[/**WAIT_TIME=** *time _in_seconds*]
[/**NAMESERVER=***name_server_address*]
[/**PASSWORD=***master_password*]
[**/MEMORY=***memory_flags*]
[**/STACK**=integer]
[**/LOGGER=***logger_flags*]
[**/SOURCE_CODE_MANAGER=***source_code_manager*]

The following table describes the flags:

Flag	Purpose
-n *repository_server_name* **/SERVER_NAME=** *repository_server_name*	The repository server name to use for the specified repository. This is the name developers will use to access the central repository.
-fr *repository_name* **/REPOSITORY=** *repository_name*	Specifies the repository name whose server you wish to start. For B-tree repositories, specify "bt:" and a full path as part of the *repository_name*, for example, "bt:d:\myreposdir\myrepos" on Windows NT.
-w **/WAIT_TIME=** *time_in_seconds*	Specifies time in seconds that the repository server will wait before reporting an error in startup. If specified, this value overrides the setting for the environment variable FORTE_RPSTART_WAIT.
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.

Flag	Purpose
-p *repository_name* **/PASSWORD=** *master_password*	(Standard repositories only) The master password for the repository. If no password existed before, this master password is set for the repository. If a password did exist before, then this master password replaces the previous password. This master password is the master password for the repository until the password is again changed using this flag. If you specify the `-p` flag with a secure repository, this flag is ignored.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags in the `rpstart` command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.
-scm *source_code_manager* **/SOURCE_CODE_MANAGER=** *source_code_manager*	Specifies the name of the service object to use as the source code manager. The name of the service object is in the format *appname/projname/svcobjectname*. For more on iPlanet UDS source code management, see *iPlanet UDS Programming Guide*.

Stopping Central Repository Servers

You need to stop a repository server to perform maintenance on a central repository or at any time you want to make the central repository temporarily unavailable.

You can stop a central repository server by:

- using the `rpstop` command, described in this section
- using the `Shutdown` command on the RpServer agent, which is described in "Shutting Down Repository Servers" on page 290

rpstop Command

The `rpstop` command stops a repository server. If there are active users of the repository server, this command will not stop the server unless you include the optional `-k` flag.

The syntax is:

Portable (all platforms)

rpstop -n *repository_server_name* [**-fns** *name_server_address*]
[**-k**] [**-fm** *memory_flags*] [**-fst** integer] [**-fl** *logger_flags*]

OpenVMS

VFORTE RPSTOP
[**/SERVER_NAME=***repository_server_name*]
[**/NAMESERVER=***name_server_address*]
[**/KILL**]
[**/MEMORY=***memory_flags*]
[**/STACK**=integer]
[**/LOGGER=***logger_flags*]

The following table describes the `rpstop` command flags:

Flag	Purpose
-n *repository_server_name* **/SERVER_NAME=** *repository_server_name*	The repository server to stop. You must specify the repository server name that was assigned to the server when it was started with the `rpstart` command.

Flag	Purpose
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-k **/KILL**	If there are active users of the repository server, the server is forced to stop anyway. The default is to give an error and reject the shutdown if there are active sessions.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX you must specify the memory flags in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags in the `rpstop` command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.

Maintaining Repositories

Because development repositories contain all of your organization's development work, it is essential that you maintain them properly. Maintaining a repository entails three basic tasks:

- compact the repository
- back up the repository, and restore it when necessary
- improve repository performance

Compacting a Repository

Because a central repository stores the incremental saved work of multiple developers, the size of the repository can grow quite rapidly. The `rpclean` command allows you to reclaim unused space and shrink the repository database files. You should use `rpclean` to compact the repository on a regular basis. If your repository is heavily used and is growing very large, you can even compact it every night.

Although private repositories do not store the work of multiple developers, you may wish to compact them. However, if the private repository is on a Windows 95 platform, you must first copy it to a UNIX, Windows NT, or VMS machine to run `rpclean` on it.

Quick vs. full optimizing With the `rpclean` command, you can perform either a quick space reclamation, using the command's `-q` option, or a full reclamation. A quick reclamation uses the repository's data manager to recover space, while a full reclamation deletes all unused components from the repository. A quick reclamation therefore recovers a smaller amount of space than a full reclamation, but completes in a fraction of the time. For example, on a large repository (300 MB or more), a quick reclamation takes usually only a few minutes, while a full reclamation may take a minimum of three times longer.

Before using rpclean During a full reclamation, iPlanet UDS deletes any components in the repository that are not included in any workspaces or in the system baseline. Therefore, it is a good idea to delete any obsolete workspaces before giving the `rpclean` command. This way, iPlanet UDS can reclaim the space used by components that are only in obsolete workspaces. In addition, it is a good idea for all developers to update their workspaces before you optimize the repository. When the workspaces are up to date with the system baseline, `rpclean` will be able to reclaim more space. See *A Guide to the iPlanet UDS Workshops* for information about workspaces and the system baseline.

The following are some basic rules of thumb for optimizing space reclamation.

➤ **To optimize space reclamation before using the rpclean command**

1. Delete all unneeded workspaces.
2. Delete all unneeded projects.
3. Have developers update and/or integrate as many workspaces as possible.
4. Have developers leave as few components checked out or branched as possible.

Note that none of these rules are necessary for correct functioning of the `rpclean` command. They simply allow the utility to reclaim more space.

rpclean Command

Stopping and starting the repository server Before using `rpclean` on a central repository, you must stop the repository server using the `rpstop` command. If developers wish to continue working while the central repository is down, they can detach their shadows and work in detached mode until the compaction is complete. When the compacting is complete, you must restart the repository server using the `rpstart` command.

The syntax for the `rpclean` command is:

Portable (all platforms)

rpclean -fr *repository_name* [**-t** *temporary_directory_name*] [**-q**]
[**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]

OpenVMS

VFORTE **RPCLEAN**
/REPOSITORY=*repository_name*
[**/TEMPORARY=***temporary_directory_name*]
[/QUICK]
[**/MEMORY=***memory_flags*]
[**/STACK**=*integer]*
[**/LOGGER=***logger_flags*]

The following table describes the `rpclean` command's flags:

Flag	Purpose
-fr *repository_name* **/REPOSITORY=** *repository_name*	Specifies the central repository name for the `rpclean` command to compact.

Flag	Purpose
-t *temporary_directory_name* **/TEMPORARY=** *temporary_directory_name*	Specifies where temporary files used by the rpclean process are placed. This directory should have free space equal to about 1/8 the size of the repository being compacted. By default, all temporary files are stored in the FORTE_ROOT/repos directory. These temporary files can be as large as10 MB larger than the size of the index file (.btx) for your repository, so you might want to place them on a storage disk other than the one you are using for your central repository. This directory must be on a local device, not NFS-mounted. Also, this directory should not be cleared automatically during initialization, like /tmp, because the B-tree transactional log file is also stored in this directory, and you would need this file to recover the repository if the machine abnormally shut down.
-q **/QUICK**	Perform a quick compaction of the repository, recovering space using the repository's data manager but not deleting unused project components from the repository.
-fm *memory_flags* **/MEMORY=** *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX you must specify the memory flags in double quotes.
-fst *integer* **/STACK=***integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fl *logger_flags* **/LOGGER=** *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags in the `rpclean` command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.

Backing up Repositories

Usually the system manager is only responsible for maintenance of the central repositories. Typically, application developers will back up their own shadow and private repositories. However, this section provides information on how to back up and restore all three kinds of repositories so that you can provide assistance whenever necessary.

Always test your backup repository before assuming it was copied correctly. See that it can be opened, that it contains the proper workspaces, and so on. Only when it has passed those tests is it safe to remove the repository it was copied from.

If the repository also has a .btb file, do one of the following:

- Recover the repository by opening the repository, then copy its .btd, .btx, and .rop files.
- Copy the .btx, .btd, .btb, and .rop files to the new location, then recover the repository by opening it.

Backing up Central Repositories

Because the central repository contains the work of multiple developers, you should back it up on a very regular basis. We recommend that you do so daily, if possible.

There are two ways to back up a central repository:

- Use the `rpcopy` command to make a copy of the repository, and then copy the B-tree files to another physical disk or tape.

  The disadvantage of this technique is the `rpcopy` command takes some time to run. Also, the .rop file is not automatically backed up by the `rpcopy` command.

  Other developers can be using a central repository during the rpcopy, so long as they are all connected read-only and the rpserver is running. If the rpserver is running, specify the repository server name (for example, “CentralRepository”) as the `-s` flag to rpcopy. Otherwise, specify the repository file name (for example, “bt:centralrepository”) as the `-s` flag to rpcopy.

  For information about `rpcopy`, see “rpcopy Command” on page 255.

- Use your operating system copy commands to copy the repository files directly onto another physical disk or tape. No other program can be using the repository, even read-only, while you are backing the files up this way.

Before backing up a central repository, you must stop the repository server using the `rpstop` command (as described in "rpstop Command" on page 265). The repository server must not be running at the time you backup the repository files. If another program has the repository open during the backup, the resulting copy is likely to be completely unusable.

If you stop the repository server to make the backup copy, and developers want to continue working while the central repository is down, they can detach their shadows and work in detached mode until the backup is complete. When the backup is complete, you must restart the repository server using the `rpstart` command (as described in "rpstart Command" on page 262).

This technique is faster than using `rpcopy`.

If you try to use the repository without letting iPlanet UDS recover the repository with the information in the .btb file, your repository will be in an unknown state.

NOTE Remember to always back up the .rop repository file. This file contains important repository information that can help to diagnose problems in the central repository.

Shadow repositories are not copied along with the central repository; therefore you cannot restore them. After restoring the central repository, you must delete all the shadow repositories associated with it. The developers will need to create the new shadows that they need.

Backing up Shadow Repositories

When a developer works in a detached shadow, the work she does locally is not backed up in the central repository. If she runs out of disk space or has a problem with her local disk, her work in the detached shadow could become corrupted. Therefore, she should backup her changes in the detached shadow by doing one of the following:

- Use the Utility > Backup Repository... command of the Repository Workshop.
- Make copies of the shadow B-tree files using operating system commands to copy the B-tree .btd and .btx files.
- Export the plans or components to a file using the iPlanet UDS Workshops.

If you try to use the repository without letting iPlanet UDS recover the repository with the information in the .btb file, your repository will be in an unknown state.

To backup the detached shadow by copying the B-tree files, use your operating system to copy the shadow's .btd and .btx files. To restore a detached shadow, copy the backup files into the original location.

Backing up Private Repositories

To backup a private repository, do one of the following:

- Use the Utility > Backup Repository... command of the Repository Workshop.
- Make copies of the B-tree files using operating system commands to copy the B-tree .btd and .btx files.
- Use `rpcopy` to back up a private repository. You must be the only one using a private repository when you back it up this way. For information about `rpcopy`, see "rpcopy Command" on page 255.

To restore a private repository, copy the backup files to the original locations.

Improving Repository Performance

There are several steps you can take to improve the performance of a central repository:

- ensure that developers use shadow repositories
- tune the central repository server environment
- reduce repository overhead
- keep shadows efficient
- use multiple repositories
- use detached shadows

Each of these topics is described below.

Shadow Repositories

Each developer who uses a shadow helps to reduce the load on the repository server, which improves performance for all developers accessing the repository. An attached shadow serves as a persistent cache—objects fetched when using an attached shadow are cached, resulting in much higher performance on subsequent references. Without shadows, every object reference results in a fetch from the central repository, which dramatically reduces the performance of the central repository.

Tuning the Central Repository Server Environment

If the repository server is not allocated sufficient CPU and disk I/O resources, its performance suffers. Therefore, it is a good idea to minimize the number and size of other processes running on the node where the repository server is running. Especially during the tuning process, exclude other resource intensive activity, for example, RDBMS servers, iPlanet UDS partitions, and so on.

Excluding nodes Exclude the node where the repository server is running when you test configurations in simulated environments. For information on excluding nodes from a configuration, see *A Guide to the iPlanet UDS Workshops*. After you establish your baseline performance, you can consider adding processes on the repository server node and measure the impact.

To reduce the repository's size and improve response time, regularly use `rpclean`, iPlanet UDS's utility to compact the central repository. To decide how often to invoke `rpclean`, consider that `rpclean` performs two jobs: it removes from the repository objects no longer in use, and it compresses objects. When you invoke `rpclean` with the `-q` flag, rpclean performs only the compression step, which is faster than performing both tasks.

Monitoring disk consumption and reserving the space necessary to perform `rpclean` are critical components in managing your iPlanet UDS development environment.

BtreeCache agent To improve the performance of a B-tree central repository, increase the size of the B-tree cache by increasing value of the TotalPages instrument on the BtreeCache agent. For information about the BtreeCache agent, see *Escript and System Agent Reference Guide*.

Reduce Repository Overhead

Updating a workspace is an expensive operation and should be done only when there is a need to update the workspace with changes integrated by another developer. Updating may require the entire workspace to recompile and cause a subsequent detach of your shadow to take longer.

Compiling the workspace Compiling the entire workspace before integrating it improves performance for a number of reasons. The system baseline will contain fully compiled code, and subsequent workspace updates will access fully compiled code, greatly reducing the number of compilations needed. Since the result of those compilations are written to the repository, this lowers repository traffic as well.

Using Shadows Efficiently

The most efficient way to use an attached shadow is to turn off the Save Commits to Central check box in the Repository Workshop Preferences dialog. With this option turned off, iPlanet UDS saves changes to the shadow until either you request that iPlanet UDS save them to the central repository or when iPlanet UDS performs a large operation like integrating a workspace.

Using an attached shadow repository without always committing to the central repository provides performance similar to that of detached shadows, while letting you access the central repository to check out components and include plans. However, because your changes are more secure in the central repository than in a shadow, you should only use this option when performance is an important consideration.

NOTE Replace your shadow regularly. You cannot perform an `rpclean` on a shadow, and it eventually get large and slow. In general, replace your shadow every two weeks of heavy development, or when you notice that response is getting slow.

Delete attached shadows after committing changes to the central repository. Before deleting detached shadows, re-attach them to the central repository to commit changes to the central repository. After you delete the shadow, connect to the central repository and create a new shadow.

NOTE A shadow's performance depends on what proportion of the central repository is cached in the shadow—the larger the proportion, the better the performance. When you create a new shadow, it is essentially empty. The shadow normally becomes incrementally more efficient as objects are copied from the central repository into the shadow as you access objects through workspaces open on the repository.

To cache all the objects at once, detach your shadow. You can immediately re-attach to the central repository, if you wish.

Using Multiple Repositories

This section lists some guidelines for using multiple repositories.

➤ **To use multiple development repositories for an application**

1. Organize the plans of the application into sets.

   For example, divide the application so that the projects for the window classes are in one set, and the projects for the database access classes are in another set.

2. Organize the developers into teams based on the sets.

3. Designate a central repository as the *master* repository for each set of projects, which contains the master copies of the projects.

   For example, the master copies of the projects containing the window classes for an application might be in one repository, and the master copies of the projects containing the database access classes night be in another repository.

   Only make changes to the projects in the master repository for the set.

4. Other repositories that contain copies—or *snapshots*—of these projects are considered *slave* repositories. Ensure that these snapshots are read only in the slave repositories.

   Each central repository is the master for some projects and possibly the slave for others. To avoid losing changes when the slaves are updated, ensure that changes are made only to the appropriate master repository. The development team must enforce this policy.

   To ensure that a snapshot of a project cannot be changed, you can deploy the project from the master repository as a library, then import the generated .pex file for the library in the slave repositories. For more information about deploying libraries and importing their information into other repositories, see *A Guide to the iPlanet UDS Workshops*.

5. Update the project snapshots regularly.

➤ **To update project snapshots**

1. Update a workspace in the master repository to the system baseline using the File > Update Workspace command in the Repository Workshop, or the Fscript `UpdateWorkspace` command.

2. Export updated projects to flat files, using the Plan > Export command in the Repository Workshop or the Fscript `ExportPlan` command.

3. In the slave repository, check out all components of each project being updated, using the Plan > Checkout All Components command in the Project Workshop or the Fscript `CheckoutAllComps` command.
4. In a workspace of the slave repository, import each exported project using the Plan > Import command in the Repository Workshop or the Fscript `ImportPlan` command.
5. Integrate the workspace using the File > Integrate Workspace command in the Repository Workshop or the Fscript `IntegrateWorkspace` command.
6. Update other workspaces in the slave repositories to get the new version of that project.

The Repository Workshop is described in *A Guide to the iPlanet UDS Workshops*. Fscript commands are described in *Fscript Reference Guide*.

Using Detached Shadow Repositories

The caching performed by an attached shadow repository can significantly reduce the load on the central repository.

Attached shadows are short-term caches. If the Save Commits to Central option of the Repository Workshop Preferences dialog is turned on, iPlanet UDS writes everything written to the shadow into the central repository when you commit your changes. If this option is turned off, iPlanet UDS writes these changes to the central repository only when you do one of the following:

- Use the Utility > Commit To Central command in the Repository Workshop.
- Exit the Repository Workshop, and are prompted to commit your changes.
- Update or integrate the workspace using the File > Update Workspace or Integrate Workspace commands in the Repository Workshop, or the Fscript `UpdateWorkspace` or `IntegrateWorkspace` commands.

However, detached shadows only access the central repository during attach and detach operations, so the interaction with the central repository is much less.

The following are guidelines for successfully using detached shadows:

- Divide the application's components into autonomous sets, with each set owned by a single developer. The more insulated a developer is from other developer's changes, the less need there is to check out, update, and integrate (and therefore attach). Attaching and detaching are expensive operations and should be minimized.

- Prevent several developers from attaching or detaching at the same time, which creates slow repository performance. You might want to schedule when developers attach or detach.
- You can create up to 50 new projects in a detached shadow during the time that it is detached. When you reattach the shadow, the new projects are flushed to the central repository. If you then detach the shadow, you can again create up to 50 more new projects in that shadow before you need to reattach the shadow.
- Encourage developers to check out all components they need before detaching. They cannot check out anything while detached. Checking out what they need up front saves them from having to re-attach and detach again. Note that it is possible to branch a component while detached; developers can convert the branches to check outs after re-attaching as long as the components are available to be checked out.
- Have the developers back up their detached shadows regularly. (Attached shadows cannot be backed up, but should be flushed regularly to the central repository.) Detached shadows, on the other hand, are the only places that contain all the work done since detaching, and should thus be backed up regularly. Shadows are B-tree repositories, which are two files (*shadowname*`.btd` and *shadowname*`.btx`).

  Backing up the repository is as simple as copying the files to another directory. It is preferable to copy the file to a directory on a different disk to protect against crashes. B-tree repositories are architecture-independent, so they can be copied from a PC to a UNIX machine, for example. If you are using a file transfer utility, do a binary transfer (not ASCII).

The Repository Workshop is described in *A Guide to the iPlanet UDS Workshops*. Fscript commands are described in *Fscript Reference Guide*.

Maintaining a Secure Repository

This section explains how to maintain secure repositories that you have created using the `rpcreate` command with the `-secure` flag (described in "rpcreate Command" on page 253) or the `rpcopy` command (described in "rpcopy Command" on page 255).

Creating New Workspaces in a Secure Repository

To create a new workspace for a secure repository, you need to specify the administrator password.

➤ **To create a new workspace in a secure repository using the Repository Workshop**

1. Choose the File > New Workspace command.
2. In the New Workspace dialog, specify the workspace name, the administrator password, and the workspace password.
3. Click the OK button to create the workspace.

➤ **To create a new workspace in a security repository using Fscript**

Enter the `NewWorkspace` command, using the following syntax:

NewWorkspace *workspace_name* [*initial_password* [*admin_password*]]

Argument	Description
workspace_name	The name of the new workspace to create. This name must not be the name of an existing workspace in the current repository.
initial_password	The initial password for the workspace. If the current repository is a secure repository, you must specify an initial password. If the current repository is a standard repository, a workspace password is not required.
admin_password	The administrator password for the current repository. In a secure repository, you must specify the administrator password. In a standard repository, do not specify this password. This password is ignored if you specify it.

For example, to create a new workspace using the Fscript `NewWorkspace` command on a secure repository, specify the following command:

```
NewWorkspace PrivateWorkspace mysecret123 pwadmin8
```

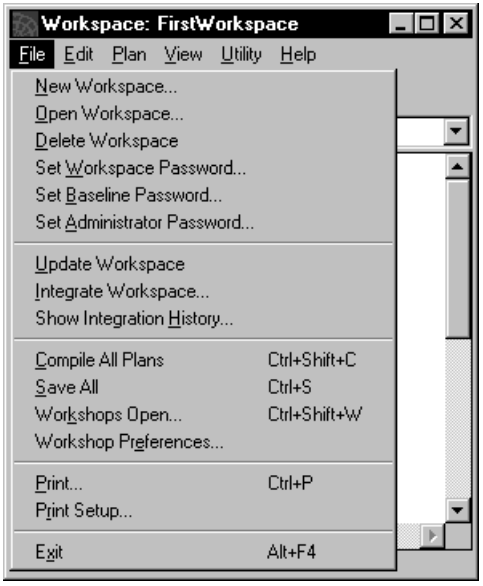
Changing Passwords in a Secure Repository

You can change the administrator, baseline, and workspace passwords in either the Repository Workshop or in `Fscript`. However, you can only change the master password in `Fscript`.

NOTE You cannot change the master password for a secure repository using the `rpstart -p` flag.

In the Repository Workshop

All the commands for changing passwords are in the Repository Workshop, as shown:



Administrator password To change the administrator password on the current repository, click the File > Set Administrator Password command. You need to specify the existing password and enter the new password twice.

Baseline password To change the baseline password on the current repository, choose the File > Set Baseline Password command. You need to specify the existing password and enter the new password twice.

Workspace password To change the password for the current workspace, choose the File > Set Workspace Password command. You need to specify the existing password and enter the new password twice.

In Fscript

To change the master, administrator, baseline, or workspace passwords, use the `SetPassword` command:

SetPassword *password_type new_password* [*current_password*]

Argument	Description
password_type	The type of password you are setting.
new_password	The new password you are setting for the repository or workspace.
current_password	The current password for the repository or workspace.

The *password_type* argument specifies whether the kind of password you are setting for your current repository or for your workspace. You can set the current repository using the `SetRepos` command. You can set the current workspace using the `SetWorkspace` command.

password_type can have one of the following values:

***password_type* value**	**Description**
admin	For a secure repository, sets a password for creating workspaces and copying the repository. You cannot set this password for a standard repository.
baseline	Sets a password for integrating a workspace into the system baseline in the current repository.
master	Sets a master password for the repository, which allows complete access to the repository.
workspace	Sets a password for accessing the current workspace.

The *new_password* argument specifies the new password for the repository or workspace. A legal password is a string of 7-bit ASCII characters, of any length with no spaces. In a secure repository, a non-null password is required. In a standard repository, you can remove a password by specifying a null string, as shown in the following example:

```
SetPassword workspace '' secretpassword
```

The *current_password* argument specifies the current password that is being replaced by the new password. If a password is being set for the first time, then this value is not required.

If you want to set the administrator password for a repository that is a standard repository, you need to convert the repository to a secure repository using the `rpcopy` command, as described in "Making a Standard Repository a Secure Repository" on page 261.

Using Repository Agents

This section introduces iPlanet UDS system agents that you can use to monitor and control the repository server for a central repository. These agents, their commands, instruments, and positions in the agent hierarchy are described in detail in *Escript and System Agent Reference Guide*:

BtreeCache Corresponds to the cache used with the B-tree central repository to improve repository performance.

BtreeRepository Corresponds to the B-tree repository data files.

ObjectCache Corresponds to the object cache used by a repository session (see "Repository sessions" on page 245).

Repository Corresponds to a read-write repository, such as a central, shadow, or node repository.

RepositoryServer Corresponds to a running repository server partition.

RepositoryServerInfo Provides information about a running repository server.

RepositorySession Corresponds to a repository session.

This section explains how the agents are related and how you can use them to monitor and control your central repositories.

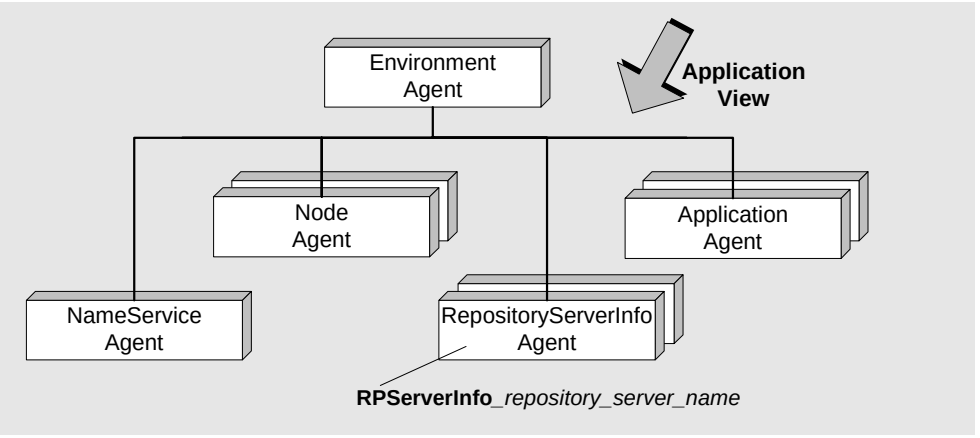
How the Agents are Related

This section explains how you can locate repository server agents and navigate through their agent hierarchy.

Finding Running Repository Servers and Their Agents

There is a RepositoryServerInfo agent for each repository server that is running in the current environment. The RepositoryServerInfo agent is a place holder for information about the running repository server and is a subagent of the Environment agent, as shown in Figure 8-1.

Figure 8-1 Locating information about running repository servers



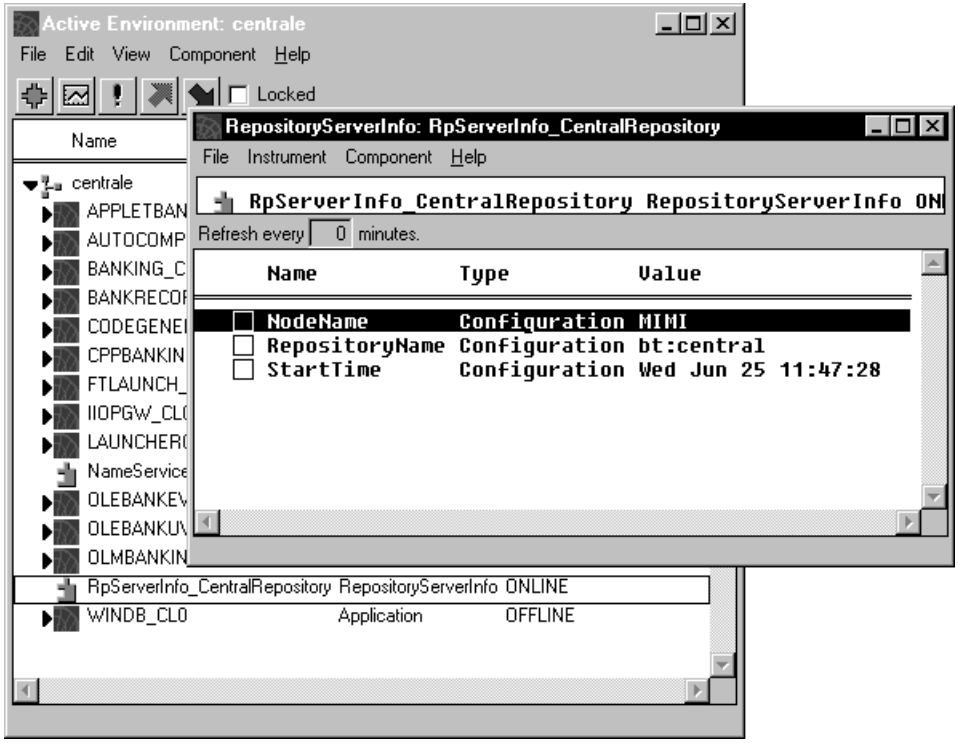
➤ **To find the RepositoryServerInfo agent for a repository server**

1. Open the Application View, using the View > Application Outline command in the Environment Console Active Environment window.
2. Look for the desired RepositoryServerInfo agent.

   Each RepositoryServerInfo agent's name contains the name of the repository server in the format "RpServerInfo_*repository_server_name*." For example, the RepositoryServerInfo agent for the CentralRepository repository server is "RpServerInfo_CentralRepository."

This agent has instruments that provide information about the node on which the repository server is running, the name of the repository, and the time that the repository server started, as shown in the following figure:

Figure 8-2 RepositoryServerInfo agent in the Application View



This agent has no subagents, however, you can use the information in this agent to locate the other repository agents. For example, the RepositoryServerInfo agent shown in Figure 8-2 identifies a node name of “MIMI” and a repository name of “bt:central”. You can then navigate to the Node agent that represents the node “MIMI” to locate the agents that represent the running repository server.

Navigating Through the Repository Agents

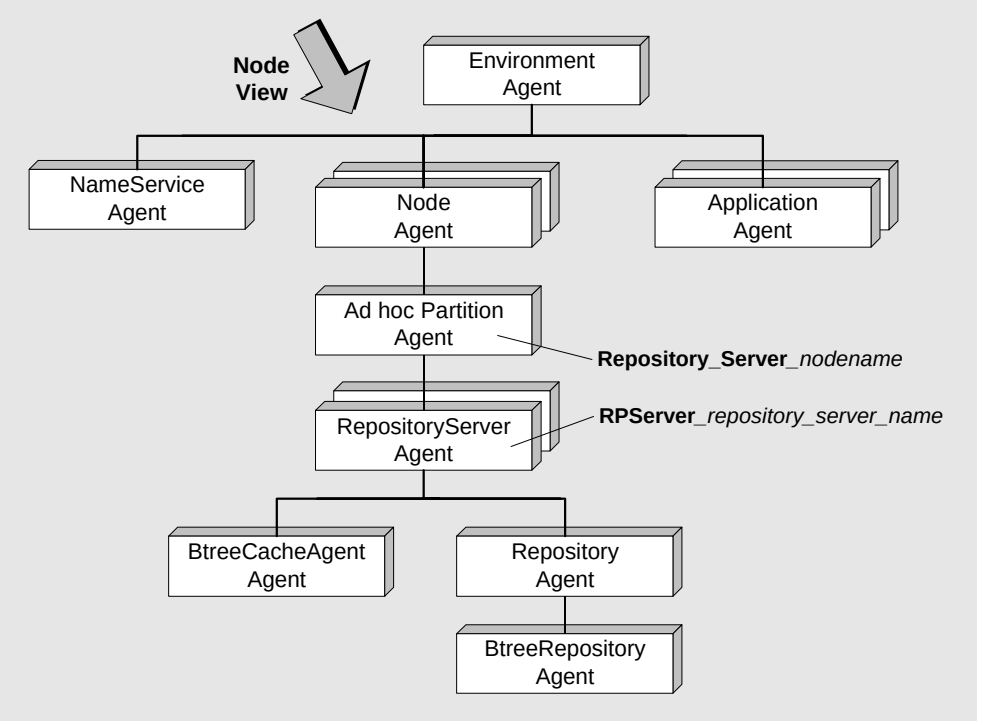
Each repository server has a corresponding RepositoryServer agent and its subagent hierarchy. The following repository agents are subagents of this RepositoryServer agent, as shown in Figure 8-3:

BtreeCache Corresponds to the cache used with the B-tree central repository to improve repository performance. For information about the commands and instruments available for this agent, see *Escript and System Agent Reference Guide*.

Repository Corresponds to the repository being served. For information about the commands and instruments for this agent, see *Escript and System Agent Reference Guide*.

BtreeRepository Corresponds to the B-tree repository data files. This agent is a subagent of a Repository agent. For information about the commands and instruments for this agent, see *Escript and System Agent Reference Guide*.

Figure 8-3 Agents for a central repository server



➤ **To find the agents for a repository server**

1. Open the Node Outline view, using the View > Node Outline command.
2. Find the node on which the repository is running, based on the information in the RepositoryServerInfo agent (see "Finding Running Repository Servers and Their Agents" on page 282).

   In Figure 8-2, the CentralRepository repository server is running on the node "MIMI."

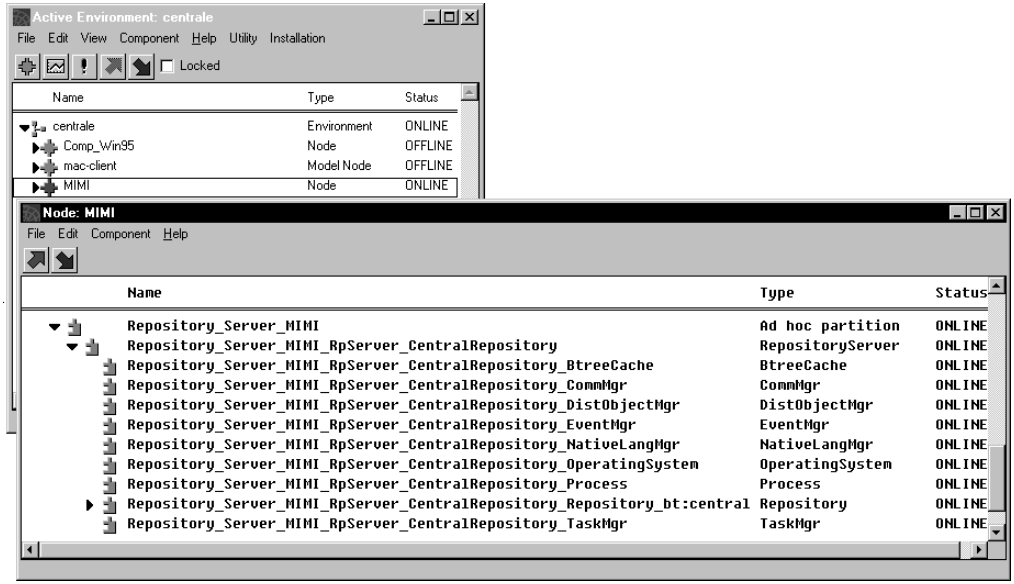
3. Click the expansion arrow to see the subagents of the Node agent.
4. Find the ad hoc partition agent that has a name of the format "Repository_Server_*nodename*."

   This agent represents the iPlanet UDS ad hoc partition for all repository server instances running on the node. (An ad hoc partition is the equivalent of an installed partition for iPlanet UDS system applications.)
5. Double-click this agent to open another window that shows RepositoryServer agents for any repository servers running on the node.

   RepositoryServer agent names end in "RpServer_*repository_server_name*."
6. Click the expansion arrows to see all the RepositoryServer subagents, as shown in Figure 8-4.

   The RepositoryServer agent whose name ends "RpServer_CentralRepository" represents the repository server called CentralRepository.

Figure 8-4 Repository Agent Hierarchy for CentralRepository Central Server in Node Outline View

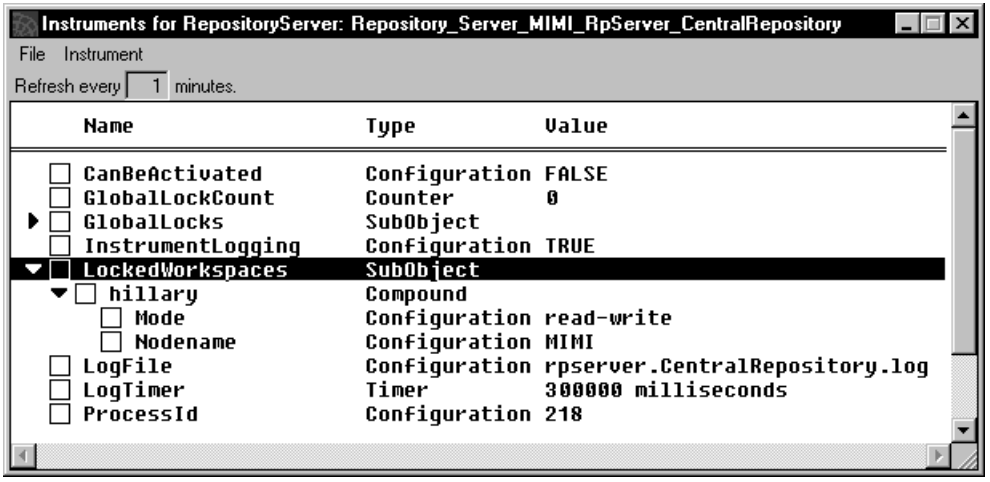


Finding Information about Locked Workspaces

The RepositoryServer agent has a LockedWorkspaces instrument that contains information about the workspaces in the repository that are currently locked. In other words, these workspaces have been opened in read-only or read-write mode.

➤ **To see the LockedWorkspaces instrument**

1. Locate the RepositoryServer agent, as described in "Navigating Through the Repository Agents" on page 283.
2. Double-click the RepositoryServer agent to open a new window for that agent.
3. Choose the File > Instruments command in the new window to display all the instruments of the RepositoryServer agent.
4. Click the expansion arrow on the LockedWorkspaces instrument to see a list of locked workspaces.



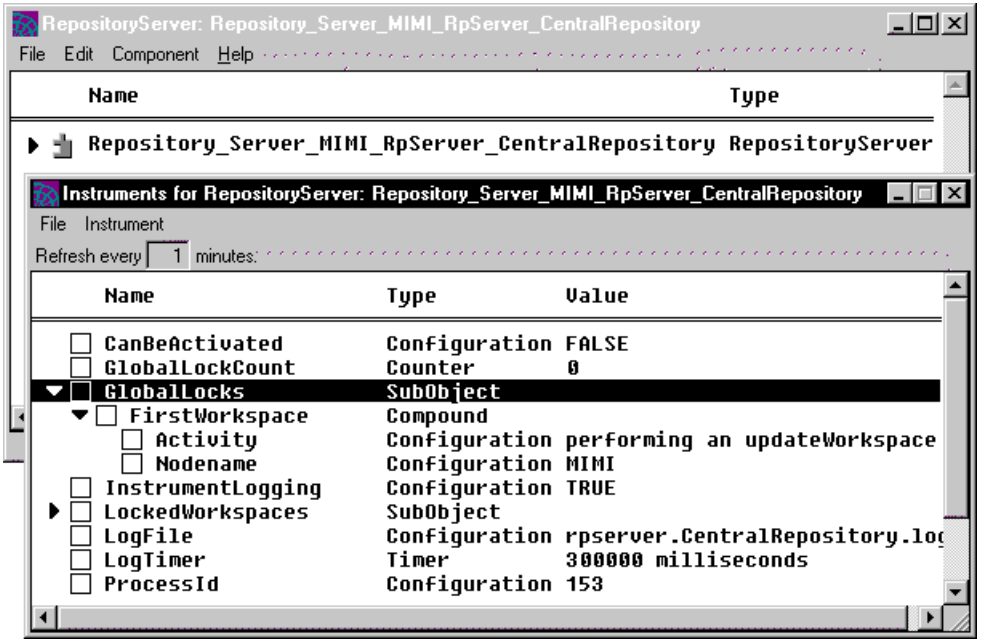
Each workspace contains information about the node that has locked the workspace, and indicates whether the lock is read-only or read-write mode. For more information about the RepositoryServer agent and its instruments, see *Escript and System Agent Reference Guide*.

Finding Information about Global Locks

The RepositoryServer agent has a GlobalLocks instrument that contains information about any global locks on the repository. This agent identifies the workspace that holds the lock, the type of lock, and the operation that is holding the lock, such as integrating or updating activities.

➤ **To see the GlobalLocks instrument**

1. Locate the RepositoryServer agent, as described in "Navigating Through the Repository Agents" on page 283.
2. Double-click the RepositoryServer agent to open a new window for that agent.
3. Choose the File > Instruments in the new window to display all the instruments of the RepositoryServer agent.
4. Click the expansion arrow on the GlobalLocks instrument to see a list of global locks on the centralrepository.



For more information about the RepositoryServer agent and its instruments, see *Escript and System Agent Reference Guide*.

Finding Information about Repository Sessions

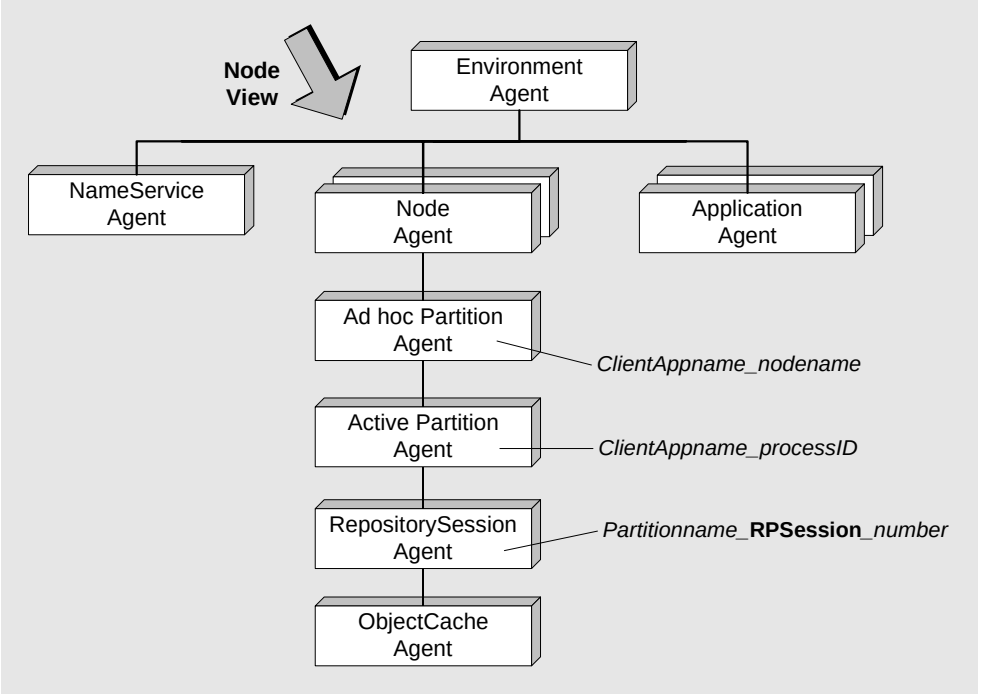
You can find information about repository sessions opened by a client application of a repository server, such as the Repository Workshop, using the following agents:

RepositorySession Corresponds to a repository session for the client application connected to a repository server. For information about the instruments available for this agent, see *Escript and System Agent Reference Guide*.

ObjectCache Corresponds to the object cache used by a repository session. A subagent of a RepositorySession agent. For information about the instruments available for this agent, see *Escript and System Agent Reference Guide*.

Figure 8-5 shows the hierarchy of parent and subagents for agents that describe a repository session:

Figure 8-5 Agents for a repository session

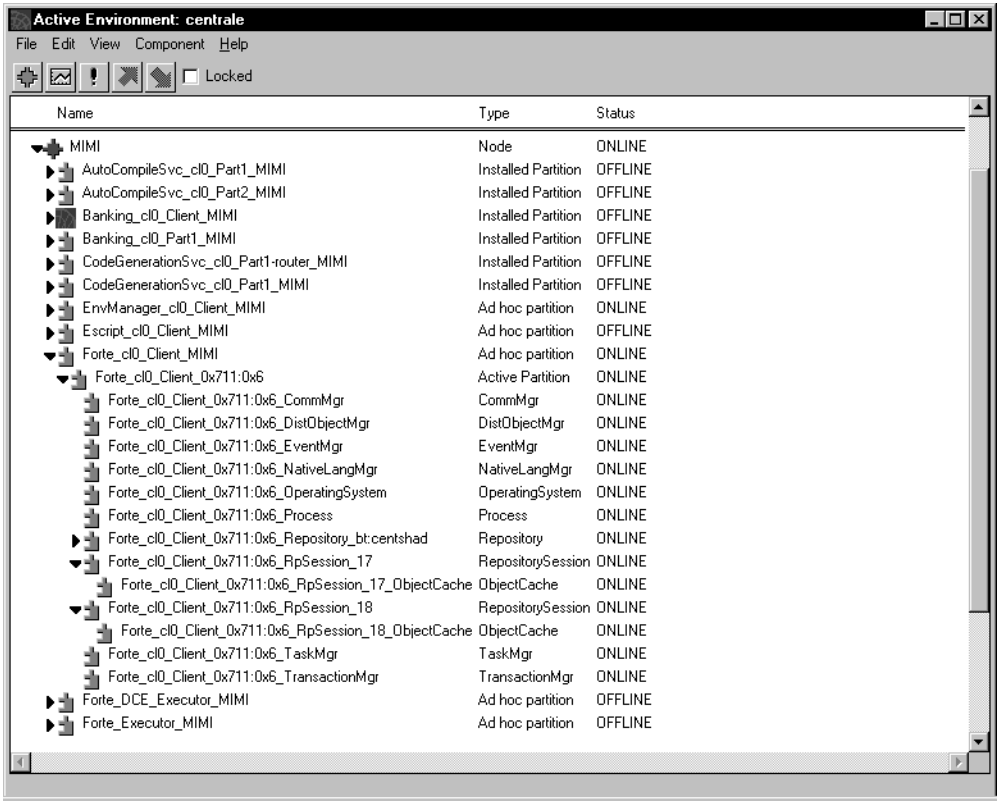


If a client node does not have a Node Manager application (such as the Environment Console, Escript, or the Launch Server) running, then you cannot access the Active Partition agents to monitor the repository sessions running on that node.

➤ **To locate the repository session agents of a client application**

1. Open the Node Outline view, using the View > Node Outline command.
2. Find the node that is running the repository session.
3. Click the expansion arrow to see the subagents of the Node agent.
4. Find the ad hoc partition agent that has a name of the application that started the repository session, for example, "Forte_cl0_Client_mimi" in Figure 8-6. Click the expansion arrow to see its subagents.

Figure 8-6 Agents for Repository Sessions in Node Outline View



5. Find the Active Partition agent for the application that started the repository session, for example, "Forte_cl0_Client_0x711:0x6" in Figure 8-6. Click the expansion arrow to see its subagents.
6. Find the RepositorySession agents. These agents have names with the format "*active_partition_name*_RpSession_*number*."

   This agent has instruments that identify the repository name, the type of connection, and the workspace name.
7. Click the expansion arrow for the RepositorySession agents to locate the ObjectCache agent that is the subagent of the RepositorySession agent.

   This agent has instruments that describe how efficiently the repository session is caching objects.

Shutting Down Repository Servers

You can shut down the repository server by using the `Shutdown` command on the following agents:

Ad hoc partition agent for all repository server partitions on a node Shuts down all the repository servers running on that node. This agent is a subagent of the Node agent, with the name "Repository_Server_*nodename*."

RepositoryServer agent Stops just the repository server managed by that agent. This agent is a subagent of the ad hoc partition agent "Repository_Server_*nodename*."

RepositoryServerInfo agent Stops the repository server described by this agent. This agent appears in the Environment Console Application View as a subagent of the Environment agent.

The `Shutdown` command stops the repository server only if no repository sessions are open.

In case of emergency, if you need to stop the repository server even when repository sessions might be running, use the `ForceShutdown` command, as described in *Fscript Reference Guide*.

Chapter 9

Launching iPlanet UDS Applications and Applets

This chapter describes how to use the Launch Server, an iPlanet UDS service that enables you to deploy and launch client applications in a easier and more efficient way than by installing one node at a time.

Topics covered include:

- deploying and managing applets and applications using the Launch Server
- using the iPlanet UDS Launcher (an application that uses the Launch Server to download and launch applications)
- using the Ftcmd utility

For information about configuring client applications as applets, see *A Guide to the iPlanet UDS Workshops* and *iPlanet UDS Programming Guide*.

For information about writing applications that use the AppletSupport Library to take advantages of applets and the ability to launch applications, see *iPlanet UDS Programming Guide*.

About Launching iPlanet UDS Applications and Applets

The *Launch Server* is an iPlanet UDS service that runs on client nodes and starts iPlanet UDS applets and applications. The Launch Server can run several standard iPlanet UDS client partitions under the same operating system process, which enables the applications to start faster and use less memory. It can also ensure that

the most recent copy of an application is installed on the client; if it is not, the Launch Server automatically downloads the newer copy. You can start the Launch Server directly using the Launch Server icon (Windows 95 and Windows NT) or the `ftlaunch` script, which starts the ftlaunch application.

NOTE The Launch Server starts and automatically downloads only standard client partitions.

End users can launch their applications by using the iPlanet UDS Launcher, an iPlanet UDS application that allows end users to start and shutdown applications. The iPlanet UDS Launcher provides online help for your end users, so that they will find the application easy to use. Starting the iPlanet UDS Launcher application starts the Launch Server on the client node.

You can also program your own interface to the Launcher Server, providing your end users with customized access to your applications. The iPlanet UDS Launcher application provides an example of how you can write TOOL code that accesses the Launch Server, and the code for this application is available as the Launcher sample application.

For system managers, the Launch Server simplifies how client applications are installed on client nodes. The system manager can set up the client nodes in the environment so that end users can access the Launch Server. This provides the following benefits:

Improved performance The Launch Server can run several iPlanet UDS applications under the same `ftexec` process and operating system process. When you start an iPlanet UDS client application through the Launch Server, the application uses the same set of runtime system objects—including the Log Manager, the Distributed Object Manager, and others—as the Launch Server.

By using the same process and the same runtime objects as the Launch Server, the client applications start significantly faster and use less memory than when they start as separate processes.

These applications also share a trace window with the Launch Server, which means that all log messages are printed to the same trace window (or log file, if you divert the trace information to a file). The application ID of the application printing each log message is added to the beginning of the log message.

Files automatically downloaded If you start an application that is assigned to the client node using the Launch Server, the Launch Server ensures that the most recent image repositories for the client partition are installed. The Launch Server compares the creation time and date of the image repository to that in the environment, and automatically downloads the newer image repository, if needed.

Publicly-available client applications You can define a list of client applications that are *publicly-available*, which can be downloaded and run from any client node. When an end user starts a publicly-available client application, iPlanet UDS downloads the necessary files, launches the application, then deletes the files when the application completes.

Ftcmd utility The `Ftcmd` utility provides a command interface to the Launch Server. This utility can be run as a command interface to the Launch Server or can be issued from icons to launch client applications under the Launch Server. You can also use it access information about available and running applications.

Using any Ftcmd command starts the Launch Server on the client node if the Launch Server is not already running on that node. By default, iPlanet UDS creates icons Windows 95 and Windows NT for interpreted client partitions that use the `Ftcmd` utility to start the client application through the Launch Server. iPlanet UDS applications, including the iPlanet UDS Workshops, the Environment Console, and others, are now started using the `ftcmd run` command when you invoke them using the installed icons or scripts.

Applets Using the AppletSupport library, you also can design modular client applications by specifying parts of the application as *applets*. Applets are small applications or application modules that are designed to be started only by other client applications, not to be started independently. Most deployed client applications can be used as applets. *A Guide to the iPlanet UDS Workshops* and *iPlanet UDS Programming Guide* describes how to configure client applications as applets. The AppletSupport library is described in Framework Library online Help.

This chapter describes the Launch Server from two points of view:

- End user, who uses the iPlanet UDS Launcher application, to download and launch applications. See "About the Launcher Application" on page 294.
- System manager, who deploys applets and applications as assigned or publicly-available. The system manager defines how and whether an end user can download and run an application. The system manager also defines icons that launch applications using the `Ftcmd` utility. See "Setting up the Launch Server and Applications" on page 296.

About the Launcher Application

The *Launcher* application is a TOOL application that is provided as part of the runtime system on client nodes. An end user can use the Launcher application to start applications on her machine.

iPlanet UDS provides both an application distribution and the source TOOL code for the Launcher application. You can customize the Launcher application using the AppletSupport library. For information about using the AppletSupport library, see *iPlanet UDS Programming Guide* and Framework Library online Help. The Launcher application TOOL code is available in the `FORTE_ROOT/install/examples/frame` directory as `launcher.pex`.

Launcher application uses the Launch Server On UNIX platforms, Windows NT, and Windows 95, the Launcher application automatically starts the Launch Server, if the Launch Server is not already running.

The Launcher application uses the services of the Launch Server to display, run, and shut down applications. Applications started by the Launch Server run in the same process as the Launch Server, and can be shutdown by the Launch Server. iPlanet UDS provides the source code for the Launcher as an example. You can customize this application to suit your needs, as discussed in *iPlanet UDS Programming Guide*.

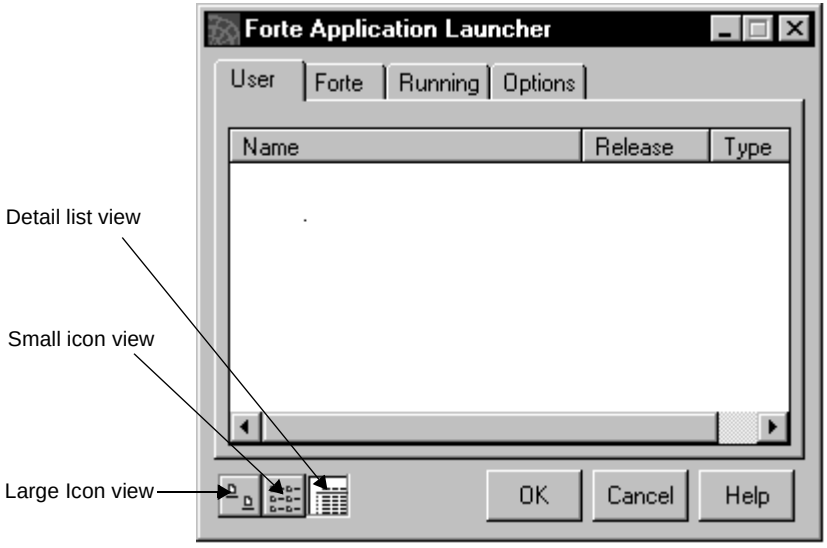
Alternatively, the end user can use icons that invoke commands using the Ftcmd utility to have the Launch Server start an application. You can define these icons to have the Launch Server start either assigned or publicly-available client applications. For more information about defining icons using the `Ftcmd` utility, see "Using the Ftcmd Utility" on page 307.

iPlanet UDS Launcher Application

This section describes how the Launcher works from the end user's perspective. For specific information about using the windows and controls of the Launcher, see the online help for the Launcher application.

iPlanet UDS provides a TOOL application, called Launcher, that is installed on an end user's machine with the iPlanet UDS runtime system. Figure 9-1 shows the Launcher:

Figure 9-1 Launcher



The Launcher application displays the following tab pages, which let the end user start and shutdown applications using the Launch Server:

User Applications available to the end user. You can start applications from this window.

iPlanet UDS iPlanet UDS product applications, such as the Environment Console, the iPlanet UDS Workshops, and so forth, which can be started by the Launch Server. You can start some of the iPlanet UDS product applications from this window.

Running Running applications that have been started by the Launch Server. You can shutdown applications from this window.

Options Shows the options that the user can set when running the Launcher application.

Launcher online help The online help for the Launcher application (available through the Help button) provides specific information about using the application to start and stop applications.

Starting the Launcher Application

The following table describes how to start the Launcher application on each platform:

Platform	How to Start the Launcher Application
Windows 95 or Windows NT	Click the Launcher Distributed or Launcher Standalone icon.
UNIX	Run the launcher script.

Setting up the Launch Server and Applications

As the system manager for a production environment, you are responsible for the deployment and maintenance of applications in iPlanet UDS environments.

This section explains how you can use the Launch Server to deploy applications efficiently to your end users. This section discusses the following topics:

- advantages of using the Launch Server
- restrictions for using the Launch Server
- deploying applications so that they are available to client nodes
- setting up the Launcher and icons so that an end user can start applications using the Launch Server
- using the `ftcmd` utility
- installing applications that use applets

Advantages of Using the Launch Server

There are several advantages to setting up client nodes in your environment so that end users can use the Launch Server:

Improved performance The Launch Server can run several iPlanet UDS applications under the same operating system process. When you start an iPlanet UDS client application through the Launch Server, the application uses the same set of runtime system objects—including the Log Manager, the Distributed Object Manager, and others—as the Launch Server.

By using the same process and the same runtime objects as the Launch Server, the client applications start significantly faster and use less memory than when they start as separate processes.

These applications also share a trace window with the Launch Server, which means that all log messages are printed to the same trace window (or log file, if you divert the trace information to a file). The messages from each application start with the name of that application, as shown:

The Launch Server automatically turns on the trace flag that has the trace messages start with the name of the application: “cfg:sp:8.” You can turn this trace flag on or off in other situations, if you wish.

Files automatically downloaded If you use the Launch Server to start an application that is assigned to the client node, the Launch Server ensures that the most recent image repositories for the client partition are installed. The Launch Server compares the creation time and date of the image repository to that in the environment, and automatically downloads the newer image repository, if needed.

If you start a publicly-available client application, iPlanet UDS downloads the necessary files, launches the application, then deletes the files when the application completes.

Restrictions

You can only launch standard (interpreted) client applications using the Launch Server. Compiled client applications cannot run as part of another process.

The Launch Server downloads only the image repository for the client partition. If the client partition requires that other libraries be deployed on the client node, you need to install the libraries separately.

After you start the Launch Server, the working directory for all applications that are started by the Launch Server is the working directory for the Launch Server. Because an application cannot predict its working directory, the application itself and environment variables used by the application should always specify absolute paths for files.

When the `ftexec` process for the Launch Server starts, it reads all the environment variables that are set for the client node. If any environment variables are changed for the benefit of an application started by the Launch Server, that application can only see the original value that was set when the Launch Server started.

If you need to have an application pick up an environment variable after the Launch Server has started, you can do one of the following:

- If the client node has either a running Node Manager or the Launch Server acting as a Node Manager, start `Escript` or the Environment Console. Locate the Active Partition agent for the FTLaunch application, which is named FTLaunch_cl0_client_*processID*. Set the environment variables using the `SetEnvRemote` command on this Active Partition agent. This command is described in *Escript and System Agent Reference Guide*.
- Stop the Launch Server, set the environment variable, and restart the Launch Server. This action kills all applications and applets started by the Launch Server.
- Set the environment variable and start the application using the `ftexec` command or a compiled executable.

NOTE The trace window for the Launch Server process also serves as the command-line window for any launched client applications that have command-line interfaces, such as Escript or Fscript. For this reason, we recommend that you not launch applications with command-line interfaces through the Launch Server.

Deploying Applications to Client Nodes

As the system manager, you need to deploy applications so that your end users can access the appropriate applications on their client nodes.

Assigned or publicly-available applications iPlanet UDS lets you define whether the client partition of an application (which represents the application on the client node) are assigned to a node or publicly available. *Publicly-available* applications can be downloaded and run from any client node (except Open VMS clients). When an end user starts a publicly-available client application, iPlanet UDS downloads the necessary files, launches the application, then deletes the files when the application completes.

When an application with an interpreted client partition is available to a client node by being assigned or publicly available, users on the client node can start the application by using one of the following interfaces to the Launch Server:

- Launcher application, described in "About the Launcher Application" on page 294

- `ftcmd run` command, which is used to define icons, described in "Setting up Icons or Scripts That Use the Ftcmd Utility" on page 303 and "Using the Ftcmd Utility" on page 307

Advantages of assigned and publicly available Both assigned and publicly-available applications have their own advantages when you are using the Launch Server:

- Assigning applications to specific client nodes is more efficient at runtime because the files needed to run the application do not need to be downloaded and deleted whenever the application is run.
- Defining client applications as publicly-available is more flexible and less work for you, because you do not need to assign the client partition to all possible nodes or model nodes where the user might want to run the application.

Applications can be both assigned and publicly available. If an application is assigned to a certain node when the user tries to start it using an interface to the Launch Server, then the application is considered an assigned application to that node. If the application is not assigned to that node, and the application is publicly available, then the application behaves as a publicly-available application for that node.

Publicly available only If you want a client partition to only be publicly available, but not assigned to any client node, you need to create a dummy client node (a node that does not actually correspond to any node in the environment), and assign the client partition to that node, because a client partition must be assigned to some node in the environment when you configure your application.

Deploying applets to client nodes *Applets* are applications with client partitions that have been deployed as parts of other applications. Applets are deployed in exactly the same way as other applications, except that no icons are generated for applets, and applets cannot be started directly by the Launcher application. Applets can either be explicitly assigned to a client node, or can be publicly available to all client nodes, just as other applications.

In the following sections—"Assigning Application Partitions to Client Nodes" on page 300 and "Defining Publicly-Available Applications" on page 301—the term "application" applies to both applications and applets.

For more information about deploying applets, see "Deploying Applications that Launch Other Applications and Applets" on page 313.

Assigning Application Partitions to Client Nodes

When you assign the partition of an application to a node or a model node, you explicitly define what nodes can install and use that partition. Applications assigned to certain nodes are installed on those nodes, either when you use the standard iPlanet UDS installation procedure or when the end user starts the application using the Launch Server. After assigned applications are installed, you have the choice of starting the applications with the Launch Server or starting them using ftexec or a compiled executable, as described in "Starting iPlanet UDS Applications" on page 185.

Assigning the application partition You can assign the client partition of an application to nodes or model nodes using commands in the Partition Workshop, Fscript, the Environment Console, and `Escript`. The steps for assigning partitions of applications to be started by the Launch Server are the same as for any other interpreted application. These steps are described in Chapter 5, "Deploying iPlanet UDS Applications."

Loading the application distribution You then need to—at minimum—make a distribution and load the distribution into the environment so that the environment is aware of the application and its partition assignments. Loading the distribution is required as the first step of installing the application, so if the application has been installed in the environment (and can be seen using the Environment Console or `Escript`), then you know that the distribution has also already been loaded. The Launch Server cannot locate an application until the application distribution has been loaded. See Chapter 5, "Deploying iPlanet UDS Applications" for information about loading and installing application distributions.

Automatic install and update of image repositories When the client node runs the Launch Server, the Launch Server automatically manages the task of installing the application files for the client partition. When an end user chooses to run an assigned application—for example, by double-clicking an icon in the Launcher application—the Launch Server checks that the application image repositories installed on the client node are as recent as the image repositories installed in the environment. The Launch Server automatically downloads the newer image repositories, if necessary.

Assigned server partitions downloaded If the client partition and one or more server partitions are assigned to a client node that can run server partitions, such as UNIX with Motif or Windows NT, the Launch Server downloads any assigned server partitions at the same time it installs the client partition.

For information about assigning partitions using the Partition Workshop or `Fscript`, see *A Guide to the iPlanet UDS Workshops* or *Fscript Reference Guide*. For information about using `Escript`, see *Escript and System Agent Reference Guide*.

Defining Publicly-Available Applications

Publicly-available applications are defined as available to any client node, even if the application is not explicitly assigned to a node or a model node. If an application is publicly available, but not assigned to a given node, the application files are downloaded when the user starts the application using an interface to the Launch Server. The Launch Server deletes the application files when the application completes.

Both applications and applets can be deployed as publicly available applications.

Client nodes see applications assigned to them as assigned applications, even if they are publicly available applications for nodes to which they are not assigned. Therefore, the application will be permanently installed on any node to which it is assigned.

You cannot start a publicly-available application on a client node unless you use one of the interfaces to the Launch Server.

Loading the application distribution and installing servers Before anyone can access a publicly-available application, you need to load the application, then assign and install the server partitions of the application, if any. You need to load the distribution into the environment so that the environment is aware of the application and its partition assignments. The Launch Server cannot locate the application until the application distribution has been loaded.

Defining what applications are publicly available To control which applications in the environment are publicly available, you need to edit and manage the following file provided by iPlanet UDS:

```
$FORTE_ROOT/install/scripts/pubaplts.inf
```

The copy of the file that controls what applications are publicly available is in the `FORTE_ROOT` directory on the node where the Environment Manager is running. The account or user that runs the Environment Manager must have read permission on this file. Any changes to this file take affect immediately, the next time the Launch Server needs to check the file.

The following table explains how to define what applications are publicly available:

Publicly Available Applications	Status of the pubaplts.inf file
No applications	The default. This file is present but empty.
All applications	Delete the file completely. If you later want to specify no or some applications as publicly available, you need to recreate this file.
Some applications	Define the list of applications, as described below.

In the `FORTE_ROOT/install/scripts/pubaplts.inf` file on the node where the Environment Manager is running, enter the application identifiers as text lines in the following format:

application_name[**_cl#**]

The *application_name* is not case-sensitive.

The following example shows how you could specify these applications:

```
TestApplication_cl0
TimeIt_cl4
Banking
```

Specifying just the application name, as with Banking in the above example, makes only the most recent release of the application publicly available.

Specifying both the application name and release, such as TimeIt_cl4, makes that specific release of the application publicly available. To make multiple releases of an application publicly available, you need to specify each application release separately, such as Accounting_cl0, Accounting_cl1, and Accounting_cl2.

Setting up Icons or Scripts That Use the Ftcmd Utility

In previous releases, iPlanet UDS generated icons for client partitions that start the application as a separate process in the client node's operating system.

iPlanet UDS now generates icons that use the `ftcmd run` utility to start the applications.

Windows platforms iPlanet UDS, generates icons on the Windows platforms that start the installed client partition using the Launch Server. These icons specify `ftcmd run` commands that have the Launch Server run the specified application.

UNIX On the UNIX platforms, you can define scripts or aliases that start client partitions using the `ftcmd run` command.

NOTE The `Ftcmd` utility is not available for OpenVMS platforms (VAX VMS or Alpha VMS).

For example, if you assign the client partition of the TimeIt example to a Windows client node and install the application on this node, the Command field of the Properties dialog for the icon would contain the following line:

```
ftcmd run TimeIt
```

The generated icons contain the `ftcmd run` command with the application name without the compatibility level.

You can also define icons that have the Launch Server start a client partition by using the following command syntax with the icons:

ftcmd run *application_name* [*release*] [*arguments*] [*update*]

For information about using this command, see "run" on page 311.

Generating icons with ftexec commands To have iPlanet UDS generate icons that start applications using the `ftexec` command, set the following configuration flag on each client node where you want this type of icon generated:

cfg:em:2

Set this configuration flag in either the Log Flags tab page of the iPlanet UDS Control Panel or the FORTE_LOGGER_SETUP environment variable. This configuration flag must be set before a Node Manager or a Launcher Server acting as Node Manager is started on the client node.

Starting the Launch Server

You can start the Launch Server by either using the icon for Launch Server or the `ftlaunch` script. When you start the Launch Server, it runs as a client-only application on the client node. The Launch Server contains a user-visible service object that manages how applications are started and shutdown in its process.

On Windows 95 and NT, and on UNIX platforms, the Launcher application and Ftcmd commands automatically start the Launcher Server, if it is not already running.

All iPlanet UDS client nodes on Windows platforms have a Launch Server Distributed and Launch Server Standalone icons defined.

NOTE If you plan to run any applications in distributed mode, you must start the Launch Server using the Launch Server Distributed icon.

Setting up the Port for the Launch Server

The Launch Server listens at a socket to receive commands from the Ftcmd utility. By default, a Launch Server listens at socket number 3783.

Setting up Launch Servers for UNIX On UNIX, you need to define a unique port number for each user, so that each user can run a separate instance of the Launch Server for himself. Before you can start the Launch Server using the `ftlaunch` command or the `ftcmd` command, you need to set the FORTE_FTLAUNCH_PORT environment variable, using a shell script or a profile, for each user session. You can also use the `-port` flag for the `ftcmd` command, as described in "Flags on the ftcmd Command" on page 307.

FORTE_FTLAUNCH_PORT environment variable If you have more than one Launch Server running on a machine, or if you have already assigned socket 3783 to another machine, you can change what socket the Launch Server listens at by changing the value of the FORTE_FTLAUNCH_PORT environment variable or by using the `-port` flag on the `ftlaunch` or `ftcmd` commands.

ftlaunch Command

The syntax for `ftlaunch` is:

Portable

ftlaunch [**-port** *port_number*] [**-fnd** *node_name*] [**-nonode**] [**-fs**]
[**-fns** *name_server_address*] [**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]

On Windows, you can specify these flags on the icons that start the Launch Server.

The following table describes the flags:

-port *port_number*	Identifies the port number at which this Launch Server will be started. On UNIX, either this flag must be specified, or the FORTE_FTLAUNCH_PORT environment variable must be set uniquely for each user.
-fnd *node_name*	Specifies the node name to use for the Launch Server. If the Launch Server becomes a node manager, this is used as the node name. Otherwise, this is used by the Launch Server to locate an existing node manager.
-nonode	Specifies that the Launch Server not check for a running Node Manager and not perform the functions of a Node Manager when it starts. Normally, the Launch Server checks whether a Node Manager is running on the client node, and if there is none, then the Launch Server behaves like a Node Manager.
-fs	Specifies that the Launch Server run as a standalone application. A Launch Server that starts as a standalone application can only launch iPlanet UDS applications, such as the iPlanet UDS Workshops and the Environment Console. When the Launch Server is started as a standalone application, it cannot run any publicly-available applications or update assigned applications because it cannot access the Environment Manager to get the information it needs.
-fns *name_server_address* **/NAMESERVER=** *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want the Launch Server to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in "Environment Manager Failover for Partitions" on page 122.
-fm *memory_flags*	Specifies the space to use for the memory manager. See "-fm Flag (Memory Manager)" on page 375 for syntax information. If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.
-fst *integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.

-fl *logger_flags*	Specifies the logger flags to use for the command. See "-fl Flag (Log Manager)" on page 371 for information about the syntax for specifying logger flags. If you do not set the logger flags, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. On UNIX, you must specify the logger flags in double quotes.

For example, the following command in a Windows icon or shortcut starts the Launch Server:

```
ftexec -fi bt:%{FORTE_ROOT}/userapp/ftlaunch/cl0/ftlaunch
```

Launch Server Details

Launch Server and Node Manager By default, the Launch Server checks whether a Node Manager is running on the client node, and if there is none, then the Launch Server behaves like a Node Manager. If there is a Node Manager running, then the Launch Server uses the services provided by the running Node Manager.

You can specify that the Launch Server not check whether a Node Manager is running and not perform then functions of a Node Manager by specifying either the `-fs` flag or the `-nonode` flag when you start the ftlaunch application to start the Launch Server.

Starting Launch Server at system startup If you start the Launch Server as part of the system startup routine, all the runtime objects and the Node Manager that are needed by any client partitions are started at that time, instead of when the first client application is launched by the Launch Server. Later in the session, when the user starts a client application using the Launch Server, the client application shares the runtime objects and Node Manager associated with the Launch Server, which can considerably reduce the start time.

Shutting down the Launch Server On Windows NT and Windows 95 platforms, you can shut down the Launch Server using the Launch Server Shutdown icons. You can stop the Launch Server on UNIX using the `ftcmd shutdown server` command.

Log files in UNIX The log file for a Launch Server on UNIX is named `ftlaunch_`*port*`.log`, in the `FORTE_ROOT/log` directory, where *port* is the socket number set for the Launch Server using either the `-port` flag on the `ftcmd` command or the FORTE_FTLAUNCH_PORT environment variable.

On Windows platforms, the Launch Server writes its messages and output to the Launch Server trace window.

Using the Ftcmd Utility

The Ftcmd utility is the command interface to the Launch Server. You can define scripts containing these commands to launch, list, update, and shutdown client applications through the Launch Server.

On Windows 95, NT, and UNIX, using any Ftcmd command starts the Launch Server on the client node, if the Launch Server is not already running on that node.

The Ftcmd utility provides a command interface to the Launch Server, but is not a command-line interface application. In other words, you enter commands to the Ftcmd utility from the command line for your environment or a script without actually entering a separate command-line window.

All commands start with ftcmd All Ftcmd commands must specify `ftcmd` with the command name, as shown in the following example, which lists all available client applications:

```
ftcmd list all
```

The `Ftcmd` utility supports the following commands:

list Lists available client applications running inside the Launch Server.

run Launches a client application.

shutdown Shuts down the client partition of one or more client applications or the Launch Server itself.

update Updates the installed application, if its image repositories are not as current as the image repositories stored in the environment.

These commands are described in "Ftcmd Commands" on page 310.

Flags on the ftcmd Command

The syntax for the `ftcmd` command is:

ftcmd [**-v**] [**-nolog**] [**-port** *port_number*] [**-fnd** *node_name*] [**-nonode**] [**-fs**]
[**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]
{**-i** *input_file* | *command*}

Flags that affect Ftcmd commands The following flags affect how the `Ftcmd` utility invokes a command:

Flag	Description
-v	Displays detailed information about the steps performed by the Ftcmd command. This flag works only on the current `Ftcmd` command. This flag is useful for diagnosing command syntax errors.
-port *port_number*	Identifies the port number that this `Ftcmd` command will use to locate a Launch Server. If a Launch Server is not already running at this port number, a Launch Server is started at this port number.
-i *input_file.*	Run a single `ftcmd` command in the specified file.
command	One of the following commands: `list`, `run`, `shutdown`, and `update`. For more information about these commands, see "Ftcmd Commands" on page 310.

Input files Using the `-i` flag, you can specify that the `Ftcmd` utility invoke a single command stored in a file. To use this flag, define a file that contains one `Ftcmd` command without the `ftcmd` prefix, as shown in the following example, which represents the contents of a file named `cmd.txt`:

```
run banking
```

You can then run the command in the file by entering the following command at the command prompt:

```
ftexec -i cmd.txt
```

Flags that affect the Launch Server On Unix, Windows 95, and Windows NT, the `ftcmd` command also starts a Launch Server if a Launch Server is not running at the port number used by the `Ftcmd` utility.

If the `ftcmd` command starts a Launch Server, the flags in the following table affect that Launch Server. If the Launch Server is already running, these flags are ignored.

Flag	Description
-nolog	UNIX only. Prevents the `Ftcmd` utility from redirecting the Launch Server's output to a log file if the Ftcmd utility needs to start the Launch Server. If the Launch Server is already running, this flag has no effect. By default, the started Launch Server logs its output to a file in `FORTE_ROOT/log` named `ftlaunch_`*port_number*`.log`, where *port_number* is the port where the Launch Server is running.
port_number	Identifies the port number that this `Ftcmd` command will use to locate a Launch Server. If a Launch Server is not already running at this port number, a Launch Server is started at this port number.
-fnd *node_name*	Specifies the node name to use for the Launch Server. If the Launch Server becomes a node manager, this is used as the node name. Otherwise, this is used by the Launch Server to locate an existing node manager.
-nonode	Tells the Launch Server not to check for a running Node Manager on the client node and not to act as a Node Manager. Normally, the Launch Server checks whether a Node Manager is running on the client node, and if there is none, then the Launch Server behaves like a Node Manager
-fs	Specifies that the Launch Server run as a standalone application. A Launch Server that starts as a standalone application can only launch iPlanet UDS applications, such as the iPlanet UDS Workshops and the Environment Console. When the Launch Server is started as a standalone application, it cannot run any publicly-available applications or update assigned applications.
-fm *memory_flags*	Specifies the memory settings for the ftexec process that runs the Launch Server.
-fst *integer*	The thread stack size in bytes for iPlanet UDS and POSIX threads. See "-fst Flag (Stack Size)" on page 378 for syntax information. This specification overrides default stack size allocation.
-fl *logger_flags*	Specifies the log flags for the `ftexec` process that runs the Launch Server.

The `-fm` and `-fl` flags are as described in "iPlanet UDS Logger and Memory Manager Flags" on page 371.

You can also use the FORTE_FTLAUNCH_FLAGS environment variable to set flags that will be used when the `ftcmd` command starts a Launch Server.

Ftcmd Commands

You can use the following commands for the `Ftcmd` utility to deploy, update, start, stop, and manage running applications:

- list
- run
- shutdown
- update

list

The `list` command lists the available client applications. The list is printed to the command line where you entered the command.

The argument of this command specifies whether to list applications that are assigned, publicly available, iPlanet UDS product, or all of these.

ftcmd list all | assigned | **public** | forte | **running**

Argument	Description
all	Lists all applications available to the current client node.
assigned	Lists the applications containing client partitions assigned to the current client node.
public	Lists the publicly-available applications available to the current client node.
forte	Lists the iPlanet UDS product application available to the current client node.
running	Lists all running applications that were started by the Launch Server. Applications running in the Launch Server are given integer identifiers (Applet IDs) that are unique for the Launch Server.

The lists produced by the `ftcmd list` command with the `all`, `assigned`, `public`, or `forte` arguments contain the name and release, and indicate whether the application is assigned, publicly available, or an iPlanet UDS product application.

Applets not listed This command does not list applets, which are client partitions that are not intended to be started independently. For more information about applets, see *iPlanet UDS Programming Guide*

The list produced by the `ftcmd list running` command contain the identifier, the application name, and the release for the running client application.

run

The `run` command starts the specified application using the Launch Server. You can define icons that invoke this command to start applications.

ftcmd run *application_name* [*release*] [*arguments*] [*update*]

Argument	Description
application_name	Specifies the name of the application that is being launched by the Launch Server.
release	Specifies the compatibility level to be started. By default, the Launch Server starts the release with the highest compatibility level.
arguments	Specifies command line arguments for the application being started.
update	For installed client partitions only. Indicates whether the Launch Server automatically downloads and installs the most current application distribution, if the application has not yet been installed, or if the installed application is not as current as the distribution available in the environment. By default, the value is `TRUE`, which means that the more current application distribution is automatically downloaded and installed, if necessary. To prevent this automatic update, set this value to `FALSE`. This argument is ignored for publicly-available applications. Note that only the client partition of an application is downloaded, not any libraries that the client partition might depend upon.

If you specify just `ftcmd run` *application_name* without the *release* or *update* arguments, the Launch Server runs the application with the highest compatibility level and updates the application distribution if an application distribution that is more current than the installed one is available in the environment.

If you specify `ftcmd run` *application_name release arguments* `FALSE`, the application is started without checking whether the installed image repositories are as current as the image repositories in the environment.

Specifying arguments The command line arguments need to be interpreted by Fa string in quotation marks, for example "-fr CentralRepository -fw tempworkspace".

On some platforms, such as UNIX and Windows, you need to use escape characters, such as \, to have the quotation marks interpreted correctly. Therefore, to specify arguments, you might need to specify them using something like \"-country canada\", so that the quotation marks are not removed before the `Ftcmd` utility receives them. To see the actual command string that is being sent to the Ftcmd utility, specify the `-v` flag, as discussed in "Ftcmd Commands" on page 310.

The following example demonstrates how to start an application on a Windows NT node called OLMBanking and specify an argument for that application:

```
ftcmd run olmbanking cl0 \"-country canada\"
```

shutdown

The `shutdown` command shuts down the client partition of the application that has the specified identifier, as well as any applications that might have been started by this application.

ftcmd shutdown *app_ID* | **all** | **server**

Argument	Description
app_ID	Specifies the integer identifier of the application to be shut down. To get this identifier, use the `ftcmd list running` command.
all	Shuts down all client partitions started by the Launch Server.
server	Shuts down the Launch Server.

update

The `update` command downloads and installs the most current image repositories for the client partition of the specified application. If the installed image repositories are as current as the image repositories in the environment, then the image repositories are not reinstalled.

This command is only useful for applications that are assigned to the current client node. Do not use this command with publicly-available applications.

ftcmd update *application_name* [*release*]

Argument	Description
application_name	Specifies the name of the application to be updated.
release	Specifies the release (compatibility level) to be updated. By default, the Launch Server downloads and installs the image repositories for the release with the highest compatibility level.

NOTE Only the client partition of an application is downloaded, not any libraries that the client partition might depend upon.

Deploying Applications that Launch Other Applications and Applets

Using the `LaunchMgr.RunApplet` method, application developers can design applications that start other iPlanet UDS client applications in the same `ftexec` process, but in separate partitions.

Applets An *applet* is a client application that is started as part of another application. In design and function, an applet is actually a small independent application. When the application developer makes the application distribution, she marks the application as an applet, whose client partition is to be started only by using the LaunchMgr.RunApplet from another "main" client application.

Locating deployed applets in the environment You can see whether an application distribution is an applet by checking the Partition agents for the client partition. In the Environment Console, the Type for the client partition says Partition (Applet). In `Escript`, the Partition Type for the Partition is Client (Applet).

Applets are deployed in exactly the same way as other applications, except that no icons are generated for applets, and applets cannot be started by the Launcher application. Applets can either be explicitly assigned to a client node, or can be publicly available to all client nodes, just as other applications can. For specific information about assigning applets or making them publicly available to client nodes, see "Deploying Applications to Client Nodes" on page 298.

When you deploy a client application that starts applets or applications, you must also deploy the required applets or applications independently. Ideally, you should have a script for this installation.

After you have deployed an applet, you will notice that it is not visible on the client node the way a regular client application is:

- the Launcher application does not display an applet as an application that can be started
- iPlanet UDS does not automatically create icons for this applet the way it does for other applications on the Windows platforms
- the `ftcmd list` command does not include applets in its list of available applications

Troubleshooting Client Applications That Use Applets

Check the log file or trace window If you have problems with a client application that starts other applets or client partitions, you can diagnose the source of the problems by checking the trace window or log file for the main client application. The application prints trace information about the application itself and the applications or applets it starts in the same trace window or log file.

The Launch Server automatically turns on the trace flag that has the trace messages start with the name of the application: "cfg:sp:8." You can turn this trace flag on or off in other situations, if you wish.

Log files in UNIX The log file for a Launch Server on UNIX is named ftlaunch_*port*`.log`, in the `FORTE_ROOT/log` directory, where *port* is the socket number set for the Launch Server using either the `-port` flag on the `ftcmd` command or the FORTE_FTLAUNCH_PORT environment variable.

On Windows platforms, the Launch Server writes its messages and output to the Launch Server trace window.

Debugging UsageExceptions If the main client application generates a UsageException exception with a SP_ER_INVALIDSTATE reason code, then one of the applications or applets it starts is missing. Information about exceptions that can be raised when the main client application starts another application, see the documentation for the `LaunchMgr.RunApplet` method in the AppletSupport library in Framework Library online Help.

Appendix A

Special Setup for Development Environments

A number of development situations require special environment setup tasks above and beyond the regular setup tasks described in Chapter 3, "Setting up and Maintaining an iPlanet UDS Environment" and Chapter 8, "Managing iPlanet UDS Development Repositories."

For example, before developers can use iPlanet UDS's auto-compile option when making a distribution containing compiled components, you must first deploy a number of auto-compile services, and make them available in the development environment.

For developers to write and partition applications that integrate with OLE 2 (Object Linking and Embedding) applications, you must specify the nodes on which these products reside.

This appendix covers the following topics:

- setting up auto-compile services
- setting up support for OLE

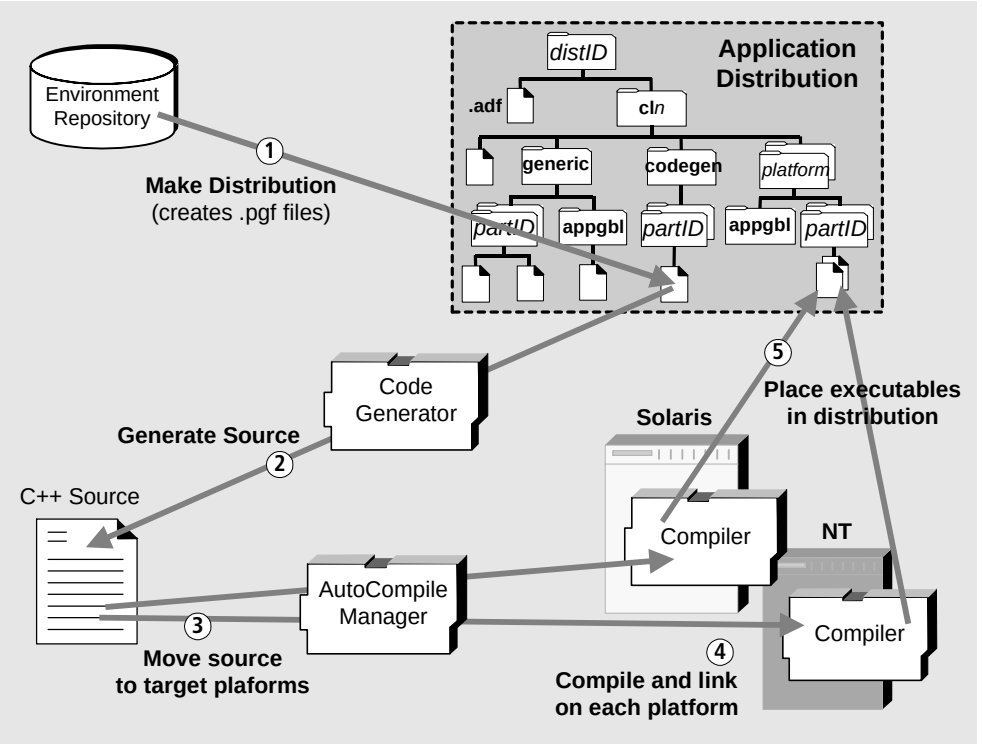
Auto-Compile Services

Developers making an application or library distribution have the option of automating the compiling and linking process required for compiled distribution components. For the auto-compile option to work, however, a number of auto-compile services must be deployed in the development environment. This section describes the auto-compile process and the application architecture employed to automate it. It also describes how to deploy the two applications that comprise the auto-compile services, AutoCompileSvc and CodeGenerationSvc.

Auto-Compile Process

This section describes the work done by the auto-compile services. The steps in the auto-compile process and the services used to automate them are shown schematically in Figure A-1.

Figure A-1 Auto-Compile Process



The auto-compile services perform the following steps to make a distribution with compiled components:

1. Create partition (.pgf) or library (.lgf) generation files with the Partition Workshop Make Distribution command or the Fscript `MakeAppDistrib` command.

   Strictly speaking, this step is not part of the auto-compile process, but it is the first step performed when a developer makes a distribution. For more information on this step, see "About Application and Library Distributions" on page 149.

2. Create platform-independent C++ source code from the information contained in the .pgf or .lgf file.

   A code generator service (CodeGenerationSvc_cl0_Part1), which has been installed on one or more server nodes in the development environment, generates this C++ code.

3. Move the C++ source code to one node of each platform on which a partition or library is to be compiled and on which a compiler service (AutoCompileSvc_cl0_Part2) has been installed.

   This process is coordinated by an auto-compile manager service (AutoCompileSvc_cl0_Part1), which knows (from assigned partition properties) on which platforms the C++ source is to be compiled and on which nodes the compiler service (AutoCompileSvc_cl0_Part2) has been installed.

4. Compile and link the C++ source code on each target platform.

   The auto-compile manager service (AutoCompileSvc_cl0_Part1) starts a compiler service on each target node to which it has moved C++ source code. The compiler service (AutoCompileSvc_cl0_Part2) compiles and links the code, creating a compiled executable. When this operation is complete, the auto-compile manager shuts down the compiler service.

5. Move compiled executable files to the appropriate directory within the distribution.

   The auto-compile manager service moves the compiled partition to the location in the distribution that corresponds to the partition's name and platform format.

Auto-Compile Application Architecture

The auto-compile services discussed in the previous section and illustrated in Figure A-1 are provided by two distributed server applications: CodeGenerationSvc and AutoCompileSvc.

CodeGenerationSvc

Part1 (Generator) Generates portable C++ source code. This partition requires a sizeable amount of memory: 20-30 Megabytes. Normally code generation takes a few seconds to complete. The partition can be replicated for load balancing in environments with a very heavy load. The Generator partition should generally be started and left running; it takes about 5 minutes to start.

Part1 Router Routes requests to multiple Generator partitions, if more than one is active.

AutoCompileSvc

Part1 (Manager) Coordinates the process of moving C++ source code to different nodes for compiling and linking, starting and shutting down the compiler partitions, and then moving the compiled partitions to the appropriate location in a distribution. The partition cannot be replicated. It communicates heavily with the Environment Manager, and is therefore best placed on the same node. The Manager partition should generally be started and left running.

Part2 (Compiler) Performs compiling and linking on the host node. The partition can be replicated, and a replicate should be placed on at least one node of each platform architecture. The Compiler partition should not be started, since it is started automatically by the Manager when needed, and then shut down.

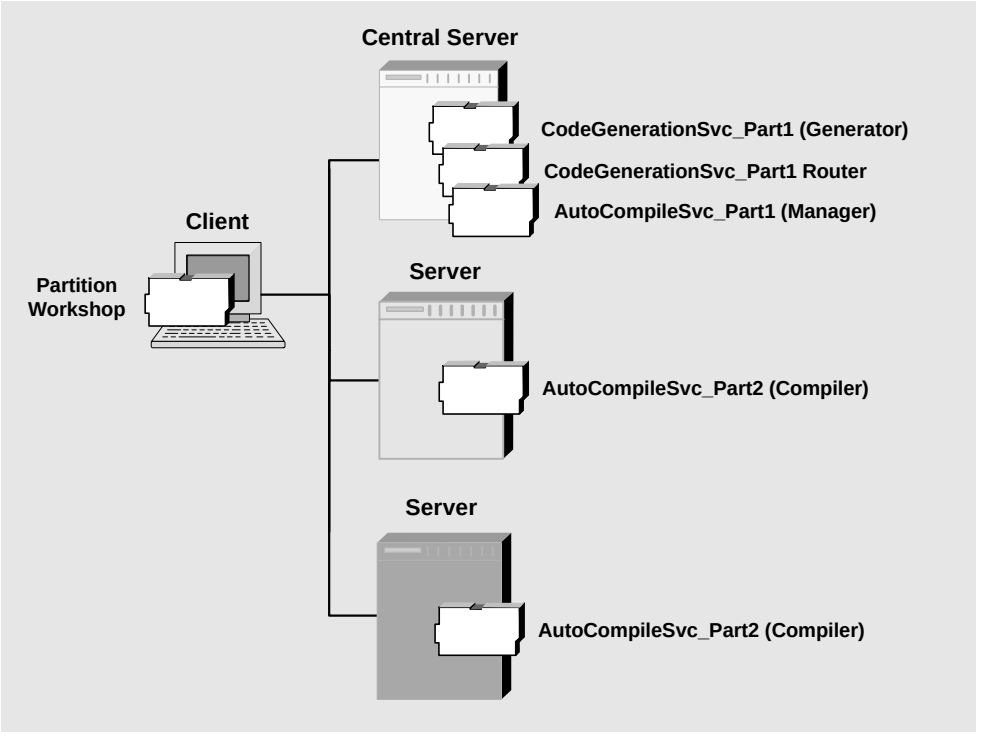
An example auto-compile configuration is illustrated in Figure A-2 on page 319. The iPlanet UDS environment consists of a large number of client workstations (only one is shown) used by developers to write their applications. There is also one or more central development repositories (not shown) on one or more of the four servers in the illustration. Servers 1, 2, and 3 are of different architectures (the central server is the same architecture as Server 2).

In Figure A-2, one instance of CodeGenerationSvc_cl0_Part1 (Generator) has been installed on the central server, along with its router. A second instance of the Generator partition could be installed on another server, or started on the central server, if the load on the first partition becomes too heavy. The Generator requires 20-30 Megabytes of memory.

AutoCompileSvc_cl0_Part2 (Compiler) has been installed on Servers 1, 2, and 3—one node of each architecture. Placement on these nodes is necessary for the auto-compile service to compile partitions for each platform architecture in the environment. Additional Compiler partitions can be installed on other nodes for backup purposes, if you wish.

AutoCompileSvc_cl0_Part1 (Manager) can be installed on any server node. Since it communicates with the Environment Manager, it has been installed on the central server node in this example.

Figure A-2 Auto-Compile Configuration Example



Setting up the Auto-Compile Feature

This section explains the steps you need to follow to configure and install the auto-compile applications for your environment. Windows 95. If you want to use the auto-compile feature with Windows 95, you need to consider some additional issues. These issues are discussed in "Using Auto-Compile with Windows 95" on page 323.

iPlanet UDS provides an auto-compile feature that automatically generates and compiles C++ code for server partitions that you have designated as compiled.

C++ compilers and linkers To use the auto-compile feature, you need to install the C++ compilers and linkers appropriate for each platform on the nodes where you plan to compile C++ code. The C++ compilers and linkers that can be used on each platform are specified in the *iPlanet UDS System Installation Guide*.

Before you can use the auto-compile feature, you need to have the following iPlanet UDS applications installed in your environment:

AutoCompileSvc Compiles C++ code on the nodes where it is installed. This application has two partitions: Part1, which manages the auto-compile process, and Part2, which compiles the C++ code on the nodes where it is installed.

CodeGenerationSvc Generates portable C++ source code. This application has two partitions: Part1, which generates the C++ code, and Router, which manages load balancing if the partition is replicated on more than one node.

The steps in deploying both the CodeGenerationSvc and AutoCompileSvc applications are the same as for any other iPlanet UDS application distribution, except that the iPlanet UDS installation program has already loaded both application distributions into your environment repository and installed them on your central server node. You have to modify their partitioning configurations, and install them in your active environment. For details on deploying applications, see Chapter 5, "Deploying iPlanet UDS Applications."

Configuring the Auto-Compile Services

When you install iPlanet UDS in your environment, these applications and their application distributions are automatically installed on the central server node. These applications are not installed on any other server nodes or any client nodes.

Since the default partition assignments in the distributions do not correspond to nodes in your development environment, they were dropped when the installer loaded the application distributions into your environment repository. iPlanet UDS then assigned each of the four partitions to the central server by default. You will have to manually re-assign each of these partitions by hand, as you wish.

You need to configure and install the Part2 partition of the AutoCompileSvc application on each node where you want the server partitions compiled.

➤ **To configure and install the Part2 partition of the AutoCompileSvc application using the Environment Console**

1. Start the Environment Console.
2. Change to the node outline view by selecting the View > Node Outline command.
3. Display the lists of applications on the central server node and the nodes where you want to install the Part2 partition of the AutoCompileSvc application by clicking the expansion arrow.

4. Lock the environment by clicking the Locked toggle on the tool bar.
5. Copy the AutoCompileSvc_cl0_Part2 partition by selecting this partition on the central server node and selecting the Edit > Copy command.
6. Paste the AutoCompileSvc_cl0_Part2 partition onto the nodes where you want to install it by selecting each node and selecting the Edit > Paste command.
7. Disable all but one of the assigned Compiler partition (AutoCompileSvc_cl0_Part2).

   This partition is automatically started and shutdown as needed, however, the partition must be enabled on at least one node (for example, on your central server).
8. Unlock the environment by clicking the Locked toggle on the tool bar.
9. Display the application outline view by selecting Application Outline from the View menu.
10. Select the AutoCompileSvc_cl0 application.
11. Install the AutoCompileSvc_cl0_Part2 partition on the newly-assigned nodes by selecting the Component > Install command.

➤ **To configure and install the Part2 partition of the AutoCompileSvc application using Escript**

1. Start `Escript`.
2. Enter the following sequence of Escript commands, substituting the name of the node for the <node_name>:

```
FindSubAgent AutoCompileSvc_cl0
LockEnv
AssignAppComp <node_name> AutoCompileSvc_cl0_Part2
# You can assign Part2 to several different nodes at this
point.

Commit  # Unlocks the environment automatically
Install  # Installs Part2 on the newly-assigned nodes
Refresh
```

Starting up the Auto-Compile Services

After you complete installation, start the CodeGenerationSvc and AutoCompileSvc applications. If you do not start these applications, they will auto-start the first time a developer tries to auto-compile her application; auto-starting these applications takes about five minutes.

To start up the CodeGenerationSvc application, use the `Startup` command on the Application agent for this application.This command starts the CodeGenerationSvc_cl0_Part1 and Router.

You should also start the partition of the AutoCompileSvc application that manages the auto-compilation process, AutoCompileSvc_cl0_Part1. To start up this partition, locate its Installed Partition agent and invoke the `Startup` command. A developer should now be able to invoke the auto-compile option successfully, starting up the appropriate instances of AutoCompileSvc_cl0_Part2.

For information about the `Startup` command for the Application and Installed Partition agents, see *Escript and System Agent Reference Guide*.

Troubleshooting the Auto-Compile Feature

If you have problems running the auto-compile feature on a particular node, make sure that the PATH and LIBPATH environment variables are set properly.

Make sure that the C++ compiler executable is in a directory specified in the PATH environment variable used by the Node Manager for each node that compiles partitions. Otherwise, the auto-compile partition will not be able to find the C++ compiler.

Debugging Errors When Using Auto-Compile

If the auto-compile feature encounters an error, you can look in the following places for error information:

- Compile Distribution dialog, which shows the progress of the auto-compile feature when you auto-compile a distribution
- log files for the AutoCompileSvc application
- trace information for the iPlanet UDS Workshops—either the Console window or the file where stdout is being sent

Using Auto-Compile with Windows 95

If you want to use the auto-compile feature to compile partitions that run on the Windows 95 platform, you have two choices:

- Compile partitions as Windows NT partitions, then include the compiled executables or libraries as part of the Windows 95 distribution.

  This approach is recommended, because most Windows NT machines are usually more powerful than Windows 95 machines, which makes compilations more efficient.

- Set up a Windows 95 node to act as a server that can auto-compile Windows 95 partitions.

  This option is recommended if your environment does not include any Windows NT nodes.

Compiling Partitions as Windows NT partitions

You can compile partitions as Windows NT partitions, then include the compiled executables or libraries as part of a Windows 95 distribution because Windows 95 code and Windows NT code are binary compatible. You might want to use this approach if you have Windows NT nodes already available with auto-compile services installed. Compiled Windows NT executables and libraries also run on Windows 95.

➤ **To compile partitions in Windows NT and use them in Windows 95 distributions**

1. In Fscript or the Partition Workshop, assign to Windows NT nodes the partitions that you want to compile and run on Windows 95.
2. Make a distribution for the application using auto-compile, but not auto-install.
3. Copy the compiled executable or library for the partition from pc_nt to the pc_win95 directory for that partition in the `FORTE_ROOT/appdist` directory.

   For example, if you have assigned the client partition for the Banking_cl0 application to a Windows NT node to auto-compile it, you need to copy the following file:

   `FORTE_ROOT/appdist/`*environment_ID*`/banking/cl0/pc_nt/bankin0/bankin0.exe`

   Copy this file to the following directory (make the pc_win95 and client partition directories first, if they does not already exist):

   `FORTE_ROOT/appdist/`*environment_ID*`/banking/cl0/pc_win95/banking0`

For more information about the Partition Workshop, see *A Guide to the iPlanet UDS Workshops*. For information about Fscript commands, see *Fscript Reference Guide*.

➤ **To install the Windows 95 distributions**

1. In the Environment Console or `Escript`, load your application, lock the active environment, then assign the client partitions to Windows 95 nodes and set the partitions as compiled. Unlock the active environment when you are done.

   You might get a warning on this step that says that the distribution does not contain the compiled partition for Windows 95. You can ignore this warning and continue.

2. Install the application using the `Install` command for the Application agent.

For more information about the Environment Console, see Chapter 2, "The iPlanet UDS Environment Console."

For information about `Escript` and iPlanet UDS system agent commands, see *Escript and System Agent Reference Guide*.

Setting up a Windows 95 Node to Auto-Compile

You can have a Windows 95 node act as a server node to run the auto-compile compiler service (AutoCompileSvc_cl0_Part2).

NOTE This is the only situation in which you should use a Windows 95 machine as a server node.

➤ **To set up a Windows 95 node to run the auto-compile compiler service**

1. Set aside a Windows 95 node to be used to compile partitions. This node must have the correct C++ compiler installed. You should use a node that is not also simultaneously used to run the iPlanet UDS workshops because the compiler service is memory-intensive and will slow the performance of the iPlanet UDS workshops.

2. Define the Windows 95 node as a unique Windows 95 node in the active environment. Do not define this node as a model node or as part of any model group. For example, if your Windows 95 nodes are defined by a model node named Win95_clients, you must create a unique node that represents this Windows 95 node.

3. Start the Launch Server on the Windows 95 node to run a node manager on that node. For information about starting the Launch Server, see "Starting the Launch Server" on page 304.

4. Copy the AutoCompileSvc_cl0_Part2 partition to the Windows 95 node and reinstall the AutoCompileSvc, as described in "Configuring the Auto-Compile Services" on page 320.

Support For OLE

Developers writing applications that use OLE services or applications can also use interface classes provided by iPlanet UDS. These interface classes, in the form of projects (or system libraries), are automatically installed by the iPlanet UDS installation program on each node, and can be referenced by developers through supplier projects (or system libraries) residing in your central development repositories.

Any partition referencing an OLE object must reside on a node where the respective service or application resides. For an application to be properly partitioned, therefore, the iPlanet UDS partitioning facility must know which nodes in your development environment host the OLE services or applications. You supply this information by specifying the name of the appropriate interface project in the Installed Library property of the node on which the DCE, ObjectBroker, or OLE service or application resides.

Accordingly, for each node in your environment on which OLE software has been installed, you should modify the Installed Library property of the node (in the environment definition of your active environment). You can use the Environment Console or Escript utility to make this change.

➤ **To specify support for DCE, ObjectBroker, or OLE on a node**

1. In the Environment Console, select the node in the Active Environment window.
2. Lock the environment definition.
3. Select Properties… from the Component menu.

   The Node Properties dialog opens.
4. Select Installed Libraries from the drop-down list at the bottom of the Node Properties dialog.
5. Enter the name of the appropriate interface: "OLE."
6. Close the Node Properties dialog and save the change.

Appendix B

iPlanet UDS Environment Variables

This appendix describes the iPlanet UDS environment variables, listed in alphabetical order. You can set these variables on any platform, unless otherwise noted.

This appendix also lists the logical names used by OpenVMS. These names allow you to specify settings that are used when iPlanet UDS automatically starts a server; they also provide international support for OpenVMS.

Finally, this appendix explains how to set environment variables with and without the iPlanet UDS Control Panel.

If you set environment variables that are used by the Node Manager for a particular node, you must set the environment variables before you start the Node Manager; otherwise, the Node Manager does not pick up the environment variable values. Because the Environment Manager is a special kind of Node Manager, this requirement also applies to the Environment Manager.

Environment Variable Descriptions

This chapter describes all the environment variables defined by iPlanet UDS.

Use upper case to specify environment variable names on all platforms.

FORTE_ALL_FILES_SHARED

(For OpenVMS only) Specifies that all OpenVMS files created or opened by iPlanet UDS are opened as shared and can be read by multiple processes.

FORTE_ALL_FILES_SHARED *any__non_blank_char*

If this variable is not set, then iPlanet UDS treats files in the OpenVMS default manner; files are created as unshared and only one process may open a file for writing or reading. To change this treatment, set this variable to `TRUE` or any non-blank character. To override the default for a given file, use the `SetFileAttribute` method on the File class.

FORTE_AUTOTESTER_DELAY

Sets the delay in milliseconds between input bursts when "playing back" an application using the AutoTester project. The default is `1000`.

FORTE_AUTOTESTER_DELAY *integer*

FORTE_AUTOTESTER_ROOT

When using AutoTester, this variable specifies that file names should be captured in a portable format, and indicates the directory path to be used to determine the location of the files.

FORTE_AUTOTESTER_ROOT *directory_specification*

FORTE_CG_RESERVED

Specifies a file containing reserved words to supplement the list of iPlanet UDS reserved words.

FORTE_CG_RESERVED *file_specification*

The reserved word file is optional. You should provide this file when your project uses names for class elements that are already reserved by your C++ compiler. The iPlanet UDS code generator uses the reserved words file to rename the class components in order to avoid conflicts.

If you do not set the variable, the default is `$FORTE_ROOT/install/scripts/cgreserv.lst`.

FORTE_CODEGEN_OCTAL

Specifies that code generation will convert non-ASCII characters in strings into equivalent octal constants. By default, characters are not converted.

FORTE_CODEGEN_OCTAL {**TRUE**|**FALSE**}

Use this variable when you are generating TOOL code that contains strings with characters that are not legal for some native compilers, such as Kanji characters. Setting this environment variable to `TRUE` (which is typically done by the system manager) ensures that native compilers can correctly compile generated code containing any type of character.

FORTE_COSSHR

Specifies the location and name of the ObjectBroker name service shared library.

```
FORTE_COSSHR file_specification
```

Also see "FORTE_OBBSHR" on page 337.

FORTE_CTLIB_LOCK

If TRUE, disables multi-threaded database access. Use this variable to implement serial database access on UNIX platforms with native-threaded partitions.

```
FORTE_CTLIB_LOCK {TRUE|FALSE}
```

For more information, refer to the *Accessing Databases* manual.

FORTE_DB_MAX_STATEMENTS

Sets the number of TOOL SQL statements kept prepared in the statement cache.

```
FORTE_DB_MAX_STATEMENTS integer
```

Increasing this value may improve performance of your application if your application uses a large set of TOOL SQL statements. The default varies by database vendor; see the manual *Accessing Databases* to see the value for your database.

FORTE_EDITOR

Specifies the editor to be invoked by the Fscript `Vi` command.

```
FORTE_EDITOR editor_executable
```

FORTE_EP_WRKDIR

Used when running a number of the iPlanet UDS example programs, specifies the current working directory.

```
FORTE_EP_WRKDIR directory
```

Note that a few other environment variables that begin with FORTE_EP are used for one or more example programs. This variable is used most often.

FORTE_FTLAUNCH_FLAGS

Specifies command line flags in effect when the Launch Server is started with the `ftcmd` command.

```
FORTE_FTLAUNCH_FLAGS '-flag [-flag ...]'
```

If this environment variable is set, any flags it specifies are added to the command line when ftcmd starts the launch server. The valid flags for the Launch Server (ftlaunch) are described in "ftlaunch Command" on page 304. Some flags are described under the `forte` command. An example follows:

```
FORTE_FTLAUNCH_FLAGS '-fs -fm (n:4000,x:10000)'
```

FORTE_FTLAUNCH_PORT

Specifies the port number for the launch server.

`FORTE_FTLAUNCH_PORT` *`tcp_port_number`*

If this variable is set, then `ftcmd` uses the specified port, instead of the default (port 3783), to contact the launch server. The port number must be a numeric value which is a legal and unused TCP port for the underlying platform and operating system.

Because this variable should be set individually for each iPlanet UDS user, it should not be set in a centralized fortedef file which is accessed by multiple iPlanet UDS users.

For more information about this environment variable, see "Setting up the Port for the Launch Server" on page 304.

FORTE_GC_SPECIAL

Sets the memory allocated for iPlanet UDS partitions. The syntax for this variable is identical to the syntax for the `-fm startup` flag (also used to set memory).

`FORTE_GC_SPECIAL` (*`memory_option`* `{:|=}` *`number`* `[,` *`memory_option`* `{:|=}` *`number`*`])`

The syntax and default values for the memory flags are described in "-fm Flag (Memory Manager)" on page 375. An example follows:

```
FORTE_GC_SPECIAL (n:5000, x:10000)
```

Memory flags are set according to the following rules:

- If the `-fm startup` flag is set, then FORTE_GC_SPECIAL is ignored.
- If the `-fm startup` flag is not set, then the setting for FORTE_GC_SPECIAL is used.
- Any memory flags that are not explicitly set use the default values.

FORTE_ISFLOATOVERENABLED

Specifies whether float-over help is turned on. By default, float-over help is enabled (TRUE).

FORTE_ISFLOATOVERENABLED {**TRUE**|**FALSE**}

To turn off float-over help, set this variable to FALSE.

FORTE_KEEP_COUNT

Specifies the number of pings that the keepalive feature attempts before it closes the connection. Default is 3.

FORTE_KEEP_COUNT *integer*

FORTE_KEEP_CYCLE

Specifies the length of time, in seconds, that a connection can be inactive before keepalive processing starts. The default value is 60 seconds.

FORTE_KEEP_CYCLE *integer*

If this value is set to f, all keepalive processing is disabled. If a client has this value set to 0, the client sends a message to each partition it connects to, telling that partition not to check on the client's connection using the keepalive feature.

FORTE_KEEP_INTERVAL

Specifies the interval, in seconds, after a ping message is sent and during which a reply is expected. The default value is 10 seconds.

FORTE_KEEP_INTERVAL *integer*

FORTE_LAUNCHER_CLOSE

Specifies whether or not the Launcher should shutdown after the user double-clicks an application to launch it. The default is that it will not (FALSE).

FORTE_LAUNCHER_CLOSE {**TRUE**|**FALSE**}

This variable stores one of the Launcher preferences set by a user in the Launcher Options tab page. The Server Launcher automatically updates this variable; so although a user can set it, the Launcher overwrites the current setting whenever "Save Options" is entered.

FORTE_LAUNCHER_LISTTYPE

Determines the default display format for the iPlanet UDS and user applications shown in the launcher. ListType values are the same as the ListStyle integers used for the ListView widget. The default list type is 0.

FORTE_LAUNCHER_LISTTYPE *list_type_integer*

Valid list types are shown in the following table:

Value	Option
0	LT_DEFAULT
1	LT_LIST
2	LT_IMAGE
3	LT_SMALLICON
4	LT_DETAIL

This variable stores one of the Launcher preferences set by a user in the Launcher Options tab page. The Server Launcher automatically updates this variable; so although a user can set it, the Launcher overwrites the current setting whenever "Save Options" is entered.

FORTE_LAUNCHER_REFRESH

Specifies the time in minutes between checks by the launcher to see if new applications have been installed. The default is `0`.

FORTE_LAUNCHER_REFRESH *refresh_interval*

To have the Launcher perform such checks, set FORTE_LAUNCHER_REFRESH to any decimal value greater than zero.

This variable stores one of the Launcher preferences set by a user in the Launcher Options tab page. The Server Launcher automatically updates this variable; so although a user can set it, the Launcher overwrites the current setting whenever "Save Options" is entered.

FORTE_LAUNCHER_SHOWALL

Specifies whether the Launcher should display all available releases of an application in the User Page. The default is that it will not (FALSE).

FORTE_LAUNCHER_SHOWALL {**TRUE**|**FALSE**}

This variable stores one of the Launcher preferences set by a user in the Launcher Options tab page. The Server Launcher automatically updates this variable; so although a user can set it, the Launcher overwrites the current setting whenever "Save Options" is entered.

FORTE_LAUNCHER_UPDATEREL

Specifies whether the Launcher should automatically install the most current release of an application. The default is that it will (TRUE).

FORTE_LAUNCHER_UPDATEREL {**TRUE**|**FALSE**}

This variable stores one of the Launcher preferences set by a user in the Launcher Options tab page. The Server Launcher automatically updates this variable; so although a user can set it, the Launcher overwrites the current setting whenever "Save Options" is entered.

FORTE_LC_COLLATE

Sets the locale used for collating sequence information.

FORTE_LC_COLLATE *language_territory.codeset*[*@collate_sequence*]

See FORTE_LOCALE for information on the arguments. See the *iPlanet UDS Programming Guide* for information about using this environment variable.

FORTE_LC_CTYPE

Sets the locale used for character classifications.

FORTE_LC_CTYPE *language_territory.codeset*[*@collate_sequence*]

See FORTE_LOCALE for information on the arguments. See *iPlanet UDS Programming Guide* for information about using this environment variable.

FORTE_LC_MONETARY

Sets the locale used for currency formatting.

FORTE_LC_MONETARY*language_territory.codeset*[*@collate_sequence*]

See FORTE_LOCALE for information on the arguments. See *iPlanet UDS Programming Guide* for information about using this environment variable.

FORTE_LC_NUMERIC

Sets the locale used for numeric formatting.

FORTE_LC_NUMERIC *language_territory.codeset*[*@collate_sequence*]

See FORTE_LOCALE for information on the arguments. See *iPlanet UDS Programming Guide* for information about using this environment variable.

FORTE_LC_TIME

Sets the locale used for date and time formatting.

FORTE_LC_TIME *language_territory.codeset*[*@collate_sequence*]

See FORTE_LOCALE for information on the arguments. See *iPlanet UDS Programming Guide* for information about using this environment variable.

FORTE_LOCALE

Specifies the default locale.

```
FORTE_LOCALE language_territory.codeset[@collate_sequence]
```

Argument	Description
language_territory	Specifies the language and the territory conventions to use for locale information.
codeset	Specifies the codeset to use when running any partition.
collate_sequence	Specifies the collating sequence to use within the locale definition file.

If you set this environment variable, iPlanet UDS consults the specified locale file in the iPlanet UDS locale directory, and uses the settings in that file that pertain to all categories that are not already specified by one of the FORTE_LC_* environment variables (such as FORTE_LC_COLLATE).

See the *iPlanet UDS Programming Guide* for more information about using this environment variable. This manual also lists the set of iPlanet UDS locales, which perform the same across platforms.

FORTE_LOCATIONS

Allows a manual selection of the network address (for example, TCP host and port) to be used to communicate with objects created in a partition on this machine.

```
FORTE_LOCATIONS address [::protocol_name][;address
[::protocol_name]...]
```

The syntax of *address* is protocol-dependent, as shown in the following table:

Syntax for Address	Syntax for Protocol_Name
{hostname | IP_address}:*port_number*	TCP/IP
nodename:*object_name*	DECnet
path_name	UnixDomain (*note*: no space)

TCP is the default protocol for all platforms. If FORTE_LOCATIONS is not specified, a TCP port number is automatically assigned by the operating system/communications software. You can specify a different protocol by specifying the optional *protocol_name*.

For example, setting FORTE_LOCATIONS is useful if your site has a firewall that allows access only on a specific port.

Be sure that the address you specify is not already used by any other software, including iPlanet UDS.

FORTE_LOGGER_SETUP

Specifies the default logger settings for the iPlanet UDS session.

```
FORTE_LOGGER_SETUP
file_name(file_filter)[file_name(file_filter)...]
```

You can override the settings for FORTE_LOGGER_SETUP by using the `-fl` flag on any of the iPlanet UDS command lines. In fact, we recommend that you use the `-fl` flag rather than resetting FORTE_LOGGER_SETUP when you want to change message filtering.

File name The `file_name` argument is any valid file name where you want to log certain messages. The special file names "%stdout" and "%stderr" log the messages to standard output or standard error, respectively.

On Windows only, you can use the name "%stdwin" to create a simple, scrollable output window for textual output. "%stdwin" is particularly useful when using the FORTE_LOGGER_SETUP variable to specify an alternative file for the output from `Fscript` or the development environment.

File filters Each file name is associated with a *file_filter*. The `file_filter` argument may include one or more filter options separated by a space. The syntax for the `file_filter` parameter is the following:

message_type[:*service_type*[:*group_number*[:*level_number*]]]

For information on the file filters, see the description of the LogMgr class in the Framework Library online Help. The LogMgr object examines the FORTE_LOGGER_SETUP environment variable and opens one or more text files or display windows based on the settings it finds there. If FORTE_LOGGER_SETUP is not set (and you have not specified in the `-fl` flag in your command line), no specially filtered messages are logged.

Modifying logging during runtime You can also modify the types and detail level of messages being logged while a program is running. The ModifyFlags methods defined on LogMgr provides a runtime interface to the same settings that FORTE_LOGGER_SETUP (and the `-fl` flag on iPlanet UDS command lines) provides at application start-up. The Environment Console provides an interface to a running server's LogMgr object and can be used to modify the LogMgr settings of the server being monitored. Finally, the Repository Workshop allows you to change LogMgr setting while developing iPlanet UDS applications.

FORTE_LOS_EXPTIME

Specifies the desired number of objects in the repository cache. The default is `2500`.

FORTE_LOS_EXPTIME *integer*

You may want to increase this setting, especially during development, to help repository performance at the cost of using more iPlanet UDS memory.

FORTE_MODELNODE

Specifies the name of a model node in the active environment, which the current node uses for definition.

FORTE_MODELNODE *model_node_name*

This variable is normally set during the installation process. If the environment variable is not set, the node is not treated as a model node. See also FORTE_NODENAME.

FORTE_NEXT_AFFINITY

(For Windows NT only) Specifies the processor number which is to be used next.

FORTE_NEXT_AFFINITY *integer*

This variable is provided only for R2 backward compatibility on Windows NT. Under normal circumstances, you should not set this environment variable.

FORTE_MOTIF_CLIPBOARD

(For Motif only) Specifies that iPlanet UDS will use the Motif clipboard rather than a local iPlanet UDS clipboard.

FORTE_MOTIF_CLIPBOARD {**TRUE**|**FALSE**}

If you set this variable to `TRUE`, all Cut, Copy, and Paste operations that use a clipboard will use the window system clipboard rather than a local iPlanet UDS clipboard.

FORTE_NODENAME

(For PCs only) Specifies the name of the current node. (On UNIX and OpenVMS, the host is taken from the hosts file.)

```
FORTE_NODENAME node_name
```

This variable is normally set during the installation process.

FORTE_NS_ADDRESS

The FORTE_NS_ADDRESS environment variable is used one way for the Environment Manager process and another way for all other processes.

- For the Environment Manager, FORTE_NS_ADDRESS specifies one or more network addresses at which the Environment Manager listens for incoming requests.
- For *all other* processes, FORTE_NS_ADDRESS specifies one or more addresses to use when contacting the Environment Manager. Addresses are tried in the order they are specified. If a connection fails while in use, the next address is automatically tried.

In both cases, however, the syntax is the same:

```
FORTE_NS_ADDRESS address [::protocol_name][;address
[::protocol_name]...]
```

See "FORTE_LOCATIONS" on page 334 for a description of *address* and *protocol_name*. For further information about FORTE_NS_ADDRESS, see "Environment Manager Failover for Partitions" on page 122.

FORTE_OBBSHR

Specifies the location of the ObjectBroker runtime libraries.

```
FORTE_OBBSHR file_specification
```

Also see "FORTE_COSSHR" on page 329.

FORTE_OLDEST_19YY

Used to anticipate dates in year 2000 and beyond, this variable specifies the oldest year that iPlanet UDS will set to a 19xx date.

```
FORTE_OLDEST_19YY yy
```

In Release 3 the default for this variable is `30`. This setting indicates that any "yy" format year data that is equal to or greater than 30 (that is, 30 to 99) should be prefixed with 19, and any "yy" format data less than 30 (that is, 00 to 29) should be prefixed with 20.

Set this variable to `00` to have date data treated as it is in iPlanet UDS Release 2. See the Framework Library online Help for more information on this variable.

FORTE_OLESHR

Specifies the location of the OLE libraries.

FORTE_OLESHR *file_specification*

FORTE_PROCESSORS

(For Windows NT only) Contains the number of processors that exist on the system. On a uniprocessor machine the value is 1, while for a 6-way Sequent the value is `6`. iPlanet UDS automatically detects this value and sets this variable automatically.

FORTE_PROCESSORS *integer*

This variable is provided only for R2 backward compatibility on Windows NT. Under normal circumstances, you should not set this environment variable.

FORTE_PROVIDERS

Specifies the transport provider for a given platform. Normally you need not set this variable as the installation does so for you.

FORTE_PROVIDERS *protocol_name*

However, you can use this variable to specify a different transport provider for your platform. For example, for Windows the default communication protocol is Winsock, but you can specify the use of the native interface to the PC-NFS product. For Windows the valid values for this environment variable are:

Communication Provider	Communication Protocol
W3TPSUN	PC-NFS
W3TPWSS	Winsock

FORTE_REPOSNAME

Specifies the name of the repository that will be used as the default repository for all iPlanet UDS commands that use a repository.

FORTE_REPOSNAME *repository_name*

To specify the repository name, enter:

- "Central Repository" for the default central repository
- A repository service name.
- The name of a private C-tree or shadow repository using the following format: **ct:** *private_repository_name.*

  Specify the private repository name as the root file name of the C-tree repository (that is, the full path name of the .dat or .idx file without the trailer).

By default, this has the value "CentralRepository" which represents the default central repository in the distributed environment.

On Windows, if you give a New Shadow command in the Repository Workshop, this automatically resets the value of FORTE_REPOSNAME to the shadow name. This way, if you leave and then reenter iPlanet UDS, the same shadow repository will automatically be open.

Note that the `-fr` flag on several iPlanet UDS commands, such as `forte` and `fscript`, overrides the value of FORTE_REPOSNAME.

FORTE_ROOT

Specifies the root directory path for iPlanet UDS.

`FORTE_ROOT` *`file_specification`*

All the files that make up the iPlanet UDS system as well as the default locations for files that iPlanet UDS writes and reads are stored in FORTE_ROOT. The value of this environment variable is set during the installation process; you should not reset it.

FORTE_RPSTART_WAIT

Specifies the amount of time in seconds that rpstart will wait for the repository server to start before issuing an error message. The default value is `60 seconds`.

`FORTE_RPSTART_WAIT` *`integer`*

FORTE_SCREEN_HEIGHT_MILS

Sets the actual height of the screen, in mils.

`FORTE_SCREEN_HEIGHT_MILS` *`integer`*

This is intended primarily for use on X servers that are unaware of or incorrectly report the screen height. You can use this environment variable to correct for distortion. Measure your screen size, and set the actual height as an integer value in mils (thousandth's of an inch).

FORTE_SCREEN_WIDTH_MILS

Sets the actual width of the screen, in mils.

FORTE_SCREEN_WIDTH_MILS *integer*

This is intended primarily for use on X servers that are unaware of or incorrectly report the screen width.You can use this environment variable to correct for distortion. Measure your screen size, and set the actual width as an integer value in mils (thousandth's of an inch).

FORTE_STACK_SIZE

Sets the thread stack size in bytes for iPlanet UDS and POSIX threads. The syntax for this variable is identical to the syntax for the `-fst` startup flag, which can also be used to set the thread stack size.

FORTE_STACK_SIZE *integer*

If both the `-fst` flag and the FORTE_STACK_SIZE variable have been specified, then the larger of the two values is used.

The default value for the thread stack size ranges from 28K to 48K, depending on the platform and release of iPlanet UDS. Motif clients have a minimum size of 100K. On NT, the system default is 1MB and is not adjustable. A thread stack size setting below the default value for that system is ignored.

You can increase the stack size if necessary. Because the new stack size you specify is used for every thread, it will increase memory usage by the stack size times the number of concurrent active threads. iPlanet UDS rounds the value up to the nearest system MMU (Memory Management Unit) pagesize (8K on many machines, but sometimes more or less). iPlanet UDS adds an MMU pagesize guardword to the size; this page is memory protected to try and catch stack overflow cases.

FORTE_THREAD_AFFINITY

Under normal circumstances, you should not set this environment variable.

FORTE_TIMEZONE

Specifies the time zone for the machine as the number of hours east or west of GMT.

FORTE_TIMEZONE *integer*

This variable is only needed on OpenVMS and in the southern hemisphere on Win3.1. On UNIX systems and PCs running PC/NFS), the system provides the time zone setting. However, if you set this environment variable, its value overrides the operating system timezone.

Acceptable values are integers between -23 and 23. For example, use `8` for Pacific time and `5` for Eastern time. For time zones east of GMT, set this to a negative number. If daylight savings time is in effect, use the FORTE_TIMEZONEDST environment variable in addition.

FORTE_TIMEZONEDST

Specifies whether or not daylight savings is in effect. If it is set to `TRUE`, one hour is added to the current time setting.

FORTE_TIMEZONESDT {**TRUE** | **FALSE**}

This variable is only needed on OpenVMS and in the southern hemisphere on Win3.1.

FORTE_TIMEZONEMIN

Specifies the number in minutes west or east of GMT. Use negative numbers for east of GMT.

FORTE_TIMEZONEMIN *integer*

This variable is provided for regions that use half-hour timezones. Like the variable FORTE_TIMEZONE, this variable is only needed on OpenVMS and Win3.1 (southern hemisphere).

FORTE_VISUAL_STYLE

For Windows and Motif only—Specifies whether iPlanet UDS widgets should use two- or three-dimensional style. The default is `three-dimensional` (3D).

FORTE_VISUAL_STYLE {**3D** | **2D** | **2DONLY**}

To turn off the three-dimensional style completely, use the 2DONLY setting. To specify that the default visual style is two-dimensional, use the 2D setting. When the default visual style is two-dimensional, you can still make an individual window three-dimensional by setting the window's DefaultVisualStyle attribute to VS_3D.

FORTE_WORKMSG

Specifies the directory where application catalogs are stored during the development cycle. The default is "`FORTE_ROOT/workmsg`."

FORTE_WORKMSG *directory_specification*

The use of this variable allows multiple developers to use different message catalogs during the development process. See *iPlanet UDS Programming Guide* for more information on using this environment variable.

FORTE_WORKSPACE

Specifies the name of the iPlanet UDS workspace to use for the default workspace for the `forte` command or when the user gives an `Open` command in Fscript.

`FORTE_WORKSPACE` *`workspace_name`*

If you do not set this environment variable and do not specify a value with the `-fw` flag on the `forte` or fscript command, iPlanet UDS opens the Repository Workshop without a workspace and starts `Fscript` with the prefabricated workspace called "FirstWorkspace."

On Windows, the value of this environment variable is automatically set by the iPlanet UDS workshops to the last workspace that was opened. This way, if you leave and then reenter iPlanet UDS, the same workspace you left will automatically be open.

You can override the setting of FORTE_WORKSPACE by using the `-fw` flag on the `forte` and `fscript` commands. In the Repository Workshop, you can open any workspace in the repository with the Open Workspace command on the File menu.

FORTE_X_HEADERDIRS

Specifies the directory in which the header files for X Windows and Motif are stored. This is for use when you are making the distribution for a compiled partition.

`FORTE_X_HEADERDIRS -I`*`/dir_name/`***`include`**

You need to set this variable if the Display Project is contained in the partition. To find out if the Display Project is in the server partition, run Fscript, and use the `FindPlan`, `FindApp`, and `ShowApp` commands (see *Fscript Reference Guide* for more information about `Fscript`).

FORTE_X_LIBDIRS

Specifies the directory for the X header files. This is for use when you are making the distribution for a compiled partition (see *A Guide to the iPlanet UDS Workshops* for more information).

`FORTE_X_LIBDIRS -L`*`/dir_name/`***`lib`**

You should set this variable if you are unsure whether your X Window header files reside in a standard location. Generally, you should set this variable if you are running on a Sparc. If your hardware vendor also supplied your windowing software, then you probably do not need to set the variable.

FORTE_X_SOLID_GHOST

(For Motif only) Specifies that ghost lines used by the iPlanet UDS display system will be solid rather than dashed. (`Dashed ghost lines` are the default for Motif).

```
FORTE_X_SOLID_GHOST { TRUE | FALSE }
```

Logical Names for OpenVMS

The following logical names allow you to set system parameters to apply to all servers that are automatically started by iPlanet UDS. See "Starting iPlanet UDS Applications" on page 185 for information about starting servers.

FORTE_DETACHED_ASTLM

Sets the limit for the AST queue. This is the total number of asynchronous system trap (AST) operations and scheduled wake-up requests that the user can have queued at one time. The default value is `100`.

FORTE_DETACHED_BIOLM

Sets the buffered I/O count limit for the BIOLM field of the UAF record. The buffered I/O count limit is the maximum number of buffered I/O operations, such as terminal I/O, that can be outstanding at a given time. The default value is `64`.

FORTE_DETACHED_BYTLM

Sets the buffered I/O byte limit for the BYTLM field of the UAF record. The buffered I/O byte limit is the maximum number of bytes of non-paged, system-dynamic memory that a user's job can consume at one time. The system uses non-paged, dynamic memory for operations such as I/O buffering and file access windows. The default value is `100000`.

FORTE_DETACHED_DIOLM

Sets the direct I/O count limit for the DIOLM field of the UAF record. The direct I/O count limit is the maximum number of direct I/O operations (usually disk) that can be outstanding at a given time. The default value is `64`.

FORTE_DETACHED_ENQLM

Sets the lock queue limit for the ENQLM field of the UAF record. The lock queue limit is the maximum number of locks that can be queued by the user at one time. The default is `2000`.

FORTE_DETACHED_FILLM

Sets the open file limit for the FILLM field of the UAF record. The open file limit is the maximum number of files that can open at a given time, including active network logical links. The default value is `10`.

FORTE_DETACHED_JTQUOTA

Sets the initial byte quota with which the job-wide logical name table is to be created. The default value is `0`.

FORTE_DETACHED_PGFLQUOTA

Sets the paging file limit. The paging file limit is the maximum number of pages that the user's process can use in the system paging file. The default value is `128000`.

FORTE_DETACHED_TQE

Sets the total number of entries in the timer queue plus the number of temporary common event flag clusters that the user can have at the same time. The default value is `200`.

FORTE_DETACHED_WSDEFAULT

Sets the default working set size. This is the initial limit for the number of physical pages that the process can use. The default value is `1024`.

FORTE_DETACHED_WSEXTENT

Sets the working set maximum. This represents the maximum amount of physical memory allowed to the process. The system provides memory to a process beyond its working set quota only when it has excess free pages. The additional memory is recalled by the system if needed. The value you specify must be an integer equal to or greater than FORTE_DETACHED_WSQUOTA. The default value is `32000`.

FORTE_DETACHED_WSQUOTA

Sets the working set quota. The working set quota is the maximum amount of physical memory a user process can lock into its working set. It also controls the maximum amount of swap space that the system reserves for this process, and the maximum amount of physical memory that the system allows the process to consume when the system-wide memory demand is significant. The default value is `800`.

Using the iPlanet UDS Control Panel

The iPlanet UDS Control Panel provides a simple user interface that allows you to view and/or set the most commonly used iPlanet UDS environment variables. For Windows, the Control Panel allows you to view and change the current settings. For other platforms, the Control Panel only provides read access to the settings. For these platforms, you must set the environment variables as described under "Setting Environment Variables Without the iPlanet UDS Control Panel" on page 353.

Opening the Control Panel

To start the Control Panel from Windows, double-click the iPlanet UDS Control Panel icon, or choose the Start > Programs > iPlanet UDS > iPlanet UDS Control Panel command.

To start the Control Panel from UNIX or VMS, use the `fcontrol` command. The portable syntax is:

fcontrol

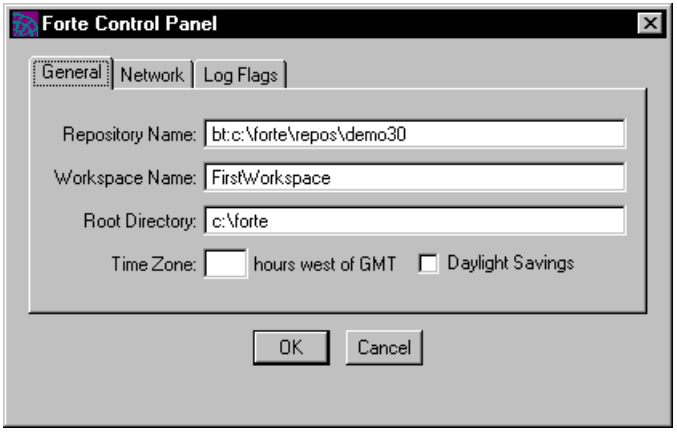
The OpenVMS syntax is:

VFORTE FCONTROL

The Control Panel Window

The Control Panel window is a dialog box, shown below, where you can view or change the iPlanet UDS environment variable settings.

Figure B-1 iPlanet UDS Control Panel



If you use the utility on Windows, the window displays OK and Cancel buttons. Any changes you make to settings take effect when you click the OK button. Clicking the OK button also closes the dialog.

If you use the utility on any platform other than the Windows, you cannot change any of the settings, so the Control Panel simply displays a Close button. Click the Close button to close the dialog when you have finished viewing the settings.

When you open the Control Panel, the Control Panel window displays a tab folder with the following tab pages:

Tab Page	Available Settings
General	Repository name, workspace name, root directory, and time zone.
Network	Model node, node name, name server address, and communication provider.
Log flags	Default logger settings for the iPlanet UDS session.

The General tab page is displayed first. To display or set the network settings, click the Network tab. To display or set the log flags setting, click the Log Flags tab. Do not click the OK on the Control Panel until you are ready to close the Control Panel.

The following sections provide detailed information about the settings you can view or change on each of the tab pages.

Closing the Control Panel

On Windows, you can use either the OK or Cancel button to close the Control Panel. The OK button changes the settings as specified and then closes the Control Panel. The Cancel button discards the new settings and then closes the Control Panel.

On all other platforms, simply click the Close button to close the Control Panel.

General Tab Page

The following table lists general settings you can control with the Control Panel and shows the environment variable that corresponds to each setting:

General Setting	Environment Variable
Repository Name	FORTE_REPOSNAME
Workspace Name	FORTE_WORKSPACE
Root Directory	FORTE_ROOT
Time Zone	FORTE_TIMEZONE
Daylight Savings	FORTE_TIMEZONEDST

The following sections provide information on the individual settings.

Repository Name

The repository name setting specifies the name of the repository to be used as the default repository for all iPlanet UDS commands that use a repository.

To specify the repository name, enter one of the following options:

- "Central Repository" for the default central repository.
- A repository service name.
- The name of a private B-tree or shadow repository using the following format: **bt:** *private_repository_name.*

  Specify the private repository name as the root file name of the B-tree repository (that is, the full path name of the file without the trailer).

By default, the repository name has the value "CentralRepository," which represents the default central repository in the distributed environment.

B-tree specifications The following examples illustrate how to specify a B-tree repository name:

Platform	Command Syntax
UNIX	bt:$FORTE_ROOT/repos/myShadow
OpenVMS	BT:FORTE_ROOT:[REPOS]MYSHADOW
Windows	bt:\forte\repos\myshadow

New Shadow command On Windows platforms, if you choose the New Shadow command in the Repository Workshop, iPlanet UDS automatically resets the value of the Repository Name setting (and the FORTE_REPOSNAME environment variable) to the shadow name. As a result, if you leave and then reenter iPlanet UDS, the same shadow repository will automatically be open.

The `-fr` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Repository Name setting. For information about the `-fr` flag, see *A Guide to the iPlanet UDS Workshops*.

Workspace Name

The Workspace Name setting specifies the name of the workspace to use for the default workspace when you start the iPlanet UDS Workshops or when the you give an Open command in `Fscript`.

The `-fw` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Workspace Name setting. For information about the `-fw` flag, see *A Guide to the iPlanet UDS Workshops*.

If you do not specify a value for Workspace Name and do not specify a value with the `-fw` flag on the command line, iPlanet UDS opens the Repository Workshop without a workspace and starts `Fscript` with the prefabricated workspace called "FirstWorkspace."

On Windows, the value of the Workspace Name setting (and the corresponding FORTE_WORKSPACE environment variable) is automatically set by the iPlanet UDS workshops to the last workspace that was opened. As a result, if you leave and then reenter iPlanet UDS, the same workspace you left will automatically be open.

You can override the Workspace Name setting by using the `-fw` flag on the commands that start the iPlanet UDS Workshops. In the Repository Workshop, you can open any workspace in the repository by choosing the File > Open Workspace command (see *A Guide to the iPlanet UDS Workshops*).

Root Directory

The Root Directory setting specifies the root directory path for iPlanet UDS. All the files that make up the iPlanet UDS system as well as the default locations for files that iPlanet UDS writes to and reads are stored in the root directory. The value of this setting is usually specified during the installation process using the FORTE_ROOT environment variable. The Control Panel displays the current setting mainly for your information only. You should not specify a new root directory unless you are moving your entire iPlanet UDS installation.

Time Zone and Daylight Savings

The Time Zone setting specifies the time zone for the machine as the number of hours west of GMT. On some systems (on UNIX and on PCs running PC/NFS), this is not needed, as the system provides an accurate value for the time zone setting. However, if you do specify a value for this setting, note that the setting will override any other value it gets from the operating system.

You must set the Time Zone setting to an integer value between +11 and -13. For Pacific time, set the value to `8`, and for Eastern time, set the value to `5`. For time zones east of GMT, set the Time Zone setting to a negative number. If daylight savings time is in effect, set the Daylight Savings toggle to on.

The Daylight Savings toggle specifies whether or not daylight savings is in effect. If the toggle is set to on, one hour is added to the current time setting.

Network Tab Page

The following table lists the network settings you can control with the Control Panel and shows the environment variable that corresponds to each setting:

Network Setting	Environment Variable
Model Node	FORTE_MODELNODE
Node Name	FORTE_NODENAME
Name Server Address	FORTE_NS_ADDRESS
Communication Provider	FORTE_PROVIDERS

The following sections provide information on the individual settings.

Model Node

The Model Node setting specifies the name of the model node, in the active environment, which this node uses for its definition. The Model Node is usually set during installation. If you do not specify a model node name, the node is not treated as a model node.

The `-fmn` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Model Node setting. For information about the `-fmn` flag, see *A Guide to the iPlanet UDS Workshops*.

Node Name

The Node Name setting specifies the name of the node on which you are running. The Node Name setting in the Control Panel is used only on client machines; on UNIX, OpenVMS, and NT servers, the host name is taken from the hosts file. The Node Name is usually set during installation, but you can change the value from the Control Panel if necessary.

The `-fn` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Node Name setting. For information about the `-fnd` flag, see "Startup Flags" on page 192.

Name Server Address

The Name Server Address setting specifies the address of the iPlanet UDS name server process. The Name Server Address is usually set during installation, but you can change the value from the Control Panel if necessary.

The syntax for the Name Server Address setting is:

address [::*protocol_name*]

The syntax of *address* is protocol dependent, as shown in the following table.

Protocol	Address Syntax
TCP/IP	*machine_name*:*port_number*
DECnet	*machine_name*:*object_name*
UNIX Domain	*path_name*

The optional *protocol_name* allows you to use a different protocol than the default for the platform. The default protocol for OpenVMS is DECnet and for all other platforms TCP/IP.

The *protocol_name* is one of the following values:

TCP/IP
DECnet
UNIX Domain

CAUTION If you change your name server address, the next iPlanet UDS application that you start, including the iPlanet UDS Workshops, will use the new value. This effectively changes you to a different iPlanet UDS environment. You should discuss this with your system manager.

For further information about the name server, see the "Environment Manager Failover for Partitions" on page 122.

Communication Provider

The Communication Provider setting specifies the communication transport providers to use for the node. A transport provider is a program that enables iPlanet UDS to access a particular communication package.

Choosing a provider for PC Windows In the iPlanet UDS Control Panel, the only platform for which you can change the communication provider is PC Windows. The default communication provider for PC Windows is Winsock. You can choose PC-NFS if appropriate.

Log Flags Tab Page

As you develop and test applications in the iPlanet UDS Workshops, iPlanet UDS logs messages in the trace window or log file as specified by FORTE_LOGGER_SETUP environment variable (see "FORTE_LOGGER_SETUP" on page 335). If you did not specify a log file name with FORTE_LOGGER_SETUP, iPlanet UDS logs the messages in the trace window.

The Log Flags setting in the Control Panel allows you to specify the filter settings used for logging the messages. This is equivalent to the log settings you can specify the FORTE_LOGGER_SETUP environment variable. However, the Log Files setting does not allow you to specify the log file names. Instead, all messages are logged in the default log file, "stdout."

CAUTION If you change any of the filter settings using the Control Panel, the new settings override any file specifications set by the FORTE_LOGGER_SETUP environment variable. All messages will be logged in the default log file, "stdout."

To change the filter settings for an individual message, edit the fields in the array row as follows:

Field	How to Fill It in
Message	Choose the message type from the drop list.
Service	Choose the service type from the drop list.
Group	Enter integers in the two data fields to specify a range. The integers can be from 1 to 63.
Level	Enter an enter from 1 to 255.

See "-fl Flag (Log Manager)" on page 371 for specific information about each of these settings.

Inserting and deleting log settings You can request an additional log setting by inserting a new row in the array field or eliminate a log setting by deleting the row from the array field. The Insert button adds a new row above the selected row, using default values for each of the fields. The Delete button removes the currently selected row.

The LogMgr object (described in Framework Library online Help) examines the Log Flags setting (or the FORTE_LOGGER_SETUP environment variable) and based on the information it finds, opens one or more text files or display windows. If you do not specify the log flags with the LogFlags setting or FORTE_LOGGER_SETUP (and you have not specified in the `-fl` flag in your command line), no specially filtered messages are logged.

The `-fl` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Log Flags setting. For information about the `-fl` flag, see "-fl Flag (Log Manager)" on page 371.

Modify Log Flags command If you wish to change the default filter settings at any point during your development session, you can use the Modify Log Flags command in the Repository Workshop (see *A Guide to the iPlanet UDS Workshops*). The Modify Log Flags command opens a window, where you view and/or change the filter settings in an array field.

Runtime modification of log specifications You can also modify the types and detail level of messages being logged while the program is running. The ModifyFlags method defined on the LogMgr class provides a runtime interface to the same settings the Log Flags setting (and the `-fl` flag on iPlanet UDS command lines) provides at application start-up. The Environment Console provides an interface to a running server's LogMgr object and can be used to modify the LogMgr settings of the server being monitored.

Setting Environment Variables Without the iPlanet UDS Control Panel

You may find you need to set environment variables for Windows that are not included in the iPlanet UDS Control Panel. To do so, you must use the operating system's standard format. For platforms other than Windows, you must set all your environment variables using the operating system's standard format.

The following sections provide information about setting environment variables in the operating systems supported by iPlanet UDS.

Setting Environment Variables on NT

On Windows and Alpha NT, the default settings for the iPlanet UDS environment variables are set in the Registry. NT allows you to set environment variables in several different places, all of which are described below. We recommend that you set your environment variables using the iPlanet UDS Control Panel or the Registry Editor (regedit). In general, you should avoid using the NT Control Panel.

The order of precedence is as follows:

1. individual process window (set at the DOS command prompt)
2. NT Control Panel's User variables

3. NT Control Panel's System variables
4. Registry's Current User Tree (HKEY_CURRENT_USER)
5. Registry's Global Tree (HKEY_LOCAL_MACHINE)

Using the Registry

The environment variables in the Registry are set in two different directories.

Registry's Global Tree iPlanet UDS machine-wide information and default iPlanet UDS environment variables are kept in the iPlanet UDS Global Tree branch:

HKEY_LOCAL_MACHINE\Software\ForteSoftwareInc\Forte*version number*

Registry's User Tree All user-specific iPlanet UDS environment variables are kept in the iPlanet UDS User Tree branch:

HKEY_CURRENT_USER\Software\ForteSoftwareInc\Forte

Environment variable settings in the iPlanet UDS User Tree override those in the iPlanet UDS Global Tree. Unless you are a system administrator who is setting up a machine for use by several users, we recommend that you set your environment variables in the *User Tree*.

To change the value of an environment variable or add a new one, use regedit.exe to open the Registry Editor and modify the iPlanet UDS User Tree.

Using the NT Control Panel The NT Control Panel allows you to set any of the iPlanet UDS environment variables. Like the Registry, the NT Control Panel contains System and User sections. The settings in the User section override the settings in the System variables.

Settings in the NT Control Panel override the settings in the Registry (and therefore settings in the iPlanet UDS Control Panel). Therefore, we recommend that you do *not* use the NT Control Panel to set your environment variables. Setting environment variables in too many places can make your setup confusing and inconsistent.

Using the DOS command line For an individual process window, you can set environment variables using the `set` command in DOS. Using the `set` command for an individual window overrides the environment variable settings for that individual process.

Using the iPlanet UDS Control Panel You can set the most commonly used environment variables using the iPlanet UDS Control Panel as described under "Using the iPlanet UDS Control Panel" on page 345. Setting environment variables in the iPlanet UDS Control Panel changes the value of the equivalent environment variable in the User Tree of the Registry (*not* in the NT Control Panel).

iPlanet UDS command-line flags Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Setting Environment Variables on Windows 95

On Windows 95, the default settings for the iPlanet UDS environment variables are stored in the User Registry.

Using the Registry The following file in the Registry contains the iPlanet UDS environment variables:

HKEY_CURRENT_USER/Software/ForteSoftwareInc/Forte

To change the value of an environment variable or add a new one, use regedit.exe to open the Registry Editor and modify this file.

autoexec.bat file You can also set iPlanet UDS environment variables in the autoexec.bat file. Setting environment variables in the autoexec.bat file overrides the settings in the User Registry. For consistency, we recommend that you set your environment variables in the Registry as described above, and *not* in the autoexec.bat file.

iPlanet UDS Control Panel You can set the most commonly used environment variables using the iPlanet UDS Control Panel as described under "Using the iPlanet UDS Control Panel" on page 345. Setting an environment variable in the iPlanet UDS Control Panel changes the value of the corresponding environment variable in the Registry. Therefore, environment variables set in autoexec.bat file will override those set by the iPlanet UDS Control Panel.

In summary, the following environment variable settings take precedence:

1. autoexec.bat file
2. User Registry

iPlanet UDS command-line flags Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Setting Environment Variables on UNIX

Using the fortedef file On UNIX, the default settings for the iPlanet UDS environment variables are stored in the following files:

C-Shell

```
FORTE_ROOT/fortedef.csh
```

Bourne-Shell

```
FORTE_ROOT/fortedef.sh
```

These two files are created by the iPlanet UDS Installer, and save your input for the iPlanet UDS system information.

Using the fortedef file To change the default settings or add new environment variable settings, simply edit the appropriate fortedef file.

Using the command line You can also enter settings for iPlanet UDS environment variables on the command line. Following the UNIX standard, environment variable settings specified at the command line override those in the fortedef file.

iPlanet UDS command-line flags Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Setting Logical Names on OpenVMS

On OpenVMS, the default settings for the iPlanet UDS logical names and DCL foreign symbols are stored in the following file:

```
FORTE_ROOT:[INSTALL.SCRIPTS]FORTE_LOGIN.COM
```

Using the FORTE_LOGIN.COM file To change the default settings or add new logical names or symbols, simply edit the `FORTE_LOGIN.COM` file or create a new file:

```
FORTE_ROOT:[INSTALL.SCRIPTS]SITE_LOGIN.COM
```

In order to define the FORTE_ROOT logical name you can simply execute:

```
@SYS$LIBRARY:FORTE_LOGIN_Vversion_number
```

Using your personal login.com file If you use iPlanet UDS frequently, you may wish to include the following command within your personal `LOGIN.COM` file:

```
$ @SYS$LIBRARY:FORTE_LOGIN_Vversion_number
```

Using the command line You can also define settings for iPlanet UDS logical names on the command line. Logical name settings specified at the command line override those in the `FORTE_LOGIN.COM` file. The logicals defined in `FORTE_LOGIN.COM` are stored in two logical name tables:

```
FORTE_PRCTABLE_Vversion_number
```

```
FORTE_GBLTABLE_Vversion_number
```

Therefore, if you use the DEFINE command to set your own logicals in either the LNM$PROCESS table (with the default /PROCESS qualifier to DEFINE) or the LNM$JOB table (with the /JOB qualifier to DEFINE) you can leave the FORTE_PRCTABLE_V*version_number* and FORTE_GBLTABLE_V*version_number* tables undisturbed for other users.

iPlanet UDS command-line flags Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Appendix C

iPlanet UDS Command Summary

This appendix provides the syntax for iPlanet UDS commands in alphabetical order. An asterisk (*) next to the command name indicates that the command is available for development only.

The appendix also provides information about the `-fl` and `-fm` flags, which appear on most of the iPlanet UDS commands.

iPlanet UDS Commands

Compmsg Command

The `compmsg` command starts the `Compmsg` utility, which takes a file containing source message text and numbers, and compiles it into a portable binary format for use at runtime by the iPlanet UDS runtime system.

Portable

compmsg -m [**@**]*input_msg_file* **-c** *output_msg_catalog* [**-o**] [**-d** *msg_catalog*]
[**-e** *log_file*] [**-flc** *locale*]

OpenVMS

VFORTE COMPMSG
/**MSG_TEXT=***input_msg_file*
/**CATALOG=***output_msg_catalog*
[/**OVERWRITE=TRUE**]
[/**DISPLAY=***msg_catalog*]
[/**ERROR_LOGFILE=***log_file*]

For more information about using the `compmsg` command, see the *iPlanet UDS Programming Guide*.

Econsole Command

The `econsole` command starts the Environment Console.

Portable

econsole [**-fns** *name_server_address*][**-fl** *logger_flags*][**-fm** *memory_flags*]

OpenVMS

VFORTE ECONSOLE
[/**NAMESERVER=***name_server_address*]
[/**LOGGER=***logger_flags*]
[/**MEMORY=***memory_flags*]

For more information about using the `econsole` command, see "Using the econsole Command" on page 64.

Escript Command

The `escript` command starts the `Escript` utility.

Portable

escript [**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-i** *input_file*] [**-o** *output_file*] [**-fns** *name_server_address*]

OpenVMS

VFORTE ESCRIPT
[/**LOGGER=***logger_flags*]
[/**MEMORY=***memory_flags*]
[/**STACK=***integer*]
[/**INPUT=***input_file*]
[**/OUTPUT=***output_file*]
[/**NAMESERVER=***name_server_address*]

For more information about using the `escript` command, refer to the *Escript and System Agent Reference Guide*.

Extmsg Command

The `extmsg` command starts the `Extmsg` utility, which takes a TOOL project export file as input and replaces any single-quoted strings found in the text within the file with invocations of the `GetString` method on the default message catalog for the application.

Portable

extmsg -i *input_TOOL_file* **-o** *output_TOOL_file*
-m *output_msg_file* [**-s** *set_string*] [**-n** *first_msg_number*]
[**-t** *substitution_text_string*] [**-l**]

OpenVMS

VFORTE EXTMSG
/**INPUT_TOOL=***input_TOOL_file*
/**OUTPUT_TOOL=***output_TOOL_file*
/**MSG_TEXT=***output_msg_file*
[**/SET=***set_string*]
[/**FIRST_MSG=***first_msg_number*]
[/**SUBSTITUTE=***substitution_text_string*]
[/**LEAVE_MSG=TRUE**]
[/**ERROR_LOGFILE=***log_file*]

For more information about using the `extmsg` command, see the*iPlanet UDS Programming Guide*.

*Fcompile Command

The `fcompile` command generates code, compiles, and links a compiled partition, TOOL library, or external library.

Portable

fcompile [**-c** *component_generation_file*] [**-d** *target_directory*] [**-o** *output_file*]
[**-cflags** *compiler_flags*] [**-lflags** *linking_flags*]
[**-iflags** *IDL_compiler_flags*] [**-r** *repository_name*]
[**-fns** *name_server_address*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-fl** *logger_flags*] [**-cleanup**]

OpenVMS

VFORTE FCOMPILE
[/**COMPONENT=***component_generation_file*]
[/**DIRECTORY=***target_directory*]
[/**OUTPUT=***output_file*]
[/**COMPILER=***compiler_flags*]
[/**LINKING=***linking_flags*]
[/**IDL=***IDL_compiler_flags*]
[/**REPOSITORY=***repository_name*]
[/**NAMESERVER=***name_server_address*]
[/**MEMORY=***memory_flags*]
[/**STACK=***integer*]
[/**LOGGER=***logger_flags*]
[/**CLEANUP**]

For more information about using the `fcompile` command, see *A Guide to the iPlanet UDS Workshops* and *Integrating with External Systems*.

Fcontrol Command

The `fcontrol` command opens the iPlanet UDS control panel.

Portable

fcontrol

OpenVMS

VFORTE FCONTROL

For more information on using the control panel, refer to "Using the iPlanet UDS Control Panel" on page 345.

*Forte Command

The `forte` command opens the iPlanet UDS Workshops to start a development session.

Portable

forte [**-fs**] [**-fr** *repository*] [**-fw** *workspace*] [**-fnd** *node_name*]
[**-fmn** *model_node_name*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-fl** *logger_flags*] [**-fcons**]

OpenVMS

VFORTE FORTE
[/STANDALONE]
[/REPOSITORY=*repository_name***]**
[/WORKSPACE=*workspace_name***]**
[/NODE=*node_name***]**
[/MODEL_NODE=*model_node_name***]**
[/MEMORY=*memory_flags***]**
[/**STACK=***integer*]
[/LOGGER=*logger_flags***]**
[/**NAMESERVER=***name_server_address*]

For more information about using the `forte` command, refer to *A Guide to the iPlanet UDS Workshops*.

*Fscript Command

The `fscript` command starts the `Fscript` utility, `which` is a command-line interface to the functions provided by the iPlanet UDS Workshops.

Portable

`fscript` [**-fs**] [**-fr** *repository_name*] [**-fw** *workspace_name*] [**-fcons**]
[**-fns** *name_server_address*] [**-fnd** *node_name*] [**-fmn** *model_node_name*]
[**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]
[**-i** *input_file*] [**-o** *output_file*]

OpenVMS

VFORTE FSCRIPT
[/**STANDALONE**]
[/**REPOSITORY=***repository_name*]
[/**WORKSPACE=***workspace_name*]
[/**NAMESERVER=***name_server_address*]
[/**NODE=***node_name*]
[/**MODEL_NODE=***model_node_name*]
[/**MEMORY=***memory_flags*]
[/**STACK=***integer*]
[/**LOGGER=***logger_flags*]
[/**INPUT=***input_file*]
[/OUTPUT=*output_file*]

For more information about using the `fscript` command, refer to the *Fscript Reference Guide*.

Ftcmd Command

The ftcmd command starts the command line interface to the iPlanet UDS launch server.

ftcmd [**-v**] [**-nolog**] [**-port** *port_number*] [**-fnd** *node_name*] [**-nonode**] [**-fs**]
[**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]
{**-i** *input_file* | {[**list** {**all** | **assigned** | **public** | **forte** | **running**}] |
[**run** *application_name* [*release*] [*arguments*] [*update*]] |
[**shutdown** *app_ID* | **all** | **launcher**] |
[**update** *application_name* [*release*]] }

For more information on the `ftcmd` command, see "Using the Ftcmd Utility" on page 307.

Ftexec Command

The `ftexec` command starts a standard partition on a node. The syntax differs for client partitions and server partitions.

➤ **To start a client partition, use the following ftexec command syntax**

Portable

ftexec -fi *image_repository_name* [**-fs**] [**-fns** *name_server_address*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-fcons**] [**-fnw**] [**-fterm**] [**-fss**]

OpenVMS

VFORTE FTEXEC
/IMAGE_REPOSITORY=*image_repository_name*
*[***/STANDALONE***]*
[**/NAMESERVER=***name_server_address*]
[**/LOGGER=***logger_flags*]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]
[**/FCONSOLE**]
[**/FNW**]
[**/FTERM**]

➤ **To start a standard server partition, enter the following command**

Portablex

ftexec -fi *image_repository_name* **-ftsvr 0** [**-fns** *name_server_address*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*] [**-ftsvr 0**]

OpenVMS

VFORTE FTEXEC
/IMAGE_REPOSITORY=*image_repository_name*
[**/NAMESERVER=***name_server_address*]
/SERVER_ONLY
[**/LOGGER=***logger_flags*]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]

For more information about the `ftexec` command, refer to "Manual Startup" on page 190.

Ftexecd Command

The `ftexecd` command starts a standard partition on a server node using DCE/POSIX threads instead of iPlanet UDS threads.

Portable (all platforms)

ftexecd -fi *image_repository_name* **-ftsvr 0** [**-fns** *name_server_address*] [**-fnd** *node_name*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*]

For more information about this command, see "Manual Startup" on page 190.

Ftlaunch Command

The `ftlaunch` command starts the iPlanet UDS Launch Server.

Portable

ftlaunch [**-port** *port_number*] [**-fnd** *node_name*] [**-nonode**] [**-fs**]
[**-fns** *name_server_address*] [**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]

For more information on the `ftlaunch` command, see "ftlaunch Command" on page 304.

Nodemgr Command

The `nodemgr` command starts the `Nodemgr` utility.

Portable (all platforms)

{nodemgr | start_nodemgr}
[**-e** *environment_name* [**-b** *environment_definition_file*]]
[**-fns** *name_server_address*] [**-p** *master_password*]
[**-fl** *logger_flags*] [**-fm** *memory_flags*] [**-fst** *integer*]
[**-fnd** *node_name*] [**-i** *initialization_file*]

OpenVMS

{VFORTE NODEMGR | NODEMGRSTART | ENVMGRSTART}
[**/ENVIRONMENT=***environment_name*
[**/BOOT=***environment_definition_file*]]
[**/PASSWORD=***master_password*]
[**/NAMESERVER=***name_server_address*]
[**/LOGGER=***logger_flags*]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]
[**/NODE=***node_name*]
[**/INITIALIZATION_FILE=***initialization_file*]
[**/DETACH**]
[**/ERROR=***error_file*]
[**/OUTPUT=***output_file*]
[**/PROCESS_NAME=**"*process_name*"]
[**/UIC=***uic*]

For more information about using the `nodemgr` command, see "Startup Commands (nodemgr and start_nodemgr)" on page 104.

Olegen Command

To run the `Olegen` utility, you must be running on Windows. You can start this utility in the Windows dialog that you can access by selecting the Run command from the File menu in Program Manager.

olegen *input_specification* [*output_specifications. . .*] **[-ai]**

olegen *input_specification* **[-of** *output_file_name*] **[-ai]**

For more information about using the olegen command, see *Integrating with External Systems*.

*Rpclean Command

The `rpclean` command compacts a central or private repository.

Portable

rpclean -fr *repository_name* **[-t** *temporary_directory_name***] [-q] [-fm** *memory_flags*]
[-fst *integer*] [**-fl** *logger_flags*]

OpenVMS

VFORTE RPCLEAN
/REPOSITORY=*repository_name*
[/TEMPORARY=*temporary_directory_name***]**
[/QUICK]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]
[**/LOGGER=***logger_flags***]]**

For more information about using the `rpclean` command, see "rpclean Command" on page 268.

*Rpcopy Command

The `rpcopy` command makes a new central repository by copying an existing central or private repository.

Portable

rpcopy -s *source_repository_name* **-fr** *target_repository_name* **[-r]**
[-fns *name_server_address*] [**-fs**] [**-fm** *memory_flags*]
[-fst *integer*] [**-fl** *logger_flags*] **[-secure] [-nonsecure]**

OpenVMS

VFORTE RPCOPY
/SOURCE_REPOSITORY=*source_repository_name*
/REPOSITORY=*target_repository_name*
[/REPLACE]
[/NAMESERVER=*name_server_address*]
[/STANDALONE]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]
[**/LOGGER=***logger_flags*]
[/SECURE|NOSECURE]

For more information about using the `rpcopy` command, see "rpcopy Command" on page 255.

*Rpcreate Command

The `rpcreate` command creates a new, empty repository.

Portable

rpcreate -fr *target_repository_name* **[-r]**
[-fm *memory_flags*] **[-fl** *logger_flags*] **[-secure]**

OpenVMS

VFORTE RPCREATE
/REPOSITORY=*target_repository_name*
[/REPLACE]
[/MEMORY=*memory_flags*]
[/STACK=*integer*]
[/LOGGER=*logger_flags*]
[/SECURE | /NOSECURE]

For more information about using the `rpcreate` command, see "rpcreate Command" on page 253.

*Rpshadow Command

The `rpshadow` command creates a shadow repository.

Portable

rpshadow -fr *target_shadow_name* **-n** *repository_server_name* **[-r]**
[-fns *name_server_address*]**[-fm** *memory_flags*] **[-fst** *integer*] **[-fl** *logger_flags*]

OpenVMS

VFORTE RPSHADOW
/REPOSITORY=*target_shadow_name*
/SERVICE_NAME=*repository_server_name*
[/REPLACE]
[/NAMESERVER=*name_server_address*]
[/MEMORY=*memory_flags*]
[/STACK=*integer*]
[/LOGGER=*logger_flags*]

For more information about using the `rpshadow` command, see "Creating Shadow Repositories" on page 258.

*Rpstart Command

The `rpstart` command starts a repository server on the specified central repository.

Portable

rpstart -n *repository_service_name* **-fr** *repository_name* [**-w**]
[**-fns** *name_server_address*] [**-p** *master_password*]
[**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]

OpenVMS

VFORTE RPSTART
[**/SERVICE_NAME=***repository_service_name]*
[**/REPOSITORY=***repository_name]*
[**/WAIT_TIME=** *time _in_seconds*]
[**/NAMESERVER=***name_server_address*]
[**/PASSWORD=***master_password*]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]
[**/LOGGER=***logger_flags*]

For more information about using the `rpstart` command, see "rpstart Command" on page 262.

*Rpstop Command

The `rpstop` command stops a repository server.

Portable

rpstop -n *repository_service_name* [**-fns** *name_server_address*]
[**-k**] [**-fm** *memory_flags*] [**-fst** *integer*] [**-fl** *logger_flags*]

OpenVMSx

VFORTE RPSTOP
[**/SERVICE_NAME=***repository_service_name]*
[**/NAMESERVER=***name_server_address*]
[**/KILL**]
[**/MEMORY=***memory_flags*]
[**/STACK=***integer*]
[**/LOGGER=***logger_flags*]

For more information about using the `rpstop` command, see "rpstop Command" on page 265.

*Tclient Command

The `tclient` command starts the `TestClient` utility, which allows you to test a shared service object by running the application on multiple clients.

Portable

tclient [**-fnd** *node_name*] [**-fmn** *model_node_name*]
[**-fm** *memory_flags*] [**-fst** *integer*]
[**-fl** *logger_flags*] [**-fns** *name_server_address*] **[-fterm] [-fcons]**

OpenVMS

VFORTE TCLIENT
[/NODE=*node_name***]**
[/MODEL_NODE=*model_node_name***]**
[/MEMORY=*memory_flags***]**
[**/STACK=***integer*]
[/LOGGER=*logger_flags***]**
[/NAMESERVER=*name_server_address*]
[/FTERM]
[/FCONS]

For more information on the Tclient command, see the *iPlanet UDS Programming Guide*.

iPlanet UDS Logger and Memory Manager Flags

The following sections describe how to use the `-fl` and `-fm` flags, which are available for all iPlanet UDS commands.

-fl Flag (Log Manager)

The `-fl` flog allows you to specify logger flags to be used for the command. The logger flags set the file or files used by the LogMgr object for logging messages, and specify the types of messages logged in each file. See `LogMgr` class in Framework Library online Help for information on how to produce the actual messages.

The `-fl` flag overrides the setting of the FORTE_LOGGER_SETUP environment variable.

-fl *file_name*(*file_filter*)[*file_name*(*file_filter*)...]

For UNIX and VMS, any arguments that contain parentheses must be enclosed by double quotes.

The following sections provide information specifying the file name and file filters.

File Name

The log file name is any valid file name where you want to log messages. The special file names "%stdout" and "%stderr" log the messages to standard output or standard error, respectively.

You can specify several files for logging messages. Multiple logging files are useful, for example, in an application where you want to display general tracing on standard output (%stdout), but want detailed tracing logged to a file for later review.

On Windows only, you can use the name "%stdwin" to create a simple, scrollable output window for textual output. "%stdwin" is particularly useful to specify an alternative file for the output from `Fscript` or the iPlanet UDS Workshops.

File Filter

Each file name is associated with a *file filter*.

message_type[:*service_type*[:*group_number*[:*level_number*]]]

A description of each file filter option follows.

Message Type Option

The most general filter is message type. The value of message type differentiates messages such as errors, debugging information, or performance data. The message types appear in the following table. Each type is paired with a runtime LogMgr constant that corresponds to the message type when used with the more complex versions of the Put and PutLine methods:

Type	Message Field in the Control Panel	Meaning	Put or PutLine Constant
aud	Audit	Audit messages	SP_MT_AUDIT
cfg	Configuration	Configuration modification	SP_MT_CONFIGURATION
err	Error	Error Messages	SP_MT_ERROR
prf	Performance	Performance information	SP_MT_PERFORMANCE
res	Resource	Resource information	SP_MT_RESOURCE
sec	Security	Security messages	SP_MT_SECURITY
trc	Debug	Debugging Information	SP_MT_DEBUG
*		All of the above	Any of the above

By using the message type categories, you can print different types of messages to different files. For example, you may want to print trace messages on standard output, error messages on standard output and an error log file, and performance information in a performance log file. The specification for this setup might be the following:

```
%stdout(trc:user err:user) err.log(err:user) perf.log(prf:user)
```

Service Type Option

Within message types there are service types. Service types are the large subdivisions you make within your program and typically map to projects. The service type parameter is optional. If used, the service type value must be between “user1” and “user10”. Typically, a service is a large portion of your application, such as inventory control, accounts receivable, or employee administration.

The LogMgr constant that corresponds to the service types “user1” through “user10” is SP_ST_USER1 through SP_ST_USER10. You can use these constants with the advanced version of the Put or PutLine methods or with the Test method. For convenience, you can use the name “user” or the asterisk symbol (*) to specify all user service types. Previous examples used the specification “user” without a trailing digit to indicate all user services.

For example, if you want all tracing to go to standard output, but tracing from service types “user1” and “user3” to be logged in a special file as well, you would use the following specification:

```
%stdout(trc:user) trc1_3.log(trc:user1 trc:user3)
```

Group Number Option

Within a service type there are group numbers. Group numbers are smaller subdivisions you make within a particular service and typically map to a group of related facilities. The optional group number provides further filtering within the service. A group number is between 1 and 63 inclusive.

For example, within a particular service (say, “user3”) you may have subdivided the modules into groups (for example, “transactions in progress”, “queued work lists”, and “problem reports”). Each module is large enough to warrant a group number within the service. “Transactions in progress” may be group number 2, whereas “problem reports” may be the group number 4. The following specification puts performance information from group number 2 into one file and trace information from group number 4 into another file:

```
xactprog.prf(prf:user3:2) probrep.trc(trc:user3:4)
```

The group number you specify in a Put method may be a constant that you defined to be equivalent to the numeric literal that you specified in FORTE_LOGGER_SETUP. For example, even though the literal 2 indicates the “transaction in progress” group, your specification to print the related performance information may be the following:

```
task.Part.LogMgr.PutLine(SP_MT_PERFORMANCE,
       SP_ST_USER3,TRANSACT_IN_PROGRESS,1,perfTextData);
```

This code assumes the value of the TOOL constant TRANSACT_IN_PROGRESS is 2.

You can also specify a range of group numbers using the syntax *group#-group#*. In the previous example, if you want trace information from groups 2 through 4 to go to a specific file, you would use the following statement:

```
some_trc.log(trc:user3:2-4)
```

Level Number Option

Within a group there are level numbers that you use to specify particularly detailed levels of information. The greater the level number value, the more detailed the information. The optional level number indicates the detail level of the information printed. Level numbers must be from 1 to 255 inclusive.

As with group numbers, the level number is determined by the application. Typically, developers use level numbers to filter out trace messages. Using the current example, the specification%stdout(trc:user3:2:1) indicates that all level 1 trace data from the "transaction in progress" (group 2) module of the "user3" service should be printed to standard output. Levels greater than 1 do not print. Thus, the following fragment prints only one line:

```
log: LogMgr = task.Part.LogMgr;
-- Printed (level <= 1)
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS, 1,
           'Browsing account # ');
   log.PutLine(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
             1,acc.Number);
-- Not printed (level > 1)
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
             2,acc.Owner);
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
             2,acc.LastChangeDate);
```

-fm Flag (Memory Manager)

The `-fm` flag allows you to control the space used by the iPlanet UDS memory manager.

If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system.

Note that you can change the memory configurations for a running application using the Environment Console and instruments defined on the OperatingSystem agent. See *Escript and System Agent Reference Guide* for information.

-fm(*memory_option* {: | =} *number* [, *memory_option* {: | =} *number*])

NOTE To make this flag portable across the platforms supported by iPlanet UDS, do not include any spaces in this argument, and do not enclose any part of the argument in single quotes. [This paragraph followed a mar

For UNIX and VMS, any arguments that contain parentheses must be enclosed by double quotes, as shown in the following example:

```
"-fm(n:4000,x:8000)"
```

In UNIX, if you include spaces in this argument, you need to enclose the values, including the parentheses, in single quotes. You do not need to use single quotes for any other platform. The following table describes the memory options. For options that refer to "pages," a page is 1024 bytes of memory.

Memory Option	Description
c	Specifies when the memory pool should be contracted. The value represents the percentage utilization of the active pages that will trigger a memory pool contraction. Range is 0 to 100. The default value is `80`. This option is valid only for Windows 95.

Memory Option	Description
d	Sets the level of debugging information that is provided. The value is interpreted as a bit-mask of enabled options. The default is `0`. The options are: 1—Verify memory before every collect. This checks that all of the memory manager's data structures are correct, that all pages containing user objects are correct, and that all pointers point to something legal. 2—Verify memory after every collect. 4—Verify memory before every allocation. 8—Zero-Fill free memory. 16—Pattern-Fill free memory.
e	Specifies when the memory pool should be expanded. The value represents the percentage utilization of the active pages that will trigger a memory pool expansion. Range is 0 to 100. The default value is `80`.
g	Sets the percentage by which the memory pool is expanded. The default is `10 percent`.
i	Incremental unit in pages for memory expansion or contracting. Range is 64 to 1,048,576. Default is `256`.
n	Minimum number of pages managed by the memory manager. The value specifies the absolute minimum number of pages that will be allocated to the memory heap. Range is 1024 to 4194304 (32384 on WIndows 3.1). Must be less than the x memory option. The default value is `1024`. See "Setting Maximum and Minimum Size of the Memory Heap" on page 377 for information about how n and x interact.
r	Sets the minimum number of free pages needed to perform a shutdown. Range is 64 to 1,024. The default is `64`.
u	Target average memory use. The value specifies the target percentage utilization of the memory heap, calculated as the proportion of allocated pages that are active. Specify this as a percent of currently allocated memory. Legal range is 25 to 95. The default is `85`.
x	Maximum number of pages managed by the memory manager. The value specifies the absolute maximum number of pages that can be allocated to the memory heap. Range is 1024 to 4194304. Must be greater than the n memory option.The default value is `8192`. See "Setting Maximum and Minimum Size of the Memory Heap" on page 377 for information about how n and x interact.

For more information about how iPlanet UDS memory management works, see "Memory Issues" on page 234.

Setting Maximum and Minimum Size of the Memory Heap

To specify the maximum and minimum sizes of the iPlanet UDS memory heap, use the n and x memory options as described in the previous table.

For most operating systems, iPlanet UDS follows these rules to determine the actual maximum and minimum sizes, based on the values specified:

When you specify only the value of n:

- If n is less than 1024, n is set to 1024.
- If n is smaller than the default value of x (8192), then x is 8192.
- If n is larger than the default value of x (8192), then x is also set to n. The values of the maximum and minimum memory heap sizes in this case are equal.

When you specify only the value of x:

- If x is larger than the default value of n (1024), then n is 1024.
- If x is smaller than the default value of n (1024), then n is also set to x. The values of the maximum and minimum memory heap sizes in this case are equal.

When you specify both the n and x values:

- x is set to the larger value specified, whether by x or n. The value of n is always the value specified.

-fst Flag (Stack Size)

Sets the thread stack size in bytes for iPlanet UDS and POSIX threads.

-fst *integer*

You can also specify the thread stack size with the FORTE_STACK_SIZE variable. If both the `-fst` flag and FORTE_STACK_SIZE variable have been specified, then the larger of the two values is used.

The default value for the thread stack size ranges from 28K to 48K, depending on the platform and release of iPlanet UDS. Motif clients have a minimum size of 100K. On NT, the system default is 1MB and is not adjustable. A thread stack size setting below the default value for that system is ignored.

You can increase the stack size if necessary. Because the new stack size you specify is used for every thread, it will increase memory usage by the stack size times the number of concurrent active threads. iPlanet UDS rounds the value up to the nearest system MMU (Memory Management Unit) pagesize (8K on many machines, but sometimes more or less). iPlanet UDS adds an MMU pagesize guardword to the size; this page is memory protected to try and catch stack overflow cases.

Index

SYMBOLS

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W