Lotus

Inside Notes

The Architecture of Notes and the Domino Server

Contents

Preface

In 1989 Lotus Notes® made the world a little smaller.

Since its first release, Notes™ has enabled teams of people to work together, even when separated by great distances or when individual contributions are made at different times. Notes was the world's first viable groupware product, and it remains today the world's best groupware product. Release after release, Notes continues to evolve innovative, new capabilities that support its initial core concept: Collaboration made possible through shared, secure, online databases.

Much of what has been written about Notes over the years does not contain enough information about how Notes works to satisfy the needs of IT managers, application developers, and other technical individuals who are responsible for purchasing equipment, integrating Notes into their organizations, and designing appropriate, efficient applications. The purpose of *Inside Notes: The Architecture of Notes and the Domino Server* is to describe the internal architecture of Notes and Domino™ in sufficient detail to enable you to make optimal deployment and programming decisions.

Inside Notes provides a single place where you can quickly learn about the overall architecture of Notes. It clearly presents gist of Notes design, without getting bogged down in details. While some of the information included here has appeared elsewhere, other information has never before been published.

Chapter 1
Overview of Notes and Domino Architecture

This chapter describes the technology that forms the basis of Notes and Domino architecture up to and including Release 5.

Critical terms and definitions

Within the software industry, the use and meaning of certain terms varies widely. To ensure that you understand the use of these terms within the context of *Inside Notes*, the following is a small list of the terms and definitions that are essential for your understanding this book. A complete glossary appears at the end of this book.

Program
A program is written in C or C++, complied into machine code, and then distributed as an executable (EXE file). Examples of Notes programs include the Notes Client, the Domino Designer, the Domino Administrator, the Domino Server program, and Domino server tasks.

Program component
A program component is written in C or C++, complied into machine code, and then distributed as a dynamic link library (DLL file). Program components contain reusable code and/or resources — for example, text strings — that can be used by one or more running programs. An example of a Notes program component is the Notes Object Services (NOS).

Notes application developer
An application developer designs Notes databases.

Notes application
A Notes application is the design of a Notes database. A complex Notes application may consist of several individual database designs that work together to perform a specific task. A typical Notes application consists of a set of design elements that specify, among other things, the following:

- The type of documents (data notes) in the database.
- The way that documents can be indexed and viewed.
- The application's logic, which is written in one of four interpreted languages — Notes Formula Language, LotusScript®, Java, or JavaScript — and which can be activated based on an event, such as a button click, and/or based on schedules. Because logic written in any of these languages is interpreted the same way on all Notes and Domino platforms, a Notes application runs on all platforms to which Notes and Domino have been ported.

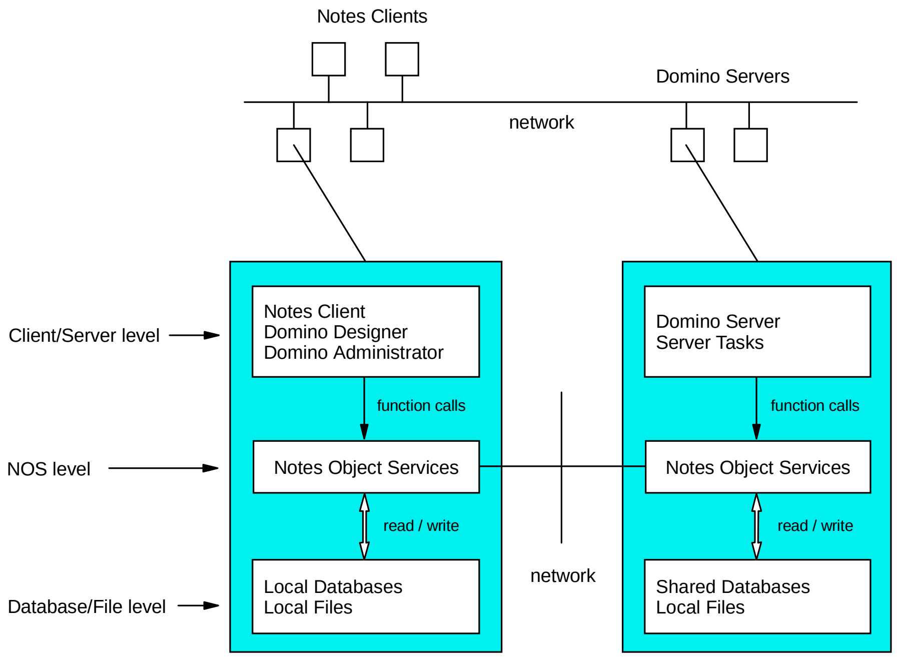
Notes database
A Notes database is a single file that physically contains both a set of documents and a copy of the application design elements that control the creation and modification of those documents.

Notes and Domino components

The following figure shows the key Notes and Domino hardware and software components. The hardware components are Notes client computers, Domino server computers, and the network that connects them. The software components reside on the client and server computers.

The same three-level software architecture is used on both client and server computers, and each key software component belongs to one of the levels:

- Client and server programs
- Notes Object Services (NOS)
- Databases and files



Client and server programs

Client and server programs use NOS to create, modify, read, and maintain databases and files.

Client programs

On client computers, the Notes Client, the Domino Designer™, and the Domino Administrator give interactive, window-based access to databases and files local to the client computer and to shared databases.

Server programs

On server computers, the Domino Server program supports the connection between clients and the server and also manages a set of server tasks, which are programs that either perform schedule-driven database chores — such as routing messages to mailboxes and updating user accounts — or connect various types of clients — for example, Notes, Web browser, and CORBA — to the server.

Notes Object Services (NOS)

The Notes Object Services (NOS) is a set of portable C/C++ functions that create and access information in databases and files, compile and interpret formulas and scripts, and interface to operating system services in a consistent, portable way. Using C-language callback functions, you can customize many NOS functions.

Databases and files

Server computers have shared databases; client computers have local databases; and both have local files.

A database is shared if it can be accessed over the network by a program running on another computer. The Domino Server program is the only program that contains the logic to respond to incoming requests from another computer on the network for access to a database. Because the Domino Server program runs only on server computers, only databases that are on Domino server computers are shared. Because NOS implements the logic that posts requests to access a shared database and because NOS runs on all client and server computers, programs running on any client or server computer can request access to a shared server database. The server may deny specific requests, however, if the requester lacks the proper access rights. When a program running on one computer accesses a shared database residing on another computer, the shared database is considered to be a remote database, with respect to the program accessing it.

A database or file is local if it can be accessed only by programs running on the same computer. Databases on client computers are local because client programs lack the logic implemented in the Domino Server program to respond to incoming database requests. Only programs running on a client computer can access databases on the client computer.

While databases contain most of the data in a Notes network, some data is kept in non-database files, such as ID files and the NOTES.INI file. These files exist on client and server computers and are always local because neither the client nor the server program contains the logic required to request or provide shared access to non-database files.

Fundamental design principles

A software design principle is a rule or a goal that guides decisions made when designing software. For example, if performance is the main goal of a software product and you have a choice between one design that results in better performance and another that results in a system that is easier to use, you would choose the better performing design.

Notes software designers had some very specific goals in mind prior to designing and developing any code. Understanding those principles is key to understanding the design of Notes. This section describes those principles as well as some of the more important design features derived from those principles.

NOS is at the heart of everything

To understand the fundamental design principles, you need to know that the Notes Object Services (NOS) is the common thread that ties the entire Notes architecture together. NOS implements the services that create and maintain Notes databases, which are the foundation of the Notes architecture. NOS functions implement a secure, note-oriented data model and contain the groupware logic that allows many programs to access notes simultaneously and resolve conflicts that occur when two or more programs try to change the same note at the same time.

NOS is the key to the multiplatform capabilities of Notes. NOS itself is portable — that is, it runs on many operating systems, thereby providing a foundation that supports portable client and server programs. In addition, NOS enforces a common database structure, regardless of the platform on which a database is created or resides. NOS contains the compilers and interpreters for the portable interpreted languages that Notes supports: the Notes Formula Language, LotusScript®, Java, and JavaScript. Programs written in these languages run on every Notes platform because NOS has been ported to all those platforms. Finally, NOS contains globalization services that enable client and server programs to input, display, and print textual information in dozens of languages used throughout the world.

Design principles

Notes software architecture relies on a combination of principles all working in concert to provide its unique groupware capabilities. Each separate design principle contributes to a whole that is greater than the sum of the individual parts:

- Notes is multiplatform.
- Notes is designed for global use.
- Notes must perform well on different platforms and at different scales.
- A Notes application can run on any Notes client or Domino server computer.
- Client programs can run on servers, and vice versa.
- The note is the basic data structure.
- Later releases of Notes support databases created using earlier releases.
- Fundamental support for security and groupware support is built into all Notes software.
- Notes uses its own databases to control many of its own activities.
- Notes uses its own databases to implement domain-wide features.

Notes is multiplatform

A software product is multiplatform if it runs on many different platforms. A platform is a specific operating system running on a specific computer — for example, four different platforms are IBM® OS/400® running on an AS/400®, AIX® running on an RS/6000®, Microsoft Windows NT running on an Intel Pentium processor, and Microsoft Windows NT running on a DEC Alpha.

As a groupware product, Notes was designed from the very start to be multiplatform. A groupware product should be multiplatform because groups can be large or small and each group requires a choice of platforms to match its technical and budgetary needs. Also, because many companies employ a mix of platforms, it is easier and more economical to deploy a multiplatform product such as Notes, rather than deploy a variety of single-platform products.

How Notes supports multiple platforms

There are a number of ways to write multiplatform software:

- The classic model involves writing a program in a portable language, such as C or C++, and then compiling it to run on different platforms.
- The virtual-machine model, used by Java, implements a common runtime environment that runs on many different operating systems, and therefore hides their differences from the interpreted programs that run on the virtual machine.
- The network-brokered model, used by CORBA, enables a program that runs on only one platform to serve clients that run on many different platforms and that use a standard network-based interface to communicate with it.

Notes uses a combination of the classic model and the virtual-machine model. Notes client programs, server programs, and NOS use the classic approach. All are written in C and C++ and are compiled to run on many different platforms. Notes applications, which comprise a large part of the Notes and Domino product, use the virtual-machine approach. They contain design elements and logic (written in Notes Formula Language, LotusScript, Java, and/or JavaScript) that are processed in an identical manner, regardless of the computer on which the application resides. Because all access to an application is through NOS functions, applications work on all platforms to which NOS has been ported.

The classic model used by the client programs, server programs, and NOS is the most difficult way to create multiplatform software. There is a large initial investment in writing a common set of sources that compile and run on many different platforms. Some parts of the code make extensive use of C/C++ conditional statements, so that a single set of source modules can be used to build Notes and Domino software for many platforms. However, the investment reaps dividends because a common code base is easy to maintain and is easy to port to new operating systems that rise in popularity and use — for example, Linux. The classic model also produces software that runs much faster than software interpreted by the other models. In fact, the only way to achieve both goals of excellent performance and multiplatform support was to write the core client, server, and NOS programs in a portable language that can be compiled into fast, native machine code for all platforms.

Further support for portability comes from the OS portability services in NOS, which allocate memory, read and write files, launch programs, and perform other OS-like functions. These services interface directly to and hide the details of the specific operating systems on which Notes runs. It is easier to port programs that call NOS portability functions than to port programs that directly call the services of any one particular operating system.

This approach to providing multiplatform support ensures that Notes provides superior performance, runs seamlessly and is scalable on many of the most popular operating systems in use today and in the future.

The following table describes the operating systems to which NOS, Notes client programs, and Domino server programs have been ported. Note that because all client and server programs require NOS, NOS runs on all platforms. The table represents what is supported in Notes Release 5, as well as in releases prior to Notes Release 5 (even if no longer supported in Notes Release 5).

Platform		**Programs ported to the platform**		
Operating system	**Processor**	**NOS**	**Client programs**	**Server programs**
MAC OS	Power PC	Yes	Yes	
OS/2®	Intel x-86	Yes	Yes	Yes
Windows 95/98	Intel x-86	Yes	Yes	Yes
Windows NT	Intel x-86	Yes	Yes	Yes
	DEC Alpha	Yes	Yes	Yes
Solaris	Intel x-86	Yes	Yes	Yes
	SunSparc	Yes	Yes	Yes
AIX	IBM RS/6000	Yes	Yes	Yes
HP-UX	HP-PA	Yes	Yes	Yes
OS/400	IBM AS/400	Yes		Yes
OS/390®	IBM 390	Yes		Yes
Linux	Intel x-86	Yes		Yes

Notes must perform and scale well on all platforms

Performing well and scaling well are two closely related and very important design goals for Notes and Domino. While the exact definition of these goals is often the subject of fierce debate, especially when trying to quantify them, the absence of these goals is quickly noted. Defined subjectively and qualitatively:

- Performing well. A product must complete its work in a timely manner in the view of the people who use it.
- Scaling well. A product scales well if it works well on a range of computers, from small to large.

In general, people want instant response from their computer applications and are frustrated with delays of even a few seconds, so making a product that performs according to the definition above can be difficult. Also, organizations, as well as the groups within them that need groupware products, come in a range of sizes. If Notes/Domino worked for only certain sizes of groups, large organizations would not be able to deploy them.

The goals of performing and scaling well are often at odds with all other design goals in a product. They can complicate the product development, installation, and ease of use. Despite these issues, performing and scaling well rank very high in the list of design principles. From the very beginning, attention to performance and scale issues has permeated the design of nearly every component of

Notes/Domino. Sometimes attention to these design goals has negatively affected the design of a feature — for example, by complicating its use or appearance — but even so, development decisions favor improving how the product performs and scales, whenever possible.

Writing software that performs well across a range of scales is very difficult. And achieving both goals across a variety of platforms, each with different memory-management services, network and I/O capabilities, display features, and so on, can seem impossible sometimes. For example, features that need to search and sort data are generally easier to develop for platforms that offer large amounts of memory because the algorithms that take advantage of large amounts of memory are easier to understand, program, and test. However, if the code must work on smaller platforms too, then using simple algorithms is often not an option. Given such a choice, the more complicated algorithm is almost always chosen so that the product scales well — that is, it works well on both memory-limited and memory-rich computers. An alternative is to use two or more algorithms to implement a feature — some algorithms that run best on memory-limited environments and some that run best on memory-rich ones — and pick the one that yields the best performance, depending on the runtime environment.

Notes is designed for global use

As a groupware product, Notes was designed from the very start for global use, since its users are likely to be anywhere in the world. A product that is designed for global use can input, manipulate, and display text information — that is, menu choices, file names, and so on — in different languages. It can also resolve issues that arise when users in different time zones interact with each other — for example, when scheduling meetings or corresponding via mail.

Lotus Multibyte Character Set (LMBCS)

A number of different character sets — that is, binary codes that represent specific text characters — are used throughout the world. The character set used on any given computer depends on the operating system running on the computer and the location of the computer.

The multitude of character sets used throughout the world creates problems for multiplatform software such as Notes. For example, how does the software convert all lowercase characters in a text string into uppercase characters? The lowercase-to-uppercase conversion rule can be different depending on the character set used to encode the string. To resolve this and other similar problems, Notes uses a single character set, the Lotus® Multibyte Character Set (LMBCS), to encode all text data used internally by its programs. Whenever Notes first inputs text encoded in a character set other than LMBCS, it translates the text into a LMBCS string, and whenever it must output text in a character set other than LMBCS, it translates the internal LMBCS string into the appropriate character set. Because all text is internally formatted by LMBCS, all text-processing operations — for example, converting lowercase characters to uppercase characters — are done in only one way.

LMBCS uses up to three bytes in memory to represent a single text character — for example, the Roman letter "a" or a German umlauted "o." The LMBCS specification defines hundreds of characters — for example, Latin, Cyrillic, Japanese, and so on — and can represent just about any character used by any language in the world.

The functions that translate text to and from LMBCS format are in NOS. These functions are called directly by client and server programs, as needed. They are also called by other NOS functions that work with text strings. For example, when a client program calls the NOS function OpenDatabase, the program file name is encoded in LMBCS. The function that opens a database, in turn, calls the NOS function that converts the LMBCS file name into one encoded in the platform-specific character set before passing it to the platform function that opens the database file.

Because NOS manages most text translation, the majority of Notes client and Domino server programs do not have to do so. There are exceptions, however. For example, if after opening a file, a client program finds that its contents are encoded in a platform-specific character set, the client program calls the NOS translation functions to convert the contents into LMBCS text.

Time and date stamps

Like text, data representing time and/or date values can also be problematic for multiplatform software used on a global scale. To resolve platform-specific differences in how time and date values are represented, Notes stores time and date information in one consistent way on all Notes/Domino computers. It defines a TIMEDATE structure and stores within it both absolute time and date values — that is, a time value represented as standard Greenwich Mean Time (GMT) and a date value represented as a Julian date — as well as some locale-specific time and date values — for example, the time zone of the computer that creates the TIMEDATE structure. The absolute time and date values guarantee that when comparing two TIMEDATE values, Notes can always resolve whether one time is before, is the same, or is after the other. The locale-specific values make it possible for Notes to present a TIMEDATE value to a user in a meaningful local time format, rather than in GMT format.

A Notes application can run on any Notes client or Domino server computer

A Notes database is a single file that contains documents (data notes), a copy of the application design elements that control the creation and modification of the documents, and administrative information. In particular, a Notes database contains some, if not all, of the following:

- A header, which contains information about the database, such as its creation time
- Documents (data notes), which hold user data
- Form notes, which control how users create, edit, and view the documents
- Code (filter notes), which validate user input, generate data, and control the flow and states of the application
- Indexes (view notes), which organize data within a database into relevant subsets
- Security information (ACL notes), which control who can create, edit, and/or see data documents and design elements (form notes, view notes, and so on)
- Other types of notes which hold miscellaneous administrative and design information (help information, the database’s icon, and so on)

Because NOS contains all the services needed to run a Notes application and because NOS has been ported to run on any client or server computer, a Notes application can run on any Notes client or Domino server computer.

By combining all design, data, and administrative information in one database file, and by guaranteeing that an application runs equally well on both client and server computers, Notes makes it easy to share an application in a groupware environment, easy to deploy an application across a network, easy to balance application loads between servers, and easy to let a mobile user run the application when disconnected from the network. This is in stark contrast to other, more traditional databases that use multiple files to hold all the different types of information that comprise and/or support a database and that, as a result, require complicated, error-prone procedures to enable minimal groupware support.

Note There are some exceptions to the model in which everything relevant to a database is included in a single database file. For example, a complex application may use two or more databases to store its data. Or if a mail message is sent to multiple mail databases on the same Domino mail server, it can optionally be stored just once in a special Single Common Object Store (SCOS) database rather than in each mail database so that disk space can be saved on the mail server. While this latter example technically violates the convention that everything relevant to a database is included in the database’s file, Notes and Domino software hide this message-sharing feature from the user and make it appear as if each mail database has its own copy of the message. In fact, a copy of the message will indeed be copied into the mail database if the mail database is copied to a client computer or to another server.

Client programs can run on servers, and vice versa

Because NOS runs on every Notes client and every Domino server, it is possible to run client programs on servers and run server programs on clients. This, of course, is true only for those platforms to which both client and server software have been ported — for example, Windows NT, Windows 95, and Windows 98.

While it makes neither practical nor economic sense to configure every computer in a Notes/Domino network as both a client and a server, this client-server duality is a design feature that provides benefits that are unavailable in a traditional client-server architecture, where clients run only client software and servers run only server software. For example, you can conveniently design and test a new database application on a client computer before you deploy it on a server. Alternatively, there are many platforms to which client programs and server programs have been ported, making it possible, for example, to run the Notes client on a Domino server without needing an extra computer.

The note is the basic data structure

One pillar of Notes/Domino design is the use of a single and simple data structure, called a note, to store all information — including design elements such as forms and views, as well as standard user documents — within a Notes database. This simple design feature leads to an incredibly powerful benefit: NOS implements a single set of note-oriented services to create and manage all the different types of information that can be in a database. Competitive products use distinct programs to create, manage, and disseminate different types of important data in the system. Developing a program to run in a multiplatform, multinetwork environment is hard enough without having to worry about multiple data structures, too. Using one simple note-oriented data model allows Notes developers to concentrate on making one small set of efficient, bug-free programs.

Later releases of Notes support databases created using earlier releases

Each new release of Notes has new features; therefore, new types of information must be stored in a Notes database. This creates a classic programming problem — what should a new release of the program do when it encounters a database created using an earlier release? There are many options, ranging from handling the older databases in stride to requiring the use of conversion tools to upgrade old databases to the latest release so that day-to-day production software has to deal only with files in the latest release.

Notes handles databases created in an earlier release in stride. This means, for example, that Notes Release 5 can work with a database created using Release 3 and that it will not add Release 4 or Release 5 information that could later prevent Release 3 from accessing the database.

The inverse, however, is not true. Earlier releases of Notes cannot work directly with databases created using later releases of Notes — but they can work *indirectly* with them. What does this mean? Say, for example, a database created using Release 5 is copied onto a Domino server running Release 3. The Release 3 software will not open a local copy of a database created using a later release of Notes. However, when using the network to access a database on a remote server, an earlier release of Notes can indirectly operate on databases created and maintained by later releases of Notes. So, for example, Notes Release 3 can access a database created using Notes Release 5 by asking the Release 5 software on the server to access the database on its behalf.

While the convention that later releases of Notes are compatible with databases created using earlier releases makes it difficult to develop new versions of Notes, customers benefit because they can run different releases of Notes in different parts of their company at the same time. For example, if a Notes Release 5 user adds a document to a Release 3 database, Release 3 users will continue to be able to work with that Release 3 database.

That multiple releases can coexist simplifies the process of upgrading to new releases. An upgrade can be staged at a pace most suitable to the customer. A company does not need to redeploy a new release to all users at once, nor does it need to upgrade all databases to the new release simultaneously. In fact, an entire company does not need to switch to a new release of Notes; some parts of a company may upgrade, while others may choose not to.

Fundamental support for security and groupware is built into all Notes software

Fundamental support for security and groupware features can be found in all Notes software — in client and server programs, as well as in the NOS functions that support them.

Regarding security, for example, the Notes Client program and the Domino Server program work together to authenticate a user prior to letting the user access a database, and the NOS function used to update a note in a database first checks the user's rights to do so against information in the database ACL note.

A large portion of support for groupware features is implemented in NOS because many Notes groupware capabilities are tightly linked to the note-oriented data model, and all of the low-level note-oriented functions are implemented in the NOS. For example, NOS contains the logic that updates a newly edited note in a database. If NOS detects that two users are trying to update the same note at the same time, NOS accepts the first update and rejects the second.

Notes uses its own databases to control many of its own activities

Virtually every computer program uses files that contain configuration and runtime information. These files do not contain user data; instead, they contain information that controls the operation of the program. Although Notes needs such files, too, it most often uses its own databases to store this information.

There are several reasons to use database files — that is, NSF files — rather than text or binary files, to hold program data. First, a database contains structured information in the form of notes that different programs can simultaneously read from and write to. These notes can be copied and then later replicated. These characteristics are critical in a networked environment where to coordinate their activities, several Notes programs often need to share data. Second, many Notes and Domino features are actually implemented as Notes databases, which are easier to understand, upgrade, port, and deploy than the core Notes and Domino programs, which are written in C and C++. Third, it implicitly makes many configuration databases accessible over the network, for the simple reason that Notes databases are transparent and easily accessible over a network connection. This makes the system that much easier to administer remotely.

For example, every server in a domain has a replica of the domain's Domino Directory (NAMES.NSF), a database that controls the user name space and contains Person documents, Group documents, public encryption keys, Connection documents that enable and schedule mail routing and replication, Server documents that contain configuration and restriction settings, and so on. How Notes works depends on the Domino Directory, which is an NSF file that you can modify and extend if you have the appropriate access rights. The same is true for several other directory-related databases that, together with the Domino Directory, give Domino a rich and capable directory mechanism.

Almost all of the Notes server tasks — for example, Router, Replicator, Indexer, and so on — read from and write to one or more Notes databases. In addition to NSF files, other databases play an important role in how Notes works. The following are a few examples of important client and server databases that do not have an NSF extension but that do have the same underlying file structure as an NSF file:

- Design templates contain design elements, security information, and computational logic and have the extension NTF.
- The MAIL.BOX file is on both a mobile client and a server. On a mobile client, the file holds outgoing mail messages. On a server, the file temporarily stores a message until the Router moves it to the next stop on the route to its destination.
- The DESKTOP.DSK file maintains the Notes client workspace — namely, the workspace pages and the databases on each page.

Notes uses its own databases to implement domain-wide features

A Notes domain is a network of client and server computers whose users, servers, connections, and access control information are described in a single database called the Domino Directory.

For groupware to work, some features require coordinating data and/or programs spread across an entire domain. For example, messaging is done by Router tasks, several of which may have to work together to route mail successfully from a source to a destination. Other features that Notes implements across a domain include administration and security.

It is often extremely difficult to implement domain-wide features in a client-server network. However, Notes builds upon its core groupware capabilities to implement some of these complex domain-wide features. To facilitate coordination between servers, Notes uses the most basic of all groupware capabilities: replication. For example, Domino administration processes depend on replication to synchronize directory information across all servers in a domain. By using replication as a design foundation, Notes developers more easily implemented domain-wide features, such as administration and security.

Overview of Notes Object Services

The Notes Object Services (NOS), which are written as a library of C functions and grouped into distinct service areas, are at the heart of all Notes software. Client and server programs — the Notes Client, the Domino Designer, the Domino Administrator, the Domino Server program, and server tasks — call NOS to create and modify Notes databases, access the network, run formulas embedded in a database, allocate memory, get time-date information, and so on.

You can create stand-alone applications — such as menu add-ins, server add-in tasks, and client/server extensions — that call into NOS. To do so, you need the Lotus C API Toolkit for Domino and Notes. The Lotus C API Toolkit is a set of include files, libraries, and documentation that you use along with a development environment, such as IBM VisualAge® for C++ or Microsoft Visual C++, to write client and server programs that call NOS C functions directly. Although not all NOS functions are available through the Lotus C API Toolkit, the toolkit does expose hundreds of the most useful NOS functions as the C API.

Programs that you write and that call NOS functions directly can manipulate Notes databases in ways that cannot be done by using only the Notes and Domino product programs. In addition, you can use Extension Manager (EM) service in NOS to “hook” hundreds of NOS functions — for example, the NSFDbOpen function — into your own code to customize its runtime behavior. Using the Extension Manager makes it easy and efficient to implement some Notes and Domino applications. Your custom EM code could, for example, cause NSFDbOpen to signal when certain databases are opened, or it could prevent the opening of certain databases.

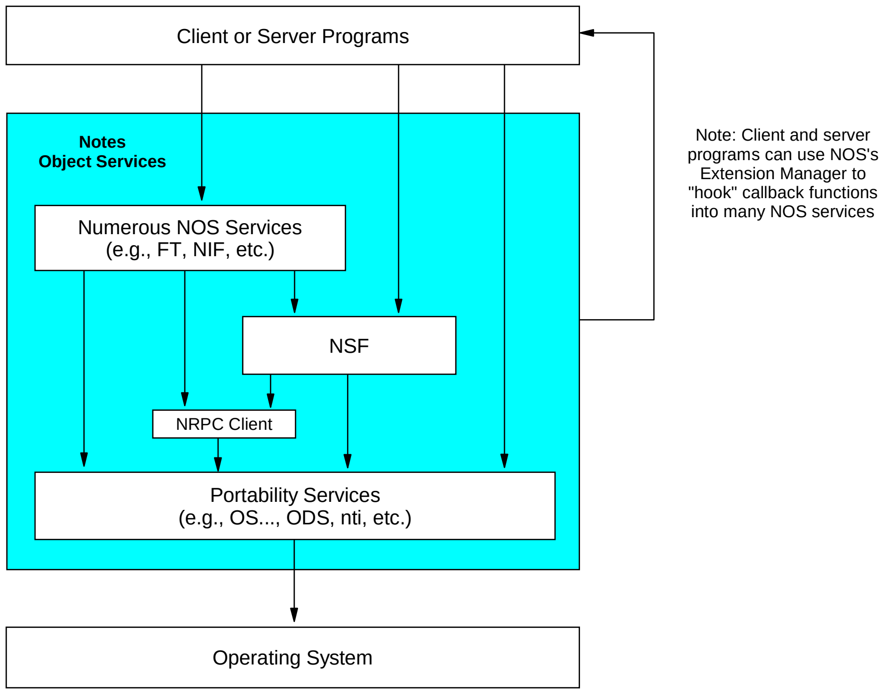
NOS is thread safe

Notes is used on multiprocessing/multitasking systems, where two or more processes — for example, the Domino Server and the Replicator server task — can simultaneously call the same NOS function. NOS is written so that if multiple processes call into it, the processes serialize their access to shared data. Serializing the access ensures that only one process at a time can access the shared data, while the others wait their turn. Programs written to work this way are called thread safe.

Because NOS itself is thread safe, programs that call into it do not have to serialize their calls. With NOS handling call serialization, programs that call into NOS operate efficiently, perform well, and do not corrupt NOS program data or Notes databases.

How NOS is organized

The following figure illustrates at a high level how NOS is organized. NOS services are grouped into a few distinct categories. Within a service, each function has a prefix that identifies the service to which it belongs. For example, the Notes Storage Facility (NSF) functions create, read, and write the contents of an NSF (database) file. Client and server programs can call any NOS services. Many NOS services, in turn, are layered relative to each other. For example, NSF functions use Portability services to accomplish their own jobs.


Client or Server Programs
Notes
Object Services
Numerous NOS Services
(e.g., FT, NIF, etc.)
NSF
NRPC Client
Portability Services
(e.g., OS..., ODS, nti, etc.)
Operating System
Note: Client and server programs can use NOS's Extension Manager to "hook" callback functions into many NOS services

The portability layer

The portability layer is key to the portability of the entire Notes/Domino product. Portability services are specifically designed to isolate high level NOS services and client and server programs from the underlying hardware and operating systems and to facilitate the development of portable applications. The services in the NOS portability layer fall into these groups:

- Operating system services
- On-disk structure (ODS) services
- Network transport services

Operating system services

These services provide a uniform API on the most common operating system services found across all Notes and Domino platforms — for example, memory-management services, file services, character-translation services, and so on.

On-disk structure (ODS) services

ODS services solve a classic portability problem that arises when programs written in C/C++ — such as, the Notes Client program and the Domino Server program — run on different operating systems and try to exchange data structures through a network or in a shared data file. Because of structure-size and byte-order differences between in-memory C/C++ data structures, a program running on one system cannot simply transfer a byte-by-byte copy of an in-memory structure to a program running on another system and expect the other program to understand it.

This problem is based primarily on differences between computer processors, not on differences between operating systems. Some processors require alignment of in-memory multibyte variables — for example, C/C++ int and long variables — to 2-, 4-, or 8-byte memory boundaries; and others do not. A C/C++ compiler adds padding bytes between structure elements to force alignment on those processors that require it. Also, processors differ in how they store the bytes of a multibyte variable in memory. On some processors, the low byte comes first; on others, the high byte comes first. These alignment and byte-order differences are the source of the problem.

Notes resolves this problem by defining a standard, or canonical, format for structures stored on disk or sent across a network. The ODS services in NOS transfer structures from a platform-specific in-memory format, which is also called host format, to canonical format, and vice versa. As a result, Notes databases are completely portable. For example, a Notes client that runs Windows or Macintosh can use a database that is on a Domino server that runs UNIX. Similarly, a Domino server that runs UNIX can use a database that is on a Domino server that runs NT.

Network transport services

Network transport services provide a single interface to drivers for different network protocols and to higher level protocols — such as, NRPC, POP3, IMAP, and so on. NOS network transport services include:

- NETBIOS, which is a peer-to-peer communication service used by clients and servers in IBM Token Ring and PC LAN networks
- SPX/IPX, which is a Novell Netware protocol used by clients and servers in PC LAN networks
- TCP/IP, which is the standard Internet LAN protocol
- XPC, which is a Lotus serial-port communication protocol
- VINES

The Notes Storage Facility (NSF)

NSF is one of the largest, most complicated pieces of NOS. NSF manages all of the databases that are active on one computer. On a large server, NSF may track operations being performed simultaneously on hundreds of databases.

NSF implements the most basic, universal, low-level database operations — such as, creating a database, creating a note, adding an item to a note, and so on. To perform these operations efficiently and maintain the integrity of databases, NSF uses buffer caching and provides for transaction logging and recovery.

Other NOS services

There are many higher level NOS services layered on top of the NOS portability and NSF services, as depicted in the diagram.

A number of these services implement most of the type-by-type differences between notes. For example, the Notes Index Facility (NIF) uses NSF to read formula items from a view note, use those formulas to select the database notes to include in the index, and then write the index back to the view note. NIF is very aware of the items that are in a view note, while NSF itself is less aware of the particular items that should be in a view note — or in any other type of note for that matter.

Other services do not use databases at all. Although these services have a portability "feel," they do not fall into the category of basic portability services because they are clearly layered on top of those basic services. For example, included in this category are services that compile and/or interpret programs written in one of the four interpreted programming languages supported by Notes/Domino: Notes Formula Language, LotusScript, Java, and JavaScript.

Some of these language services — for example, services that compile and interpret programs written in the Notes Formula Language — are accessible through the Lotus C API Toolkit; other language services are available only for internal use by the Notes Client and Domino Server programs. Whether available through the Lotus C API Toolkit or not, these services are key to the universal portability of Notes applications. Because these services are part of NOS and because NOS runs on every Notes client and Domino server platform, application logic written in any of these languages works the same, regardless of where the application resides.

The Extension Manager

Some NSF and high-level database services are designed to call application-provided callback functions at the start and end of their operations. These callbacks may affect the operation of the participating routines — for example, by doing additional work or by stopping operations for various reasons.

Application callback functions, or extensions, are managed by the NOS Extension Manager (EM). An application registers its extensions with EM. During registration, the application indicates, for each extension, which NOS function is being extended and whether to make the callback at the start or end of the function. Registration is initiated by using information in the NOTES.INI file to identify extension DLLs to Notes. At startup and before any potentially extensible NOS functions are called, the names of the extension DLLs are fetched from the NOTES.INI file, and the extension DLLs are loaded into memory and given a chance to register themselves with the EM. Several extensions can be registered for the same function.

The Notes remote procedure call client

The Notes remote procedure call (NRPC) client is an internal service within NOS. Therefore, client and server programs cannot directly call NRPC functions. Other functions — namely, those that open, close, read, and write databases — use the NRPC client to "project" their operation onto a database that is on a remote Domino server or, in other words, onto a database that is on a Domino server that is connected to the local computer by a network.

Because of the NRPC client, a program that uses NOS can access a local database as easily as it can access a remote database. For example, a program that uses NOS to open a database, count the number of notes in it, and then close it, works whether the database is local or remote. NOS and the NRPC client shield the calling program from any difficulty involved in accessing a remote database.

How the NRPC client works

An NRPC-enabled function is a NOS function that uses the NRPC client to project its action onto a remote database. Each NRPC-enabled function first tests whether the database on which it should operate is local or remote. If the database is local, the function calls other NOS services — for example, file I/O services in the NOS portability layer — to perform the operation on the local database. If the database is remote, however, the function calls an NRPC client function to construct a request message that describes the operation to perform, to send the request to the Domino server that has the remote database, and then to wait for the result.

The Domino server has a built-in server task called Dbserver. Dbserver listens for NRPC request messages, reads them after they arrive, and then uses them to call the exact same NRPC-enabled NOS function as was called on the requesting computer. As before, the NRPC-enabled function first tests if the target database is local or remote. This time the database is considered local — that is, it is a local database with respect to the Dbserver task running on the remote server — so the operation is performed locally. After the NOS function returns to the Dbserver task, Dbserver constructs a response message that contains the results of the operation and sends the response over the network to the requester. On the local computer that originated the request, the NRPC client function that made the request receives the response message, decodes it, and returns the results of the operation to the original caller. The net result is that the original caller cannot tell the difference between operations performed locally or remotely.

Many client and server programs rely on the hidden mechanisms implemented by the NRPC client and its partner, the Dbserver task on the Domino server, to simplify their design and coding. For example, the Replicator task simply reads and writes from two databases: a source database and a destination database. Because of NRPC, the Replicator does not care which database is local to it and which database is remote.

The Notes database

The Notes database is the cornerstone of Notes architecture. The majority of the Notes program is concerned with creating, maintaining, editing, viewing, accessing, copying, and replicating Notes databases. Each Notes database contains:

- A database header and other internal structures
- Notes, which fall into three categories: design elements, administrative notes, and documents
- (Optional) Replication history
- (Optional) Objects attached to notes — for example, file attachments

The database header and other internal structures

The database header and other internal structures keep track of key database information, such as database creation time, and of notes and their attached objects.

The database header

The database header stores a time stamp that indicates when the database was first created or when it was last fixed-up — that is, when notes that were corrupted as a result of a server crash were purged. This time stamp also serves as the database ID (DBID). In addition, the database header holds the unique replica ID, as well as links to the database replication history and to other internal structures that track database notes, attached objects, and free space in the database file.

Identifiers

Each note in a database has two identifiers — the note ID and the universal ID (UNID). The note ID is a 4-byte value that is assigned when the note is first created. Every database has a record relocation vector (RRV) table that maps a note's note ID to the position of the note within the database file. This table simplifies relocating a note within a database — when a note changes location, the RRV table updates to reflect the new location.

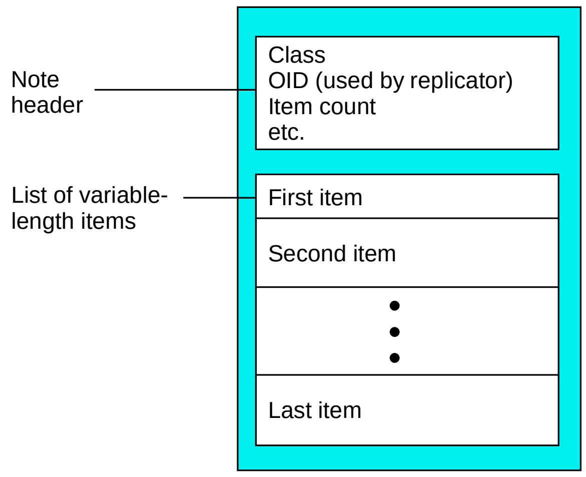
The UNID is a 16-byte value that is assigned to the note when the note is first created. A UNID uniquely identifies a note relative to all other notes in the universe, except for special copies that have the identical UNID so that they can be identified as being the same note as the original one for special purposes — for example, when replicating, or synchronizing, the notes in replica databases. Every database has a UNID table that maps the note UNID to its note ID, which in turn can be mapped through the database RRV table to the note's position within the database file. UNIDs are used when replicating database notes and when replacing or refreshing a database design notes.

The named-object table maps names to associated notes and objects. For example, this table manages per-user views, which are also known as personal or private views, and per-user unread lists. The names assigned to these views and unread lists are composed, in part, of the user's name.

The "note" in Notes

A note is a simple data structure that stores database design elements (forms, views, and so on), user-created data (documents), and administrative information, such as the database access control list (ACL). Because the same note data structure stores all these types of information, Notes requires only a single a set of NOS services to create, read, update, and replicate most of the information in a Notes database.

The following figure illustrates the logical structure of a note. Each note has a small header followed by a list of variable-length items, which are also known as fields. The header holds general information about the note, including a value that indicates the note's class — for example, document, form, or view — and its originator ID (OID). The OID contains the note's unique, universal ID (UNID), which is essential for replication. Within the item list, each item has a name, attribute flags, a value, and a value type — for example, text or number.



Note items

Every note contains a set of items that is determined by the class of note. For example, all form notes contain the same set of items, although the item values differ from form note to form note. Similarly, all view notes contain the same set of items, although the item values differ from view note to view note. Document notes are different, however, because all documents do not contain the same set of items. Because the set of items in a document depends on the form used to create the document, two document notes may have vastly different item lists.

Some items appear in almost every note, regardless of its class. For example, many notes have an item named $Revisions, which contains a list of time stamps that indicate when revisions to the items in the note occurred. Notes uses these time stamps during replication when it checks for revision conflicts.

A note may include additional, optional items, too. For example, a note might include an item containing a readers list, which contains the names of only those users who are allowed to read the document. Also, a note can have several items that have the same name. For example, one note may contain three items named $FILE (for three file attachments), and another note may contain none.

Types of notes

There are many types of notes. For some, the note class, which is a value stored in each note's header, determines the type. Other notes, however, are variations within a class. For these, Notes distinguishes the type by the presence or absence of a specific item or item value within the note. For example, agents, scripts, and script libraries are variations of the FILTER class and differ based on the value in an item called $Flags, which each of them contains.

Membership in a class group is determined solely by a note's class, which is referred to by the C-language symbol used in Notes code — for example, NOTE_CLASS_DOCUMENT, NOTE_CLASS_FORM, and so on. A word (a 2-byte value) in the header of each note contains the note's NOTE_CLASS_*xxx* value. Each NOTE_CLASS_*xxx* symbol maps to a single bit, making it easy to obtain sets of particular types of notes in a database — for example, all design elements — by using OR statements to join NOTE_CLASS_*xxx* symbols when you specify note-selection criteria.

The following table lists the 12 basic note classes and the types of notes belonging to each class. It also organizes the classes by class group.

Class group	**NOTE_CLASS_ ...**	**Note types**
Data	DOCUMENT	Document
Administration	ACL*	Access control list
	REPLFORMULA	Replication formula
	FIELD	Shared field

continued

Class group	NOTE_CLASS_ ...	Note types
	FILTER	Agent, database script, outline, script library
	FORM	Form, frameset, page, subform
	VIEW	Folder, navigator, view
Design-element	DESIGN*	Design collection (structured like a view)
	ICON*	Icon
	INFO*	About This Database document for Help
	HELP	Using This Database document for Help
	HELP_INDEX*	Index for Help

* Each database contains only a single occurrence of this class.

In addition to the 12 NOTE_CLASS_*xxx* symbols listed in the table, there are two other symbols, NOTE_CLASS_DEFAULT and NOTE_CLASS_PRIVATE, which you can join with an OR statement to the class word in the note header. NOTE_CLASS_DEFAULT identifies one note as the default note of that class. You can use an OR statement to join NOTE_CLASS_PRIVATE to any of the design-element classes. It indicates that the design-element note belongs to a particular user and, among other things, should not be changed when the database is refreshed with a newer version of its initial design elements.

Data notes

Data notes, or documents, typically comprise the bulk of a Notes database. Each document can be associated with the form note that was used to create the document and that is used by default to view or modify the document. For example, many users can use a Main Topic form to create main topic documents in a database. Each main topic document contains the items defined by the Main Topic form. In addition, the main topic document is associated with the name of the form used to create it. This association ensures that the correct form will later be used to present the document when users want to view or modify it.

When presenting a document, Notes uses a late-binding model to apply a form to the document. This approach provides more flexibility than does a model that tightly binds a document to one specific form. For example, although each document has a default form associated with it, alternative forms can be applied to the document so that the contents is presented in many different ways. In addition, through the use of field values and/or the result of a formula computation, Notes can dynamically control which form to use. Both the Notes client and Web browsers support this unique late-binding model of presenting information.

Although forms and documents are usually stored separately, a document's form may be stored within the document itself. The form is actually stored as a set of items that belong to the document. Storing a form this way makes it possible to copy a document from one database to another database that does not contain the form necessary to view and edit the document. This option, however, has a drawback in that if used too frequently, it can significantly increase the size of a database.

A document is the only type of note where the designer has some say in the fields that comprise the note. For all other types of notes — for example, view notes — Notes determines the fields. Even for documents, Notes adds some of its own fields to the list of fields specified by the form used to create the document. For example, Notes adds a field that contains the name of the form used to create the document, and when the document is a response linked in a response hierarchy, Notes adds a field called $Ref which contains the UNID of the document's "parent" document.

Administration notes

There are two types of notes that are created and managed by the database manager: the access control list note and the replication formula note. Each database has only one access control list note, which lists the access rights that various users, servers, and groups have to other notes in the database. Replication formula notes, which are optional, specify, on a server-by-server basis, which subset of notes to replicate when the database replicates with replicas stored on other servers.

Design-element notes

The database designer can create any of these design-element notes:

- Field notes define shared fields or fields that can be in more than one form.
- Filter notes hold the code of an application — that is, event handlers, agents, and so on.
- Form notes control creating, viewing, and modifying individual documents.
- View notes specify how to index the documents in a database and provide access to a specific subset of documents.
- The design-collection note, which is similar to a view, indexes the design elements, replication formula notes, and the help-index note.
- The icon note contains the database icon.
- The info note contains the About This Database document that appears the first time a user opens a database.
- Help notes contain help information about the application.
- The help-index note indexes the help notes.
- Private design-element notes contain the design elements that individual users add to a database. Each of these notes contains a field that indicates the "real" class — for example, form, view, and so on — of the note. When the Designer task refreshes design elements in a database, private design-element notes do not change.

Notes in hierarchy

To support an application that categorizes and subcategorizes documents, individual data notes can be arranged hierarchically. A note can be a main note, a response to a main note, or a response to a response note. There is a limit of up to 32 levels between a main note and its "deepest" response.

To maintain the linkage in a note hierarchy, Notes puts information into a note's header and its item list. Each document in a Notes database is uniquely identified by its UNID. Response and response-to-response documents contain a special field named $Ref, which contains the UNID of the "parent" document. This field serves to maintain the linkage of the response hierarchy.

Response notes support two important groupware features: the threaded discussion and replication-conflict identification.

Threaded discussions

One of the most common groupware applications is a threaded discussion, into which individuals can publicly share ideas, comment on ideas, comment on comments, and so on. In a threaded discussion, the large topics under discussion are usually main topics. Users comment on these topics by adding response documents. Still other users can respond to the responses, and so on.

For example, these documents are in a response hierarchy:

1.0 What is your favorite color? (2 responses)

1.1 Mine is blue.

    1.2 So is mine.

If the UNID for document 1.0 is FF863D8A:CB6E2210-852561BD:005867C7, the $Ref item on response document 1.1 contains the same number. The $Ref item on response document 1.2 contains the UNID of response document 1.1, which is its "parent" document.

Replication-conflict identification

Replication uses response notes to identify replication conflicts. A replication conflict occurs when the same note has changed in both database replicas that are being synchronized. Replication picks one note be the “winner,” makes the other note the “loser,” and makes the loser a response note of the winner.

Overview of database replication

Replication is the process of resynchronizing the contents of one database replica with another database replica. Core replication functions in NOS make it possible for both client and server programs — such as, the Notes Client program or the Replicator server task running on a Domino server — to initiate database replication. Replication can be between a local computer and a server computer or between two servers. Replication can be unidirectional or bi-directional. If you edit one replica, you can use replication to merge the edits into another replica of the same database.

Replication resolves the most difficult problem that arises in a distributed, real-time groupware product: how to keep multiple copies of databases synchronized while multiple users and programs work on them. Notes replication was designed to do just that in a secure, reliable, and flexible way. Although there may be a lag between the time when a note changes in one database replica and when the change is replicated to another database replica, replication ensures that the information never gets lost.

The note is the unit of replication. Through replication, new notes added to one replica are easily added to another replica. Similarly, notes deleted from one replica are easily deleted from another replica. When a note is modified in one replica and not the other, the modified note replaces the unmodified note. When the same note has changed in two replicas, replication merges the fields, provided different fields have changed and provided field replication is enabled. If there is no way to merge fields, replication adds one of the notes as a “conflict response” note to the other, thus preserving both sets of changes in a way that lets a user later examine them and resolve how to merge the conflicting fields.

Replication compares note originator ID (OID) values and $Revisions values to determine which notes have remained the same, which have been added, and which have been updated and/or deleted. The note OID is stored in the note header. The OID contains a UNID, which uniquely identifies the note and all replicas of the note. In addition, the OID contains a sequence number and a time stamp that together indicate how often the note has been modified and when it was last modified. Replication uses all three OID values to synchronize changes between replicas of the note.

For two databases to replicate, they must share the same replica ID. A replica ID is a unique number that is generated when you first create a database. When you make a replica of the database, the replica inherits the replica ID, the OID values, and the $Revisions values of the original database. If you make a copy of a database, the copy gets a new replica ID, and every note in the copy gets a new OID. Because a copy of a database is assigned a new replica ID, the copy is not recognized as a replica of the original database and, therefore, the original and the copy cannot replicate.

Replication can be “selective,” meaning you can specify programmatically — for example, by time or by note type — which notes to replicate. Selective replication increases overall system efficiency, especially when you want to synchronize only a subset of the data. Consider how note format and efficient note- and field-level replication together define the meaning of groupware:

- A mobile user can “take the office along” by creating a local database replica. For efficiency, the user need replicate only the relevant notes.
- A domain administrator can propagate employee move-and-change information, including the critical security information, almost instantly.

- A corporation can use clustered Domino servers to enhance Web site access, performance, reliability, and availability. In a cluster configuration, each server stores replicas of selected databases. As updates occur, the servers replicate the databases among themselves. Different users can access different replicas on different servers in the cluster. If one server experiences trouble, Notes fails over to another server that is in the cluster and that stores a replica of the database.

Domain administration

This section presents an overview of domain administration, the process that is used to create and maintain a Notes domain. Additional domain-oriented features, such as messaging and security, are described in more detail in the sections that follow.

Administration design issues

Several requirements guide the design of Domino administration software:

- Implement distributed processing
- Associate a database with the server that administers it
- Manage the Domino Directory

Implement distributed processing

Early versions of Notes implemented all administration functions in the Notes client. It became clear over time, however, that using a client-based administration program was time-consuming, tedious, and error-prone. For example, to delete a user from an early version of Notes, the administrator had to do more than simply delete the user's Person document from the Domino Directory. The administrator had to locate every database that listed the user in its ACL and manually delete the user's name from each ACL.

By implementing distributed processing, the server, rather than the administrator, manages deleting a user, as well as many other administration functions.

Associate a database with the server that administers it

Using server-based administration requires associating each database with the server that administers it. The reason has to do with replication. Consider the situation where many servers have a replica of a database. Unless only one server is responsible for making administrative updates to a database, two or more servers might apply the same administrative changes to the replicas, and there would be many conflicts when the databases replicate at a later time.

To avoid replication conflicts, each database is assigned to an administration server. Then updates made to the database on that server — the so-called primary replica — will replicate without conflicts to other replicas on other servers. In practice, this means that the assigned administration server must store a local copy of the database and that the database ACL must designate the server as its administration server.

The term "administration server" is often a source of confusion because it implies that a domain contains some special servers that are specifically configured to perform administration functions. But in fact, *every* server in a Notes domain performs administration functions on some of the databases it stores — that is, on those that are local to it and that designate it as the administration server. It makes no sense to say "Server A is an administration server." Better would be to say, "Server A is the administration server for database B," and even this is just a shorter, but less precise, way of stating "The primary replica of database B is on Server A; therefore, Server A is responsible for applying server-based administration updates to it."

Manage the Domino Directory

Most client and server programs reference and/or modify documents in the Domino Directory. Therefore, it is important that updates to the directory are reliable and universally available. As for all other databases, one server is designated to administer updates to the primary replica of the

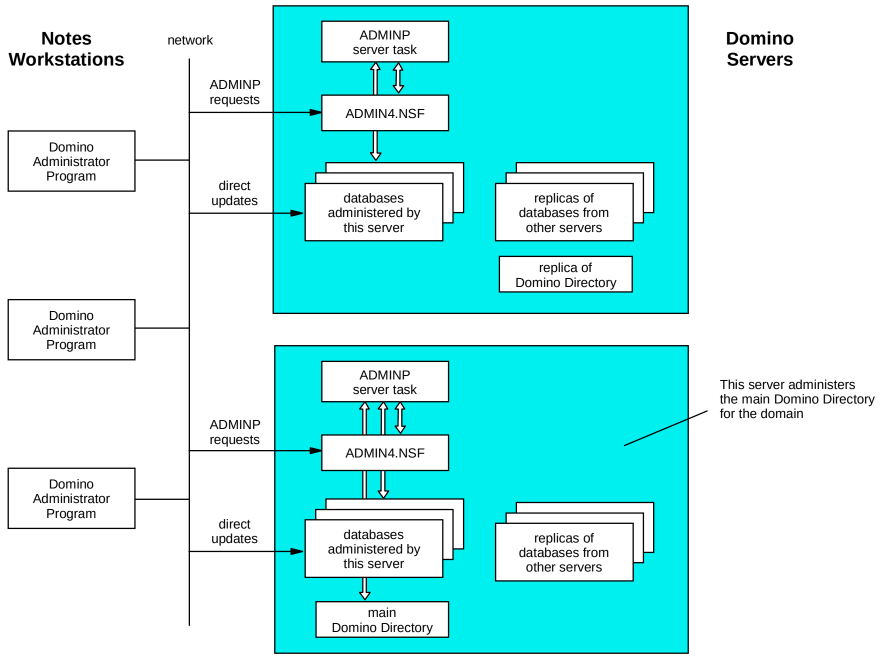
Domino Directory. Unlike other databases, though, which may or may not have replicas on other servers in the domain, every server in the domain stores a replica of the Domino Directory.

The Domino Directory is also special in that administration programs that modify it must often do more than update a field in a document to complete an administrative task. For example, after deleting a user's Person document, it is necessary to purge references to the user from the ACLs of all databases in the domain. Administration programs must be aware when they are modifying a Domino Directory and may have to perform some additional work as well.

Administration design

As the following figure illustrates, the software components used to administer a Notes domain are distributed throughout the domain.

- Any client computer can run the Domino Administrator, which the administrator uses to make administrative updates. Not shown but also available is the Domino Web Administrator, which the administrator accesses by using a Web browser and which implements a subset of the features in the Domino Administrator. In addition, the administrator can use the Notes client to initiate some administration operations. Finally, the action of updating a server can also initiate some administrative requests.
- All Domino servers should run the Administration Process server task, which processes the administration requests that are posted to a local replica of the Administration Requests database (ADMIN4.NSF).
- Every database has a server designated to administer it — that is, to update its ACL, set its size quota, and so on — and no other server will administer it or any of its replicas.
- One Domino server administers the domain's Domino Directory (NAMES.NSF), and all other servers have a replica of the Domino Directory.


Notes Workstations
network
Domino Servers
Domino Administrator Program
Domino Administrator Program
Domino Administrator Program
ADMINP requests
direct updates
ADMINP server task
ADMIN4.NSF
databases administered by this server
replicas of databases from other servers
replica of Domino Directory
ADMINP requests
direct updates
ADMINP server task
ADMIN4.NSF
databases administered by this server
replicas of databases from other servers
main Domino Directory
This server administers the main Domino Directory for the domain

Administration processing flow-of-control

Administrators use one of two programs — the Domino Administrator or the Domino Web Administrator — to specify administration tasks, such as registering users and modifying database ACLs. Administration tasks fall into three categories:

- Tasks that are simple and can be done immediately without involvement of any server processing — such as, adding a name to a database ACL

- Tasks performed by one specific server — such as, creating many replicas of a database that the server administers
- Tasks that trigger domain-wide activities — such as, deleting a user from the Domino Directory

Because some administration processing must be done on every server, every Domino server in a domain runs the Administration Process and
has an Administration Requests database (ADMIN4.NSF), which the Administration Process uses as a job-request queue: administration clients post requests in the database, and the Administration Process periodically reads the requests and acts on them.

How the Administration Process works

The Administration Process requires the replication of the Administration Requests database to distribute its activities across the domain. The following sequence of steps is typical of how an administration task is accomplished:

1. The administrator, as required by the Domino Administrator, selects a server to administer — for example, Server A.
2. The administrator specifies a task — for example, make three replicas of a database stored on Server B and put them on Servers C, D, and E. Note that none of these servers is the currently selected one, Server A.
3. Because Server A is the selected server, the Domino Administrator posts the replication request to the replica of ADMIN4.NSF on Server A.
4. Over time, the replication request in ADMIN4.NSF on Server A replicates to all ADMIN4.NSF databases on all of the servers in the domain. (Note that the administrators must specifically configure replication to do this.) Eventually, the Administration Process on Server B recognizes that it is responsible for processing the request that replicated into its local copy of ADMIN4.NSF, and it performs the task. The Administration Process that runs on each of the other servers ignores the request.

Tasks that require domain-wide processing are complicated and typically involve modifying the Domino Directory. For example, consider the case of deleting a user. Assume that Server A is the currently selected server and that Server E is the server designated to administer the Domino Directory.

1. In the ADMIN4.NSF database on Server A, the Domino Administrator posts a request to delete a user from the Domino Directory.
2. The request eventually replicates to ADMIN4.NSF on Server E, which administers the Domino Directory. Server E deletes the entry and then posts a secondary request in its own ADMIN4.NSF database to remove the user's name from all ACLs on all databases.
3. Over time, the deletion of the user's Person document from the Domino Directory replicates to all other servers.
4. Over time, the secondary request to purge the user name from all ACLs replicates to all ADMIN4.NSF databases on all servers. This time, each server responds to the request by scanning and removing the user's name from the ACL of all of the local databases that the server administers.
5. Over time, the databases replicate, and eventually no ACLs contain the name of the deleted user.

The Domino Administrator and the Domino Web Administrator

Administrators can use the Domino Administrator, which runs on any Notes workstation, to perform all administration tasks — register users, create user mail files, specify database ACLs, specify replication settings, define connections to other domains, and so on.

The Web Administrator is run by an administrator using a Web browser and implements a subset of the administration features available through the Domino Administrator. The HTTP server task, which transforms a simple Domino server into a Domino Web server, automatically creates the Web Administrator database (WEBADMIN.NSF) and assigns to it a unique replica ID so that the database cannot replicate.

The Administration Process task and the Administration Requests database

The Administration Process automates routine and/or time-consuming administrative tasks — such as purging the name of a deleted user from ACLs, modifying Readers and Authors fields in database documents, moving mail files, and so on.

The Administration Process acts on administration requests that are posted to its local replica of the Administration Requests database (ADMIN4.NSF). All servers run the Administration Process, and all servers in the domain have a replica of ADMIN4.NSF.

The Administration Process operates only on databases local to the server on which the AdminP server task is running, and performs name-management only on the subset of local databases that designate the server as the administration server. The latter restriction keeps the Administration Process from operating on local replicas of a database whose primary replica — that is, the one replica that should have administrative updates applied to it — resides on another server. Administrative updates made to the primary replica of a database eventually replicate to other replicas if replication is properly scheduled and if database ACLs specify the appropriate access rights.

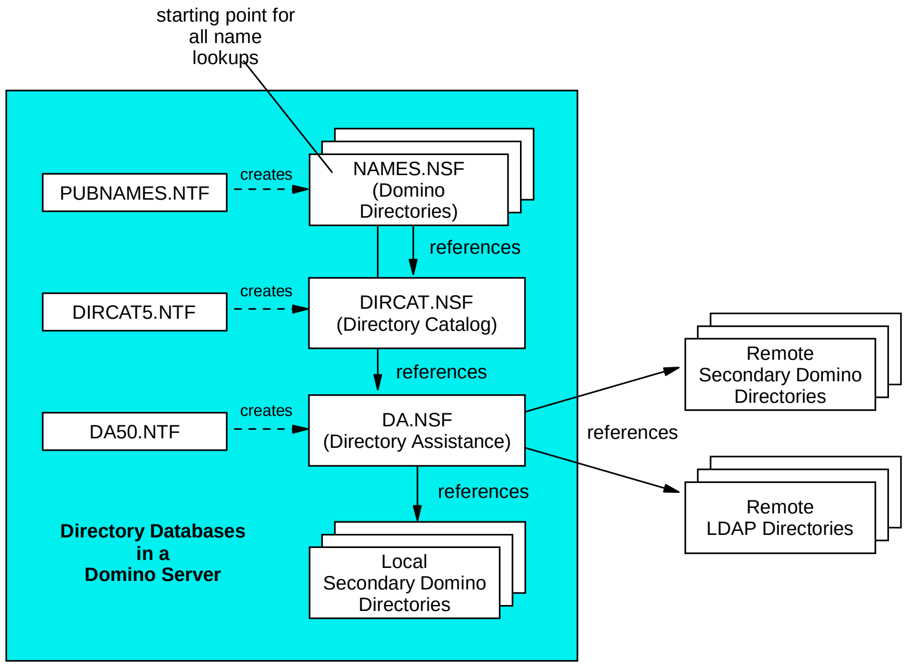
Domains and directories

A Notes domain is a collection of users, servers, and other entities that share a common Domino Directory. Or, conversely, a Domino Directory is a list of users, servers, groups, and other assets — for example, printers — that comprise a domain.

A domain is used to organize related entities — for example, a domain might describe the employees in the Marketing division of a company and the hardware and software assets they use to do their job. The design and operation of certain features of Notes and Domino — in particular, messaging (mail), security, cluster and database replication, and administration — are centered around domains. In other words, to do their jobs, these features create, access, and/or modify information in the Domino Directory.

Directory databases

This section briefly describes the databases that hold directory information. The following figure illustrates the databases used to store directory information and the templates used to create them. The figure uses default database names.


starting point for all name lookups
PUBNAMES.NTF
creates
NAMES.NSF (Domino Directories)
references
DIRCAT5.NTF
creates
DIRCAT.NSF (Directory Catalog)
references
DA50.NTF
creates
DA.NSF (Directory Assistance)
references
Directory Databases in a Domino Server
Local Secondary Domino Directories
references
Remote Secondary Domino Directories
Remote LDAP Directories

The Domino Directory (NAMES.NSF)

By default there is a single Domino Directory that has the default name NAMES.NSF. The Domino Directory inherits its design from PUBNAMES.NTF. You can change the name of this directory and/or create additional Domino Directories from the template. The Domino Directory is where clients and servers first look for information about a user, server, group, or other asset in a Notes domain. Each Domino Directory contains many entries itself, and each may contain the name of two additional databases — namely, the Directory Assistance database and the Directory Catalog database — which contain or lead to information about additional users and assets.

Directory Assistance (DA.NSF)

The Directory Assistance database, which inherits its design from DA50.NTF, lists the names of directories that store information about local or remote secondary Domino Directories and remote LDAP directories. Secondary Domino Directories inherit their design from PUBNAMES.NTF and usually contain directory information about other Notes domains.

The Directory Catalog (DIRCAT.NSF)

The Directory Catalog, which inherits its design from DIRCAT5.NTF, is used as an extension to NAMES.NSF to store a compact version of some user information. Mail-oriented applications use a directory catalog because it can present, in a very small database file, basic directory information about a huge community of users.

LDAP directories

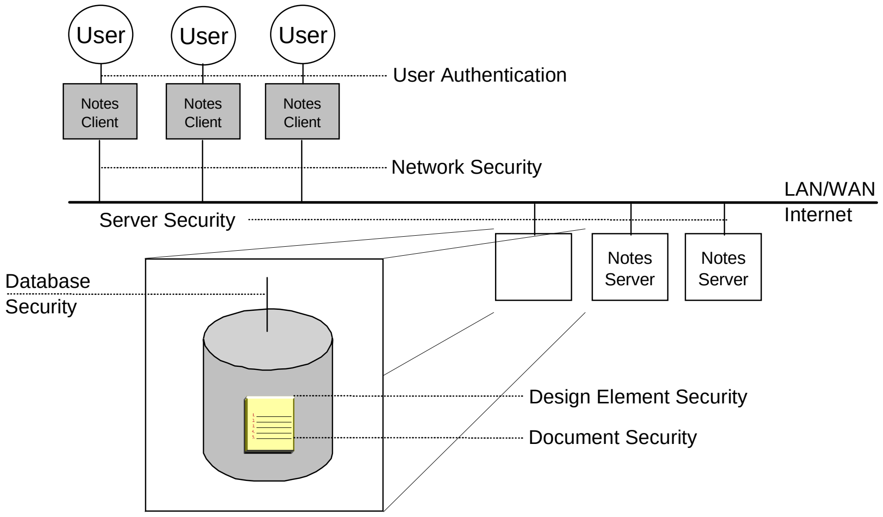
Lightweight Directory Access Protocol (LDAP) is a protocol for accessing information directories over the Internet. Domino supports LDAP in two ways. First, the Domino server can run the LDAP server task, thus making information from the Domino Directory available to other computers that are on the Internet and that use the LDAP protocol to request directory information. Second, by referencing LDAP servers in the Directory Assistance database, both the Notes client and the Domino server can be LDAP clients and retrieve directory information from an LDAP server that is located elsewhere on the Internet.

Overview of security

Notes/Domino has a layered security model that both system administrators and application developers can tailor to meet specific organizational requirements. This layered security model is similar to a home security system. In a home security system, a visitor cannot enter the grounds of the house without access to the front gate. A visitor cannot get into the house without the front-door key, and so on. The idea is that no one can gain entry to specific parts of the house without passing through each layer of security.

Notes/Domino has security features that protect its primary components — the users, network, servers, databases, design elements, and documents. Security spans the system vertically, from the user level to the document level, and horizontally, across each component. Each component has a security mechanism that applies consistently across all components at that level. For example, all users who try to use Notes clients must be authenticated. All databases have an ACL to prevent unwanted access to data. All documents can restrict who can read and write them.

The following figure the levels of Notes/Domino security.



User security

To prevent someone from impersonating a user and gaining unauthorized access to the user's data, Notes/Domino provides each user with an ID file. An ID file contains encryption keys and certificates that Domino servers use to verify the authenticity of the file and, indirectly, the authenticity of the person who has the file. To provide additional security, Notes/Domino allows the use of a password to encrypt sensitive portions of an ID file. It is unlikely that an unauthorized user can obtain an ID file and the password that encrypts it.

It is very important that users never divulge the password that encrypts their ID files. If an unauthorized user gets the password and a copy of an ID file, changing the password on the original ID file does not repair the compromise that has taken place. The unauthorized user can still use the ID to read encrypted mail and to access a server, unless password-checking has been enabled on the server.

Network security

Notes/Domino provides some network security by encrypting messages sent between client and server computers. Notes/Domino cannot enforce other types of network security, such as preventing unauthorized users from breaking into or eavesdropping on a network. The best protection against eavesdropping is to encrypt network traffic.

Server security

Server security is primarily controlled by information stored in the Domino Directory. The Domino Directory specifies which users, servers, and groups of users and servers are given or denied access to a server and/or to specific server ports. In addition, each server can specify exactly which users, servers, and groups can use passthru connections, create new databases, and/or can create database replicas on the server. Note, though, that the database ACL may restrict who can create a database replica.

Database security

Each database has its own ACL that specifies which users and servers can access the database and which tasks they can perform.

Document security

Although many users may have access to read and/or edit the documents in a database, individual documents can restrict user access. Restricting access can be achieved by the use of Readers, Authors, and Signers fields; hide-when formulas; and encryption keys that are attached to specific fields in a document. In addition, an encryption key can encrypt an entire document.

Security issues

In any computer system, many features that appear to secure data provide merely the illusion of security. In addition, even true security features have limitations. Someone who is clever and determined and who has enough time and resources can bypass every defense. As someone once said, “When it comes to security, only the paranoid survive.” To use Notes/Domino to secure data, you should:

- Use encryption
- Limit access to data by providing physical security

Use encryption

Encryption is the best mechanism for protecting data in a database and data transmitted across a network. Often the easiest way to breach security is to obtain a copy of a database or to eavesdrop on a network. However, after stealing encrypted data, the work of decrypting is very daunting. You should always encrypt sensitive data.

Limit access to data by providing physical security

A database ACL provides the security you expect from it when the database is on a Domino server and access to the database is through a network. Security is enforced by the Domino Server program, which authenticates network users and then enforces database ACLs. If you can bypass the Domino Server program, either by gaining physical access to the server computer or by obtaining a copy of a database, then you can bypass ACL security.

In addition to using the built-in Notes/Domino security features, you should provide physical security for all Domino servers. Whenever possible, locate a Domino server in a ventilated, locked room. Also consider password-protecting the server console. If servers are not secure, unauthorized users might circumvent other security features — for example, ACL settings — and access applications on the server, use the operating system to copy or delete files, or physically damage the server hardware itself.

Security tradeoffs

Notes/Domino is a groupware product, meaning it has features that make it easy to share data between members of a group. Its security features, however, serve to restrict sharing. Each organization has to find a balance between sharing data and restricting access to data.

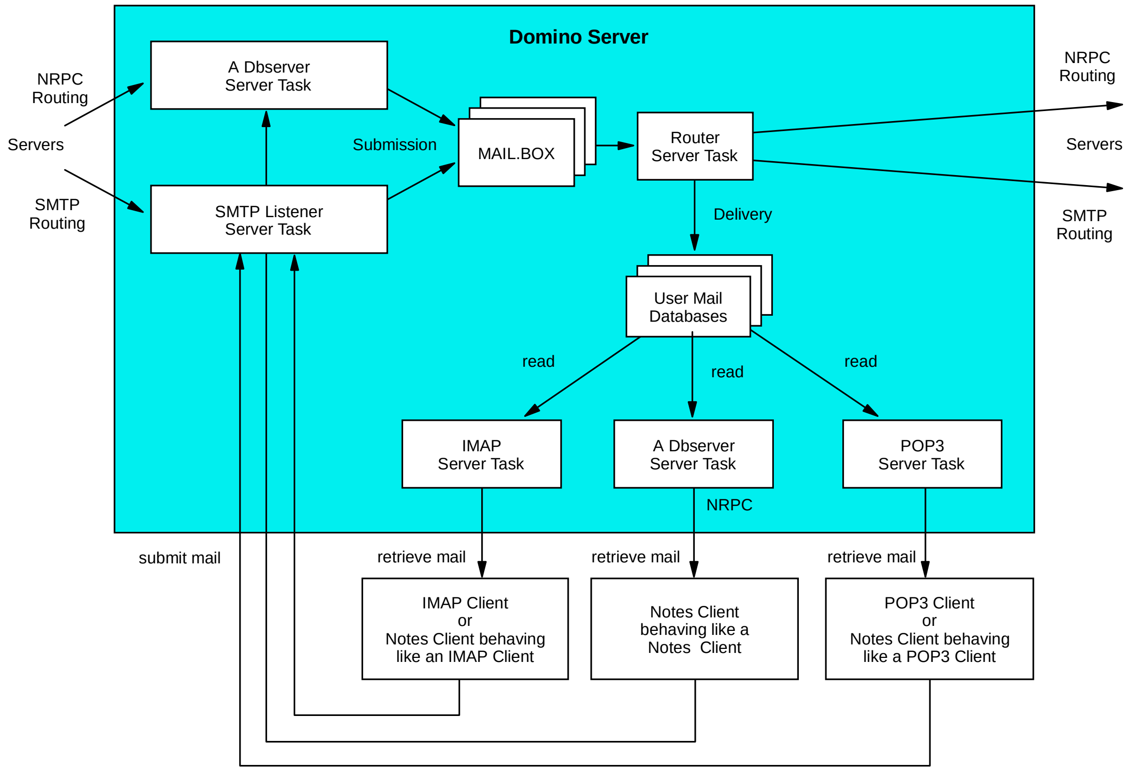
Overview of messaging

Messaging provides the foundation for secure mail communication and workflow applications. This section describes the software components that implement messaging and presents a high-level overview of how those components work together. Messaging leverages all of the strengths of the Notes/Domino architecture.

Messaging components

The following figure illustrates mail clients, a Domino mail server, and the key messaging software components:

- Mail clients: Notes, IMAP, and POP3
- Server tasks that enable mail clients to read mail: Dbserver, IMAP, and POP3
- Server tasks that enable mail clients to send mail: Dbserver and SMTP
- The Router server task, which deposits mail into user mail files or transfers mail to another Domino server or SMTP router
- The MAIL.BOX database, which holds mail that the Router server task delivers or forwards



Mail clients

A mail client is used to compose, reply to, forward, and receive mail messages. Domino messaging supports three types of mail clients: Notes, IMAP, and POP3. Each client uses one or more network protocols to send and/or receive mail and one or more formats to represent mail.

The Notes client

The Notes client is a "universal" mail client, meaning it can be used equally well in a Notes domain or on the Internet. Stated another way, you do not need a Domino server in order to use the Notes client as a mail client.

When a user composes a mail message, the Notes client has its own method of representing text internally, and it can convert its internal text into and out of two external text formats equally well: Notes rich text format (NRTF) and Internet MIME format. When a user sends a message, the Notes client examines the list of addressees and determines, on a case-by-case basis, whether to send the message in NRTF or MIME format. For an addressee whose name is in the Domino Directory, the Notes client uses the addressee's preferred mail format, which is listed in the addressee's Person document. For others — for example, an Internet addressee — the format depends on preferences specified in a Location document in the Personal Address Book or Domino Directory.

The Notes client supports three mail-retrieval protocols: Notes remote procedure calls (NRPC), IMAP, and POP3. It uses NRPC to access mail in a user's mail file on a Domino mail server. IMAP, which stands for Internet Message Access Protocol, allows the Notes client to operate wherever IMAP mail clients can be used. POP3, which stands for Post Office Protocol Version 3, allows the Notes client to operate wherever POP3 mail clients can be used. Examples of mail clients that also support IMAP and POP3 are Microsoft Outlook Express and Netscape Communicator. The Notes client can receive messages formatted in either NRTF or in MIME format. The Notes client recognizes the format of the messages it receives and generates messages in its own internal format from them.

The Notes client supports two mail-submission protocols: NRPC and SMTP. It uses NRPC to deposit a mail message directly into a MAIL.BOX database on the user's mail server. SMTP, which stands for Simple Mail Transfer Protocol, is used by Internet mail clients and mail servers to transfer mail on the Internet.

The IMAP mail client

An IMAP mail client uses IMAP to retrieve mail and uses SMTP to send MIME-formatted mail. Examples of IMAP clients are Microsoft Outlook Express, Netscape Communicator, and a Notes client that is configured for IMAP.

The Domino server runs the IMAP server task, which implements IMAP and allows users of IMAP clients to retrieve mail messages from their mail files. Messages delivered to a user's mail file are stored in the text format specified as the "preferred" format in the user's Person document in the Domino Directory — either NRTF or MIME format — with the Router translating the original message from one to the other if necessary. For a Notes client that is configured for IMAP, it is either format. For other IMAP clients, it is typically MIME format.

The Domino server runs the SMTP server task to accept MIME-formatted mail submittals from IMAP clients. SMTP deposits the mail into a MAIL.BOX file for later routing or delivery by the Router server task.

The POP3 mail client

A POP3 mail client uses POP3 to retrieve mail and uses SMTP to send MIME-formatted mail. Examples of POP3 clients are Microsoft Outlook Express, Netscape Communicator, or a Notes client behaving like a POP3 client.

The Domino server runs the POP3 server task, which implements the POP3 protocol and allows users of POP3 clients to retrieve mail messages from their mail files. Messages delivered into a user's mail file are stored in the user's preferred text format, as specified in the user's Person document in the Domino Directory — either Notes rich-text format or MIME format. The Router translates the original message from one format to the other if necessary. This could be either format for a Notes client behaving like a POP3 client, and it would typically be MIME for other POP3 clients.

The Domino server runs the SMTP server task to accept MIME-formatted mail submittals from POP3 clients. SMTP deposits the mail into a MAIL.BOX file for later routing or delivery by the Router server task.

Domino mail servers

Each mail user in a Notes domain has an assigned mail file on a Domino mail server. Each Domino mail server is responsible for receiving messages and doing one of two things with them: either depositing them into a user's mail file on that server or routing them to or toward the user's mail server.

Mail sent to a Domino mail server is first deposited into a MAIL.BOX file, a file structured and accessed as a Notes database file (NSF) despite its file extension BOX. To reduce the contention that could occur if a busy mail server had only a single mailbox file, a Domino mail server can have several MAIL.BOX files.

The Router, which runs as a Domino server task on a mail server, creates and controls the MAIL.BOX file(s); users should never make copies or replicas of MAIL.BOX files. The Router reads mail messages deposited into MAIL.BOX files, examines the addressees, and either deposits the messages into local user mail files if they are for mail users whose mail files are on that server or routes them to the target mail server.

A Domino mail server receives incoming mail via NRPC or SMTP, an Internet standard used by Internet mail servers to route Internet mail and by IMAP and POP3 mail clients to send mail. A Domino mail server runs two server tasks to receive mail, Dbserver and SMTP, one for each type of routing protocol, NRPC and SMTP, respectively. Each Domino server runs a pool of Dbserver server tasks, which process requests from Notes client computers and Domino servers by implementing the "server side" of the NRPC mechanism. Notes clients and other Domino Routers use NRPC to deposit mail directly into one of the mail server's MAIL.BOX databases. The SMTP server task manages Internet mail and deposits mail from the Internet into one of the mail server's MAIL.BOX databases.

A Domino mail server uses NRPC or SMTP to route outgoing mail. When using NRPC routing to a server that runs a release that is earlier that Release 5, the Router translates MIME messages to NRTF. When using NRPC routing to other Domino servers (Release 5 and later), no translation is required because those servers support both message formats.

A Domino mail server can run two server tasks, IMAP and/or POP3, to retrieve mail messages by IMAP mail clients and POP3 mail clients, respectively.

Mail performance

Using multiple MAIL.BOX databases and implementing shared mail enhances messaging performance. Using multiple mailbox databases lets clients or other servers deposit messages into different mailboxes simultaneously.

Shared mail stores a single copy of a message addressed to multiple recipients on one server in a shared mail database on the server. Each recipient receives a header for the message, but to save space in users' mail files, the body of the message is stored in the shared mail database. Users can still forward and reply to mail as usual.

Mail availability

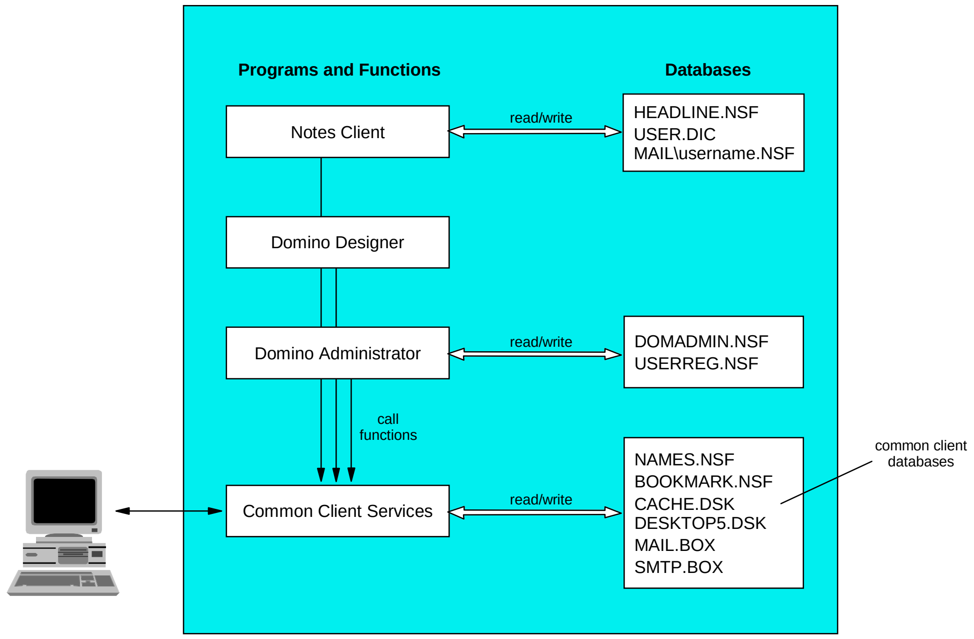
To improve mail availability, the Router can take advantage of clustering. The Router can detect if a target mail server is down and, if configured to do so, will deposit a message into another mail server in the same cluster as the intended target mail server. Similarly, it will detect if a mail server through which a message would be routed is down and, if configured to do so, will route the message through another mail server in the same cluster as the one that is down.

Client programs

There are three client programs shipped in the product family:

- The Notes Client, which is for Notes users, provides interactive access to user data.
- The Domino Designer, which is for application developers, provides access to design elements.
- The Domino Administrator, which is for system administrators, provides access to administration data.

The following figure shows the client programs and software components they use. Although not shown, NOS is used by the client programs and by the common client services that support them to read and write databases and to access system services, such as memory-management and time-date services.



Common client features

While different in the basic services they provide, the three client programs also share some common traits. For example, the clients display a similar "look and feel" by using Lotus SmartIcons® and common UI controls, and they present a common editor to the user whenever editing is required. To facilitate development and testing of client programs, features common to all of them are implemented in a library of C/C++ functions and in several common databases.

Common client services

The common client services are a set of C/C++ functions that have been ported to the platforms on which the client programs run. Included are services that create and manage windows, process keyboard and mouse input, manipulate UI resources such as menus and dialog boxes, interface to the Lotus SmartIcons and Properties box services, read and write common databases, and so on.

Common client databases

All three client programs use these common databases, which reside locally and support functionality that is not meant to be manipulated by the user:

- BOOKMARK.NSF contains bookmarks that point to both Notes and Internet elements, including databases, views, documents, Web pages, and newsgroups.
- CACHE.DSK contains design elements — forms, subforms, scripts, and so on — used in server-based databases, as well as an Unread Journal log. Storing design elements in this file improves performance because design elements are stored locally so that the client does not have to go back to the server to get them. The Unread Journal log keeps unread lists synchronized between various replicas of a database. The log records when a document's status changes from read to unread and vice verse.
- DESKTOP5.DSK stores the settings and the layout of the workspace. It contains the names of the Notes databases that the user adds to the workspace, and for each database it stores private views and an index of unread marks.

- MAIL.BOX is a database that temporarily holds outgoing Notes mail messages that a user creates when not connected to the server. When the user connects to the server, the outgoing mail moves to the server's MAIL.BOX, where it awaits further routing to its destination.
- NAMES.NSF is the user's Personal Address Book, which contains Location documents, Connection documents, and information that is required to send mail to specific people.
- SMTP.BOX is a database that temporarily stores any outgoing SMTP mail messages that the user might create when not connected to the network. When the user connects to the SMTP server, the outgoing mail will be sent.

The Notes client

The Notes client includes an interface to Domino servers and a Web browser; therefore, the client can be used to connect to Domino servers or Web servers. A user can interact with Notes databases, as well as read mail on and send mail to Internet mail servers, read and post topics to Internet newsgroups, search Internet directories, view HTML from any Web server, and use X.509 certificates for security.

In addition to the common files and databases shared by all clients, these databases provide specific functionality to the Notes client:

- HEADLINE.NSF maintains subscriptions to Notes databases. After setting up database subscriptions, users are automatically notified when changes to the databases occur.
- MAIL\*username*.NSF is the user's mail file. To facilitate mail, this file is usually stored on both the user's workstation and on the user's home mail server.
- USER.DIC is a personal directory that contains the words that the user adds to the dictionary when using the spelling checker.

The Domino Designer

Domino Designer provides a stand-alone integrated development environment (IDE) that Notes application developers use to build and deploy secure applications. Using either a Notes client or a Web browser, users can view these applications. The IDE gives the Notes application developer access to:

- Notes design elements, such as forms and views
- Web design elements, such as framesets, pages, and outlines
- The Notes Formula Language, which includes @commands and @functions
- LotusScript, JavaScript, and Java

The Domino Administrator

The Domino Administrator provides an integrated administration interface that system administrators use to manage and monitor users, databases, and servers.

The Domino Administrator provides an integrated administration interface that system administrators use to manage and monitor users, databases, and servers.

In addition to the common files and databases shared by all clients, these databases provide specific functionality to the Domino Administrator:

- DOMADMIN.NSF is the Domino Administrator database, which contains the functionality required to run the Domino Administrator.
- USERREG.NSF is the User Registration Queue database, which stores information about Notes users who are pending registration.

The Domino Server program

To start the Domino server software, you run one server program that, in turn, starts a series of other programs that are known as Domino server tasks. (The specific name of the server program is different on different platforms.) The Domino Server program and the server tasks collectively implement all of the functions that the Domino server performs.

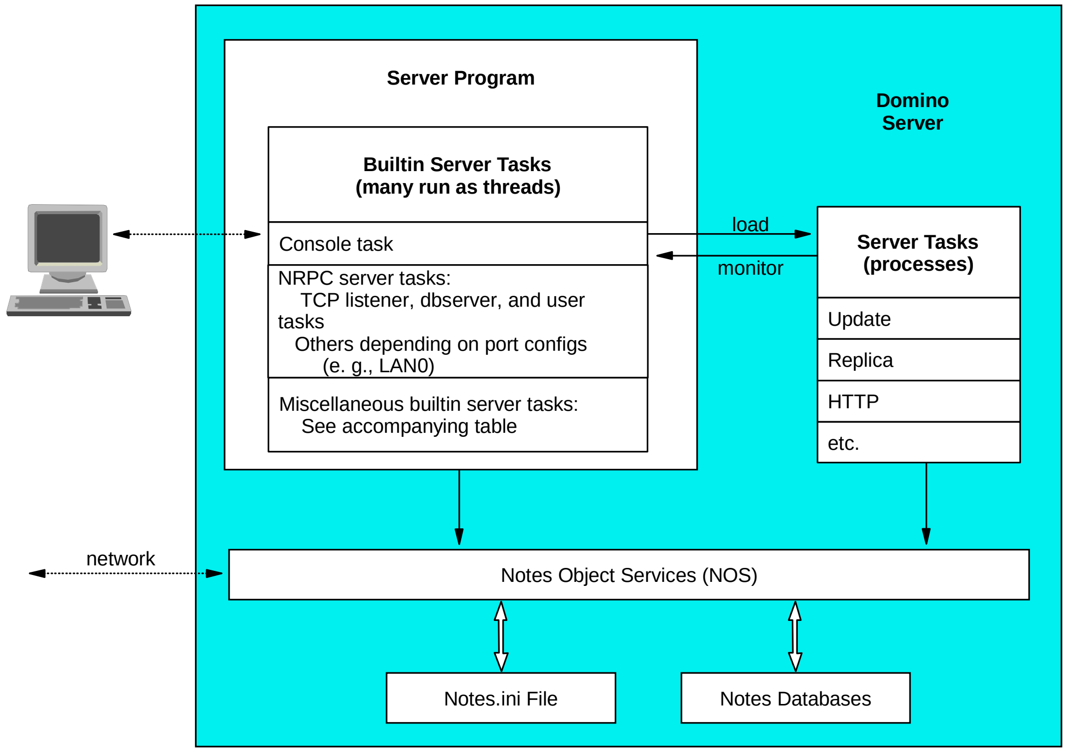
Each server task performs a specific job — for example, responding to an NRPC request from a Notes client, responding to a Web request from a browser, replicating a file, refreshing or replacing a database design, and so on. While some server tasks coordinate their activities with each other, many work independently and focus on performing one specific, narrowly defined job. At any given time, there may be hundreds, if not thousands, of server tasks running on a Domino server.

A server task is written in C/C++ and uses NOS to access the data and programs residing in server databases. The Domino Server program includes many server tasks — for example, Router, Indexer, Replicator, and so on. Some of the server tasks — namely, Server, Router, Indexer, Replicator, Administration Process, and Cluster Replicator — are described in detail elsewhere in this book.

The logic of a server task can be packaged as a stand-alone executable program — for example, the Router or the Replicator — or can be linked directly into the Domino Server program itself. Those packaged as stand-alone executable programs are always run as operating system processes. The server program decides how to run those linked directly into it — the so-called “built-in” server tasks. The server program runs some as distinct operating system processes and runs others as threads under its own operating system process.

When the server program first starts, it automatically starts several built-in tasks, including the Console task, and then checks the NOTES.INI file for settings that direct it to start other server tasks. For example, the server program evaluates the ServerTasks setting in the NOTES.INI file to determine which tasks to start immediately. Only server tasks packaged as stand-alone executable programs may appear in the ServerTasks setting. Other settings in the NOTES.INI file may direct the server program to start additional server tasks as well. If Domino doesn’t provide a server task for some functionality that you need, you can use the Lotus C++ API Toolkit and the Lotus C API Toolkit to write your own custom server task.

The following figure illustrates the Domino Server program and Domino server tasks.


Server Program
Domino Server
Builtin Server Tasks (many run as threads)
Console task
NRPC server tasks: TCP listener, dbserver, and user tasks Others depending on port configs (e. g., LAN0)
Miscellaneous builtin server tasks: See accompanying table
load
monitor
Server Tasks (processes)
Update
Replica
HTTP
etc.
network
Notes Object Services (NOS)
Notes.ini File
Notes Databases

Built-in server tasks

The logic for many server tasks is directly linked to the server program. These built-in tasks perform many essential server operations, such as running the server console. For optimal performance, on operating systems that support multithreading, the Server program runs many of its built-in server tasks as threads. Otherwise, the server program runs the tasks as processes. Some built-in tasks run only if a specific Domino feature is enabled. For example, if you enable transaction logging, the server program runs RM Checkpoint, which creates checkpoint records, and RM Flush, which clears transaction log records.

Whether built-in or not, each server task is a peer of every other server task; each plays a role as one of many tasks that collectively implement Domino server functionality. When you enter the Show Tasks command at the server console, Domino lists the built-in tasks first and indicates that they are running under the Database Server program.

Types of built-in tasks

Built-in server tasks fall into one of these categories: console, miscellaneous, and Notes remote procedure call (NRPC). Console and miscellaneous tasks are fairly simple. The Console server task manages the server console. Miscellaneous server tasks perform housekeeping chores on the server.

NRPC server tasks respond to NRPC requests that come from programs running on any Notes client or Domino server computer — for example, the Notes Client program, the Domino Designer program, the Domino Server program, and Domino server tasks or any program that calls NOS functions that generate an NRPC request, even programs written by customers who use the Lotus C API Toolkit to link to the Notes API in NOS.

For each configured server port, the Domino Server program starts a single Listener server task that "listens" for new NRPC clients that are trying to establish a connection to the server on that port. The Listener, in turn, starts Dbserver user tasks to serve NRPC requests from connected users. For all ports except the TCP/IP port, the Domino Server program runs the port's Listener task and its Dbserver tasks as operating system processes, and it creates one Dbserver task for each user connected through the port. To get the best performance on the TCP/IP port, the Domino Server program runs its Listener task and Dbserver tasks as threads under the Domino Server program's own operating system process. In addition, because there can be literally thousands of users connected to the Domino server through the TCP/IP port, the Domino Server program creates a fixed-size pool of Dbserver tasks for the TCP/IP port. This pool of Dbserver tasks acts like a set of checkout counters at the front of a supermarket: each task is capable of handling any individual NRPC request that comes in over the TCP/IP port. The pool of Dbserver tasks is made large enough to handle the number of connections expected to be active at any given time. This number is much smaller than the total number of users connected to the server at a given time. The result is a high-performing Domino server that scales well.

Table of built-in server tasks

The following table describes the built-in server tasks.

Task	Description
Cluster Manager	Probes availability of cluster members and updates cluster statistics
Console	Manages server console input and display
Db Cache Ager	Ages cached databases (those pended for closing)
Db Cache Processor	Reopens cached databases and closes fully aged databases
Db Fixup	Fixes inconsistent and/or bad data in databases listed in the fixup queue
Db User	Manages a user's connection to the server (one per NRPC user)
Listener	Listens for connection requests by NRPC clients (one per NRPC port)

continued

Task	Description
Loadmon	Updates server load-balancing statistics
Name Server	Maintains list of networked servers and the services they provide
Poll	Performs periodic housekeeping — for example, scheduling other tasks to run
RM Checkpoint	Prepares the checkpoint record for the transaction log
RM Flush	Flushes cached database information in order to free up space in transaction log
UBM Cleaner	Ages modified database information in the Unified Buffer Manager and schedules its cleaning
UBMIO	"Cleans" modified database information cached in Unified Buffer Manager by writing it to databases

Add-in server tasks

Add-in server tasks are grouped into five categories:

- Tasks that maintain Notes applications
- Tasks that monitor server and administration activities
- Tasks that manage mail, calendars, and scheduling
- Tasks that manage protocols
- Tasks that monitor server activity

Table of server tasks that maintain Notes applications

These tasks maintain the integrity of Notes applications and ensure that applications perform efficiently when users use them.

Task	Executable	Description
Agent Manager	AMGR	Runs agents on one or more databases.
Cataloger	CATALOG	Updates the database catalog, which users use to find databases and add them to their desktop.
Database compactor	COMPACT	Compacts all databases on the server to free up disk space.
Designer	DESIGN	Updates all databases to reflect changes to templates.
Directory Cataloger	DIRCAT	Populates directory catalogs and keeps the catalogs up to date.
Domain Indexer	DOMIDX	Creates a full-text index of all databases in a domain.
Replicator	REPLICA	Replicates databases with other servers.
Updall	UPDALL	Updates view indexes and full-text indexes daily. Purges deletion stubs from databases and discards view indexes for views that have not been used for a specified period of time.
Update	UPDATE	Runs continuously and updates view indexes and full-text indexes that have been tagged to be updated immediately. Also detects and attempts to rebuild corrupted view indexes and full-text indexes.

Table of server tasks that manage server and administration activities

Basic server administration tasks automate routine tasks and ensure that changes are properly coordinated and updated.

Task	Executable	Description
Administration Process	ADMINP	Automates routine administrative requests, such as renaming, deleting, or moving a user, and other minor, repetitive tasks. Uses the Administration Requests database (ADMIN4.NSF).
Cluster Administration Process	CLADMIN	Oversees the correct operation of all components of a cluster.
Cluster Database Directory Manager	CLDBDIR	Updates the cluster database directory and manages databases with cluster-specific attributes.
Cluster Replicator	CLREPL	Performs database replication in a cluster.
Map Generator	MAPS	Examines domain server topology and deposits it into the Domino Directory (NAMES.NSF) for display by the Domino Administrator.

Table of server tasks that manage mail, calendars, and scheduling

These tasks enable the connectivity necessary to allow users of various mail protocols to access their mail on a Domino server and to schedule meetings with other users.

Task	Executable	Description
Calendar Connector	CCALCON	Processes requests for free-time information from another server
IMAP Service	IMAP	Enables a Domino server to act as a mail drop for IMAP clients
POP3 Service	POP3	Enables a Domino server to act as a mail drop for POP3 clients
Router	ROUTER	Uses either NRPC or SMTP routing to route mail from MAIL.BOX to a user's mail file or to another mail server
Schedule Manager	SCHED	Returns meeting times and dates and available invitees from information stored in the Free Time database (BUSYTIME.NSF)
SMTP Listener	SMTP	Receives mail transmitted in SMTP from IMAP clients, POP3 clients, and Domino and non-Domino mail servers

Table of server tasks that manage protocols

When running on the Domino server, these tasks enable clients who use various Internet protocols to access applications on the server and enable applications running on the server to access data sources over the Internet.

Task	Executable	Description
DECS	DECS	Provides a real-time forms-based interface to enterprise data
DIIOP	DIIOP	Allows Domino and the browser client to use the Domino Internet Inter-ORB Protocol (DIIOP) server program
HTTP	HTTP	Enables a Domino server to act as a Web server so browser clients can access databases on the server
LDAP	LDAP	Enables a Domino server to provide LDAP directory services to LDAP clients
NNTP	NNTP	Enables a Domino server to act as a news server to NNTP clients

Table of server tasks that monitor server activity

Tasks that monitor server activity collect data and then store it in specific databases. Domino system administrators can then analyze the collected data.

Task Name	Executable	Description
Billing	BILLING	Collects billing information about agents, database access, document creation and retrieval, Web access, mail, replication and server sessions. Stores this information in BILLING.NSF.
Database Statistics	STATLOG	Records database activity in the log file (LOG.NSF).
Event Monitor	EVENT	Monitors events on a server. Stores event definitions in EVENTS4.NSF.
ISpy	RUNJAVA ISPY	Sends server and mail probes and stores the statistics.
Mail Tracking	MTC	Tracks a message to its final destination.
Reporter	REPORT	Reports statistics for a server. Stores the statistics in the REPORTS.NSF.
Statistics Collector	COLLECT	Collects statistics for multiple servers. Stores the statistics in STATREP.NSF.
Stats	STATS	Provides statistics for a remote computer on demand.

Programmability

This section covers those sections of Notes that:

- Allow you to program Notes applications, server add-in tasks, and LotusScript extensions (LSXs)
- Allow other programs, such as Microsoft Visual Basic applications and Java applets, to access data in a Notes database
- Allow you to connect Domino to enterprise databases

You can use a variety of development environments and products to achieve these goals.

Development environments

Domino Designer

Domino Designer is an integrated programming environment for developing applications written in Notes Formula Language, LotusScript, Java, and JavaScript. Use Domino Designer to develop applications that take advantage of all Domino services and that both the Notes Client and a Web browser can use.

Other Web authoring tools

In addition to Domino Designer, you can use many commercially available Web application development tools to create Domino Web applications. You can also use Domino Design Components™, a set of Java applets that emulate Domino Designer design elements, to develop Web applications that enhance the Web browser experience.

COM development environments

Use COM-enabled tools — such as, Microsoft Visual Basic — to access the Domino objects through the Component Object Model (COM) interface.

C and C++ development environments

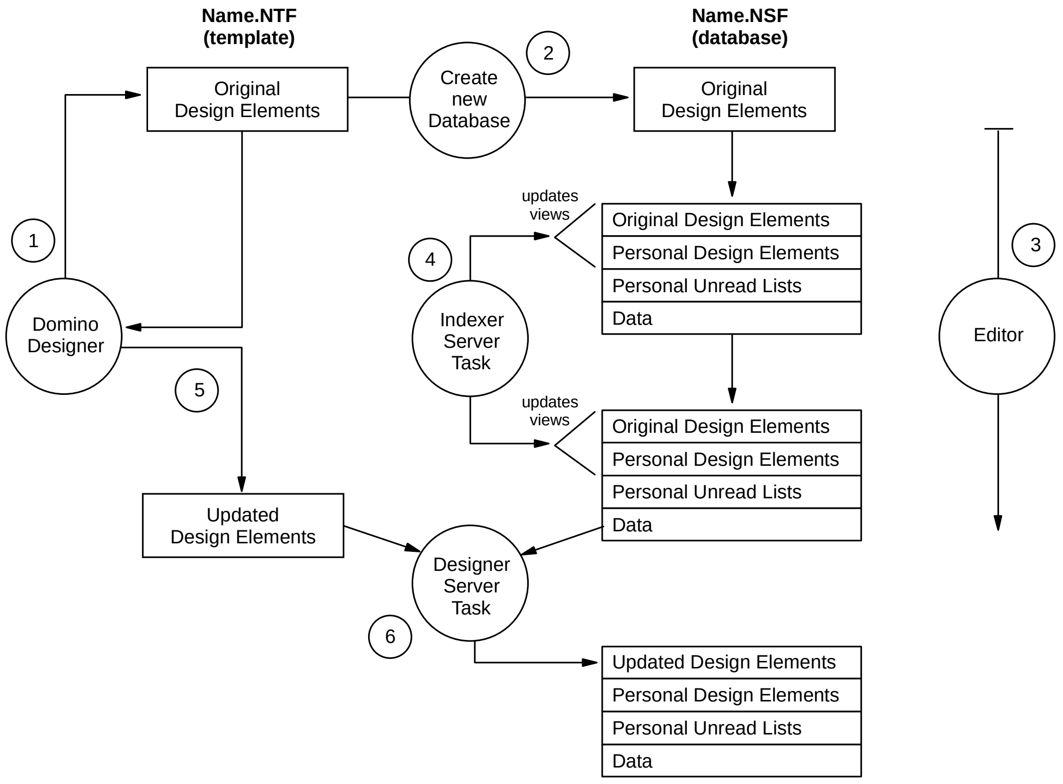
When native Domino programmability cannot accomplish the task, you can call a C DLL from within LotusScript. The DLL may be a custom-built DLL or an already existing operating system DLL. Alternatively, you can use the LotusScript Extension Toolkit to create LotusScript extensions (LSXs). LSXs are custom classes that are written in C++, are exposed to Domino, and can be scripted just like native Domino classes. You typically use an LSX to integrate Domino with other data sources and systems. Use C or C++ development environments with the Lotus C API Toolkit and the Lotus C++ API Toolkit.

Java integrated development environments

Use any integrated development environment, including IBM VisualAge® for C++, Symantec Visual Cafe, and Borland Inprise JBuilder, to develop Java applets for use in Domino applications; to write Java servlets that integrate with Domino applications; to write, test, and debug Java-based Domino agents; and to build Common Object Request Broker Architecture (CORBA) applications that remotely access Domino objects and services.

Developing, creating, and maintaining Notes applications

The following figure illustrates the steps most often used to develop and deploy a Notes application. While typical, the operations described here do not represent every way to create and maintain a Notes database. For example, while the most common way to create a database template is to use the Domino Designer, you can alternatively write a C program that uses the Lotus C API Toolkit to create one.



1. Create a template.
2. Create a new database from the template.
3. Edit and view the database.
4. Revise an existing template and refresh the design.

Creating a template

A Notes application developer typically creates a template and then uses the template to create many instances of the Notes application. A template is, in essence, an application without data. A template contains all of the design elements and computational logic that support the application. In addition, the database administrator, working in conjunction with the application developer, can seed a template with default security information before the template is used to create databases.

There is no difference between the internal structure and content of a template and that of a working database derived from it. You can, for example, use the Notes Client to open a template and add documents to it, although doing so might ruin its usefulness as a template. Despite structural similarities, templates and working databases have different roles. Therefore, to distinguish a template from a database, Notes uses different file extensions. Template databases have the file extension NTF, while the working databases derived from them have the extension NSF.

Notes itself comes with a collection of ready-to-use templates. These include a discussion template, a mail template, and a room-reservation template, to name just a few. You can use these templates as is, or you can customize them or customize a copy of them.

Creating a new database from a template

Creating a database from a template is almost exactly the same as creating a replica of a database. In particular, the originator ID (OID) of each database design element is the same as the OID in the template note from which the design element was created. The OID makes it possible to later update only a subset of the design elements with newer ones from an enhanced version of the template, even though documents have been added to the database since its creation.

In other regards, the new database is different from the template from which it inherits its design. For example, the new database has its own database ID, and its extension is NSF rather than NTF.

Editing and viewing the database

Over time many users edit a database — for example, they add documents, unread lists, and personal design elements.

The content of a document depends on the form used to create it.

Notes can create an unread list for each database user. Each unread list enumerates all of the database documents that a particular user has not yet read. The Notes client uses the unread list to highlight unread documents listed in a view and to aid the user's navigation through the documents.

A personal design element is one that is available to only the user who created it. Using personal design elements, a user can customize a shared database. The most common way to create a personal design element is to use a private-on-first-use view or folder that the application developer designed into the application's source template. Then, for example, if 100 users open such a view, Notes creates 100 personal view notes in the document. A typical personal view might be named "Show My Documents."

Revising and refreshing a design

Applications evolve and improve over time. As application developers receive suggestions for new features and reports about problems, they use the Domino Designer and other application-authoring tools to enhance and fix templates. Then, after testing the changes, the application developers redistribute the new version of the template.

The Designer server task is used to redeploy an updated version of a Notes template. This task removes all original design elements from a database and then copies into the database a revised set of design elements from the updated template.

The Domino Designer

The primary way to develop a Notes application is to use the Domino Designer, Domino's native integrated development environment (IDE). Using a group of graphical editing tools, you can quickly create a design, construct forms and views, and use one of several supported programming languages to add application logic and control. Using a special "preview in browser" feature, you can preview how your application will look on the Web.

The Domino Designer is closely integrated with additional development tools. For example, using the Domino Global WorkBench you can create databases in multiple languages so that users can choose a preferred language from a list of available languages. In addition, you can assign a specific language attribute to design elements and automatically serve the application in that language.

Notes applications use an event-driven programming model, meaning that code runs in response to events that occur in objects. The objects are databases, agents, actions, views, folders, forms, subforms, pages, fields, hotspot buttons, and hotspot actions. Using Domino Designer, you can attach code to various objects. For example, if you create a computed field in a form, you attach a formula to compute the value of the field. If you attach JavaScript code to the onFocus event of a field, the code runs whenever a user selects the field. Or you might decide to create an agent in the formula language, LotusScript, or in Java to perform scheduled updates of all documents in a database.

Domino Designer supports programming in four portable, interpreted languages: Notes Formula Language, LotusScript, JavaScript, and Java. The following table summarizes these languages.

Language	Provides	Use to
Java	Full support for Java, with a Java interface to the Domino objects and a means to store Java applets and agents in a Notes/Domino database	• Create Java agents and applications • Create servlets that read and write Domino data • Develop CORBA applications that use Domino data
JavaScript	Interactive programmability in a browser	• Program events in forms • Create graphical user interfaces for both Notes and Web clients
LotusScript	An embedded BASIC scripting language with a set of language extensions that enable object-oriented application development for Lotus products	• Program applications that the Notes client, Domino server, or SmartSuite® products host
Notes Formula Language	Provides @functions and @commands	• Automate menu commands, views, forms, agents, actions, buttons, and fields • Write selection formulas for views • Retrieve data from other sources • Calculate field values

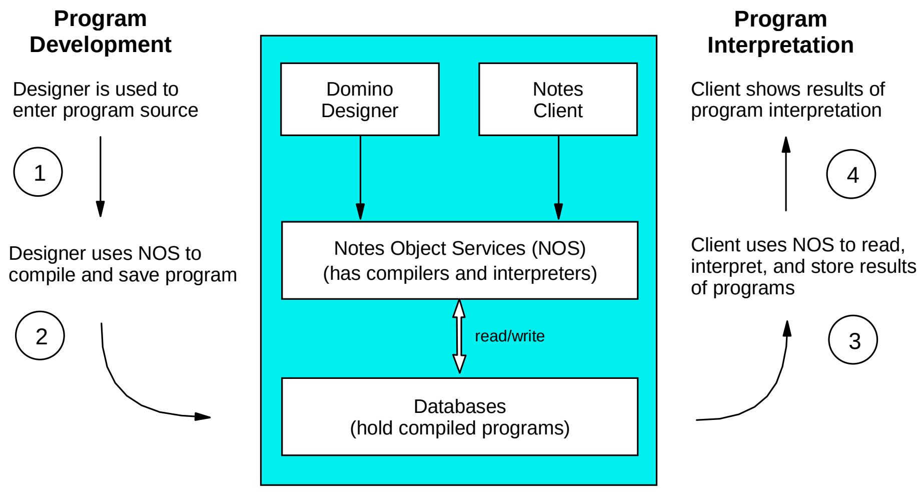
All four languages are interpreted languages, meaning that each has a program called an "interpreter" that reads and executes the logic of the program. In particular, the languages do not produce programs that must be compiled, or converted to native machine code and then executed directly on the computer's processor. While interpreted programs run slower than compiled ones, they have one big advantage: they run on any computer to which their interpreter has been ported. Consequently, an application developed in the Domino Designer on one platform will run on any Notes client or Domino server because NOS runs on all those platforms. In other words, the application is multiplatform: after creating and testing an application once on one platform, the developer can be assured that the application will run on any Notes client or Domino server.

How Notes/Domino process the built-in interpreted languages

With respect to storing and running interpreted programs, three major elements of Notes architecture come into play: the client and server programs, NOS, and databases. Notes stores and runs all programs written in its interpreted languages in a similar way. The following steps describe how Notes/Domino process the built-in interpreted languages.

1. The developer uses Domino Designer to write a program in the Notes Formula Language, LotusScript, Java, or JavaScript.
2. For each language, NOS checks program syntax and compacts the original program into a smaller format that is easier and faster to interpret. Domino Designer applies the appropriate compiler to the program (depending on the language) and uses other NOS functions to save the compacted program as data within an item of the note being designed.
3. NOS has interpreter services for each language. These services are called by client and server programs, such as the editor in the Notes client or the Agent Manager server task on a Domino server. To interpret, or run, a program, NOS reads the compacted program from the note item into an internal buffer and then applies the appropriate interpreter to the program in the buffer. Running a program usually results in modifications to the database.
4. After interpreting one of these programs, the client or server program uses NOS functions to read, process, and/or display updated database information

The following figure illustrates a database that was developed and is accessed on one client.


Program Development
Designer is used to enter program source
1
Designer uses NOS to compile and save program
2
Domino Designer
Notes Client
Notes Object Services (NOS) (has compilers and interpreters)
read/write
Databases (hold compiled programs)
Program Interpretation
Client shows results of program interpretation
4
Client uses NOS to read, interpret, and store results of programs
3

Example of a client "running" a program (formula, LotusScript, Java, JavaScript) created using Domino Designer

Web authoring tools

In addition to using the Domino Designer to develop Domino Web applications, you can use commercially available Web development environments, such as NetObjects Fusion and Microsoft FrontPage, as well.

Using Domino Designer or other Web development tools, you can include Domino Design Components in your Web applications. Domino Design Components are Java versions of a significant subset of the design elements available in Domino Designer. The Domino Design Components work with the Domino Import Service task. When you publish an application from a third-party Web authoring tool to the Domino server, the Domino Import Service task creates a document for each HTML page and stores the documents in a Domino database. The Import Service stores HTML as native HTML and marks it as passthru HTML. Storing HTML this way allows you to use cutting-edge Web design features — such as DHTML, cascading style sheets, and layered objects — without losing fidelity when the design is rendered to the browser. Site assets — such as, GIFs, applets, Shockwave files, JavaScript files, and so on — are stored as attachments to documents in the database. The Import Service uses the $File argument to convert URLs that reference these assets into Domino URLs.

While you should always use the original third-party authoring tool to edit the user interface of a Web site, you can use Domino Designer to edit any Domino design elements that you used the Web authoring tool to create. When you publish, any views or forms you created in a third-party tool are created as native Domino views and forms, as if you created them using Domino Designer. You can modify the views; add security restrictions; add agents, formulas and script to the forms; and so on. Republishing from the HTML authoring tool does not overwrite these types of changes.

Using toolkits, drivers, and connectors

Domino provides support for:

- Toolkits
- Drivers
- Connectors

Using these products, developers can access a variety of Domino services and external data sources, including:

- Access data, other than Notes databases, from within portable Notes applications — for example, data available from proprietary or special-purpose interfaces, files, and devices
- Write Domino server add-ins and NOS extensions to customize the behavior of Domino servers for particular business applications
- Develop Java programs that create and access Domino databases
- Access relational databases from Notes applications

Supported application toolkits

A toolkit is a collection of developer tools for building software components that customize or extend a specific base software product. For example, in the general software market there are toolkits for building UNIX device drivers, Netscape plug-ins, Windows printer drivers, and so on. Components developed from toolkits can be anything from a small device driver for a simple input device to a complete application built as a Java applet. Toolkits contain definition files, object libraries, build procedures, test harnesses, example code, and documentation — all of which collectively specify the application programming interface (API) between the base software product and the components developed from the toolkit.

Developers use Lotus toolkits to build applications that:

- Access Domino data programmatically, without using the Domino Designer
- Extend the LotusScript language, usually to add interfaces to special-purpose subsystems

Toolkit	Functionality	Use to
Lotus C API Toolkit for Domino and Notes	Provides subroutines and data structures	• Write C programs that perform most of the operations available through the Notes UI • Write custom server tasks that extend the capabilities of the Domino server
Lotus C++ API Toolkit for Domino and Notes	Provides a set of C++ classes and data types	• Create, access, and manage Domino databases, database design elements, and data • Perform session-level tasks, such as user registration
Lotus Domino Toolkit for Java/CORBA	Contains sample programs, documentation, and library files	• Deploy local and remote (CORBA) Notes/Domino Java agents, applications, servlets, and applets for Notes and browser clients
LotusScript Extension Toolkit (LSX)	Expands the functionality of the Domino object classes (LotusScript and Java)	• Write custom modules tightly integrated with both the Notes/Domino and SmartSuite environments

Supported database drivers

Database drivers enable users to use industry-standard APIs — namely, ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity) — to access Notes/Domino data.

Database driver	Functionality	Use to
Lotus NotesSQL®	An ODBC driver that makes Domino databases resemble another relational back-end source to an SQL tool or application interface	Gain read/write access to Notes/Domino data using any application that supports ODBC — for example, use Seagate's Crystal Reports or Microsoft Access to produce reports from Notes/Domino data
Lotus Domino Driver for JDBC	A Type II JDBC driver that makes Domino databases resemble another relational back-end source to an SQL tool or application interface	Gain read/write access to Notes/Domino data using any JDBC-enabled application — for example, servlets on an HTTP server

Supported Lotus connectors

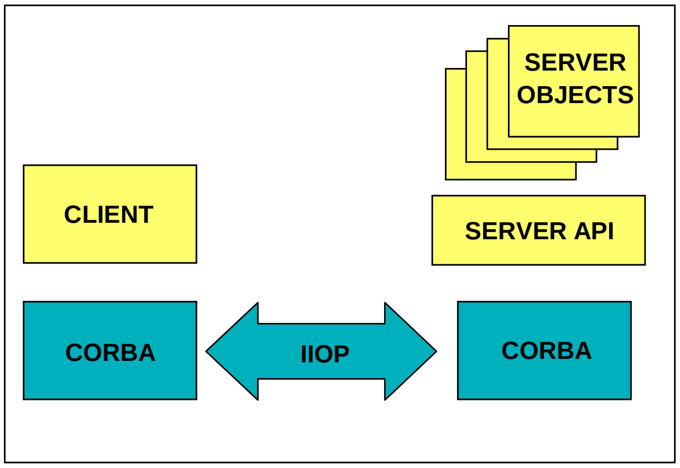
Lotus connectors permit access to external data sources from Domino for relational database-management systems, enterprise resource-planning systems, transaction-processing systems, directory services, and other services.

Connector	Provides	Available
Domino Enterprise Connection Services (DECS)	A real-time, forms-based interface to enterprise data	As an add-in task included with the Domino Release 5 server
Lotus Enterprise Integrator™ (LEI)	Schedule- and event-driven high-speed data-transfer capabilities between Domino and enterprise systems	As a separate product, formerly known as NotesPump
Lotus Connector LotusScript Extension (LC LSX)	LotusScript access to enterprise systems	On the Lotus Enterprise Integration Web site* or included with Domino Release 5 and LEI 3.0
Lotus Connector Java Classes	Java access to enterprise systems	On the Lotus Enterprise Integration Web site* or included with LEI 3.0
Lotus Connector API	C/C++ access to enterprise systems	On the Lotus Enterprise Integration Web site*

* The Lotus Enterprise Integration Web site is www.lotus.com/dominoei

CORBA support

CORBA (Common Object Request Broker Architecture) is an open standard defined by the Object Management Group (OMG), an industry standards body that includes IBM. CORBA allows applications to communicate with each other, regardless of location or platform. The transport protocol for CORBA communication over a TCP/IP network is IIOP (Internet Inter-ORB Protocol). CORBA is supported by IBM, Netscape, Oracle, and Sun and is promoted as an alternative to Microsoft's DCOM (Distributed Component Object Model).



With respect to Domino, CORBA allows Java programs on remote clients — for example, applets in browsers and stand-alone Java applications — to access Domino objects on the Domino server. From an implementation standpoint, a remote client instantiates and references Domino objects as if they were resident on the client. In fact these objects are instantiated at the Domino server. When the client is referencing these objects, it is actually communicating with the objects on the server. This is invisible to the programmer.

The same CORBA-enabled applet will use remote calls to the server within a browser. The most compelling example of this technology is the ability to place a custom applet on a form and have that applet access Domino objects in the Notes client or a browser.

Stand-alone Java applications can use CORBA to access the Domino server directly. A client does not need to be installed locally.

Java is just one of many languages that have been used to implement CORBA architecture. Domino Release 5 implements a CORBA interface for Java clients, allowing remote Java applications and Java applets to use the server side C++ Domino object model via a C++ CORBA server interface. A Domino CORBA client interface could be implemented in C, C++, SmallTalk, Ada, or COBOL.

XML support

While XML provides the resources required to describe and share data across a network, Domino provides the tools needed to make data sharing secure, reliable, and efficient. In addition to providing a medium for writing and serving XML data to an XML parser, Domino Designer provides:

- Programming tools for building collaborative applications
- Security mechanisms that protect data at many levels
- Search capabilities that enable users to locate data
- Messaging services that facilitate workflow operations such as order confirmation, mail notification, and document review
- Connectivity services that you can use to connect a Notes/Domino application with major back-end systems

Chapter 2
Notes Object Services

This chapter describes the Notes Object Services (NOS), a set of core functions that are part of the Notes client and the Domino server.

Notes Object Services

The Notes Object Services (NOS) consist of a large collection of functions. To access these functions, programmers use the Notes application programming interface (API). NOS functions perform all of the basic, portable, thread-safe services that the client and server use.

NOS functions are organized into groups of related functions. Within a group, function names start with a specific string that represents the group — for example, the string "EM" represents the group of Extension Manager functions. Groups of functions may also be layered with respect to each other — for example, some groups use services in other groups, and those groups may call the services of yet another group, and so on.

Portability

Portability is one of the primary strengths of Notes. A large organization often requires a combination of operating systems to manage its enterprises. For example, the human resources department may use a benefits package that runs on Windows, while the engineering department may run a CAD/CAM package that requires UNIX. Because Notes runs on multiple operating systems, these departments can use their department-specific software packages and use Notes.

To achieve this portability, Notes implements just one set of NOS source code that works the same on every operating system on which Notes runs. From the architecture design perspective, this portability is the result of extensive use of conditional compilations within the portability layer of NOS — that is, the layer within Notes that interacts with the operating system. Ported to interact directly with the native operating system, common service functions dynamically take advantage of extra memory or I/O features that are available on some operating systems but not on others.

Implementing NOS as a kind of virtual operating system restricts direct interaction with underlying platforms and effectively isolates much of the Notes-specific functionality from inherent differences in the hardware and software.

NOS is thread-safe

A multiprocessing operating system can run several processes simultaneously. Multiprocessing systems do this in several ways. Some use multiprocessor hardware — that is, the computer has multiple processors, each of which runs one or more processes. Other single-processor multiprocessing operating systems use process-scheduling software, which rapidly switches the single hardware processor from software process to software process, giving the appearance that several programs are running at the same time.

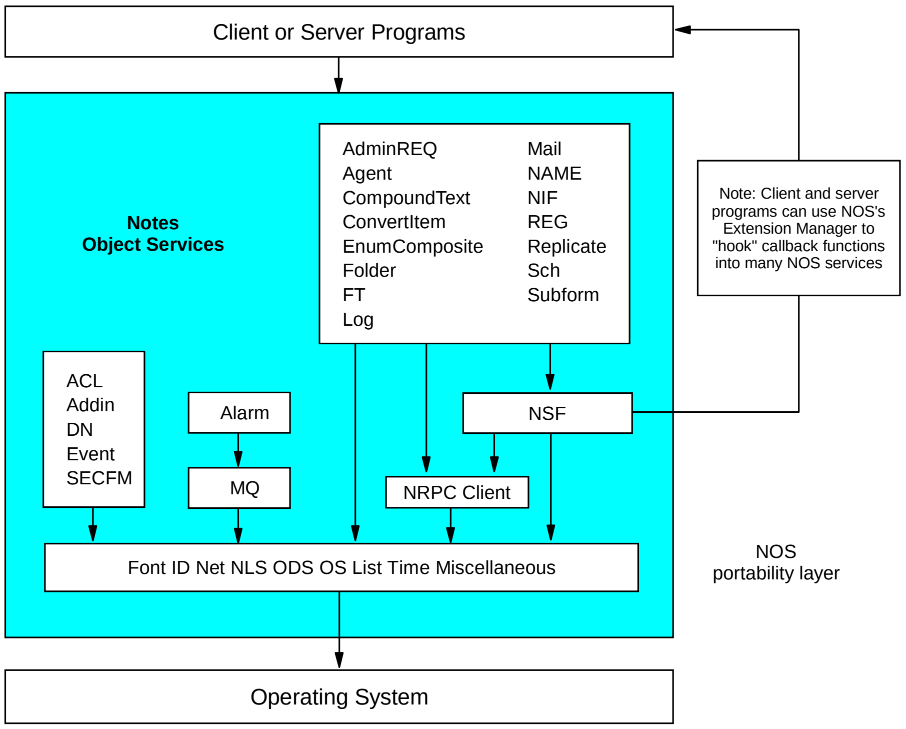
On a multiprocessing system, two or more Notes applications could try to use NOS functions simultaneously. Moreover, because of the Notes groupware features, those applications might try to access the same data simultaneously — for example, both might try to open the same document in a Notes database. If Notes allowed this behavior, data could become corrupted as two or more programs intermix their reading and writing of the document. To prevent such corruption, Notes programs serialize their access to shared data.

Serialization prevents multiple processes, which are sometimes called “threads,” from corrupting data. Code that has serialization built into its logic is sometimes called “thread-safe.”

NOS functions have built-in serialization; therefore, they are thread-safe and have all the necessary logic to prevent the corruption of shared data. The services ensure the safety of threads in the most efficient way possible for the platform on which the functions run. Therefore, because the services are thread-safe already, applications that use them do not need to implement their own serialization logic

Figure of NOS architecture

The following figure illustrates layering between functional groups in the NOS.



NOS service groups

Most services are in one of the following service groups:

- Portability layer is the level at which Notes interacts with the underlying platform and where you find those Notes components that handle all interaction with the underlying system.
- Network services allow an application to use its own protocol to connect to and communicate with a foreign system.
- Database services provide access to the contents of Notes database files. Low-level database services provide functionality for tasks such as, creating, backing up, deleting, and so on. High-level database services provide functionality for tasks such as full-text indexing, mailing, and scheduling.
- Other NOS services allow you to create and manage ID tables, customize Notes behavior by creating user-written callback functions, manipulate an in-memory copy of an ACL, and so on.

Table of NOS service groups

The following table describes the service groups in NOS.

Service group	Description
ACL	Manages the contents of an in-memory ACL. These services are usually used in conjunction with NSFDbReadACL and NSFDbWriteACL.
AdminReq	Submits requests to the Administration Process (Adminp).
Add-in	Facilitates logging within Domino add-in applications.
Agent	Loads and runs "agent" notes in a database.
Alarm	Manages Notes alarms.
CompoundText, ConvertItem, EnumComposite	Manages in-memory rich text objects and converts rich text objects into plain text objects.
DN	Processes names that are made up of distinguished-name fields — for example, "CN=...."
EM	Customizes Notes functionality by "hooking" a well-specified set of Notes activities — for example, opening or closing a database — with user-written callback functions.
Event	Manages events and event queues.
FT	Creates full-text indexes of local databases and searches for documents that match a given query.
ID	Manages compressed lists of note IDs for notes that are in the same database.
List	Manages in-memory text lists, which are a common type of item in a database note.
Log	Manages logging.
MQ	Enables Notes applications to respond to service requests from other Notes applications.
Mail	Handles items — such as, header fields and attachments — within mail messages.
Name	Locates Domino Directories and address books and retrieves information from them.
Net, NTI	Provides a portable way for applications to access networks and network transport drivers.
NIF	Creates and manages indexes onto one or more documents within a Notes database.
NLS	Manages character-set conversions.
NSF, Folder, Subform	Accesses the contents of Notes database files.
ODS	Translates data in NSF files to or from a usable in-memory format for portability.
OS,	Allocates memory, get environment information, and so on.
REG	Registers clients, servers, and IDs.
Time	Uses date/time information.
Sch	Uses the Notes API to manage user schedules.
SECKFM	Manages users, public keys, and certificates.
Miscellaneous	Includes these services: Abstract, Billing, Convert, IntlTextCompare, Map, Soundex, and Stat. For more information, see the most current version of the Lotus C API Toolkit.

NOS portability layer services

The portability layer contains Notes components that are responsible for handling all interaction with the underlying operating system. To interact with a host operating system, software must correctly use the services and resources of the operating system. In particular, there are three primary interactions with the host system:

- Transfer of data from memory and disk
- Transmission of data over the network
- Use of host O/S services

Routines in the portability layer hide these differences from higher level applications and services. Underlying systems may manage and process data differently on each system, but higher level components are unaffected by these differences.

The following table describes the portability layer services.

Portability layer service	**Description**
Notes language services (NLS)	Works with text strings
On-disk structure (ODS)	Translates NSF file data to or from a usable in-memory format
Operating system services (OS)	Allocates memory, gets environment information, and so on

Notes language services in NOS

Notes language services, which all begin with the prefix NLS, manage character strings. These services:

- Search a string for a given substring
- Find the first instance of a character in a string
- Retrieve a character (given a pointer to it) and advance the pointer
- Determine the type of character (uppercase, lowercase, numeric, arithmetic, punctuation, and so on)
- Load a given character set
- Copy a character from one buffer to another
- Return the number of bytes in a string
- Return the number of characters in a string

On-disk structure services in NOS

On-disk structure services, which all begin with the prefix ODS, translate data in an NSF file to or from a usable in-memory format. These services:

- Get the length of a data structure
- Convert a structure from canonical format to machine-specific format
- Convert a structure from machine-specific format to canonical format

Operating system services in NOS

Operating system services, which all begin with the prefix OS, collect environment variables associated with the local operating system. These services:

- Get system information — such as time/date
- Create and work with a dynamic array
- Get or set the string value of an environment variable
- Load and run an external program
- Allocate and de-allocate a block of memory from the Notes/Domino system

NOS network services

Some network services support the Notes remote procedure call (NRPC) mechanism. These services permit clients and servers to access shared server databases over several types of networks, including dialup networks and LANs. Those particular network services are not available through the Lotus C API Toolkit for Domino and Notes, except indirectly through the use of remote Notes API functions.

Other network services, which are available through the Lotus C API Toolkit, provide network send and receive routines that let you connect over a serial port to a remote system and exchange data in character mode with that system. Using NOS network services, an application can connect to and use its own protocol to communicate with a foreign system.

Network services:

- Provide a portable way for applications to access networks and network transport drivers
- Provide a single interface to drivers from many networking protocols
- Can initiate and receive phone calls or create LAN protocol sessions for communication over ports that are defined by Connection documents in the Domino Directory

These routines provide two advantages over directly using the operating system serial driver. First, they allow the application to share the same serial devices that Notes/Domino use. Second, they allow the application to take advantage of the modem support built into Notes/Domino.

An application should use these routines in the following sequence:

1. Call NetLink to dial out to the remote system and establish a session to use to communicate with that system.
2. Call NetSetSessionMode to set the operating parameters for the session.
3. Call NetSend and NetReceive to communicate over the session in a manner required by the protocol of the application.
4. Call NetCloseSession to hang up the phone, release the serial device, and close the session. Because these are important steps, you should never omit them once a call to NetLink returns successfully, even if any subsequent calls to any of the other routines fail.

NOS database services

Notes database services provide access to the contents of Notes database files. A Notes database is a collection of documents, each of which is a collection of fields. Documents and fields are also known as notes and items, respectively. Notes databases also contain header information that describes, for example, the date the database was created and last modified and the universal ID (UNID).

Notes database services fall into two major groups:

- Notes Storage Facility (NSF) services, which provide low-level database services — such as, creating, backing up, and deleting
- High-level database services, which include services such as searching, mailing, and scheduling

Notes Storage Facility services in NOS

Notes storage facility services, which all have the prefix NSF, provide low-level database services. These services, which are extensible through the Extension Manager, include:

- Create, open, or close a database
- Work with database maintenance functions, such as compacting and replicating
- Copy all or some of the database contents
- Work with ACLs

- Back up a local database
- Access the database header, selectively
- Copy or delete every type of note
- Read, write, and modify data notes and all of their fields
- Read, write, and modify form and view notes
- Work with Notes @functions

High-level database services in NOS

High-level database services include services such as searching, mailing, and scheduling. Some of these are extensible through the Extension Manager; others are not. The non-extensible database services are:

- AdminReq
- Agent
- Calendar and scheduling
- Composite text, including ConvertItem and Enum Composite
- Folder
- Full text
- Log
- Mail
- Name
- Notes Index Facility
- Registration
- Subform

AdminReq services in NOS

AdminReq services create requests for the Administration Process. These services let you create the following requests:

- Check Access for Move Replica Creation
- Check Access for New Replica Creation
- Delete in Access Control List
- Delete in Address Book
- Initiate Rename in Address Book
- Move Person in Hierarchy
- Recertify Person in Address Book
- Rename Person in Address Book

Agent services in NOS

Agent services let you load and run "agent" notes in a database. Like form and view notes, agent notes are design notes that are of class NOTE_CLASS_FILTER and that reside in databases. Domino application developers design agents to automate operations on documents in a database.

Agent notes consist of a document-selection formula, a trigger, and one or more actions. An action may be a Domino action, an @function formula, a LotusScript program, or a Java program. A predetermined time or event triggers each action, and each action works on a preselected set of documents.

Using agent services, you can create, test, run, and view the results of agents that work on a particular database. These services introduce several on-disk structures that the C API requires to manage the note. When you inspect these include files, be aware that the terms "assistant" and "agent" are synonymous. Notes/Domino also provides a set of C API functions that support the running of agent notes.

These services:

- Open a note containing an agent
- Run an agent
- Set time limit for agent execution
- Determine whether the agent is enabled
- Close the agent

Calendar and scheduling services in NOS

Calendar and scheduling services, which all have the prefix Sch, use the Notes API to manage user schedules. These services are extensible through the Extension Manager. These services:

- Add an appointment or a meeting invitation to a user's schedule
- Delete a scheduled event from a user's schedule
- Query a user's busy- and free-time information
- Search the schedule database (on the local computer or on a specified server) for free-time periods that are common to a specified list of people

Composite-text services in NOS

Using composite-text services, you can work with in-memory rich text objects and convert rich text objects into plain text objects. These services include:

- CompoundText
- ConvertItem
- EnumComposite

Folder services in NOS

Just as you access documents in a view, you can access documents in a folder. In fact, a folder is a type of view note. You can create and access private and shared database folders. However, you cannot create folders that are "Personal on first use," nor can you create private folders that are stored in the desktop file. In the Notes user interface, newly created private folders are generally stored in a specified database. However, if the user does not have permission in the database ACL to create private folders, the folder is created in the user's desktop file. In a C API program, you cannot create a folder if the user does not have permission in the database ACL to create private folders. Also, a C API program cannot access private folders that are stored in the desktop file.

To access a private folder that is stored in the database, use NIFFindPrivateDesignNote and set the class parameter to NOTE_CLASS_VIEW. To access a shared folder, use NIFFindView. Then use NIFOpenCollection and NIFReadEntries to obtain the documents in the folder. For more information about these and other view-related functions, see the latest Lotus C API Toolkit.

These services:

- Create a new folder
- Add or delete documents in a folder
- Move a folder under a parent folder
- Rename a folder
- Determine the number of entries in the index of a specified folder
- Delete a folder

Full-text search services in NOS

Full-text search services, which all have the prefix FT, create full-text indexes of local databases and search for documents that match a given query specification. These services are extensible through the Extension Manager. These services:

- Create a new full-text index for a local database.

  Note that the C API does not support full-text indexing of a remote database.
- Determine the last time a database was full-text indexed
- Examine the result of a full-text search
- Delete a full-text index of a database

Log services in NOS

The log file (LOG.NSF) is a special database that is automatically created when you set up Notes/Domino. The log file records information about all types of Notes/Domino activities.

On a Domino server, the log's ACL gives the server Manager access, by default. Immediately after the log is created, the server administrator should edit the ACL to give him/herself Manager access. Reader is the default access. The administrator maintains the ACL and may change the default access or add additional managers.

On a workstation, the person whose ID is active during Notes setup gets Manager access in the log file ACL.

Look at the log to check:

- Database replication
- Disk space
- Available memory
- Phone communications

Programs can use the Log services to add entries to the log. These services:

- Create a new log entry
- Add items — for example, text, date/time, number — to a log entry
- Write the log entry to disk
- Log an add-in event in the log file
- De-allocate a log entry

Mail services in NOS

Mail services handle items — for example, header fields and attachments — within mail messages. These services are extensible through the Extension Manager. These services:

- Create a mail message
- Add items — such as, a header, recipients, body text, attachments, and so on — to the message
- Add content to one or more message items
- Copy one or more message items to a file
- Determine information — for example, author, recipients, message size, date/time — about the message
- Determine message status — that is, sent, unsent, or returned
- Send a delivery report to the sender
- Close or delete a message

Name services in NOS

Name services locate Domino Directories and local address books and retrieve information from them. These services are extensible through the Extension Manager. These services:

- Get the list of address books in use on the local computer or get the Domino Directories on a server
- Look up names in the address books or Domino Directory
- Obtain the latest modified time/date of all the address books in the process's list

Notes Index Facility services in NOS

The Notes Index Facility (NIF) is a set of functions that handle requests to update view indexes. NIF updates the view to display the most up-to-date view information and to reflect changes made by users. When a user switches to a different view, NIF immediately updates the view so that the user sees the most recent changes. In addition, NIF updates full-text indexes, opens and closes view collections, and locates index entries.

Notes Index Facility services, which all have the prefix NIF, create and manage indexes of documents in a Notes database. Using indexes, users and servers can quickly process and/or view related documents. These services are extensible through the Extension Manager:

- Open a collection of notes
- Scan a collection and return information about the entries in the collection
- View detailed information about the collection — for example, the total number of documents and the total size
- Use various search criteria to search through a collection for specific notes
- Find the note ID of a form, view, shared folder, macro, or field note
- Locate the specified note ID in the collection and update the collection position
- Update an open collection
- Close a collection

View architecture

Views display documents in an order that is determined by certain criteria, such as creation date or subject. In a database, a view is built as a special item: the view note. Every view has a view note that stores all of the information necessary to build the view. The following table describes this information.

Field in note	Description
$Title	The name of the view.
$Formula	The selection formula that defines the documents to include in the view.
$Formula Class	The class type — for example, View Note — of the selected documents.
$Collation	The formula that defines how to sort the documents in the view.
$Collection	The appearance of the view columns — for example, font size. This information is created if it does not already exist when the view is opened.

In addition, the view note includes a collection that stores three indexes: an index sorted by unique document ID (UNID), an index of parent-child documents — that is, documents and the response documents associated with them — and a collation index that describes how to sort the view. Before users, servers, or applications can access a view, the server must build the view. This requires building the indexes in the collection, which are then stored in a B-tree structure to allow quick access.

Determining which views to update

To determine if a view needs to be updated, NIF compares two times.
The DataModifiedTime represents the last time data in the database was modified, and the ModifiedTime represents the last time NIF started working on the view. If ModifiedTime is older than DataModifiedTime,
NIF refreshes the view; if ModifiedTime is as recent or more recent, NIF processes the next view in the database.

Opening a view

When a view is opened, a call is made to NIFOpenCollection. Depending
on a flag passed with this call, the function checks if the view is up to date.
If the view is out of date, NIFOpenCollection calls NIFUpdateCollection, which forces a view update. NIFOpenCollection is called frequently, and the flag to check whether the view is up to date is usually passed. When the flag is not passed — for example, when the view refresh frequency is set to Manual — NIFUpdateCollection is not called and the view collection is not refreshed automatically.

NIF pool

The memory pool allotted to NIF is divided into 384 subpools that control the maximum amount of memory allowed for views.

Prior to Domino Release 4.5.3, the server setting “Optimize for speed” caused the server to search the NIF pool for a contiguous block of memory large enough to hold the necessary information. While fast, this process caused the fragmentation of the NIF pool and led to reaching the 24MB limit faster, since smaller unused blocks were ignored even if they could be combined. After Domino Release 4.5.3, the server setting “Optimize for space” causes the server to clean up the NIF memory pool as soon as a subpool reaches 75 percent of its maximum value.

Registration services in NOS

Notes registration services, which all have the prefix Reg, register new Notes clients and Domino servers and manage Notes IDs. These services:

- Register a Notes client or Domino server
- Create a new certifier ID
- Find the Notes ID of a certifier, server, or user
- Recertify an ID
- Cross-certify an ID
- Obtain information about an ID file

Subform services in NOS

Forms are notes of class NOTE_CLASS_FORM. Subforms, which are also forms of class NOTES_CLASS_FORM, may be inserted into a form. A form note contains a $TITLE item, an $INFO item, and one or more $Body items. A subform note contains a $TITLE item, an $INFO item, a $Flags item, and one or more $Body items.

These services:

- Insert a subform into a form or subform design note
- Remove a subform from a form or subform design note

Other NOS services

The C API provides access to these additional services:

- Access control list services
- Add-in services
- Alarm services
- Distinguished name services
- Event services
- Extension Manager services
- ID Table services
- Message queue services
- Text list services
- Time services
- Security services

Access control list services in NOS

All Domino databases contain an access control list (ACL). Use the access control list services, which all have the prefix ACL, to access and modify the ACL for a database. Use the function NSFDbReadACL to access and modify the ACL. If you use the function NSFDbCreate to create a database, you must use the function ACLCreate to create an ACL for the new database. The handle that by NSFDbReadACL or ACLCreate return is to an in-memory copy of an ACL. Use NSFDbStoreACL to store the in-memory copy of the ACL in the database.

These services:

- Create an ACL
- Add, update, and delete entries in the ACL
- Assign privileges to names in the ACL
- Determine the administration server for the ACL and change it

Add-in services in NOS

Add-in services are used by a Domino add-in program to post status information to the server console, to format and add messages to the log file, and to check if the add-in program has terminated. Add-in services:

- Create a new status line for an add-in server task and return a descriptor handle
- Delete an add-in server task status line
- Format and write error messages to the log file
- Get the module handle and default status line for the add-in task
- Set the string for the default status line of an add-in server task
- Check for termination condition

Alarm services in NOS

Using alarm services, you can work with alarms. These services:

- Register interest in receiving alarms
- Reread location for the new user
- Indicate alarm has been processed
- Refresh the alarms list
- De-register interest in alarms

Distinguished name services in NOS

Distinguished name services, which all have the prefix DN, process addresses that are made up of distinguished name fields — for example, "CN=...".

To specify a hierarchical user name in a call to an ACL function, use the fully distinguished name. A fully distinguished name is in canonical format — that is, it contains all possible naming components. For example, Mary Lee/Sales/Acme is an abbreviated name, while CN=Mary Lee/OU=Sales/O=Acme is a fully distinguished name.

These services:

- Abbreviate a distinguished name
- Convert an abbreviated distinguished name to canonical format
- Parse a distinguished name into standard components

Event services in NOS

User-defined events allow one add-in server task to notify another task that something has happened. These services:

- Create an event queue
- Place an event in an event queue
- Discontinue notification of particular events
- Retrieve the user name or database name associated with an event
- Specify the type and severity of events to process
- Remove an event from an event queue
- Delete an event queue

Extension Manager services in NOS

Extension Manager services, which all have the prefix EM, hook a well-specified set of operations — for example, opening or closing a database — with user-written callback functions.

Using the Extension Manager, an executable program library in an application can register a callback routine. The callback routine will be called before, after, or before *and* after Notes/Domino performs specified internal operations.

You build an Extension Manager application as an executable program library — for example, as a dynamic link library (DLL) for Windows and OS/2 or shared object libraries for Macintosh and UNIX operating systems. After building the application, you identify the executable program library by editing the NOTES.INI file to include the ExtMgr_Addins setting. On the Macintosh, you modify the Notes preferences file, which is in the Preferences folder within the System Folder.

You can install Extension Manager hooks on either a Notes client workstation or a Domino server. Extensions are registered on a per-process basis. After an extension is registered, all threads in a process invoke that extension. If desired, individual extensions can obtain and use a recursion ID to prevent themselves from being called recursively — for example, an extension to NSFDbOpen would want to use a recursion ID if it needs to call NSFDbOpen itself as part of its own operation. For more information about recursion IDs, see the Lotus C API Toolkit.

The following services are extensible through Extension Manager:

- Calendar and scheduling services
- Full-text search services
- Name
- Notes Index Facility
- Notes Storage Facility

ID table services in NOS

ID table services manage compressed lists of note IDs. These services:

- Copy an ID table
- Insert a note ID in an ID table
- Delete some or all IDs from an ID table
- Determine if two ID tables contain the same contents
- Determine if an ID is present in an ID table

Message queue services in NOS

Using the message queue services, which all have the prefix MQ, you can send information from one Notes/Domino application to another. These services:

- Create a message queue
- Open a message queue
- Add a message to the specified message queue
- Read and remove a message from a message queue
- Get the number of messages in the message queue
- Test if the specified message queue is in a QUIT state
- Call action routine for each message in a message queue
- Close a message queue

Text list services in NOS

Text list services, which all begin with the prefix List, manage in-memory text lists, which are a common type of item in a database note. These services:

- Create a text list
- Add a string to a text list
- Duplicate a text list
- Determine the number of entries in a text list
- Determine the size of a text list
- Remove one or more entries from a text list

Time services in NOS

Time services, which all begin with the prefix Time, manage time-related environment variables. These services:

- Create, set, or clear a TimeDate value
- Compare two TimeDate values and determine the difference between them
- Increment a TimeDate value
- Extract the date from a TimeDate value
- Convert a TimeDate value to local time

Security services in NOS

Using security services, which all have the prefix SECKFM, you can manage users, public keys, and certificate contexts. These services:

- Change the password in the specified ID file
- Return the encoded password, given a non-encoded password
- Free a certifier context

- Get the certifier context
- Get the public key for a user
- Get the current user name
- Set the certificate expiration date
- Get current user name and/or license ID

Chapter 3
Notes Storage Facility

The Notes Storage Facility (NSF), one of the largest and most complicated parts of Notes Object Services (NOS), is a library of C functions that implement the most basic database-creation and database-management operations.

The Notes Storage Facility

The Notes Storage Facility (NSF) is a key component of Notes Object Services (NOS). Notes client and Domino server programs use NSF to create and manage Notes databases. When a program calls on NSF to access a local database — that is, one that resides on the same client or server computer on which the program is running — NSF calls file-system services of the local operating system to create and manage the Notes database file directly. When a program calls on NSF to access a remote database — that is, one that resides on a Domino server that is accessible over the network — NSF uses the Notes remote procedure call (NRPC) service within NOS to ask the remote Domino server to perform the NSF operation on its behalf.

Only NSF functions can create a Notes database from scratch and create, access, modify, and delete individual objects — such as, the database header, notes, and so on — that are in a Notes database. In other words, no functions outside of NSF can create a Notes database from scratch or directly access individual objects within a Notes database. Programs that need to perform these tasks must call NSF functions to do so. For example, Update task, which rebuilds all views in a database, uses the Notes Index Facility (NIF) in NOS to create the list of documents that appear in each view. NIF, in turn, uses NSF functions to read the selection formula from each view, to search the database for documents that match the selection formula, and to build and save in the database the "collection" information that comprises the index.

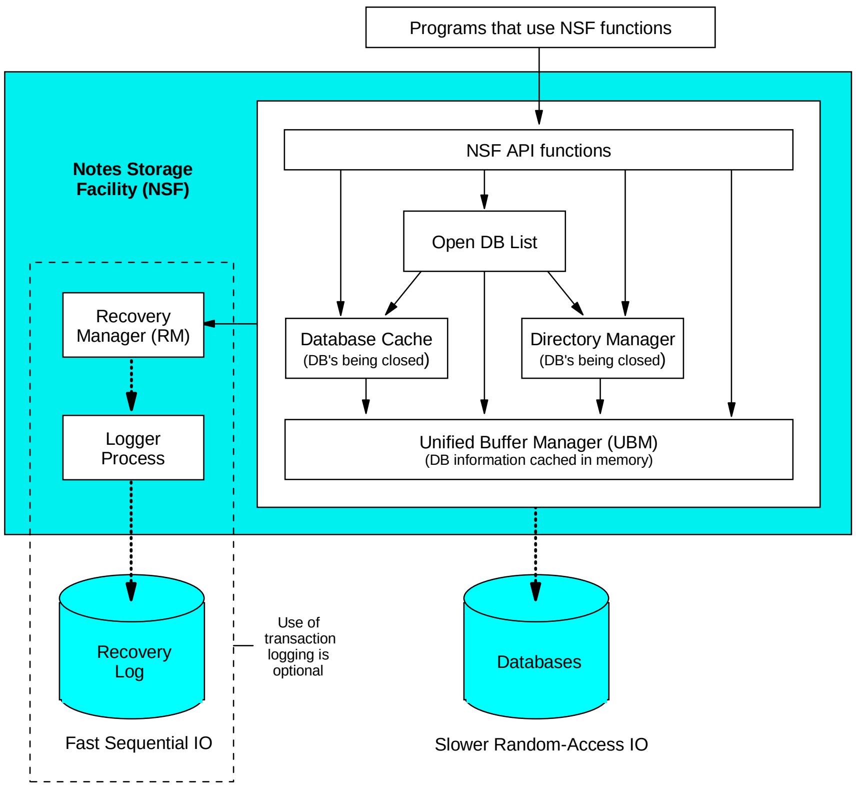
This document presents a high-level description of the software components of NSF and the database structures that NSF maintains. For more detailed information about NSF functions, see the Lotus C API Toolkit for Domino and Notes.

Figure of NSF

The following figure shows the primary software components of NSF, which are the:

- NSF API functions
- Unified Buffer Manager
- Open Database list
- Database Cache
- Directory Manager
- Recovery Manager and the Logger

While the primary components are large, complicated pieces of software with many subcomponents, this discussion does not delve into how they, in turn, are constructed. Furthermore, to simplify the figure, many secondary NSF components are neither shown nor discussed, nor is the NOS portability layer shown, although all NSF components use it to access disk files, allocate memory, synchronize access to shared data, and so on.


Programs that use NSF functions
Notes Storage Facility (NSF)
NSF API functions
Open DB List
Recovery Manager (RM)
Database Cache (DB's being closed)
Directory Manager (DB's being closed)
Logger Process
Unified Buffer Manager (UBM) (DB information cached in memory)
Recovery Log
Use of transaction logging is optional
Databases
Fast Sequential IO
Slower Random-Access IO

NSF API functions

The NSF API functions encapsulate the rest of NSF, meaning all programs that create and/or use Notes databases do so by calling these API functions. Many of the API functions are available to third-party developers who use the Lotus C API Toolkit for Domino and Notes to write C/C++ programs. For detailed information about NSF functions, see the Lotus C API Toolkit.

Unified Buffer Manager

The Unified Buffer Manager (UBM) caches information about databases that NSF is accessing. In general, about one-third of a machine's memory is assigned to the UBM. On a large server the UBM memory cache often holds tens of thousands of data structures belonging to hundreds, and even thousands, of open databases.

The UBM is largely responsible for how well Domino performs and scales. First, the UBM implements rules to guarantee maximum use of its large memory cache. These rules keep memory fragmentation to a minimum and balance memory usage across all NSF activities so that no individual database or NSF process gains more than its fair share of memory relative to other databases and processes. Second, the UBM implements rules to increase the likelihood that database information needed by NSF is already available in its cache and does not have to be read from disk. Therefore, the UBM serves database information to the rest of NSF with an almost negligible contribution to overall server overhead. In addition, enabling optional transaction logging reduces the need to flush cached information from the UBM to database files simply for the sake of maintaining database integrity, and this can also improve the performance of the server.

Open Database list

The Open Database List tracks all databases in active use on a client or server computer. NSF adds an entry to the Open Database List whenever a database is first opened. Each entry contains, among other things, a list of users that have opened the database, and each entry in the database's users list, in turn, contains information about the user's authentication state and access rights, a flag indicating if the user is a person or a server, and other user-specific information. When a program opens a database, NSF returns a DBHANDLE to be used on all subsequent accesses to the database. A DBHANDLE identifies an entry in a database's users list, which in turn leads to the database's entry in the Open Database List. This design facilitates enforcement of per-user access rights — each database operation is done in the context of the user who opened the database, and so an operation that one user may be permitted to do may not be permitted for another user.

When a user closes a database, the user's entry in the database's users list is removed. On client computers, when the last user of a database closes it, the database is closed immediately — any unwritten database information still in memory is flushed out to the database file, the database file is closed, and the database's entry in the Open Database list is removed. On server computers, when the last user of a database closes it, the Database Cache postpones closing the database for a while. NSF removes the database's entry on the Open Database list and moves it to the Database Cache list. This optimization improves performance by taking advantage of the fact that many server databases are often reopened shortly after being closed.

Database Cache

The Database Cache is responsible for closing databases after their last user is done with them. Because many databases are reopened shortly after being closed, the database cache keeps a database open for a period of time before actually closing it. When a user tries to open a database, NSF first checks the database cache and, if an entry is there for the database, then NSF removes the entry from the cache and moves it back to the Open Database list. This technique saves the time it takes to flush database information out to a file, free the memory that held the database information, close the file, reopen the file, reallocate memory to hold database information, and read the information from the file back into memory.

Client computers do not use the Database Cache. A client database is fully closed when its last user closes it. This is because the aging and closing of databases listed in the Database Cache is done by two built-in server tasks, which are not available on a client computer.

The Database Cache is enabled by default on server computers.

Directory Manager

The Directory Manager keeps critical information in memory about every database in the Domino data directory and its subdirectories. When Domino first starts, Directory Manager information is compiled and then kept current as databases are created, deleted, opened, modified, and closed. Like the UBM, the Directory Manager boosts server performance by keeping database information readily available in memory. Therefore, there is no need to incur the high overhead of opening and reading databases just to get a few items of information from them. For example, tracking the last time each database was modified helps a program such as the Replicator to determine quickly which databases need replication — those that have been modified since their last replication.

Recovery Manager and the Logger

The Recovery Manager (RM) and the Logger are optional NSF services that implement transaction logging, a feature designed to improve database integrity and the performance of database I/O operations. The Logger records information passed to it by the Recovery Manager. The Recovery Manager:

- Is the sole NSF component that interfaces to the Logger
- Writes transaction-undo records to the Logger
- Writes database-recovery records to the Logger
- Reconstructs databases after a server has crashed by replaying database-recovery records from the log and undoing partially-completed database transactions using transaction-undo from the log

Transaction logging uses write-ahead-logging (WAL) to log a sequential record of every modification to every database before the changes are made to the database itself. Unlike database update operations, which cache database modifications in memory and then later write them to database files, logging operations are synchronous, meaning they guarantee that logged data is immediately recorded in the log file. Then if the server crashes, the Recovery Manager uses the log when the server restarts to reconstruct all databases with 100 percent data integrity. Even databases for which some cached data was written to the transaction log but never recorded in database files will be recovered. All completed API operations will be reflected in logged databases after recovery is complete.

While the primary job of transaction logging is to guarantee data integrity, using RM and the Logger also can improve the performance of database recovery. Should a server crash, the Recovery Manager can rapidly “replay” log records and recover all data. It, therefore, is faster and more reliable to use a log to recover data than to use the Fixup task, which first examines the content of databases for inconsistencies and then attempts to resolve the inconsistencies, a process that may result in loss of data.

Database structures

Each database always contains a header, and it may also contain notes and/or replication-history information. In addition, a database always contains other internal structures that track notes and replication-history information in the database.

A shared database residing on a Domino server typically contains all four types of components: a database header, notes, replication-history information, and internal structures. The bulk of such database files, especially of large ones, is typically dedicated to the storage of notes. Each note has its own header structure, a collection of fields (also called items) and, optionally, a list of “response” notes.

The organization of a Notes database is different from that of a relational database, which contains tables of same-type “records,” or sets of fields, and is optimized for rapid access by table-oriented operations. A Notes database can contain any number of notes, and a note can contain any number and type of fields. In addition, there is information in the database header and in each note header that supports simultaneous access and replication between databases.

Components of the database header

Each Notes database always has a header that contains:

- Major and minor version numbers
- The database class
- An information buffer
- The database ID (DBID)
- A meaningful database instance ID (DBIID), if transaction logging is enabled
- Replication settings, which include the database replica ID

Header information can be accessed programmatically through the Notes API; consult the Lotus C API Toolkit for Domino and Notes for details.

Major and minor version numbers

Major and minor version numbers indicate which version of Notes was used to create the database file. Only Notes Releases 1 and 2 used the minor version number. The major version number indicates the level of on-disk structures (ODS) that can appear within the database. It is an ever-growing number which is incremented whenever a new version of Notes can store ODS that could not be created by any previous versions of Notes. Whenever a specific version of Notes creates a database, it stamps the header with a major version number so that other versions of Notes that may read the file know what types of ODS can be in it. The major version number is actually incremented many times during the development of a major release of Notes. For example, Notes Release 4 creates databases stamped with a major version number of 20, and because of the series of internal versions that were created as the code base evolved from Release 4 to Release 5, Notes Release 5 creates databases stamped with major version number 41. The following table lists the major version numbers for all releases of Notes.

Version of Notes	Major version number
Prehistory	< 16
Release 1 and Release 2	16
Release 3	17
Release 4	20
Release 5	41

Version numbers are used by Notes to control version-to-version software compatibility. A newer version of Notes can always work with a database created by an older version of Notes, but not vice versa. For example, a Notes Release 4 can open and access *local* databases stamped with a major version of 20 or lower but will never open and access databases stamped with a major version greater than 20.

Note It is possible for Notes Release 4 to open a *remote* database that has a major version greater than 20 — for example, a database stamped with a version of 41 by Notes Release 5 — provided that Notes Release 5 is running on the remote server and can respond to NRPCs to open and access the remote database on behalf of the Release 4 requester.

Database class

The class of a database determines the default size of some internal database structures; therefore, assigning the appropriate class to a database can positively affect its operational performance and size. Most databases use the recommended default database class setting of DBCLASS_NOTEFILE. Special databases — for example, the database that stores single copies of messages sent to multiple users — use other, predefined class settings designed specifically to take advantage of the specialized ways in which they are accessed. Database class is set when the database is first created, and it cannot change after that.

Database information buffer

Every database header has a 128-byte information buffer that contains the:

- Database title
- Database categories
- Template name (if it's a template)
- Name of the database's "inherit from" design template (if applicable)

The information in this buffer is visible through the Basics and Design tabs of the Database Properties box. In addition, several API functions provide access to the database information buffer.

Database ID

Every database has a database ID (DBID), which is a TIMEDATE value that indicates either the time when the database was first created or the time when the Fixup task last ran on it. Fixup changes a DBID so that the Replicator task knows that data has been removed from the database and needs to be restored. Replication, which normally replicates only information that has changed since the last replication, will notice that the DBID has changed and will do a full replication instead, which restores the missing data.

Database instance ID

Every database has a database instance ID (DBIID) that is meaningful only if the database resides on a server on which transaction logging is enabled. A unique DBIID is initially generated for a database when the database is first created, and it is changed later whenever the database is compacted. Because compacting temporarily disables transaction logging for a database (in order to prevent a large amount of logging data from being spewed out to the recovery log due to the large amount of I/O done during database compacting), old recovery data in the log should then never be applied to the compacted database. To indicate this, compacting changes the database's DBIID.

The Recovery Manager will not replay recovery information from a log into a database unless the DBIID value stamped on the recovery records matches the database's DBIID. It is recommended that you back up a database immediately after compacting it.

Database replication settings

Database replication settings — for example, "Remove documents not modified in the last *x* days" and "Do not send deletions made in this replica to other replicas" — are stored in a DBREPLICAINFO structure in the database header. Using the Notes API, you can access the DBPEPLICAINFO structure.

One member of the structure, named ID, is a TIMEDATE value that serves as the database's replica ID. When a database is first created from scratch, its replica ID stores the database creation time. When you create a replica of the database, the replica assumes the replica ID of the source database. Replication checks if two databases have the same replica ID — indicating that they are replicas of each other — before replicating their contents. Once set, a database's replica ID never changes.

Database replication history

Replication logs within a database store a history of all of the database's replication activity. The history is used to optimize replication, so that replication only looks for information modified since the last time the database was replicated to that destination. For each replication, it creates a REPLHIST_SUMMARY structure (time, direction, and so on) as well as the name of the database that was replicated and the server on which that database resides. All of this information can be accessed programmatically through the Notes API. Not all databases have replication history information — if a database has never been replicated, it does not have a replication history.

Database notes

All user, design, and administrative data pertaining to a database is stored in the database in notes. NSF has a common set of routines for creating and working with notes, regardless of the type of data they contain.

Each note has a header, zero or more items that store note data, and a list of zero or more notes that are "children" of the note, which is the "parent." This parent-child information is essential for implementing the topic-response relationship between notes.

After it is created, a note can be removed from a database in one of two ways — fully removed or marked as a deletion stub. A deletion stub can be further categorized as hard deleted or soft deleted. (The soft deletion became available in Notes Release 5.) A hard-deleted stub contains only enough information to indicate that a note has been deleted and is kept in the database so that replication of deletion information occurs properly between databases — there is no way to recover any information originally attached to a hard-deleted note. Like a hard-deleted stub, a soft-deleted stub is marked as deleted for replication purposes, but it also retains its data so that the data can be recovered later, if desired.

NSF itself does not require that a database contain any notes at all, although programs that use NSF to create and maintain databases sometimes require a minimum set of notes so that their own specific operations work. For example, the Notes Client program requires that a database have an ACL note and at least one view note.

There are two major categories of notes: data notes and non-data notes. Data notes, which are also called documents, store the content of a database. Non-data notes are further categorized into administration notes (ACL and replication-history notes), which control how a database can be accessed and record how it has been replicated, and design elements (forms, views, and notes, for example), which control how data notes are created and viewed, how notes are indexed, and so on.

A note can also be classified as single- or multiple-instance. A single-instance note — for example, a database ACL note — can occur only one time in a database. A multiple-instance note — for example, a view note — can occur many times.

Database notes can be arranged in a hierarchy — that is, a note can be a parent of other notes, which in turn can be parents of other notes, and so on. NSF does not limit the depth of parent-child relationships, although some programs that use NSF enforce a limit of 32 levels. Notes in parent-child relationship form the basis for databases that implement threaded discussions — with main-topic notes, response notes, and response-to-response notes — and also for the handling of replication conflict errors, in which the "loser" document becomes a child of the "winner" document.

Table of note types

The following table summarizes the types of notes that can be in a database.

Note type	Description	Instance
ACL	Specifies who may access the database and in what ways.	Single
Design Collection	Specifies a special view that indexes all form, view, filter, field, replformula, and help-index notes in a database.	Single
Document	Documents in a database, for example a mail message or To do document. Different types of data notes can be in the same database, and there can be as many instances of each type of data note as you want. For example, a database could have 1000 mail notes, 0 appointment notes, and 50 ToDo notes.	Multiple
Field	Stores the definition of a field that can be shared between forms and subforms.	Multiple
Filter	Represents agents in a database. Outlines, agent data, script libraries, and database scripts are also represented as filter notes with a flag that identifies the filter type.	Multiple
Form	Provides the structure for creating, editing, and viewing a data note. Data notes create their items from the form note's specifications. Subforms, pages, framesets, image resources, Java resources, and shared actions are also represented as form notes with a flag that identifies the form type. All shared actions in a database are represented by a single form note.	Multiple
Icon	Specifies the database icon, database title, and database category.	Single
Info	Specifies the Help About This Database document.	Single
Help	Specifies Help Using This Database document.	Single
Help Index	Specifies the index of Help information.	Single
ReplFormula	Specifies a replication formula for the database.	Multiple
View	Using a formula you specify, selects sets of data notes for viewing. Views can include a hierarchy of parents and response documents and columns that display summary items. Columns can include categories of information to group particular documents together in collapsible sections and can be sorted in ascending or descending order. Folders and navigators are also represented as view notes with a flag that identifies the view type.	Multiple

Identifiers for notes

Every note has two identifiers: an originator ID and a note ID.

The originator ID

The originator ID (OID) is a 28-byte identifier that contains a universal ID (UNID) that uniquely identifies an instance of a note and contains additional information used to resolve replication conflicts. An OID contains these three elements:

- A 16-byte universal ID (UNID).
- A 2-byte sequence number that indicates the number of times the note has been updated. This is used by replication conflict-detection.
- An 8-byte time stamp that indicates when the note was last updated. This is used by replication conflict-detection.

The universal ID (UNID) in a note's OID is the same across all replicas of a database, meaning that if a note has a specific UNID in a database and a replica copy of the database is made, then the copy of the note in the replica will preserve the original note's UNID; replication keys off of identical UNID values to determine which notes are new (no matching UNIDs between replicas), which are deleted, and which are the same between the replicas.

Except for notes in replica copies of a database, no two notes in any databases anywhere in the world should ever have the same UNID. A UNID is a structure which has two 8-byte elements, called "File" and "Note" respectively. Different formulas have been used over time to generate the values that go into these UNID elements for a newly created note. Originally, the File element of a note's UNID consisted of the 8-byte DBID of the note's database, and the "Note" element of a note's UNID consisted of the 8-byte time stamp of when the note was created. Even though the two UNID elements retain their original names for historical reasons, the current formula calculates an 8-byte random value for the File element; the Note element is still the 8-byte note-creation time stamp.

Each database contains a UNID "map," a list of all the UNIDs of all the notes within it and, associated with each UNID in the list, the note's note ID, which can be mapped to the location of the note within the database. Many NSF functions are passed to a UNID to identify the note on which the function should operate. These functions use the UNID map to locate the note within the database.

The note ID

In addition to an OID, every note is assigned a 4-byte note ID when it is first created. A note ID is unique within a database but not across replicas of the database, meaning the same note in two replicas can (and probably do) have different note IDs, even though they have identical UNIDs.

It is much faster and more convenient for programs to use a note's 4-byte note ID to identify it rather than its 28-byte ID or its 16-byte UNID. Many NSF functions return note IDs and/or expect note IDs to be passed to them as input arguments. Lists of note IDs can also be kept in a compressed form called an IDTABLE (note ID table), which is often used within Notes — for example, to specify a list of unread documents or the documents that comprise a view.

Layout of a note

Notes in a database are composed of a logical sequence of structures:

- Note header
- Item descriptors
- Summary-item values
- Response entries or overhead
- Non-summary-item values

Note header

The note header is a structure that contains, among other things, the note's originator ID (OID), which includes its universal ID (UNID); the note ID; the note's parent note (if any, and only if note hierarchy is not disabled in the database); a count of the number of items in the note; an indication about the location of non-summary items within the database; and information about the number and location of the note's children (if any). While a note's header is not directly accessible, the Lotus C API Toolkit for Domino and Notes does let you get and set some of its individual elements.

Item descriptors

Item descriptions are stored in an array of fixed-size structures, each of which describes one note item. Each structure has information describing the item name; the item type — for example, the format of the item's value; the size of the item's value; and some flags, one of which indicates if the item is a summary or non-summary item. Summary-item values are always stored with the note header and item descriptors, while non-summary-item values may be elsewhere within the database.

Summary item values

Summary item values are the values (text strings, numbers, time/date stamps, and so on) that are associated with items that the item-descriptor array flags as summary items. For example, if the fourth and ninth items are flagged as summary items and if their respective lengths are 36 and 94 bytes, then the number of summary-item bytes is 130; the first 34 will contain the value associated with the fourth item, which is followed by 94 bytes containing the value associated with the ninth item. All other items are considered non-summary items.

Response entries or overhead

The response-entries list specifies the children of a note. Provided that response information is not disabled for the database, information in the note header indicates if there are response entries and, if so, where they are located. Response entries are located after the note header, item descriptors, and summary-item values; or an indicator called the response-overhead structure appears in that location in the note and indicates the location of the response-entry list within the database.

Non-summary item values

Non-summary item values are associated with all the items flagged as non-summary items in the note's item descriptor list. Like summary values, non-summary item values are variable-length structures that are placed one after the other. The length of each structure is specified in the corresponding item-descriptor.

Physical storage of notes in a database

For performance reasons, the individual parts of a note are often physically stored in different sections of a Notes database. For example, a view shows information that summarizes the notes that it indexes. Called summary data, this information comes directly from items in the notes or is calculated using items in the notes. Performance improves when note summary items are stored near each other. If summary items and non-summary items were mixed together, constructing a view would require a high level of I/O.

NSF stores a note header, the item descriptors, and the values of those items whose descriptors are marked as summary items in internal structures called summary buckets. The class of a database determines the size of the summary buckets, which are typically big enough to hold the summary information for many notes. If a note has small response entries and/or non-summary-item values, they can be stored with the summary parts of the note in the summary bucket; otherwise, the entries and values are stored in other parts of the database file.

Because a note's summary data must fit within a summary bucket, NSF limits individual summary item values to be no more than 32K. Also, the total summary data of a note — that is, its header, list of items, and all summary-item values — cannot exceed 64K , minus a little overhead needed to manage the summary bucket.

A note's total non-summary data can be large. While no individual non-summary item can be large (typically limited to 64K), the total size of all non-summary items can be much bigger. In addition, NSF treats two or more rich text items that are in the same note and have the same item name as if they are parts of one large item with that name. In this manner, a mail message can have a rich text Body field, for example, that is much bigger than 64K. The same convention also applies to large HTML items.

How a program decides if an item is a summary or non-summary item

For non-data notes, a predetermined set of items are programmatically created as summary items. The same is true for items that Notes adds to data notes (documents) in order to manage them. For other document items — in particular, for user fields specified in the forms used to create documents by the application developer who designs the database — Notes uses two simple rules to decide if an item should be classified as summary or non-summary:

- Items categorized as TYPE_NOCOMPUTE (as defined in the nsfdata.h header file included in the Notes Software Development Kit) are always non-summary — for example, Compound Document (CD) items, which hold rich text and file attachments, are always non-summary items.
- Other items are classified based on their size. Normal size items are summary items, and very large items — that is, those greater than 60K — are non-summary items. The exact definition of "normal size" is built into the programs that use NSF — for example, the Notes Client — and not into NSF itself. These programs typically use a value that is large enough that simple items — such as text, short text lists, dates, and numbers — are usually classified as summary items.

Table of item types

Items store the content of a note — for example, a text field on a form or a view selection formula. Items are the basic unit of Notes data. Every note in a database can contain one or more items. Each item has a name, a type, one or more flags, a length, and a value.

The following table describes the item types that can be in a note.

Item type	**Content**
Action	Saved actions formated as a sequence of composite-data (CD) records.
AssistantInfo	Agent parameters.
CalendarFormat	Calendar view-format information.
Collation	The sorting rule for each of the collation columns of a view.
Composite	A sequence of composite-data (CD) records, specifying rich text and embedded objects.
Formula	A compiled Notes @function or @command.
Highlights	Full-text (FT) highlights (or "hits") in a note.
HTML	LMBCS-encoded HTML.
Icon	Bitmap of the database icon.
InvalidorUnknown	None of the other, valid item types listed in this table.
LSObject	LotusScript "object" code.
NoteLinkList	An array of NOTELINK structures, each of which identifies a note in another database by specifying the database's DBID and the UNID of the note within the database.
Number	A single floating-point number.
NumberRange	A list of floating-point numbers.
Object	A BLOB (binary large object). Specific types of objects — such as, file attachments — can be made using the Lotus C API Toolkit for Domino and Notes.
Query	Saved query CD records.
SchedList	List of busy times.
Seal	The encrypted bulk key used to encrypt the contents of the associated sealdata item. Used only in Notes Release 1.
SealData	An encrypted item
SealList	The bulk key, encrypted for each member of a list, used to encrypt the contents of an associated sealdata item.
Signature	An electronic signature affixed to the note.

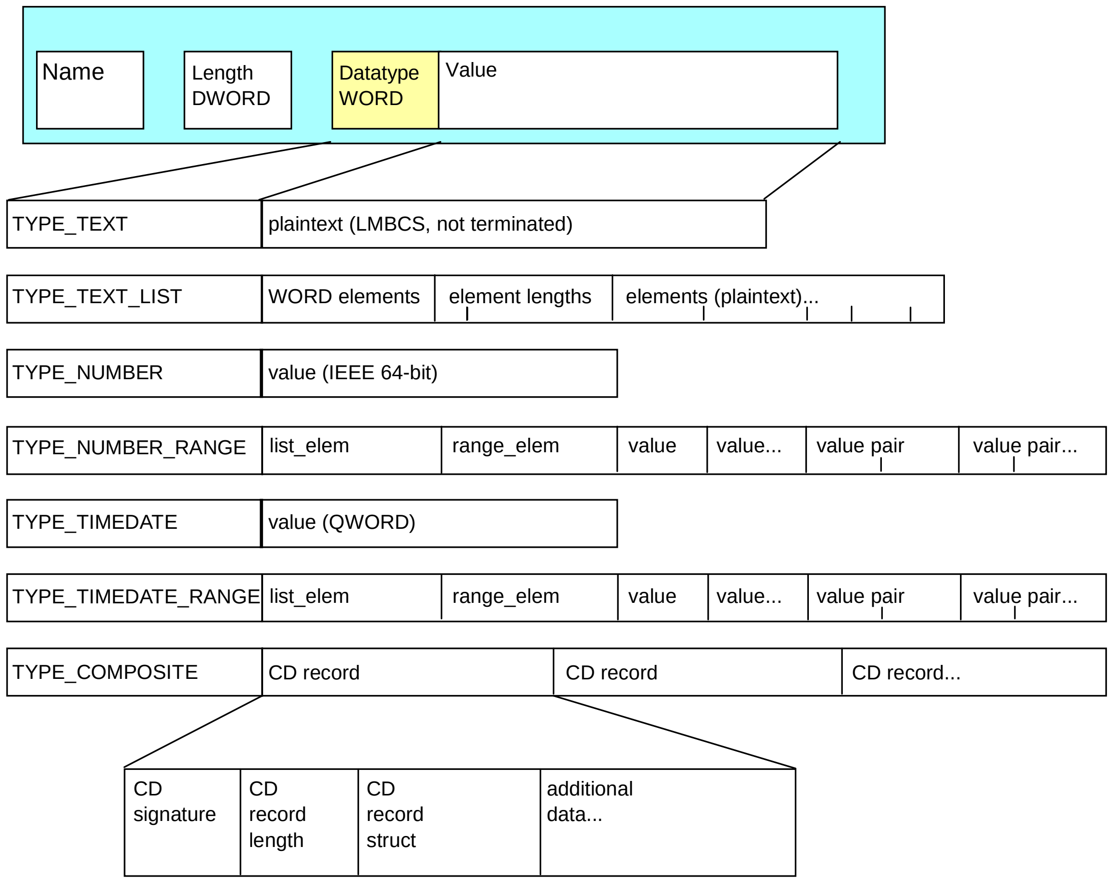
continued

Item type	Content
Text	Plain text.
TextList	A list of plain text. The exact number depends on whether the list contains many short strings or a few long strings.
Time	An 8-byte value that stores the date and time. Either the date or time can be omitted. The time is recorded in milliseconds, and the date includes the time zone.
TimeRange	A list of dates and times.
UserData	A user-formated BLOB (binary large object)
UserID	Authors field. Used in Notes Release 2.
ViewFormat	Tabular view format information.
ViewMapDataset	Navigator format information.
ViewMapLayout	Navigator format information.
WorksheetData	Information used only by the Chronicle product.

Layout of an item

Each item value in a note starts with a word containing the item's type — for example, TYPE_TEXT or TYPE_NUMBER — as defined in the API header file called nsfdata.h. The type word is followed by one or more structures packed together one after the other. All the individual elements of an item's value are stored in canonical format in a database file. The API has functions that you can use to read and write individual items and, depending on the item's type, convert the elements to/from the host format. Comments in the nsfdata.h header file indicate which ones are converted and which are not. If NSF does not convert the elements of an item's value for you, then you must call the API's on-disk structure (ODS) functions to do so.

The following figure illustrates the elements of the most commonly used item types.



Chapter 4
Server Tasks

This chapter describes the Domino server tasks, which collectively implement all the functions that the Domino server performs.

Types of server tasks

Server tasks are grouped into five categories:

- Tasks that maintain Notes applications
- Tasks that monitor server and administration activities
- Tasks that manage mail, calendars, and scheduling
- Tasks that manage protocols
- Tasks that monitor server activity

Tasks that maintain Notes applications

Agent Manager

Agents are programs that are available as part of an application and are stored in a Notes database. An agent can be run on demand by a user when the application runs on a Notes client machine. An agent can also be run on a scheduled basis by means of the Agent Manager, which is a program that can be operated both as a server task or as a program running in the background on a Notes client.

Cataloger

The Cataloger maintains the domain catalog (CATALOG.NSF), a central list of all databases in an organization. When the Cataloger runs, all databases, except mail databases, are included and updated by default. However, a database designer can use a special property to prevent a database from being included in the catalog. By default, the Cataloger runs daily at 1 AM.

Catalogs provide useful information about databases. They contain views that list databases by category, manager, replica ID, server, and title. For each database in a view, the catalog document provides the server name; file name; replica ID; the names of servers, groups, and users that have Manager access to the database; and information from the About This Database document.

Domain Indexer

The Domain Indexer updates the full-text index of one or more databases listed in the Domain Catalog (CATALOG.NSF). By default, the Domain Indexer updates the full-text index for all databases listed in the catalog although, if desired, the set of databases can be limited to those residing on specific servers and/or to databases not specifically excluded from full-text indexing by the Indexer.

How frequently the Indexer runs varies from organization to organization. Each organization must evaluate its needs and schedule the Indexer accordingly.

Database Compactor

The Database Compactor cleans up the free space created when a database is updated, new data is added, or documents are deleted. This task is never run by default. It is an optional task you can run if you wish.

Designer

To use a consistent design for multiple databases, database designers can associate an entire database or individual database design elements with a master template. When the design of a master template changes, the Designer task updates all databases that inherit their designs from the master template. A database designer can manually synchronize a database with a master template, but most designers rely on the Designer task, which runs daily at 1 AM by default, to perform synchronization.

Directory Cataloger

A directory catalog consolidates entries for users, groups, mail-in databases, and resources from one or more Domino Directories into a single, lightweight, quick-access database.

Notes clients can store a replica of the directory catalog, which is known as the mobile directory catalog. This catalog allows Notes users who are disconnected from the network to address mail to other users. Rather than searching Domino directories on a server, type-ahead addressing searches the mobile directory catalog. Therefore, network traffic is kept to a minimum.

A directory catalog on a server enables servers to search one database for names that appear in multiple Domino Directories. The directory catalog (DIRCAT.NSF) is based on the template DIRCAT5.NTF. The Directory Cataloger summarizes and combines directory entries into a source directory catalog. Subsequently running the Directory Cataloger keeps the source directory catalog synchronized with the directories combined in it.

Scheduling of the Directory Cataloger varies from organization to organization because it depends on specific organizational needs. For example, some organizations may schedule the Directory Cataloger to run every hour, while others may schedule it on a daily basis.

Replicator

For server-to-server replication, the Replicator on one server calls another Domino server at scheduled times. To schedule replication between servers, you create Connection documents that describe when servers connect to update replicas. As users add, edit, and delete documents in a database, the replicas contain slightly different information until the next time the servers replicate. Because replication transfers only changes to a database, the network traffic, server time, and connection costs are kept to a minimum.

During scheduled replication, by default, the initiating server first pulls changes from the destination server and then pushes changes to the destination server. As an alternative, you can schedule replication so that the initiating server and destination server each pull changes or so that the initiating server only pulls changes or only pushes changes.

Update and Updall

On a server, Update and Updall can run as server tasks; on a Notes client these tasks can run in the background. The tasks use the Notes Index Facility (NIF) in NOS to update database views and folders. This can reduce delays for users who visit the views and folders at a later time because they may not have to wait for the views and folders to be updated before seeing them.

On a server, Update is loaded at startup by default and runs continually, checking its work queue for views and folders that require updating. When a view or folder change is recorded in the queue, Update waits approximately 15 minutes before updating all view indexes in the database. The waiting period ensures that the update can include any other database changes made during the 15-minute period. After updating view indexes in a database, Update then updates all databases that have full-text search indexes set for immediate or hourly updates. Upon finding a corrupted view index or full-text index, Update attempts to correct the problem by rebuilding the index. In effect, Update deletes the index and then rebuilds it.

Updall is similar to Update; however, Updall doesn’t run continually or work from a queue. On a server, Updall runs daily at 4 AM by default, but you can also run Updall as needed. By default, Updall updates every index that requires updating, but you can use optional parameters with the command to change its default behavior. Also by default, Updall discards view indexes of views that have been unused for 45 days in order to save disk space, but the database designer can specify different criteria, if desired.

Tasks that manage server and administration activities

Administration Process

The Administration Process automates many routine administrative tasks. For example, if the system administrator deletes a user, the Administration Process locates that user's name in the Domino Directory and removes it, locates and removes the user's name from ACLs, and makes any other necessary deletions for that user. The Administration Process automates these tasks:

- Name-management tasks, such as rename person, rename group, delete person, delete group, delete server name, recertify users, and store Internet certificate
- Mail-file-management tasks, such as delete mail file and move a mail file
- Server-document-management tasks, such as store CPU count, store server's platform, and place network protocol information in Server document

The Administration Process primarily interacts with the Administration Requests database (ADMIN4.NSF), which is automatically created on the first server in a domain or replicated from the registration server during additional server setup. To complete tasks, the Administration Process posts and responds to requests in the Administration Requests database. Domino servers use replicas of this database to distribute requests made on one server to other servers in the domain.

Cluster Administration Process

The Cluster Administration Process performs many housekeeping tasks associated with a cluster. For example, when you add a server to a cluster, the Cluster Administration Process starts the Cluster Database Directory Manager and the Cluster Replicator server tasks, and it adds the task names — Cldbdir and Clrepl — to the ServerTasks setting in the NOTES.INI file so that these tasks start each time you start the server. When you remove a server from a cluster, the Cluster Administration Process removes these commands from the NOTES.INI file, stops the tasks, deletes the Cluster Database Directory on that server, and cleans up records of the server in the Cluster Database Directory on other servers.

Cluster Database Directory Manager

The Cluster Database Directory Manager resides on every server in a cluster. It creates the Cluster Database Directory (CLDBDIR.NSF) and keeps it up to date with the most current database information.

The Cluster Database Directory resides on every server in a cluster. It contains a document about each database and replica in the cluster. This document contains such information as the database name, server, path, replica ID, and other replication and access information. The cluster components use this information to perform their functions, such as determining failover paths, controlling access to databases, and determining which events to replicate and where to create the replicas.

When you first add a server to a cluster, the Cluster Database Directory Manager creates the Cluster Database Directory on that server. When you add a database to a clustered server, it creates a document in the Cluster Database Directory that contains information about the new database. When you delete a database from a clustered server, it deletes this document. It also tracks the status of each database, such as databases marked "out of service" or "pending delete."

When there is a change to the Cluster Database Directory, the Cluster Replicator immediately replicates the change to the Cluster Database Directory on each server in the cluster. This ensures that each cluster member has up-to-date information about the databases in the cluster.

Cluster Replicator

The Cluster Replicator constantly synchronizes data among replicas and Cluster Database Directories in a cluster. Whenever a change occurs to a database in the cluster, it immediately pushes the change to the other replicas in the cluster. This ensures that each time users access a database, they see the most up-to-date version. The Cluster Replicator also replicates changes to private folders that are stored in a database. Each server in a cluster runs one Cluster Replicator by default, although you can run more cluster replicators to improve performance.

The Cluster Replicator looks in the Cluster Database Directory (CLDBDIR.NSF) to determine which databases have replicas on other cluster members. It stores this information in memory and uses it when replicating changes to other servers. When it detects changes in the Cluster Database Directory, it updates the information in memory — for example, adding or deleting a database or disabling replication for a database.

The Cluster Replicator task pushes changes to servers in the cluster only. The standard Replicator replicates changes to and from servers outside the cluster.

Map Generator

The Map Generator examines the domain server topology and records it in the Domino Directory (NAMES.NSF) for display by the Domino Administrator.

Tasks that manage mail, calendars, and scheduling

Calendar Connector and Schedule Manager

The calendar and scheduling features allow users to check the free time of other users, schedule meetings with them, and reserve resources, such as conference rooms and equipment. These features use the Schedule Manager (SCHED), the Calendar Connector (CALCONN), and the Free Time system, which uses a combination of the SCHED and CALCONN tasks to operate. When you install Domino on a server (any server except a directory-only server), the Schedule Manager and Calendar Connector tasks are automatically added to the ServerTasks setting in the NOTES.INI file. When you start the server for the first time, the Schedule Manager creates a Free Time database (BUSYTIME.NSF for non-clustered mail servers and CLUBUSY.NSF for clustered mail servers) and creates an entry in the database for each user who has completed a Calendar Profile and whose mail file is on that server or on one of the clustered servers. Each user can keep a personal calendar and create a Calendar Profile that identifies who is allowed to access the user’s free time information and specifies when the user is available for meetings.

When users schedule appointments in their calendars and reserve resources, the Schedule Manager task collects and updates that information in the Free Time database. It opens the calendar information for each mail database on the server and computes the free time for each user and stores that data in BUSYTIME.NSF. Calendar functions in the mail template can look in the BUSYTIME.NSF database on the server of a user for free time of users or facilities.

When users invite other users to meetings, the Free Time system performs the free-time lookups. The Free Time system also searches for and returns information on the availability of resources.

When a user is in another domain, the Free Time system uses the Calendar Connector to interact with the Calendar Connector on a server in the other domain to obtain the free time information

IMAP

The Domino server supports the IMAP (Internet Message Access Protocol) service, defined in RFC 2060, for reading mail. After you set up a Domino server to run the IMAP service, IMAP users can:

- Retrieve messages from a Domino mail server that runs the IMAP service and store the messages locally
- Access messages directly from the server
- Copy messages for off-line use and then later synchronize with mail on the server
- Share mailboxes, which is a crude way to implement online discussions

POP3

POP3 (Post Office Protocol, Version 3) defined in RFC 1081, is an Internet mail protocol that allows a user of a POP3 client — for example, Netscape Communicator, Eudora Pro, or Microsoft Outlook — to retrieve mail from a server that runs the POP3 service. Users may use only a POP3 client or both a POP3 client and the Notes mail client to access mail. For each POP3-only user, you create a Person document and a mail file. Before they can access their mail files, POP3 users must authenticate with the Domino server.

Router

The Router constantly scans the MAIL.BOX database on a server for messages to deliver. To determine the destination server for each message, the Router checks the Domino Directory. For a recipient on the same server as the sender, the Router moves the message directly to the recipient's mail database. For a recipient on another server in the domain, the Router moves the message to the MAIL.BOX on that other server. If the Router does not have access to the destination server — for example, if the destination server is in another domain — the Router moves the message to the MAIL.BOX on an intermediate server. The Router on the intermediate server subsequently moves the message to the destination server and into the recipient's mail database.

SMTP Listener

The SMTP Listener receives mail transmitted in the Simple Message Transfer Protocol (SMTP) from IMAP clients, POP3 clients, and Domino and non-Domino servers and deposits it into the MAIL.BOX database for subsequent delivery or routing by the Router.

Tasks that manage protocols

Domino Enterprise Connection Server (DECS)

The Domino Enterprise Connection Server provides real-time access to external (non-Notes) database data. External data sources supported by DECS are DB2, EDA/SQL, ODBC, Oracle, and Sybase.

You use the DECS Administrator, which is created from the file DECSADM.NTF, to configure DECS connections and realtime activities. Each DECS connection includes information that specifies the external data: the external system, a database on the external system, and a table within the database. Each realtime activity specifies a connection, a Notes database to associate with the connection, a form in the Notes database, a description of how fields in the form correspond to fields in the external source, and actions to take before or after a predefined event occurs in the external database. The realtime activity may also specify options that modify and/or extend the meaning of the basic information.

Documents in the Notes database relate to records in the external source based on values in the fields designated as "key" fields by the realtime activity. A change to a data field in a Notes document is reflected as a change to the corresponding data field in the external source in real time, and vice versa.

Examples of actions that can be specified as part of a realtime activity include a Notes formula that runs prior to the creation of a new record in the external database or a stored procedure that runs in the external database after data is updated.

For more information about DECS, see the *Domino Enterprise Integration Guide*.

Domino Internet Inter-ORB Protocol (DIIOP)

There are many ways for a program to request that another program on the same or on a different computer perform certain tasks. For example, a Notes client uses a Notes remote procedure call (NRPC) to request a remote server to run code on the client's behalf to create or access a database residing on the remote server. Another way to do this is to use the Common Object Request Broker Architecture (CORBA), an industry standard that defines how service-providing programs — for example, database managers — use Object Request Brokers (ORBs) to make their services available to other programs that need them.

The DIIOP server task implements CORBA's Inter-ORB Protocol (IIOP) so that CORBA-compliant client programs — for example, Java applets running in a browser — can access Domino services across a network.

For more information about creating Java programs that use DIIOP and Domino ORB to instantiate and use Domino back-end classes, see the *Domino Designer Programming Guide, Volume 3: Java Classes*.

HTTP service

The HTTP task enables Domino to become a Web application server. As a Web application server, Domino can host Web sites and serve pages that are stored in either the file system or in a Domino database.

When a Web browser requests a page that is stored in a Domino database, Domino translates a document into HTML. When a Web browser requests a page that is stored in an HTML file, Domino reads the file directly from the file system. Then the Web server uses HTTP to transfer the information to the Web browser. Using Domino to store Web pages as documents in a database has a major advantage over storing static HTML pages: any changes to the database are automatically reflected on the Web server.

LDAP service

LDAP (Lightweight Directory Access Protocol) is a protocol that uses TCP/IP to allow clients to access directory information. LDAP defines a standard way to search for and manage entries in a directory, where an entry is one or more groups of attributes that are associated with a distinguished name. A distinguished name — for example, cn=Phyllis Spera,ou=Sales,ou=East,o=Acme — is a name that uniquely identifies an entry within the directory tree. A directory can contain many types of entries — for example, entries for users, groups, devices, and application data. The LDAP service responds to incoming LDAP requests and returns information stored in the Domino Directory (NAMES.NSF).

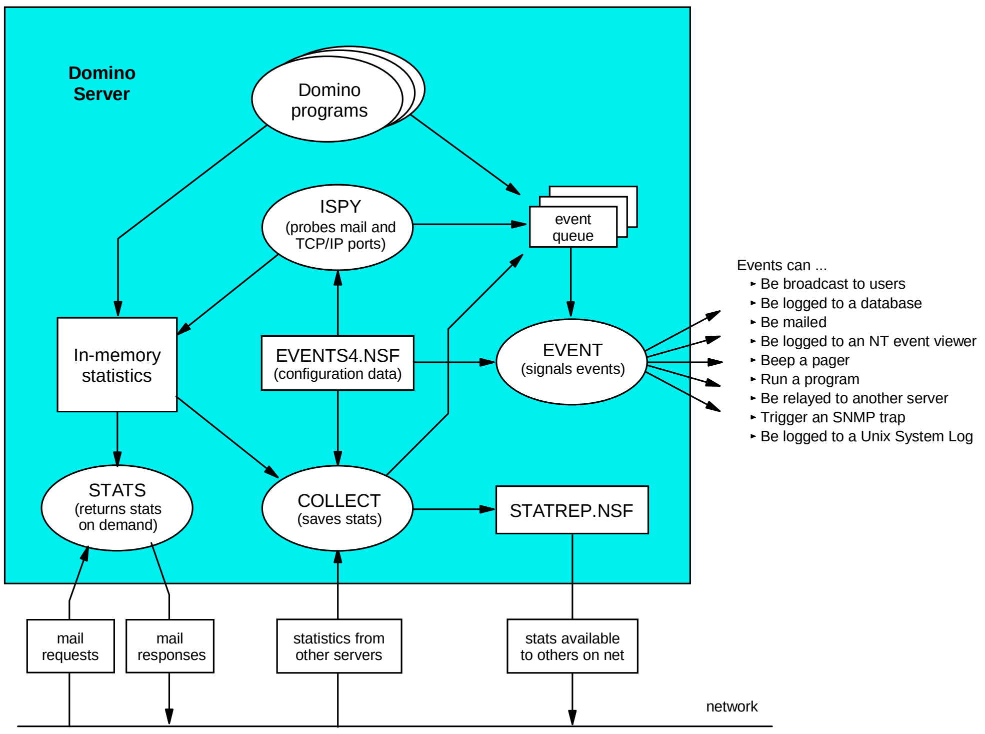
NNTP service

NNTP (Network News Transfer Protocol) uses TCP/IP to provide a reliable connection for newsfeeds, which are the periodic transfer of newly posted newsgroup articles from one server to another. With the NNTP service, you can push or pull newsfeeds, as well as read news articles remotely. A newsfeed push occurs when a server contacts the client and indicates it has news; a newsfeed pull occurs when the client contacts the server and requests news. Because newsfeeds are stored in Notes databases, users can use a Notes client or any news reader (NNTP) client to access them.

Overview of statistics and events

There are a number of server tasks that support the monitoring of server activity. Four of them — ISpy (ISPY), Statistics (STAT), Statistic Collector (COLLECT), and Event Monitor (EVENT) — all work together to report key server statistics and/or to signal significant server events. This section gives an overview of these four tasks and describes how they work together.

The following figure illustrates the Statistics and Event Monitor tasks. It also illustrates the data they use to perform their duties — the in-memory statistics table, the EVENTS4.NSF database, and the STATREP.NSF database — and it shows how other programs that run on a Domino server create statistics and/or signal events.


Domino Server
Domino programs
ISPY (probes mail and TCP/IP ports)
event queue
Events can ...
‣ Be broadcast to users
‣ Be logged to a database
‣ Be mailed
‣ Be logged to an NT event viewer
‣ Beep a pager
‣ Run a program
‣ Be relayed to another server
‣ Trigger an SNMP trap
‣ Be logged to a Unix System Log
In-memory statistics
EVENTS4.NSF (configuration data)
EVENT (signals events)
STATS (returns stats on demand)
COLLECT (saves stats)
STATREP.NSF
mail requests
mail responses
statistics from other servers
stats available to others on net
network

Domino programs use the Stat service in NOS to record statistics in an in-memory table (shown on the left side of the diagram). Statistics describes how long it takes to do a specific activity, how many times an activity occurs during a period of time, and so on. Many statistics are generated by server programs as a byproduct of their activities. Others are generated by the ISpy server task, which actively gathers statistics about Router performance by periodically generating test mail and about the status of TCP/IP ports on servers in the domain by probing them to see if they are active.

Information from the in-memory statistics table can be retrieved in several ways. First, the Statistics task provides it on demand, if desired, through the use of Domino's mail features. Second, the Statistic Collector task stores it in a shared database (STATREP.NSF) which anyone who has the appropriate access can view. The Statistic Collector can be viewed by anyone having the appropriate access rights. The Statistic Collect task always saves statistics generated on servers in the local Domino server, but it can also be configured to save statistics generated on the Domino servers in the domain so that administrators need to visit only one STATREP.NSF database to view the statistics generated by many servers. Third, you can use the Domino Administrator to retrieve and examine the statistics. Fourth, you can use the Show Stat command at the console to examine some or all statistics.

In addition to maintaining statistics, Domino server programs can use the Log, Addin, and Event services in NOS to generate events. An event is a realtime indication that something significant happened. The list of what is an event and what to do when an event occurs is in the EVENTS4.NSF database. The ISpy task signals an event whenever a mail probe or port probe fails. If the probe succeeds, then the ISpy task records information about the probe in the in-memory statistics table and does not signal an event. When the Statistic Collector notices that a statistics has crossed a significant threshold, as defined by information in EVENTS4.NSF, it signals an event. In addition to events that the administrator specifies, Domino programs generate events.

The Event Monitor manages the queue of event requests. Using information in EVENTS4.NSF, the Event Monitor determines how to signal the event. There are many ways to signal the occurrence of an event — from broadcasting it to one or more users to beeping a pager.

Tasks that monitor server activity

Billing

When you enable billing, Domino collects information about client and server activity and places this information in the billing message queue. Periodically, the Billing task polls the message queue and moves the billing information to a destination that you specify — a Notes database (BILLING.NSF), a binary file, or both.

To create billing reports, you write an application to access the billing information. If you collect the information in a Notes database, you can write a Notes API program to create billing reports. If you collect the information in a binary file, you use a third-party program to analyze the data and create billing reports. Using the information that billing collects, you can charge users for the amount they use the system, monitor usage trends, conduct resource planning, and determine if clustering would improve the efficiency of the system.

Event

The Event task receives events from an event queue that stores the events that occur in the Domino Server program. Based on information in the EVENTS4.NSF, the Event Monitor uses one of these methods to notify the administrator that a specific event has occurred:

- Broadcast a message
- Log a message in a database
- Send mail
- Log an event to an NT event viewer
- Initiate a page
- Run a program
- Relay the event to another Domino server
- Triggering an SNMP trap
- Log a message to a Unix system log

ISpy

The ISpy task, which is written in Java, sends server and mail probes as specified by information in EVENTS4.NSF. For those probes that succeed, ISpy records information about the performance of the probe in the server's in-memory statistics table. For those that fail, ISpy generates an event, and then the Event Monitor task uses the method specified in EVENTS4.NSF to notify the administrator.

Mail Tracking

System administrators and users can track mail. An administrator can track mail sent by any user, while users can track only the messages that they sent. The Mail Tracking task first reads log files that the Router produces and then writes summary data about message traffic to the Mail Tracking Store database (MTSTORE.NSF). When an administrator or user searches for a particular message, Domino searches the Mail Tracking Store database, which is created automatically when mail tracking is enabled on the server.

Reporter

The Reporter task is used on servers that run Domino Release 4.x and earlier releases. In Domino Release 5, the Statistic Collector task replaced the Reporter task. The Reporter task saves statistics in REPORTS.NSF. Each Domino server runs a copy of the Reporter task and has a copy of REPORTS.NSF. To gather statistics about all of the servers in the domain, the Domino administrator must look at each REPORTS.NSF database on each server.

Stats

The Stats task provides on-demand access to current server statistics that are stored in the in-memory statistics table. To request information, you send mail to a mail-in database, and Stats returns the requested information in a mail message.

Statistics
The Statistics task runs by default once daily at 5 AM when it collects database activity information from databases on the server and records it in the server's log file (LOG.NSF). To view the statistics, open the Database - Usage and Database - Sizes views of the log file or view the User Activity dialog box of individual databases.

Statistic Collector
Starting with Domino Release 5, the Statistic Collector task replaced the Reporter task. The Statistic Collector task is a multiserver version of the Reporter task. Using information in EVENTS4.NSF, one Statistic Collector task running on one server can gather statistics from many servers in the domain and put all the statistics in the Statistics & Events database (STATREP.NSF) so that administrators can view statistics about all participating servers in a single database file.

In addition, the Statistic Collector can be instructed by means of threshold values in EVENTS4.NSF to generate events, which are in turn signaled by the Event task.

Chapter 5
Notes and the Web

This chapter describes the architecture that Notes uses to provide its services on the Web and discusses how components of the Internet are integrated into the Notes client.

Types of Web servers

The World Wide Web revolutionized computing by making it easy for people to access and publish information in a way that defies geographic and organizational boundaries. However, there was nothing revolutionary about the underlying software technology, at least not in the early days of the Web. People were retrieving files and browsing hypertext documents long before the Web caught on. Notes is a good example of this kind of technology. The Web has gradually added features and defined standards for groupware-like capabilities that have been available in Notes since 1989. For example, Notes has many core features — for example, security that relies on certification procedures and encryption technologies and structured data standards (XML) for storing and exchanging information — that the Web is now adopting.

Early Web servers established the following standards that make the entire Internet resemble one big document store:

- A URL (Uniform Resource Locator) is a naming convention that refers to the location of a document on the Internet.
- HTTP (Hypertext Transfer Protocol) is a communications protocol for retrieving remote documents.
- HTML (Hypertext Markup Language) is a standard for creating and displaying formatted text documents.

From the point of view of software design, first-generation Web servers are basically file servers that received file requests over a network connection and responded by fetching a file from a hard disk and copying it to the network. These servers also provided the Common Gateway Interface (CGI), a crude mechanism for running a program, rather than sending a file, in response to a URL request. However, the majority of the traffic for these servers consisted of file download requests.

The next generation of Web servers were application servers that dynamically generated Web client content based on user requests. These Web servers could provide information from sources — such as, databases — and did not require that the data already be encoded in HTML. Web application servers could filter a document and customize it, based on the authenticated user who requested the document.

To customize a document, a server can:

- Change the appearance of the document.

  For example, each user can set preferences for a site’s background pattern. Alternately, an information provider might need to present identical data under different brand names and then use the identities of the users to determine which logo to display with the content.
- Restrict access to some of the content of the document.

  A user can see only the data that relates to him, although all users can use the same URL to refer to that data.
- Alter what the user can do with document.

For example, the server can allow or deny users the access to view or modify a page element. One user might see a particular field as static text, while another sees a box that allows for user input.

- Adjust to the user's software.

  The server can substitute a more sophisticated or better looking capability — such as, a specialized Java applet if the user's browser supports it — and substitute suitable work-arounds or gracious apologies in relevant parts of the document if the user's browser does not support the required functionality.

That the Web can be used for more than simply transferring files drives the current development in Web application server technology. If you think of your Web browser as a facility for implementing a general-purpose remote user interface, the Web server's role changes from serving files to implementing an application. The Web server responds to user input, updates the user interface, and updates its own internal state, as required by the semantics of the application.

The Domino Web server

Lotus Domino is a Web application server that provides an integrated set of services that developers use to create secure, interactive Internet and intranet applications. The Domino Server program runs server tasks. One task — HTTP — transforms the Domino server into a true Web application server. However, Domino is different from a traditional Web file server in three important ways.

Dynamic content

Domino automatically generates custom HTML pages from the object store based on time, user identity, user preference, client type, and data input. Different than a file store, the object separates form from content — that is, page layout from page data. When a user requests a document or page, Domino dynamically combines form and content. At the time of the request, Domino can also integrate information from other sources such as relational databases. Developers use this efficient storage and rendering method to design applications that customize content based on, for example, user identity and user profile. Domino uses a directory to manage user information and authenticate users. After a user is authenticated, an ACL authorizes user access to information.

User interaction

Domino not only serves custom information but also lets users interact with applications. Using URL syntax, Web users can issue commands to the Domino server. These commands might create, edit, or delete information or run agents on the server. Since this syntax is of little consequence to the end user, the application developer can create a point-and-click UI for the application and program its capabilities with simple @functions, instead of creating CGI or Perl programs. The editing capabilities differ from other solutions because they don't require the end user to know where the information is stored in the file system. The access levels are No Access, Depositor, Reader, Author, and Editor. The developer can define access roles that further refine the access that certain classes of users have. Just as Domino can deliver dynamic content based on class of user, it can also define application functionality, based on the class of user. This flexibility is useful in business applications that are used by a variety of users who must participate in a distinct way.

Workflow and page processing

When a user submits information changes or clicks a button to run an agent, numerous Domino services can be called into action to gather more information, process information, or push information through a serial or parallel workflow process. Enterprise integration services can submit information to relational, host, and transaction-processing systems. Agent services can act on the information to automate routine application maintenance — for example, the addition of new users. Messaging services can route information from individual to individual or department to department until the work process is completed. Other devices such as faxes and pagers can be used in the processing of information. These features are useful when developing transactive business applications that leverage existing systems and automate everyday business activities.

What distinguishes Domino from a Web application server and a Web file server are the services that allow it to present content dynamically and to manage content as it flows through a defined process. These include object stores, directory, security, workflow, agents, enterprise integration, and messaging services.

Domino Web server object model

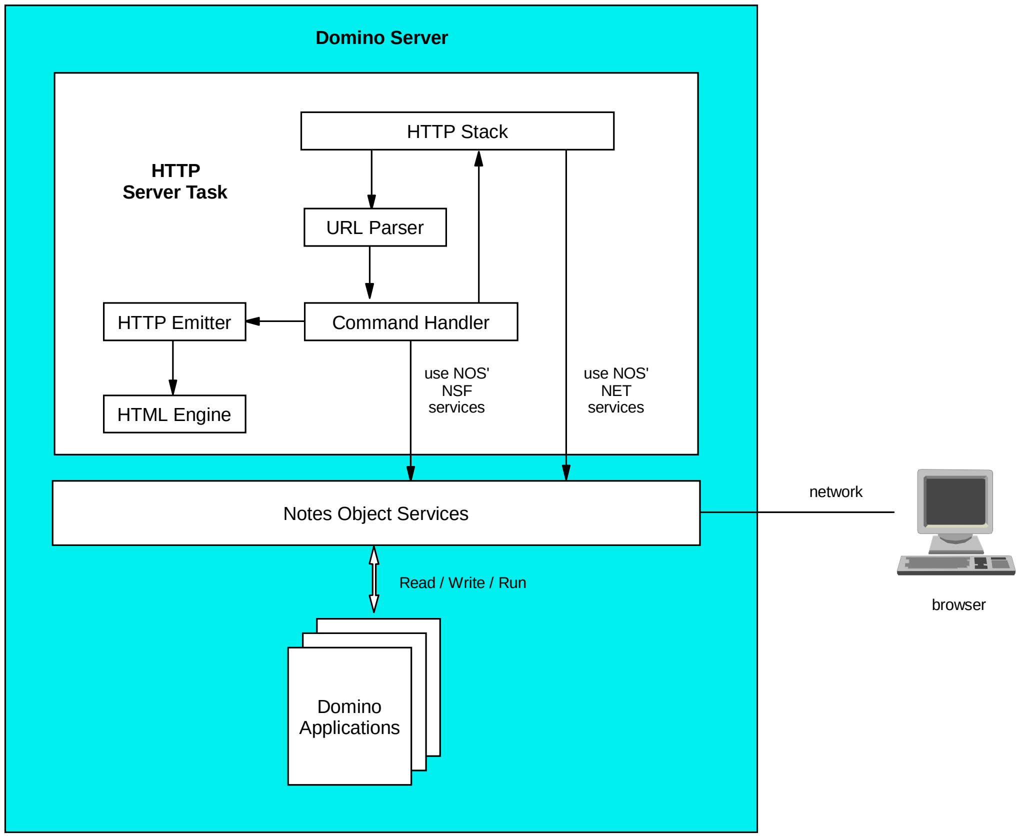
The architecture that makes Domino an efficient, service-rich, Web application server is based on a concept consistent throughout Domino: the object-oriented design. The Domino Web server is built around a series of objects that model elements of the Notes database and document structure, and define the presentation of this data. It is also a runtime environment for the applications, providing methods that implement the necessary conversions and computations that allow a browser — which has little intelligence of its own — to “run” an application. Other Web servers have no built-in data model, presentation paradigm, or application execution environment, so programmers must build data models, decide how to present information, and adapt their architecture to the limited capabilities of the Common Gateway Interface (CGI) specification or Web server vendor API. Domino excels as a Web application environment because it removes these distractions and lets developers concentrate on modeling the problem at hand.

The Domino Web server object model closely follows the user experience of a Notes user. It includes objects that represent top-level objects — for example, servers, databases, navigators, forms, views, and documents — and all the low-level elements that make up these things, such as, rows and columns in views; items in documents; rich text and fields in forms; and paragraphs, tables and even colors and fonts in rich text. In response to a URL request, the server locates the target specified by the URL, which could be any top-level object, and builds the object in memory along with any low-level objects that it includes, and then activates the GenerateHTML method associated with each object. This method specifies a read/edit argument that determines whether the method should generate HTML pages or HTML forms. This kind of intelligent HTML engine frees the individual object’s methods from having waste processing time with the syntactic and semantic rules of HTML.

In many cases, the objects that represent Domino design elements must be generated as anchor tags. Operations such as creating a response document, navigating to the next document, or expanding a view category are context dependent. The server is stateless. The HTML generated by these objects always contains whatever information the server needs to determine the context. For example, the anchor tag for the triangle that controls the expansion of a category in a view specifies a URL that contains the view name, the ?OpenView command, an &Start argument indicating the position in the view of the first document to display, an &Count argument indicating how many documents to display, and an &Expand or &Collapse argument.

Domino Web server architecture

The following figure illustrates the architecture of the Domino Web server:



The Domino Web server is a TCP/IP application that implements the HTTP protocol. Domino answers URL requests from Web clients by sending back pages of data in HTML. It also handles URL requests and HTML forms that trigger executable programs according to the Common Gateway Interface (CGI) specification. In these respects, the Domino server behaves just like any other HTTP server, responding in the standard way to standard URL requests. Domino however, is more than a typical HTTP server.

At the heart of the Domino Web application server is the HTTP server task. Requests from Web clients go directly to the HTTP task to be processed. The task has all the usual facilities for accessing HTML pages stored in the file system of the host platform and running CGI programs. Written in C/C++, the HTTP task uses the Notes Object Services (NOS) to integrate with the rest of the Domino server.

On rare occasions the HTTP task bypasses NOS. The HTTP task is multithreaded. The task listens for Web client requests and sends responses, and normal server tasks run with the identity of an ID file that resides somewhere on the host file system. However, to implement security for Web clients, the thread servicing a request must assume the identity of the logged-in user.

The preceding figure illustrates the core components of the HTTP task. The HTTP stack includes all the code that deals with both inbound and outbound HTTP communications. The URL parser handles the incoming Domino URL calls. The HTML emitter prepares the outgoing flow of information that results from URL calls. The HTML emitter uses the HTML engine, which acts as a source for standards that define the proper format of any information translated into HTML.

In the center is the command handler, which is the direct link between the HTTP task and NOS. This layer translates commands into a format that is equivalent to the format of commands made from a Notes client or called from the Dbserver task.

Let us look at the process in more detail:

When the HTTP task receives a request from a Web browser, a connection is made to the HTTP stack, which manages the connection between the Web client and server. It is at this point where secure socket layer (SSL) may be implemented.

The default action for the HTTP stack is to send the request directly to the URL parser, which determines if it's a standard URL or a special Domino URL. If it is a standard URL, the parser sends the information to the HTTP stack which processes it as simple HTTP commands. If it is a Domino URL, the parser breaks the URL into different parts, performs a series of checks, provides implicit commands when necessary, and then invokes the appropriate command handlers. These handlers manage of all the details associated with each command by establishing the correct identity for security purposes, accessing NOS, executing formulas and scripts, and retrieving information.

The HTTP task can be considered a Notes client emulator written with a tight integration with the NOS and running within the Domino server. The resulting information sent back to the HTTP server from its performed actions are returned through the command handler. These results are passed to the HTML emitter, which uses the HTML engine to translate the information into HTML pages. Then the pages are sent to the HTTP stack, which establishes a connection to the client and sends the data out properly.

Components of the HTTP Server task

Let us look at the components of the HTTP task in more detail.

HTTP stack

The HTTP stack is written in C and is the gateway to the HTTP task. All information coming into Domino from the Web and going out from Domino to the Web goes through this layer. The HTTP stack has four major functions:

- Connection management

  This establishes the TCP/IP connection to the Web and maintains all the HTTP interactions for the connection.
- URL dispatch

  This uses the HTTP protocol to send HTML text to the browser.
- HTTP command redirection

  This task executes plain URL commands for tasks such as displaying an HTML file, running a servlet, or performing any other action that does not deal with the Domino server. This service starts when the parser rejects a URL as a non-Domino URL.
- SSL implementation

URL parser

The URL parser distinguishes a Domino URL from a standard URL and translates Domino URLs into commands that the command handlers can understand. A Domino URL specifies a path to or the replica ID of a Notes database (NSF) or is a server command such as ?OpenServer. The entire Notes object path doesn't have to be found to be considered a valid Domino URL. An example could be a reference to a nonexistent form in an NSF file.

First, the parser breaks the URL into up to three parts: a mandatory object identifier, an optional command, and arguments.

Second, the parser translates Domino URLs into commands that the command handlers can understand. To do this, the parser associates the recognized object with the proper commands allowed for that object. The following is an example of a typical Domino URL:

```
http://www.xyz.com/site/app.nsf/f34a868d3e2aa2a385256345006edae6?OpenView
&start=100
```

This sample URL references a view in the database site/app.nsf on the www.xyz.com server. This view is identified by the UNID of its design document — the long string of hexadecimal digits. The rest of the URL, the command portion, is syntactically optional, but when it is left out, the URL parser determines the type of the object and supplies an implicit command appropriate for that object type. The specific command in this sample is ?OpenView and has an argument called &start that specifies that the Domino server should start at the 100th document in the view collection.

Command handlers

The command handlers use four caches to improve efficiency: the database cache, the design cache, the HTML cache, and the static cache.

Command cache

There are some specific situations in which caching HTML makes sense, but because most Web browsers locally cache pages, there is little point in the server caching HTML that is likely to be accessed by only one user. This assumes that users don’t frequently access the same server with two browsers, which is usually a good assumption.

Documents that contain Readers fields are considered to represent a view that is specific to a user. However, the Web server would have to go around the security enforced by the Notes API to determine that there are no documents with Readers fields that meet the selection criteria of that view. Since this would be terribly inefficient, HTML for view displays is not cached and shared between users.

However, if an application allows unauthenticated user access, many users will access it with the same identity — that is, “Anonymous.” Since these browsers can share the HTML for views, the HTML cache is used to store the output of ?OpenView commands. The HTML generated in response to an ?OpenDatabase command may also be cached.

If formulas are dependent on @UserName or @UserRoles or if section security is used, that HTML is considered to represent a document that is specific to a user and is not cached. Caching is also inappropriate if formulas are time-dependent. Technically, all Domino URL commands are candidates for caching. Commands are discarded only after being proved uncacheable. Commands based on authentication or POST data are immediately discarded, as are commands that encounter @functions that are themselves dependent on information residing outside the cache’s control.

HTML emitter and engine

The Domino Web server process ends with the production of HTML to send back to browsers. An intelligent HTML emitter is used by all the command handlers. The emitter uses a table-driven HTML engine that has knowledge of the syntax and semantics of HTML which ensures that the output conforms to HTML standards. For example, the emitter keeps state information to ensure that termination tags are inserted as required. This dramatically reduces the complexity of the command handlers. The result is that the HTML translation is remarkably efficient.

How Domino processes a URL

Each URL request sent to the Domino Web server specifies a top-level object. The server then activates the methods needed to carry out a request and generate HTML output. The Domino Web server implements many commands. The commands related to the display and modification of documents are worth a closer look. These are the commands that enable interactive applications, so the similarities and differences in their implementation are of particular interest.

Three commands display the contents of a document:

- ?OpenDocument opens an existing document in read mode
- ?EditDocument opens an existing document in edit mode
- ?OpenForm opens a new document in edit mode.

?OpenDocument and ?EditDocument

The implementations of ?OpenDocument and ?EditDocument are very similar. Both execute the following sequence of steps:

1. Open the existing document.
2. Read all items from document into memory.
3. Load the form referenced by the document.
4. Scan the form, calculate default-value formulas for any items that are not yet in memory, calculate all computed-field formulas, and add/update items to the in-memory document.
5. Run the appropriate agent if an item named $$QueryOpenAgent exists.
6. Use the item values from the in-memory document to render the form into an HTML page or HTML form.
7. Free the in-memory document and all items it contains.

The only significant difference between the ?OpenDocument and the ?EditDocument command is in Step 6. ?OpenDocument instructs the GenerateHTML methods to respect read-mode hide attributes contained within the form and to create HTML for a read-only page. ?EditDocument instructs the GenerateHTML methods to respect edit-mode hide attributes on the form and to create an HTML form.

?OpenForm

The ?OpenForm command executes this sequence of steps:

1. Create a new document in memory.
2. Load the form referenced in the URL.
3. Scan the form, calculate all default-value formulas and computed-field formulas, and add items to the in-memory document.
4. Run the appropriate agent if an item named $$QueryOpenAgent exists.
5. Render the form into an HTML form, respecting edit-mode hide attributes and using item values from the in-memory document.
6. Free the in-memory document and all items it contains.

The last step of the procedure for all three commands frees the in-memory document and its associated items. The ?EditDocument and ?OpenForm commands do not cache the documents for later use because that would violate the stateless nature of a Web server. If and when the browser sends edited information back, the Domino Web server re-establishes all the data structures necessary to handle the updates.

?CreateDocument and ?SaveDocument

?CreateDocument and ?SaveDocument are the two commands that receive HTTP POST data that is generated when a user presses the Submit button on an HTML form and saves the data in a Notes document. Submit buttons that are generated by the ?OpenForm command send the ?CreateDocument command. Submit buttons that are generated by the ?EditDocument command send the ?SaveDocument command. These two commands follow a similar sequence of steps.

?CreateDocument

1. Create a new document in-memory.
2. Open the form referenced in the URL.

3. Scan the form, calculate all default-field formulas and computed-value formulas, and add items to the in-memory document.

?SaveDocument

1. Read the existing document referenced in the HTTP POST data.
2. Open the form referenced in the URL.
3. Scan the form, calculate all default-field formulas and computed-value formulas, and add items to the in-memory document.

Then both ?CreateDocument and ?SaveDocument continue on essentially the same path:

4. Create items for all data sent in the POST data.
5. Scan the form; calculate all translation, validation and computed-value formulas; update items in the in-memory document; and return validation errors as HTML.
6. Scan the form and delete any computed-for-display items from the in-memory document.
7. Run the appropriate agent if an item named $$QuerySaveAgent exists.
8. Save the in-memory document to the appropriate database file.
9. If the $$Return item exists, run the appropriate formula from the form. If there is no output from a QuerySave agent, send the result of this formula back to the browser.
10. Delete the in-memory document and all associated items.

Formulas that calculate at the time that an HTML form is sent by either the ?EditDocument or ?OpenForm command calculate again when ?CreateDocument or ?SaveDocument commands process. This can cause some confusion if these formulas have time-dependent values or are based on lookups of constantly changing information.

Chapter 6 Security

This chapter describes the mechanisms that Notes and Domino use to secure applications.

The Notes/Domino security model

Notes/Domino provides a robust security model that you can tailor to meet your organization's requirements. You can think of the Notes/Domino security model in terms of an everyday example: protecting a home. You can't gain entry to the grounds of the house, unless you have access to the front gate; you can't get into the house unless you have a key to the front door; and so on. Basically, you cannot enter specific parts of the house until you pass through each layer of security. Notes/Domino has a six-layer security system.

Network

Network security protects against unauthorized access to the network on which the Domino servers reside. If you block access at the network layer, unauthorized users do not have access to any Domino servers. Network access is typically controlled using the network hardware and software, but you can further secure network access by encrypting data from the port on the Domino server. Encrypting a network port prevents unauthorized users from using a network protocol analyzer to read data. To encrypt network traffic, you use port encryption or SSL.

User authentication

User authentication is the process the Notes client and Domino server use to validate and authenticate each other when a client tries to access a Domino server. Notes and Domino use certificates stored in the Notes ID file for validation and authentication. When using Internet protocols, authentication can be based on X.509 certificates or the user's name and password.

Server

Server security controls access to the Domino server. Users must first be authenticated before server security is checked. Server access is controlled using a server access list in the Domino Directory.

Database

Database security controls access to databases on the Domino server. Users must first have access to the server and be authenticated before database security is checked. Database access is controlled using database access control lists (ACL). A local database can also be encrypted so that only a user who has the correct password to the ID file can access the database.

Design element

Design element security controls access to forms, views, and folders. Users must first have access to the database before design element security takes effect. Using design element security, you can allow users to view some types of documents in the database and block access to other types. To control form access, you use form access lists and encryption keys. To control view and folder access, you use view and folder access lists. You can also use design element security to limit the actions of formulas and scripts when they run on a Notes workstation. To control workstation access, you use an execution control list (ECL).

Document

Document security controls access to the fields, sections, and paragraphs in a document or to the entire document. This is the most granular form of security. You use Readers and Authors fields to control document access. To control access to sections and paragraphs, you use hide-when formulas. To control access to fields, you use encryption keys.

Securing local databases

The above six layers of security apply to databases that are stored on Domino servers and that are accessed by users who use a network connection. If someone has unauthorized access to a user's machine or to the server, he can bypass the security and read local databases. To prevent unauthorized access, Notes lets you encrypt a database and enforce an ACL on a local database.

Notes/Domino access control

To protect critical data, Notes and Domino provide access control mechanisms that you use to restrict the access that authenticated users have to:

- Servers and ports
- Databases
- Files
- Design elements , such as forms, views, and folders
- Workstation data
- Documents

Parts of this section are abstracted from the IBM Redbook, *Lotus Notes and Domino R5.0 Security Infrastructure Revealed*, which can be viewed in its entirety and/or ordered in hard copy by visiting the Web site www.redbooks.ibm.com/booklist.html.

Restricting access to servers and ports

The port access list is stored in the server's NOTES.INI file and is checked whenever a user tries to use that port to access the server. The server access list is stored in the Server document, which is in the Domino Directory, and is checked when the client or server performs a task that may be restricted.

Port access list

Use a port access list to allow or deny users and servers access to a specific network port. If you use a port access list and a server access list, users and servers must be listed on both to gain access to the server.

To control access to a specific port, use these NOTES.INI settings:

- Allow_Access_*portname* = *names*
- Deny_Access_*portname* = *names*

where *portname* is the name of the port, and *names* is a list of users, servers, and groups to whom you want to deny or allow access. These names must be contained in the Domino Directory.

For more information about these NOTES.INI settings, see *Administering the Domino System.*

Server access list

Administrators use the server access list to restrict the access that groups of users have to a server. In addition, administrators can further refine access by allowing only certain users to perform specific tasks on the server. On the Security tab in the Server document, you can specify these server access settings:

- Who is allowed access to the server using Notes protocols (there are separate controls for HTTP, IMAP, POP3, and LDAP)
- Who can create new or replica databases
- Who can use headline monitors
- Who can use this server for passthru and what servers the passthru server will route to, as well as additional options for passthru
- Who can run agents on the server
- Which authenticated Internet clients can use the Web Administrator to administer the server
- Which authenticated Internet clients can use the IIOP protocol to run Java or JavaScript on the server

On the Security tab, you can specify additional security checks to make it more difficult for a user to impersonate another user:

- Compare Notes public keys against those stored in the Domino Directory
- Allow users to connect to the server anonymously
- Compare the password on the Notes ID against the Domino Directory

The following example shows how Bob (the client) gains access to ServerA when Bob uses the File - Database - Open command to request a list of databases available on the server. First, Bob must authenticate with the server and establish a session; then server access control takes effect.

1. ServerA checks the Group documents in the Domino Directory to establish which groups Bob belongs to. ServerA builds a list of Bob's group memberships and stores this list in memory for quick access.
2. ServerA checks the NOTES.INI file to see if Bob is allowed access on the port that he requested.
3. ServerA checks the Server document for ServerA to see if Bob is allowed access to the server.

Restricting access to databases

Every database has an access control list (ACL) that specifies the level of access that users and servers have to the database. The ACL is stored as a note in the database. Although the types of access levels are the same for clients and servers, those assigned to clients determine the tasks that they can perform in a database, while those assigned to servers determine what information within the database the servers can replicate.

An ACL contains two special names: Anonymous and -Default-. Anonymous specifies the default database access for unauthenticated users. -Default- specifies the default database access for authenticated users and unauthenticated users if the Anonymous entry does not exist. While every database ACL must specify -Default- access, the Anonymous entry is optional.

To control the access that each client and server has to a database, the database manager specifies an access level, access privileges, and user type.

Access levels

The following table describes the database access levels.

Access level	Allows users and servers to
Manager	Modify the database ACL, encrypt the database, modify replication settings, delete the database, and perform all tasks allowed by lower access levels.
Designer	Modify all database design elements, create a full-text index, and perform all tasks allowed by lower access levels
Editor	Create documents and edit all documents, including those created by others. Read all documents unless there is a Readers field in the form. You must be able to read a document in order to edit it.

continued

Access level	Allows users and servers to
Author	Create documents only if the access privilege “Create documents” is selected. Edit the documents where there is an Authors field in the document and the user is specified in the Authors field. Read all documents unless there is a Readers field in the form.
Reader	Read documents. However, when the document contains a Readers field, only users whose names are listed in that field can read that document.
Depositor	Create documents.
No Access	None, with the exception of options to “Read public documents” and “Write public documents.”

Access privileges

A database manager selects the access level for each user, group, and server, and then enhances or restricts this level as needed by selecting or deselecting additional privileges within an access level. For some access levels, the default access privileges are hard-coded; that is, they can’t be changed for that access level. The following table describes the database access privileges.

Access privilege	Description
Create documents	Determines whether a user can create documents in the database. If a user is listed in an Authors field of a document, the user can still modify that document. This privilege is automatic for Managers, Designers, Editors, and Depositors. It’s optional for Authors.
Delete documents	Determines whether a user can delete documents in the database. If this privilege is deselected, a user can’t delete documents, no matter what the access level. Authors can delete only documents they create. If the document contains an Authors field, an author can delete documents only if his user name is specified in the Authors field. This privilege is optional for Managers, Designers, Editors, and Authors.
Create personal agents	Determines whether a user can create personal agents in the database. Once created, a personal agent can perform only those tasks allowed by the user’s assigned access level in the ACL. If the user creates an agent that runs on the server, the Agent Restrictions section of the Server document in the Domino Directory determines whether the agent can run. This privilege is automatic for Managers and Designers. It’s optional for Editors, Readers, and Authors.
Create personal folders/views	Determines whether a user can create personal folders and views in a database on a server. Personal folders and views created in a database on a server are more secure than those created locally, and they are available on multiple servers. If the “Create personal folders/views” privilege is not selected, users can create personal folders and views and store them on their local workstations. This privilege is automatic for Managers and Designers. It’s optional for Editors, Authors, and Readers.
Create shared folders/views	Determines whether a user can create shared folders and views in a database. Deselect this option to maintain tighter control over database design. Otherwise, users can create views visible to others. This privilege is automatic for Managers and Designers. It’s optional for Editors.
Create LotusScript/ Java agents	Determines whether a user can create LotusScript and Java agents in a database. Since LotusScript and Java agents on server databases can take up significant server processing time, database managers may want to restrict which users can create them. Whether or not a user can run agents is dependent on the access set by the Domino administrator in the Agent Restrictions section of the Server document in the Domino Directory. This privilege is automatic for Managers. It’s optional for Designers, Editors, Authors, and Readers.

continued

Access privilege	Description
Read public documents	Determines whether a user can read public documents. This option lets you give users with no access the ability to view specific documents without giving them full reader access to the database. **Note** A document is public if it has a $PublicAccess field with a text value of "1." Documents are not normally public; however, some specific documents — such as, calendar and scheduling documents in a user's mail file — are marked for public access. This privilege is automatic for Managers, Designers, Editors, Authors, and Readers. It's optional for Depositors and No Access.
Write public documents	Determines whether a user can write public documents. This option allows users to create and modify documents with forms designed to allow public access. This option lets you give users create and edit access to specific documents without giving them Author access. **Note** A document is public if it has a $PublicAccess field with a text value of "1." Documents are not normally public; however, some specific documents — such as, calendar and scheduling documents in a user's mail file — are marked for public access. This privilege is automatic for Managers, Designers, and Editors. It's optional for Readers, Depositors, Authors and No access.

User types

A user type identifies whether a name in the ACL is for a person, server, or group. Assigning a user type to a name specifies the type of ID that is required to access the database with that name. The user types are Person, Server, Mixed Group, Person Group, Server Group, and Unspecified.

User types provide additional security for a database. For example, assigning the Person user type instead of "unspecified" to a name prevents an unauthorized user from creating a Group document with the same person name, adding his or her name to the group, and then using the group name to access the database.

Designating a name as a Server or Server Group prevents a user from using the server ID at a workstation to access a database on the server. Be aware, though, that designating a name as a Server or Server Group is not a foolproof security method. It is possible for a user to create an add-in program that acts like a server and uses a server ID to access the server database from a workstation.

Instead of assigning a user type to each name, a database manager can automatically assign a user type to all unassigned names in the ACL. The user type assigned to each name is determined by the Domino Directory entry for that name. Using this method, a group is always designated as mixed Group, not as a Person Group or a Server Group. To assign a Person Group or Server Group to a name, the database manager must select the name and manually assign that user type.

Additional ACL options

To further control database access, a database manager can enforce a consistent ACL and/or specify a maximum Internet name and password.

Enforcing a consistent ACL ensures that an ACL remains identical on all database replicas on servers and on all local replicas that users make on workstations or laptops. If you don't enforce a consistent database ACL, a user who has restricted access to the database — for example, Author access — can create a local replica of the database and automatically gain Manager access to the local replica. Although the user can't replicate this change in access back to the server (the server copy won't allow a change in the ACL because the user only has author access in the ACL of the replica on the server), the user still has access to all the information in the database.

Specifying a maximum Internet name and password applies to clients who use name-and-password authentication or who access the server anonymously and use the TCP/IP or SSL port to connect to servers. This option limits the access level that a client may have been explicitly given in the database ACL.

Restricting access to files

File Protection documents control the access that Web browser clients have to files — for example, HTML, JPEG, and GIF — on the server. In addition, File Protection documents control access to CI scripts, servlets, and agents. File protection does not, however, extend to other files accessed by the scripts, servlets, or agents. For example, you can apply file protection to a CI script that restricts access to a group named Web Admins. However, if the CI script runs and opens other files or causes other scripts to run, the File Protection document is not checked to determine whether Web Admins has access to these files.

File protection does apply, however, to files that access other files — for example, to an HTML file that opens an image file. If a user has access to the HTML file but does not have access to the JPEG file that the HTML file uses, Domino does not display the JPEG file when the user opens the HTML file.

By default, the Domino Directory contains a File Protection document for the domino\adm-bin directory. This File Protection document, which is created when the server starts for the first time after installation, gives administrators Write/Read/Execute access to the directory and gives all other users No Access.

Restricting access to design elements

Design element security controls access to specific features of a database. For example, a database manager might want to restrict a user who has Author access to being able to create only specific types of documents.

Use of ACL roles in design-element access control

To assign special access to users, groups, and/or servers listed in a database ACL, the database manager creates a role, which is a subset of the ACL. A group formed by using a role is similar to a group in the Domino Directory because you can refer to all members of the group by a single name. However, a role differs from a group in the Domino Directory because it has meaning only within the database where it's created. Role names, like any other ACL name, can be used to refine, or restrict, access to particular views, forms, sections, or fields of a database. Roles are especially useful when the members of a group change frequently.

Roles simplify the job of database designers. For example, a designer can restrict specific users from creating a certain type of document in a database. To do this without using roles, the designer must add all the names of those users to the form's access list. Using roles, however, the designer can associate all those users in the ACL with a role name — for example, Employees — and add Employees to the form access list in place of a long list of names.

Roles have the following advantages:

- Provide a flexible method of restricting document access to a specific set of users
- Can be used in formulas
- Provide group control if you do not have the authority to create groups in the Domino Directory or if you want to create groups just for the database
- Make it easy to modify access when users leave or new users join

Form access control

Form access lists control user access to forms in the database. If users cannot access forms, they cannot read or create documents in the database. Using form access lists refines the database ACL, allowing more flexibility in the database security design. For example, if you assign users Author access in the ACL, you can restrict which forms they can use to create documents. To set form access control, you use the Security tab of the Forms Properties box.

The following table describes the available form access control options.

Security option	Use to
Default read access for documents created with this form	Allow only a subset of users in the database ACL to read documents created with a specific form. This option is also known as the form reader access list. Documents created with this form have these users as the default document reader access list. Notes creates an internal field named $Readers when you create a reader access list.
Who can create documents with this form	Allow only a subset of users with Author access or above in the database ACL to use this form to create documents.
Default encryption keys	Encrypt all fields for which encryption has been enabled with the specified key. This allows only users who have the encryption key to read documents created with this form. Since Internet users do not have Notes encryption keys, Internet users cannot access encrypted fields. Do not, however, rely on encrypted fields to hide information from Internet users.
Disable printing/forwarding/copying to the clipboard	Prevent accidental distribution of confidential information. This feature is not a true security feature since users can circumvent it by using screen capture programs and does not apply to Web users accessing the form.
Available to Public Access Users	Give users with No Access or Depositor access the ability to view and modify specific documents created with this form without giving them Reader access or above in the ACL. In addition, documents that you want available to public access users must contain a field called $PublicAccess that is a text field with a default value of 1 and have the ACL option to Read public documents or Write public documents.

View access control

To allow some users and not others to see the contents of a view, database managers create a view read access list. Users who are excluded from the access list do not see the view on the View menu. A read access list is not a true security measure; it only prevents access to the view, not to the underlying documents. Users can create private views that display the documents shown in the restricted view, unless the documents are otherwise protected. To protect the documents, use a read access list on a form.

Users listed in the read access list have access to a view as long as they already have at least Reader access in the database ACL.

The following table describes the available view access control options.

Security option	Use to
May be used by	Allow only a subset of users in the database ACL to access this view
Available to Public Access Users	Give users with No Access or Depositor access the ability to access the view without giving them Reader access in the ACL

Folder access control

Folders have the same access control options as views. In addition, database managers can create a write access list for the folder to allow some users and not others to update the contents of a folder. Database managers can add users to a write access list for a folder as long as the users already have at least Author access in the database ACL. Users specified in the write access list for the folder can move and copy documents into the folder and can remove documents from the folder. They cannot update the documents themselves unless the database managers give them access to do so.

Outline access control

Database managers can control whether users have access to each entry in an outline. Database managers can hide the outline entry from the user, but doing so does not prevent users from accessing the document. For greater security, use a read access list for a form to prevent others from accessing the document.

Restricting access to workstation data

Workstation ECLs impose security restrictions on Notes @functions, @commands, and LotusScript formulas. The database manager and system administrator can protect users' workstations and databases from unauthorized access by creating an ECL for each workstation and by signing templates and databases. ECLs determine the tasks that a procedure embedded in a database can perform in the database and on the workstation according to who signed the procedure. For example, you may give limited execution access to your Domino system administrator, but allow no execution access to unsigned scripts or formulas.

Individual users can edit their ECL. Administrators should create an Administration ECL that is stored in the Domino Directory and gets copied to the user's workstation when the system administrator registers a new user.

Workstation ECL

By default, scripts and formulas, whether signed or unsigned, cannot run on a workstation without displaying a warning message. However, scripts and formulas can run from any database created with a template that ships with Notes, and are signed "Lotus Notes Template Development/Lotus Notes." This signature has complete execution access by default.

A workstation ECL can limit access to the following:

Access option	Allows formulas and code to
Access to the file system	Attach, detach, read to, and write from workstation files
Access to current database	Read and modify the current database
Access to environment variables	Use the @SetEnvironment and @GetEnvironment variables and LotusScript® methods to access the NOTES.INI file
Access to non-Notes databases	Use @DBLookup, @DBColumn, and @DBCommand to access databases when the first parameter for these @functions is a database driver of another application
Access to external code	Run LotusScript classes and DLLs that are unknown to Notes
Access to external programs	Access other applications, including activating any OLE object
Ability to send mail	Use functions such as @MailSend to send mail
Ability to read other databases	Read information in databases other than the current database
Ability to modify other databases	Modify information in databases other than the current database
Ability to export data	Print, copy to the clipboard, import, and export data
Access to Workstation Security ECL	Modify the ECL

Java applet ECL

When a Java applet runs within Notes, certain security restrictions are imposed on that applet. This is sometimes referred to as the "Java security sandbox." This security model protects against malicious code by determining what operations an applet can perform and what system resources it can access. These restrictions can be customized on a per-signature basis.

Java applets are signed when the user inserts them in to a Notes document. The user can also insert them without a signature by not providing a password when prompted. Java applets created by using the Web Navigator are not signed.

By default, templates that are signed by Lotus Notes Template Development/Lotus Notes are allowed complete execution access. All others have no access.

Access option	**Allows the applet to**
Access to file system	Read and write files on the local file system
Access to Notes Java classes	Load and call the Domino back-end object classes
Access to network addresses	Bind to and accept connections on a privileged port (a port outside the range 0 to 1024) and establish connections with other servers
Printing	Submit print jobs
Access to system properties	Read system properties, such as color settings and environment variables
Dialog and clipboard access	Access the system Clipboard and display the “security banner” in top-level windows. The security banner is a visual indication (usually a message such as “Java Applet Window”) that this window was created by a Java applet. This is done to ensure that a user does not inadvertently enter security-sensitive information into a dialog box that is masquerading as a password dialog box, for example. Choosing this option prevents the security banner from being displayed.
Process-level access	Create threads and threadgroups, fork and execute external processes, load and link external libraries, access non-public members of classes using Java core reflection, and access the AWT event queue

JavaScript ECL

The JavaScript ECL options control security for JavaScript that runs within the Notes client, either on a Notes form or on a Web page rendered by the Notes browser. These options do not control JavaScript that other browsers, including the Microsoft Internet Explorer browser, run, even when the JavaScript is embedded within the Notes client.

The read and write options (under the general categories “Allow Read Data Access From” and “Allow Write Data Access To,” respectively) control whether JavaScript code can read or modify JavaScript properties of the Window object. The Window object is the top-level object in the JavaScript document object model. It has properties that apply to the entire window. Securing access to the Window object secures access to other objects on the page since the JavaScript program cannot access the objects lower in the object model without first traversing the Window object.

By default, templates that are signed by Lotus Notes Template Development/Lotus Notes have complete execution access.

You can control the security for these read and write options independently for three different classes of Window objects:

Window object class	Description
Source window	Controls JavaScript access to the Window object on the same page as the JavaScript code. Typically this is a very low security threat. Selecting this option does not prevent JavaScript calls if the call is made directly to the object on the source window. Doing so circumvents the Window object; therefore, this ECL option is not enforced. The default templates and templates with no signature do not allow read and write access.
Other window from same host	Controls JavaScript access to the Window object on a different page from the JavaScript code, but from a page using the same host. For example, JavaScript code on a page on www.lotus.com can access the Window object on another page on www.lotus.com. This allows two pages to interact if they are within the same frameset. This is a slightly higher security threat. The default templates and templates with no signature do not allow read and write access.
Other window from different host	This is similar to "Other window from same host," except it enables access to the Window object on a different page within a frameset that uses a different host. For example, JavaScript code on a page on www.lotus.com can access the Window object on a page on any other server. This is the highest security threat because someone could design a frameset containing a page performing malicious actions accessing data on another page in the same frameset that you "trust," where you might type a password or some other sensitive information. The default template and templates with no signature do not allow read and write access.

There are two additional ECL options that control whether JavaScript executing in the Notes client is authorized to open a new Web page or Notes document.

The following options are available in the "Allow Open Access To" category.

Option	Description
URL on same host	Controls access for opening a Web page or Notes document on the same host as the JavaScript code. The default template and templates with no signature allow open access.
URL on different host	Controls access for opening a Web page or Notes document on a different host as the JavaScript code. The default template and templates with no signature do not allow open access.

Restricting access to documents

The author of a document can determine who can read the entire document or only specific sections of a document. Documents inherit the read access property from the form; however, the author of a document can further refine the read access list of the form or of specific sections of a document.

The following table describes the document access options.

Option	Description
Who can read this document	Allows only a subset of users in the database ACL and form reader access list to read this document. This option is also known as the document reader access list. A document inherits the reader access list from the form. This option refines the document reader access list.
Encryption key	Encrypts the document using the specified private and/or public keys. This allows only users who have the encryption key to read the document. Since Internet users do not have Notes encryption keys, Internet users cannot access encrypted fields.
Readers field	Restricts read access to documents that are created with the form that contains the Readers field. A Readers field contains a list of users who are allowed to have read access to the document. If a Readers field is editable, the author of a document can specify who can read the document.
Authors field	Lets users who have Author access in the ACL edit a document that they didn't create. An Authors field often includes the name of the creator. If you create a document and don't list yourself as author, you can't subsequently modify the document. Authors fields contain names of users and are created in the same way as Readers fields. Users with Editor access or above can edit a document even if they are not in the Authors field. Users with No access, Depositor access, or Reader access cannot edit a document, even if they are listed in the Authors field, unless it's a public document.
Section	Collapses or expands information in and controls access to parts of a document. The section can be hidden based on the user's current mode (Read, Edit, and so on), the type of client a user has (browser or Notes), or a formula. Hiding a section does not protect the section from updates by agents, actions, or access by another form.
Field	Fields on the form can be encrypted with one or more keys. To decrypt and read the contents of the field, the user must have one of the keys. Application developers can specify that a field be changed only by users who have Editor access or above in the ACL. This prevents users with Author or Depositor access from changing the field on the form. Field encryption does not hide information in the full-text index files created by someone who has access to the field. In addition, full-text search returns hits on the encrypted fields, even if the user cannot access the contents of the field.
Paragraph	Paragraphs of text can be hidden based on the user's current mode (Read, Edit, and so on), the type of client a user has (browser or Notes), or a formula. This hides the paragraph from viewing. It does not protect the paragraph from updates by agents, actions, or access by another form.
Layout region	Layout regions are areas of grouped objects that can be easily moved and displayed in ways that are not possible with the use of forms and subforms. A layout region can be hidden based on the user's current mode (reading, editing, and so on) or a formula. Hiding a layout region does not protect it from updates by agents, actions, or access by another form.
Sign-enabled field	Sign-enabled fields on a form allow a digital signature to be attached when a document is saved or mailed. Digital signatures verify that authors are who they say they are and guarantee that the information in the document has not been tampered with. The private key in a user ID file generates the signature. When a user opens the document containing the signed field, Notes verifies the signature by comparing it with the author's public key in the Domino Directory.

Using access control features for security

It is important to distinguish between true security features and access control features that it make it difficult for users to access information. Some access control features can be circumvented by experienced users and should not be used as true security measures. These features are useful, however, because they can make an application easier to use by preventing accidental operations.

The following table describes the security abilities of different access control features.

Feature	Provides security in Notes	Provides security on the Web
ACL	Yes	Yes
Authors fields	Yes, for users who have Author access in the ACL.	Yes, for users who have Author access in the ACL.
Controlled access sections	No	No
ECL	Yes	Not applicable
Enforce consistent ACLs	No	Not applicable
Field encryption	Yes	Not applicable
Form access lists	No	No when "Generate HTML for all fields" is selected. Otherwise, Yes.
Hide-when formulas	No	No when "Generate HTML for all fields" is selected. Otherwise, Yes.
Prevent copying and forwarding	No	Not applicable
Public access to documents	Yes	Yes
Reader access lists and Readers fields	Yes	Yes
Signing	Yes	Not applicable
View access lists	No	Yes

Notes/Domino authentication

The basis on which all Notes/Domino security is built is user authentication. Authentication is important because it allows you to differentiate one user from another; without it, you could not identify whether users are who they claim to be. It is, therefore, the key to providing restricted access to Notes and Domino resources.

The Notes authentication procedure depends on a certificate, which is an electronic stamp added by a certifier to a Notes ID file. Every Notes client uses a Notes ID file for identification.

Certification and authentication are complex processes. Familiarizing yourself with the following terms will help you further understand these processes.

Public key encryption

Public key encryption is also referred to as asymmetric encryption. With public key encryption, a user has a key pair — private and public. The public key is distributed to everyone with whom you want to communicate. In Domino, the public key is published in the Domino Directory.

Symmetric encryption

Symmetric encryption, often referred to as secret key encryption, uses a common key and the same mathematical algorithm to encrypt and decrypt a message. If two people want to communicate securely with each other, both need to agree on the same mathematical algorithm to use for encrypting and decrypting data. They also need to have a common key, the secret key.

Digital signatures

A digital signature is the electronic equivalent of a handwritten signature — a unique block of text that verifies your identity — that is appended to a message. It can be used to confirm the identity of the sender and the integrity of the message. The block of text is encrypted and decrypted using public and private keys.

Public key certificates

A certificate is a unique electronic stamp stored in a Notes or Domino ID file that associates a name with a public key. Certificates permit users and servers to access specific Domino servers. An ID may have many certificates.

SSL certificates let servers with certificates created by Domino certificate applications exchange certificates easily between Domino and other applications. SSL certificates contain a public key, a name, an expiration date, and a digital signature, and are stored in files called key ring files. Key ring files are password protected and store one or more certificates on the client and server hard drives. Public and private keys are a unique pair of mathematically-related keys that are used to initiate SSL-encrypted transactions.

Hierarchical naming

Hierarchical naming is a system of naming associated with Notes IDs that reflects the relationship of names to the certifiers in an organization. The format of a hierarchical name is:

common name/organizational unit/organization/country code

For example, Sarah Forbes/ Toronto/Acme/CA

All hierarchical names include common name and organization components. An organizational unit (of which there can be a maximum of four) and country code components are optional.

Hierarchical naming helps distinguish users with the same common name for added security and allows for decentralized management of certification.

Notes ID files

When registering a user, the administrator specifies the user name, password, expiration date, and other default options. The registration process creates an ID for the user or server and places it in the Domino Directory and/or in a file that must be given to the user to reside on the user's workstation.

Types of ID files

Essentially, Notes IDs store certificates and encryption keys. There are three different types of Notes IDs: user, server, and certifier. A user ID is the Notes ID for a Notes client, and a server ID is the Notes ID for a Domino server. A certifier ID is treated differently because it is only used for certifying other Notes IDs and cannot be used to run a workstation or server. Therefore, a certifier ID is more important than other IDs for the organization because so much of Notes security is based on signatures and authentication, and both use certificates. The certifier ID should be stored on a disk in a safe place, not on a server hard disk where unauthorized users may be able to access it. Any untrustworthy person that gains access to the certifier ID can easily access any system set up to trust that certifier.

Contents of a Notes ID

When an administrator attempts to register a new user or server, the Domino Administrator generates two RSA key pairs (public-private keys) for that entity. One 512-bit key length pair is used for data encryption in countries outside the US and Canada. Another 630-bit key length pair is used for data encryption within US and Canada and for signatures and authentication worldwide. The Domino Administrator then builds a certificate using the certifier's private key to sign the certificate. The signed certificate is then placed in the Notes ID file.

After the registration process, the ID file contains:

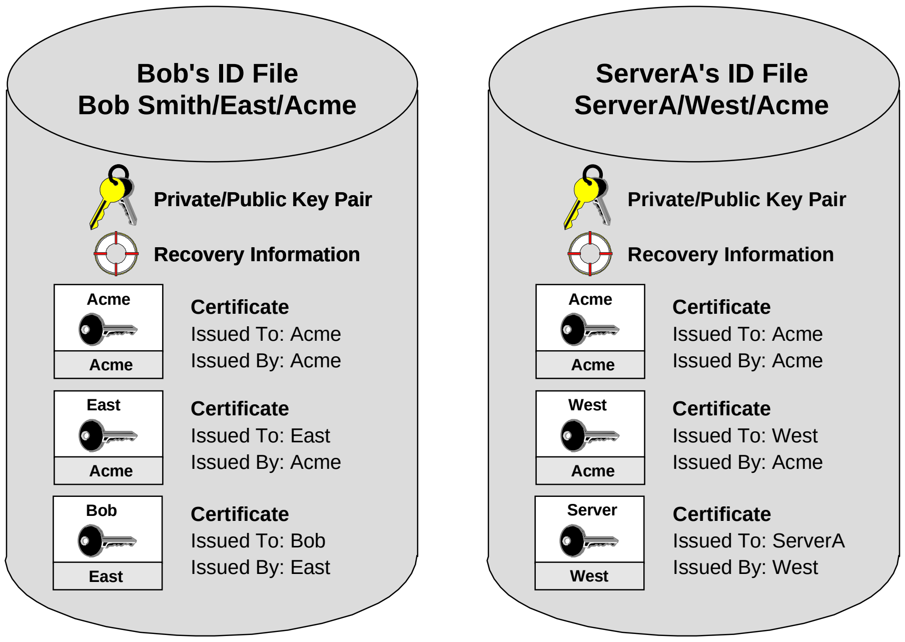
- The user's name and Notes license number
- Two public and private key pairs
- Two certificates for the user
- A certificate for each ancestor certifier
- (Optional) Recovery information for the ID file

Public and private keys are mathematically related and uniquely identify a user. Information encrypted with a public key can be decrypted only with the private key of the pair. The big advantage to public key encryption is that it does not matter who has access to the public key because it is useless without the private key. Therefore, you can make the public key available without compromising security.

Notes stores recovery information in the Notes ID file. Using the recovery information, the administrator can recover an ID file if the user forgets the password or to restore an ID file from an encrypted backup if the ID file becomes corrupted.

After registration, encryption keys distributed by application developers may be added to the ID file to allow encryption and decryption of fields in documents. The private key and the encryption keys in the ID file are encrypted using a key computed from the user's password, so that only the owner can use the ID file. Public information such as the user's name and public key are not encrypted.

The following figure shows the ID files for Bob and ServerA.



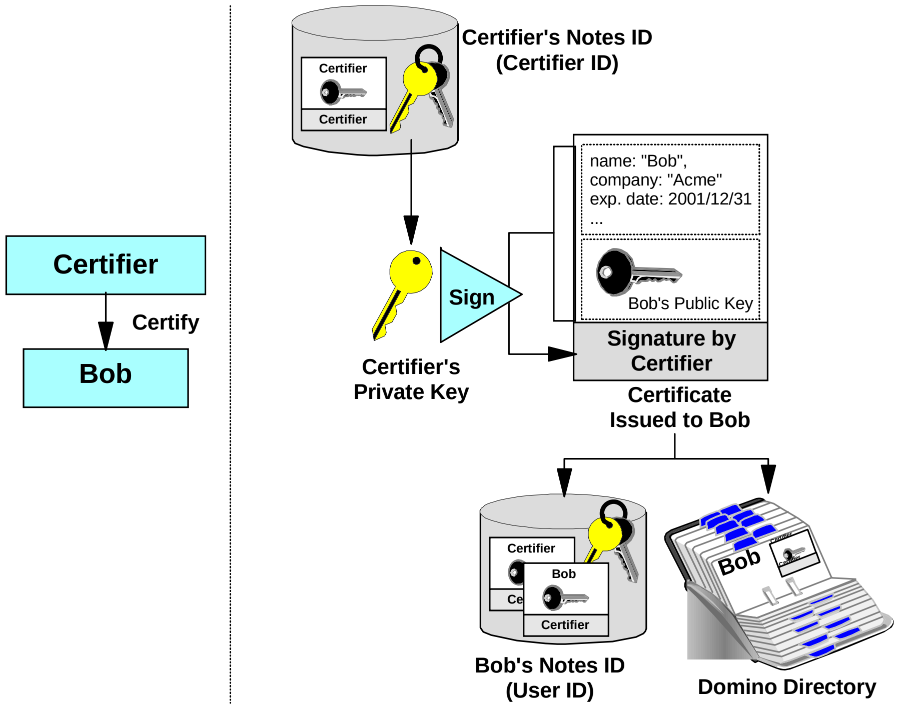
Certificates

A certificate contains:

- The certificate owner's name
- The certificate owner's public key
- The certifier's name
- The certifier's public key
- The certificate expiration date
- A digital signature by the certifier using the certifier's private key. This signature proves the certificate's authenticity.

The certificate is stored in a Notes ID file and the Domino Directory. The certificates registered to the Domino Directory are referred to by all users and servers belonging to the Domino domain when they attempt to encrypt or sign mail messages or document data. Note that the certificate itself does not contain any secret information; it is therefore open to the public and can be distributed anywhere.

The following figure shows how certificates are placed in ID files and the Domino Directory.



Example of Notes/Domino authentication

Authentication is a two-step process: before a Domino server can authenticate a Notes client or Domino server, it must validate the client or server. Validation is the process of reliably determining the sender's public key. Domino uses the following rules when deciding to trust a public key:

Rule 1
: Trust the public key of any of the validating server's ancestors in the hierarchical name tree because they are stored in your ID file.

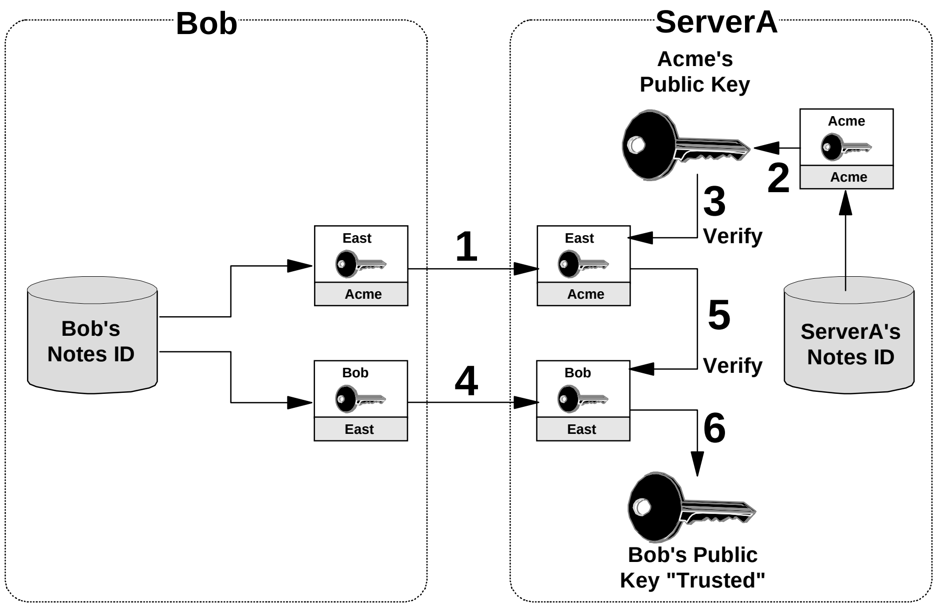
Rule 2
: Trust any public key obtained from a valid certificate issued by any of your ancestors in the hierarchical name tree.

Rule 3
: Trust any public key certified by any trusted certifier and belonging to one of the certifier's descendants.

If the public key is not certified by one of your ancestors, then Domino establishes trust using a cross-certificate .

Phase 1 -- Public key validation

Let us now see how these rules are applied in the validation process. The user ID file for Bob Smith contains everything he needs to identify himself and establish his credentials. When he requests a session with a server, the first step is to send to the server all of the certificates from the ID file (both the user's own certificate and the chain of certifiers' certificates that support it). The figure below illustrates the validation process that follows. Note that Acme is a common ancestor for both Bob and Server A.



The numbered steps in the figure are described as follows:

Step 1

ServerA reads the East certificate that Bob Smith sent from his ID file. This was signed by Acme. ServerA is interested in it because East is the certifier of Bob's certificate.

Step 2

ServerA reads the Acme public key from its own ID file. (According to Rule 1, ServerA will trust the public key of any ancestor that is stored in its ID file.)

Step 3

ServerA uses the public key of Acme (which is trusted because it is in the server's ID file) to verify that the certificate of East/Acme is valid. (According to Rule 2, you trust your ancestor's authority to issue certificates to anyone.)

Step 4

ServerA reads the certificate that was sent from Bob Smith's ID file. This was signed by East.

Step 5

ServerA uses the public key of East/Acme, which now is trusted, to verify that the Bob Smith/East/Acme certificate is valid. (According to Rule 3, trust any public key certified by any trusted certifier and belonging to one of the certifier's descendants.)

Step 6

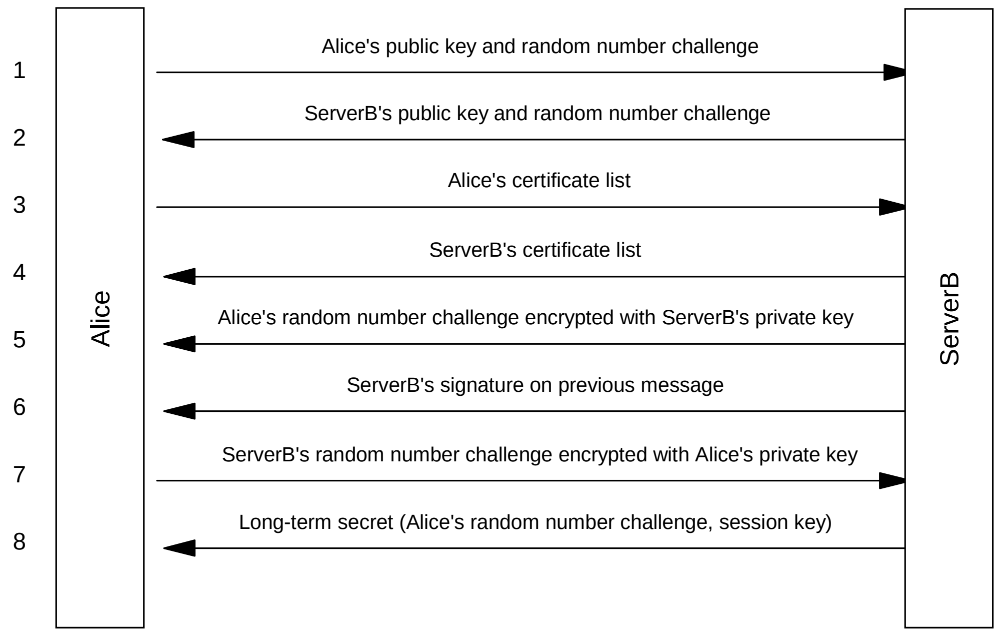
ServerA has now reliably learned Bob Smith's public key.

The same process is followed in reverse so that Bob can reliably learn ServerA's public key.

Phase 2 -- Authentication

After the validation process finishes, the authentication process begins. Authentication is a proof of identity. The validation process described above has not completely proved who each of the session partners is because all they have presented so far is certificates. A certificate associates the user with a public key and tells the recipient that the public key can be trusted, but in order to prove that user is really who they claim to be, they must show that they hold the private key that matches the public key in the certificate. The authentication process achieves this with a challenge/response dialog between a workstation and a server, or between two servers when either is running database replication or mail routing.

To continue the previous example of Bob Smith accessing ServerA, see the following figure. Although the actual implementation is complex, a simplified description follows.


Alice
ServerB
1 Alice's public key and random number challenge
2 ServerB's public key and random number challenge
3 Alice's certificate list
4 ServerB's certificate list
5 Alice's random number challenge encrypted with ServerB's private key
6 ServerB's signature on previous message
7 ServerB's random number challenge encrypted with Alice's private key
8 Long-term secret (Alice's random number challenge, session key)

Steps 1 and 2
: Alice and ServerB exchange public keys and give each other a random number challenge.

Steps 3 and 4
: Alice and ServerB exchange the complete set of certificates from their root down.

Steps 5 and 6
: ServerB sends Alice their long-term shared secret key, encrypted with Alice's public key and signed with ServerB's private key, to prove it's ServerB sending the message (Alice takes ServerB's word for it on their shared secret key.)

Step 7
: Alice proves its identity. Alice needed to know its private key in order to extract the long-term secret, which it then uses to encrypt ServerB's challenge.

Step 8
: ServerB proves his identity and securely sends Alice a session key they will share for this one conversation.

To summarize, there are three key points to remember about Notes authentication:

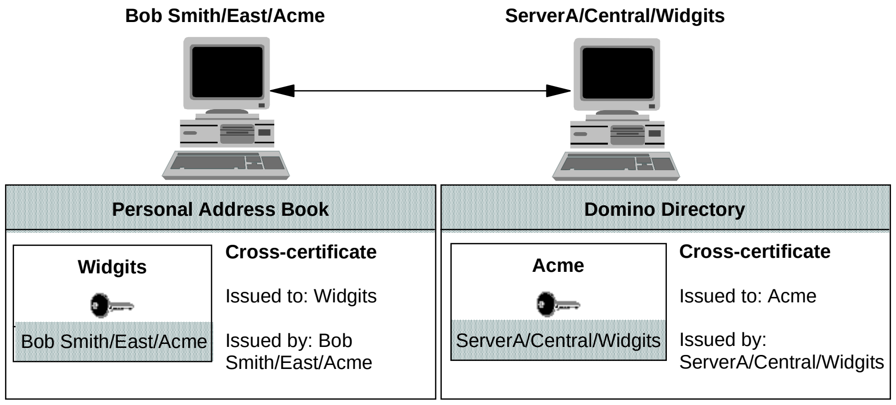
1. Notes authentication is a two-way procedure.
2. Notes authentication avoids some operations on subsequent authentications. That is, if the parties have exchanged and verified each other's certificates during a previous connection, then some steps in the authentication process — namely, Steps 3 — 6 in the example — do not occur. This speeds up processing time on subsequent authentications.
3. Notes authentication establishes a session key that can be used to encrypt the messages that follow authentication.

Cross-certificates

Cross-certificates are used to establish trust between different hierarchically certified organizations. Typically, users establish trust by determining that they have an ancestor in common. In different hierarchically certified organizations, users do not have a common ancestor; therefore, users need cross-certificates to establish trust.

Cross-certificates are stored in documents in the user's Personal Address Book and the server's Domino Directory. Cross-certificates have "Issued by" and "Issued to" fields. "Issued by" indicates who trusts this certificate. If the certificate is issued by Bob Smith/East/Acme, only the user with this Notes ID name will trust this certificate. "Issued to" indicates the certificate that Notes should trust. A certificate that is issued to /Acme trusts all users and servers in the hierarchy that includes /Acme — for example, /East/Acme and /West/Acme. A certificate that is issued to /East/Acme trusts a smaller scope of users and servers — for example, /East/Acme and /Sales/East/Acme, but not /West/Acme.

For example, user Bob Smith/East/Acme wants to access ServerA/Central/Widgets. /East/Acme and /Central/Widgets are not in the same hierarchy and, therefore, cannot be trusted. The server establishes trust by obtaining a cross-certificate for /Acme, and Bob establishes trust by obtaining a cross-certificate for /Widgets. ServerA could also establish trust by obtaining a cross-certificate issued to /East/Acme, and Bob could establish trust by obtaining a cross-certificate issued to /Central/Widgets; however, doing so may limit the scope of other servers and users that can be trusted with that cross-certificate.



Cross-certificates are also used to establish trust for digital signatures and for encrypted S/MIME messages that are sent over the Internet. In the case of digital signatures, only the recipient of the document with the digital signature needs to have a cross-certificate for the signer. In the case of encrypted Internet messages, the user who sends the message needs the cross-certificate in order to encrypt the message.

Internet client authentication

There are two authentication methods for Internet clients: name-and-password and SSL.

Name-and-password authentication

There are two types of name-and-password authentication:

- Basic name-and-password authentication
- Session-based name-and-password authentication

Name-and-password authentication, also known as basic password authentication, uses a basic challenge/response protocol. Whenever users try to access restricted information, they are prompted for a name and password. Name-and-password authentication on a TCP/IP port is not very secure, however. Information, including the name and password, passes between the client and server unencrypted. Anyone who has access to a network sniffer or trace tool can capture the information anywhere along the session path. To prevent this type of attack, name-and-password authentication can be used on an SSL port.

When setting up name-and-password authentication for an HTTP server, the administrator can use session-based name-and-password authentication. Session-based name-and-password authentication offers greater control over user interaction than basic name-and-password authentication. The administrator can customize the form in which users enter their name and password information, and users can log out of the session without closing down the browser.

SSL authentication

SSL authentication is similar to Notes authentication. The SSL client presents a certificate to a Domino server. Then the server uses that certificate to authenticate the client, and vice versa. Unlike Notes and Domino, SSL does not require that both the server and client authenticate each other. If SSL is enabled, Domino requires that the client authenticate the server; however, it is optional for the server to authenticate the client.

SSL uses an Internet certificate in X.509 format, which is an industry standard format that most applications, including Domino, understand.

An Internet certificate typically consists of the following fields:

- Version of certificate format
- Certificate serial number
- Digital signature algorithm identifier (for issuer's digital signature)
- Issuer name (that is, the name of the Certification Authority)
- Validity period
- Subject name — that is, user or server
- Subject public-key information: algorithm identifier and public-key value
- Issuer unique identifier
- Subject unique identifier
- Extensions
- Digital signature by issuer on the above fields

Internet certificates can be issued by a third-party certificate authority (CA), such as VeriSign, or the administrator can use the Domino Certificate Authority application to set up a CA.

For more information on X.509 certificates, see the IBM Redbook, *Lotus Notes and Domino R5.0 Security Infrastructure Revealed*, which can be viewed in its entirety and/or ordered in hardcopy by visiting the Web site /www.redbooks.ibm.com/booklist.html.

Supported authentication methods for Internet protocols

The following table lists the Internet protocols that Domino supports and the type of authentication that can be used for each protocol.

	Anonymous	Basic name-and-password	Session name-and-password	SSL	SSL client certificates
HTTP	Yes	Yes	Yes	Yes	Yes
IIOP	Yes	Yes	No	Yes	No
IMAP	No	Yes	No	Yes	Yes
LDAP	Yes	Yes	No	Yes	Yes
NNTP	Yes	Yes	No	Yes	Yes
POP	No	Yes	No	Yes	Yes
SMTP Inbound	Yes	Yes	No	Yes	No
SMTP Outbound	No	No	No	No	No

Basic name-and-password authentication

Name-and-password authentication, also known as basic password authentication, uses a basic challenge/response protocol to ask users for their names and passwords and then verifies the accuracy of the passwords by checking them against those stored in Person documents in the Domino Directory. When set up for this, Domino asks for a name and password only when an Internet/intranet client tries to perform a task for which access is restricted. Internet/intranet access differs from Notes client and Domino server access in that a Domino server asks a Notes client or Domino server for a name and password when the client or server initially attempts to access the server.

Before a database manager can use Domino security to assign database access to an Internet/intranet client, the system administrator must create a Person document for that client in the Domino Directory. Clients who do not have Person documents are considered Anonymous and can access only those servers and databases that allow Anonymous access.

Name-and-password authentication allows Domino to locate the Person document for the client accessing the server (if one exists). Domino uses the Person document to identify the client. After the client is identified, access to databases can be determined.

Some Internet protocols can use name-and-password authentication with SSL. Then the system prompts the user for a name and password before using SSL to secure the session.

Session-based name-and-password authentication

A session is the time during which a Web client is actively logged onto a server. Session-based name-and-password authentication includes additional functionality that is not available with basic name-and-password authentication. The Server document in the Domino Directory contains settings that enable and control session authentication.

Customized HTML log-in form

An HTML log-in form allows a user to enter a name and password and then use that name and password for the entire user session. The browser sends the name and password to the server in the server's character set; therefore, users can enter a name and password in character sets other than ASCII or Latin-1.

Domino provides a default HTML form, which is created and configured in the Domino Configuration database (DOMCFG.NSF) and which can be customized to contain additional information.

Default log-out time period

A default log-out time period logs the Web client off the server after a specified period of inactivity. Automatically logging a user off the server prevents others from using the Web client to impersonate a user if a Web client leaves the workstation before logging off. Users can also append ?logout at the end of a URL to log off a session.

Maximum user sessions

To manage performance, the administrator can adjust the maximum number of user sessions allowed on the server at the same time

Examples of name-and-password authentication

The following example shows how a client (Bob) authenticates with a server (ServerA) using a name and password on a TCP/IP port.

HTTP

1. Bob attempts a task for which access is restricted.
2. ServerA checks whether anonymous access is allowed on the server for the protocol. If it is, the following happens:
   a. The server checks whether there is an entry named Anonymous in the database ACL. If there is, then Bob is allowed anonymous access to the database using the Anonymous access level.
   b. If there is no entry named Anonymous, the server checks the -Default- access.
   c. If the -Default- access is Reader or higher, then Bob is allowed anonymous access to the database using the -Default- access level.
3. If anonymous access is not allowed and name-and-password access is enabled, the client displays a request for a user name and password.
4. Bob enters his name and password.
5. The server checks for a Person document in the Domino Directory for Bob and verifies that the name entered is in the User name field and the password presented is in the Internet password field.
6. If the name and password are verified, Domino uses the first name listed in the User name field to identify Bob.

IIOP

1. The Java applet that Bob is using attempts a task and passes to the server either a name and password or tries to connect anonymously.
2. ServerA checks whether name-and-password or anonymous access are allowed on the server, depending on what the Java applet passed to the server.
3. If access is allowed, the Java applet connects. If it is not allowed, the Java applet does not make the connection and Bob receives an error.

   Unlike HTTP, IIOP does not fail over to another type of connection. For example, if an applet tries to connect using name-and-password and name and password connections are not allowed on the server for that protocol, the applet is not allowed access using anonymous access.

LDAP

1. Bob has configured his LDAP client to connect to ServerA using name-and-password authentication.
2. Bob's name and password are retrieved from the LDAP client's configuration settings — for example, the account record in the case of the Domino client — and sent on the LDAP bind request.
3. The server checks for a Person document in the Domino Directory for Bob and verifies that the name entered is in the User name field and the password presented is in the Internet password field.
4. If the name and password are verified, Domino uses the first name listed in the User name field to identify Bob.

POP3

1. The client establishes a connection to the POP3 server on the port.
2. ServerA checks whether name-and-password access is enabled. If so, ServerA puts the client into the POP3 "greeting" state.

3. ServerA waits for Bob to enter his name and password. After Bob sends his name and password, ServerA checks the User name field in the Person document for the name Bob entered and makes sure the password entered matches the password in the Internet password field. The server also checks secondary address books and LDAP directories if Domino is configured to search secondary address books and LDAP directories. If the password matches, ServerA opens the database specified in the Mail File field of Bob's Person document.

SSL authentication

If SSL is enabled for a protocol, clients must authenticate the server. Optionally, servers can authenticate the clients. Domino lets you set up SSL on a protocol-by-protocol basis. For example, you can require SSL for clients that use HTTP to connect to the server, but not require SSL for LDAP sessions.

Server authentication

If the Internet client that connects to a server has a trusted certificate in common with the server, then the Internet client can be confident that the server is who it says it is. Only the server requires an Internet certificate. The client needs to establish trust for that certificate but does not require an Internet certificate. To control what a user can see on a secure site, however, the administrator must manage user names and passwords for each user or require users to use client authentication to connect.

Clients obtain trusted certificates from a Certificate Authority (CA). The CA serves as a third party that indicates that the server certificate can be trusted.

From the point of view of an Internet client that is using SSL to connect to a server, the whole negotiation and authentication process is transparent. For example, for a Web client to establish an SSL connection, the URL prefix must change from http:// to https://. After the SSL connection is established, the browser gives the user a visual indication — for example, in Netscape Navigator a closed padlock appears in the lower left corner of the screen.

Client authentication

With client authentication, Notes and Domino take authentication one step further. Client authentication lets administrators authenticate Internet clients accessing the server and control access based on the client identity. To authenticate with the server, the Internet client uses an Internet certificate that is signed by a CA. Using client authentication, administrators do not need to manage passwords for these users because the CA vouches for their authenticity.

Examples of SSL authentication

The following example shows how a client (Bob) authenticates with a server (ServerA) on an SSL port.

HTTP

1. Bob attempts a task for which access is restricted.
2. The client sends a request to ServerA specifying information about the SSL connection, such as supported encryption algorithms and certificate expiration date.
3. ServerA sends the client the certificate that contains ServerA's public key.
4. The client checks the CA's digital signature on ServerA's certificate against a trusted certificate on the client to validate the identity of ServerA.
5. If ServerA's CA certificate is trusted, Bob uses an algorithm to create a session key, uses the public key stored in ServerA's certificate to encrypt the key, and sends it to ServerA. To ensure security and prevent an unauthorized user from tracing the key, the session key changes for each session.

6. ServerA uses ServerA's private key to decrypt the session key and uses the session key to encrypt data passed between Bob and ServerA after this point.
7. If client authentication is enabled, the following happens:
   a. ServerA requests Bob's certificate.
   b. The client sends ServerA Bob's certificate.
   c. ServerA checks the CA's digital signature on Bob's certificate to verify the identity of Bob. If ServerA has Bob's CA certificate marked as a trusted root in the server certificate, then ServerA knows that Bob's certificate is valid.
   d. ServerA checks the User name field in the Person document for the common name on Bob's certificate and verifies that the public key in Bob's certificate matches the public key in his Person document. ServerA checks the primary address book for the Person document. ServerA also checks secondary address books and LDAP directories if the user is a Web client and Domino is configured to search secondary address books and LDAP directories.
8. If ServerA cannot validate Bob for client authentication, ServerA checks whether anonymous access is allowed on the server for the protocol. If it is, the following happens:
   a. The server checks whether there is an entry named Anonymous in the database ACL. If there is, then Bob is allowed anonymous access to the database using the Anonymous access level.
   b. If there is no entry named Anonymous, the server checks the -Default- access.
   c. If the -Default- access is Reader or higher, then Bob is allowed anonymous access to the database using the -Default- access level.
9. If anonymous access is disabled for the protocol or if the database ACL does not allow anonymous access, the server checks whether name-and-password access is allowed on the server for the protocol. If name-and-password access is enabled, the following happens:
   a. The server asks Bob to enter his user name and password.
   b. The server checks the User name field in the Person document for the user name Bob entered and verifies that the password that Bob entered matches the password in the Internet password field. The server also checks secondary address books and LDAP directories if Domino is configured to search secondary address books and LDAP directories.
   c. If the password matches, the server checks the database ACL for the first entry listed in the User name field in Bob's Person document and Bob gains access to the database using the access provided for that user name.

IIOP

1. When a user visits a Web page that contains an applet that uses IIOP for access to the Domino objects, the applet, the lotus.domino classes (ncso.jar), and ServerA's root certificate are loaded into the browser (using http or https).
2. The applet sends a request to ServerA specifying information about the SSL connection, such as supported encryption algorithms and certificate expiration date.
3. The applet uses an algorithm to create a session key, uses the public key stored in ServerA's certificate to encrypt the key, and sends it to ServerA. To ensure security and prevent an unauthorized user from tracing the key, the session key changes for each session.
4. ServerA uses its private key to decrypt the session key and uses the session key to encrypt data passed between the applet and ServerA after this point.
5. If access is allowed, the Java applet connects. If it is not allowed, the Java applet does not make the connection and Bob receives an error.
6. After the SSL connection is made with the server, the applet passes to the server either a name and password or tries to connect anonymously.
7. ServerA checks whether name-and-password or anonymous access is allowed on the server, depending on what the Java applet passed to the server.

LDAP

1. Bob has configured his LDAP client to connect to ServerA using an SSL encrypted port (636).
2. The client sends a request to ServerA specifying information about the SSL connection, such as supported encryption algorithms and certificate expiration date.
3. ServerA sends the client the certificate that contains ServerA's public key.
4. The client checks the CA's digital signature on ServerA's certificate against a trusted certificate on the client to validate the identity of ServerA.
5. If ServerA's CA certificate is trusted, Bob uses an algorithm to create a session key, uses the public key stored in ServerA's certificate to encrypt the key, and sends it to ServerA. To ensure security and prevent an unauthorized user from tracing the key, the session key changes for each session.
6. ServerA uses ServerA's private key to decrypt the session key and uses the session key to encrypt data passed between Bob and ServerA after this point.
7. If client authentication is enabled, the following happens:
   a. ServerA requests Bob's certificate.
   b. The client sends ServerA Bob's certificate.
   c. ServerA checks the CA's digital signature on Bob's certificate to verify the identity of Bob. If ServerA has Bob's CA certificate marked as a trusted root in the server certificate, then ServerA knows that Bob's certificate is valid.
8. The client sends an LDAP bind request to ServerA. The client can specify both the mechanism of bind and a set of credentials to use with that mechanism. The mechanism can be simple — that is, name and password — SASL, or unsupported.
   a. Simple
      - If the client did not specify any credentials, then he is identified as "anonymous." ServerA checks whether anonymous access is allowed on the LDAP SSL port. If anonymous access is allowed, clients who want to modify the database are restricted to access privileges granted by "Anonymous" in database ACLs. For search operations, the client is restricted to the set of attributes selected in the Domain Configuration document under the "Anonymous users can query" section. If anonymous access is not allowed on the port, ServerA returns an "Invalid credentials" response and discontinues the bind process.
      - If the client specifies a name and password, the server checks the User name field in the Person document for the user name Bob entered and verifies that the password that Bob entered matches the password in the Internet password field. The server also checks secondary address books and LDAP directories if Domino is configured to search secondary address books and LDAP directories. If the password matches, the server checks the database ACL for the first entry listed in the User name field in Bob's Person document, and Bob gains access to the database using the access provided for that user name. If name-and-password access is not allowed on the port, ServerA returns an "Invalid credentials" response and discontinues the bind process. If the client does not provide a password, ServerA returns an "Inappropriate authentication" response and discontinues the bind process.

   b. Simple Authentication and Security Layer (SASL)

      SASL describes a method of adding authentication support to connection-based protocols. Each protocol that uses SASL includes a command for identifying and authorizing a user to a server and for optionally negotiating a security layer for subsequent protocol interactions.
      - If the client specifies a SASL mechanism other than "EXTERNAL," ServerA returns the LDAP response "Unsupported SASL mechanism" and discontinues the bind process.
      - If the client specifies the "EXTERNAL" SASL mechanism, ServerA confirms that the client is using an SSL connection. If not, ServerA returns the LDAP response "SASL EXTERNAL requires a secure connection be previously established" and discontinues the bind process.

- If the client specifies the "EXTERNAL" SASL mechanism on a secure connection, ServerA checks the User name field in the Person document for the common name on Bob's certificate and verifies that the public key in Bob's certificate matches the public key in his Person document. The server also checks secondary address books and LDAP directories if Domino is configured to search secondary address books and LDAP directories. If his certificate is not found or it does not match the one Bob provided, then Bob is treated as an anonymous user (see Step 8a above). Otherwise, ServerA uses the name from the certificate for database ACLs.

c. Unsupported

- If the client specifies a Berbers or other unsupported or unknown mechanism, ServerA returns the LDAP response "Authentication method not supported" and discontinues the bind process.

POP3

1. The client establishes a connection to the POP3 server on the SSL port.
2. The client sends a request to ServerA specifying information about the SSL connection, such as supported encryption algorithms and certificate expiration date.
3. ServerA sends the client the certificate that contains ServerA's public key.
4. The client checks the CA's digital signature on ServerA's certificate against a trusted certificate on the client to validate the identity of ServerA.
5. If ServerA's CA certificate is trusted, Bob uses an algorithm to create a session key, uses the public key stored in ServerA's certificate to encrypt the key, and sends it to ServerA. To ensure security and prevent an unauthorized user from tracing the key, the session key changes for each session.
6. ServerA uses ServerA's private key to decrypt the session key and uses the session key to encrypt data passed between Bob and ServerA after this point.
7. If client authentication is enabled, the following happens:

   d. ServerA requests Bob's certificate.

   e. The client sends ServerA Bob's certificate.

   f. ServerA checks the CA's digital signature on Bob's certificate to verify the identity of Bob. If ServerA has Bob's CA certificate marked as a trusted root in the server certificate, then ServerA knows that Bob's certificate is valid.
8. If client certificates are enabled, then ServerA checks the User name field in the Person document for the common name on Bob's certificate. ServerA verifies that the public key in Bob's certificate matches the public key in his Person document. The server also checks secondary address books and LDAP directories if Domino is configured to search secondary address books and LDAP directories. If Bob's certificate is found and is valid, then the client enters the POP3 "greeting" state. If Bob's certificate is not found or is not valid, then the client enters the "terminal" state and the connection is dropped from the server.
9. If name-and-password access is enabled, the client enters the POP3 "greeting" state.
10. If client certificates and name-and-password access are not enabled, ServerA does not accept the client's connection, and the client enters the "terminal" state. Anonymous access is not allowed on POP3.

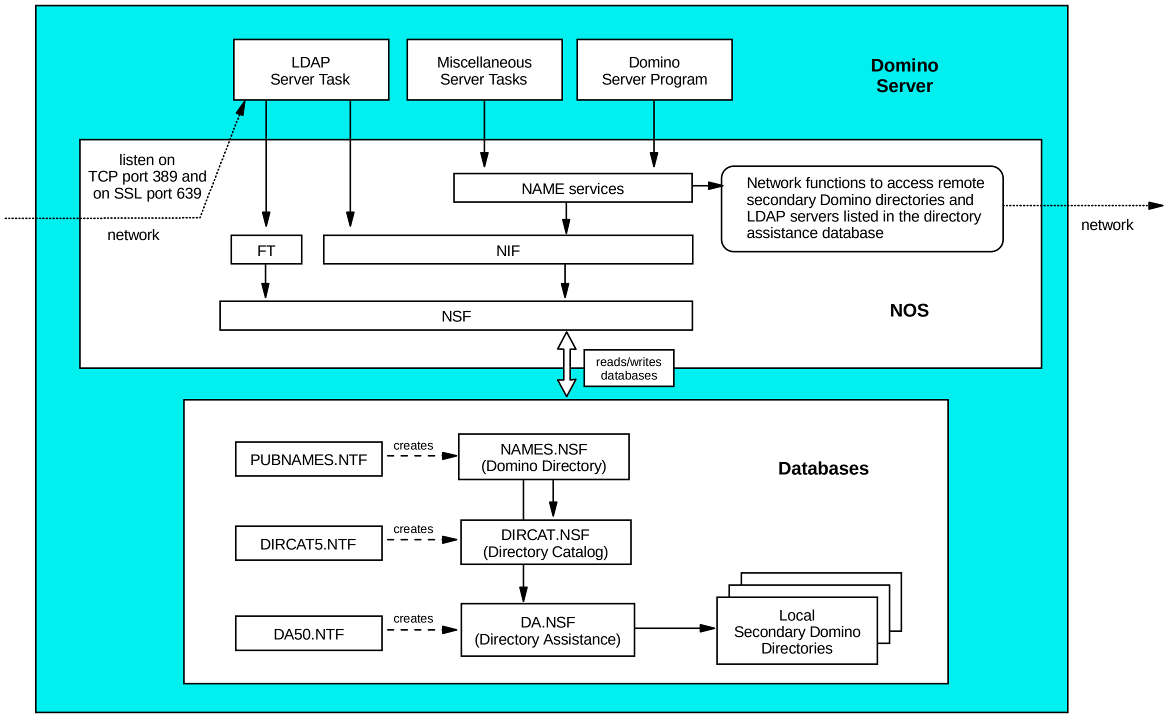
If the client is in the "greeting" state, ServerA waits for Bob to enter his name and password. After Bob sends his name and password, ServerA checks the User name field in the Person document for the name Bob entered and makes sure the password entered matches the password in the Internet password field. The server also checks secondary address books and LDAP directories if Domino is configured to search secondary address books and LDAP directories. If the password matches, ServerA opens the database specified in the Mail File field of Bob's Person document.

Chapter 7
Directories

This chapter describes, in technical detail, how Domino creates and uses directories.

Directories

Domino can process requests for directory information received through NRPC Name service calls and through Lightweight Directory Access Protocol (LDAP) operations. The following figure shows the server tasks, services, and databases involved with processing these requests.


LDAP Server Task
Miscellaneous Server Tasks
Domino Server Program
Domino Server
listen on TCP port 389 and on SSL port 639
network
NAME services
Network functions to access remote secondary Domino directories and LDAP servers listed in the directory assistance database
network
FT
NIF
NSF
NOS
reads/writes databases
PUBNAMES.NTF
creates
NAMES.NSF (Domino Directory)
Databases
DIRCAT5.NTF
creates
DIRCAT.NSF (Directory Catalog)
DA50.NTF
creates
DA.NSF (Directory Assistance)
Local Secondary Domino Directories

Databases used for directory lookups

Domino can look up directory information in three databases: the Domino Directory database, a directory catalog, and a directory assistance database. The first Domino server installed in a Notes domain automatically creates the Domino Directory database (NAMES.NSF) from the PUBNAMES.NTF template. A replica of the Domino Directory is automatically created on each additional server added to the domain. NAMES.NSF stores information about users and groups in the domain and about Domino configuration. When looking up directory entries, a server always searches its NAMES.NSF database before searching the other databases.

A directory catalog is a database created manually from the template DIRCAT5.NTF and populated by the Directory Cataloger (Dircat) server task. A directory catalog contains abbreviated user, group, mail-in database, and resource directory entries from one or multiple Domino Directories in a single, lightweight, quick-access directory. A directory catalog is small enough for mobile Notes users to store locally so they can address mail to anyone in an organization when disconnected from the network. A directory catalog also makes it easy to do directory lookups in organizations that use multiple Notes domains. A server can locate its directory catalog by looking up the directory catalog file name in NAMES.NSF, as the bottom box in the above figure illustrates.

A directory assistance database is created manually from the DA50.NTF template. A directory assistance database acts as a directory of secondary directories — directories other than a server's primary Domino Directory (NAMES.NSF). Directory assistance can point to secondary Domino Directories, for example Domino Directories from other Notes domains, stored locally or on remote Domino servers. Directory assistance can also point to LDAP directories on remote LDAP servers, for example, third-party LDAP directories. A server locates its directory assistance database by looking up the directory assistance database file name in NAMES.NSF, as the bottom box in the above figure illustrates.

Lookups using the NOS Name services

As the top box in the figure above illustrates, Domino server tasks other than the LDAP task and the Domino Server program call Name services functions to access directory information. To look up directory entries, they call NAMELookup and related functions, which in turn call NIF functions to locate directory entries and NSF functions to open, and optionally, modify, the entries. The same calls are used for lookups to the primary Domino Directory and to secondary Domino Directories. The calls are essentially the same for lookups to a directory catalog except that for certain types of directory catalog lookups — for example, Soundex lookups — the FT functions, rather than NIF functions, are used.

When using directory assistance o look up names on a remote LDAP server, NAMELookup calls are converted to LDAP search requests.

The following table describes the available Name services functions.

Name services function	Description
NAMEGetAddressBooks	Gets the list of Domino Directories used locally or on a specified server. Requires the NOTES.INI setting Name_Include_Ed=1 to include a directory catalog.
NAMEGetModifiedTime	Gets the last modified time of a Domino Directory. Doesn't apply to a directory catalog.
NAMEGetTextItem	Finds a matching name in a NAMELookup buffer and returns a specified item from it.
NAMELocateItem	Returns a specific item of a specified match of a name in a NAMELookup buffer.
NAMELocateMatchAndItem	Finds a matching name in a NAMELookup buffer and returns a specified item from it.
NAMELocateNextMatch	Returns the next match of a given name from a NAMELookup buffer.
NAMELocateNextName	Returns next batch of matches (less than 64K) from a NAMELookup buffer.
NAMELookup	Looks up name(s) in a Domino Directory or directory catalog.

LDAP directory searches

LDAP searches are similar to NAMELookup calls in that their purpose is to retrieve or manipulate entries in a directory. However, LDAP searches offer richer directory search capabilities. While NAMELookup can locate entries solely based on name, LDAP supports the use of sophisticated search filters. For example, LDAP can be used to search for all entries containing a specific attribute or attributes. In addition, LDAP supports searches based on a directory scope. For example, LDAP can be used to search only entries directly below the branch of the directory tree ou=west.

As the figure above illustrates, the LDAP server task listens for LDAP search requests over a designated TCP/IP port (by default, 389) and/or SSL port (by default, 639). The LDAP task calls FT and NIF functions to locate directory entries defined in an LDAP search operation then calls NSF functions to locate and, optionally, modify attributes (fields).

The LDAP service searches its primary Domino Directory (NAMES.NSF), a directory catalog, and secondary Domino Directories configured in its directory assistance database. The LDAP service always calls FT functions to locate entries in a directory catalog.

The Domino Directory

The Domino Directory is a database (NAMES.NSF) that Domino automatically creates from the PUBNAMES.NTF template on the first server in a Notes domain. Additional servers in a domain automatically store a replica of this database. The Domino Directory is directory of information about users, servers, groups, and other objects that administrators might add to the directory themselves — for example, printers. It is also a tool that administrators use to manage the Domino system. For example, administrators create documents in the Domino Directory to connect servers for replication or mail routing, to register users and servers, to schedule server tasks, and so on.

NAMELookup and related functions are used to access a Domino Directory. A Domino Directory is also accessible via LDAP. To process LDAP operations, the LDAP operations are translated into FT, NIF, and NSF function calls.

Directory catalogs

A directory catalog is a database that combines abbreviated user, group, mail-in database, and resource directory entries from one or more Domino Directories into a single, lightweight, quick-access directory so that users and servers can easily look up names and addresses of people throughout an organization. A directory catalog stores only the information that is important for end-user directory services and excludes other information, such as server configuration settings, that are part of a full Domino Directory.

A directory catalog is generally 80 to 100 times smaller than the combined size of the secondary Domino Directories represented in the directory catalog. For example, if the combined size of all the individual secondary Domino Directories is 3GB, the size of a directory catalog that aggregates those directories is likely to be only 30MB.

Typically an organization uses two directory catalogs, a mobile directory catalog and a server directory catalog, each of which is configured using somewhat different options. A mobile directory is replicated to Notes clients and it is used by Notes users to quickly address mail to anyone in an organization even when disconnected from the network. A server directory catalog is a directory catalog set up for use by servers so that servers in multiple-domain organizations can search for names and addresses in a single database, rather than in multiple secondary Domino Directories.

To minimize the size of a directory catalog, entries in the directory catalog include only the fields required to resolve mail addresses, although administrators can add fields. A directory catalog supplements rather than replaces the Domino Directory and the Personal Address Book.

The mobile directory catalog

Notes users that have a mobile directory catalog set up gain these benefits:

- Users who do not have a server connection can quickly look up the address of anyone in an organization.
- Laptop users can send encrypted mail. User entries in a directory catalog contain a flag indicating whether users have certificates. When a laptop user encrypts a memo, the memo is marked in the local MAIL.BOX file for “just-in-time encryption.” When the user later connects to the network and sends the mail, the client looks up the public key on a server and encrypts the mail.
- Users can address mail to groups because group names are included in a directory catalog. By default users can’t see the members of a group as they address mail however, because, by default, the group field “members” isn’t included in the directory catalog

- Users can see instantaneous address resolution when addressing a memo because type-ahead addressing — which resolves addresses as users enter names — searches a mobile directory catalog rather than a directory on a server.
- Users can search the directory catalog using Boolean search queries the way they can search a Personal Address Book. For example, if a user wants to send mail to someone by the name of Robin at the Los Angeles location but doesn't remember Robin's last name, the user can search for "First name" Robin and "Location" Los Angeles to retrieve the name from the catalog. The LDAP protocol is used for these searches.
- Users can browse and open entries in the directory catalog the way they can in their Personal Address Books.
- Users can enter the phonetic spelling of a name and use Soundex to find the exact spelling of the name.

A mobile directory catalog is a benefit to administrators because network traffic is reduced when names are searched for locally rather than on servers.

The server directory catalog

If an organization uses multiple Domino Directories — for example, if it has multiple Notes domains — servers can use the server directory catalog to look up names from these directories rather than search each full directory individually.

After searching its primary Domino Directory, a server can search a server directory catalog to:

- Look up the names of users in secondary Domino Directories on behalf of Notes users who don't use mobile directory catalogs.
- Process LDAP client search requests, if the server runs the LDAP service.
- Quickly authenticate Web browser clients who are registered in secondary Domino Directories if the server is a Domino Web server and if directory assistance is set up on the server to allow the authentication.

Although directory assistance alone can provide all of these services, typically a server uses both directory assistance and the directory catalog. A server directory catalog is not useful in organizations that use only a primary Domino Directory because the primary Domino Directory is always searched before a server directory catalog.

How a directory catalog works

Administrators create a directory catalog database manually from the Directory Catalog template (DIRCAT5.NTF) and create a configuration document in it to indicate, among other things, which secondary Domino Directories to build into the directory catalog. The Directory Cataloger (the Dircat server task) populates a directory catalog and keeps the entries in a directory catalog synchronized with the corresponding entries in the full secondary Domino Directories. When the task runs, it replicates a limited number of fields from Person, Group, Mail-in Database, and Resource documents from each secondary Domino Directory and then combines on average 200 of these abbreviated documents into a single directory catalog aggregate document. Consequently, the directory catalog uses approximately 1,000 aggregate documents to store 200,000 entries. Since the directory catalog stores fewer documents than the Domino Directory, Notes performs operations against the directory catalog very efficiently.

Databases typically use views that sort documents by a particular field to expedite searching of the field. In a directory catalog no view sorts the aggregate documents; instead, to keep the directory catalog small, the aggregate documents themselves are sorted. By default, the aggregate documents sort entries by first name then by last name. Therefore, when Notes users look up names by entering first name followed by last name, the names are very quickly located. Administrators can specify a different sort format.

When Notes users look up names without using the selected sort format or look up names using Soundex searches and when LDAP is used to search a directory catalog, full-text searching is used to locate the names. When administrators first create a directory catalog, they create the full-text index on it; but when they later replicate it for server or mobile use, the full-text index is automatically created on the replicas.

A directory catalog has three small hidden views. The $Users view contains the aggregate documents and is used for name lookups. The $Unid view contains information that the Dircat task needs to replicate the secondary directory entries to the directory catalog. The $Unid view isn't created on replicas of the directory catalog, which further reduces the directory catalog size. The $PeopleGroupsFlat view is used to display directory names when Notes users click the Address button to browse directories.

There is one visible view called Configuration that shows the document used to configure the directory catalog. There is also a "virtual" view called Users that users can open and programs can access to see the names included in the directory catalog. This view is not stored on disk but is instead created as needed.

Directory catalog configuration options

When administrators set up a directory catalog they decide:

- Which secondary Domino Directories to include in the directory catalog.
- Which entry fields to include in the directory catalog. The default field configuration includes the minimum fields required for mail addressing.
- Which sort format to use to sort entries in aggregate documents.
- Whether to allow Soundex searches. Allowing Soundex searches increases the size of the directory catalog by about 4 bytes for each entry.
- Whether to remove duplicate entries.
- Which types of group entries to include
- Whether to change the default of 255 maximum entries allowed in an aggregate document — for example, to decrease the amount to improve full-text searches of the directory catalog.
- Whether the Dircat task makes any field changes directly in the fields or stores field changes in a temporary location to reduce replication overhead.
- When to schedule the Dircat task.
- Whether to enable an agent that regularly mails status reports about the directory catalog.

Programmatic access to a directory catalog

Developers can use these methods to access a directory catalog programmatically:

- NAMELookup calls, without any modification required
- NAMEGetAddressBooks calls, if you use the NOTES.INI setting Name_Include_Ed=1.
- NIFFindByKey, NIFReadEntries, and NIFOpenNote calls.* You can't use NSFNoteOpen to open notes passed back from NIFReadEntries; you must call NIFOpenNote instead.
- LotusScript methods*
- @NameLookup function*

*Can access the Users view but not the $Users view.

Index "LDAP service" # "directory catalog"In addition, LDAP operations work against a directory catalog located on a server that runs the LDAP service.

Directory assistance

A server's primary Domino Directory is the directory in which an administrator registers the server and which is associated with the server's Notes domain. Each server stores its own primary Domino Directory under the file name NAMES.NSF. Directory assistance is a feature that enables users and servers to locate information in a directory that is not a server's primary Domino Directory. Administrators can set up directory assistance for secondary Domino Directories and for remote LDAP directories — for example, for third-party LDAP directories.

To configure directory assistance, administrators create a Directory Assistance database from the template DA50.NTF and then create Directory Assistance documents for individual directories. A Directory Assistance document defines the hierarchical naming rules that reflect the names in a directory, specifies if a directory is used for authentication, and provides the location of a directory.

Administrators replicate the Directory Assistance database to servers that should use it and enter the Directory Assistance database file name in the Domino Directory Server documents. Servers load directory information configured in the Directory Assistance database into memory.

Directory assistance for secondary Domino Directories

From the standpoint of a given server, any Domino Directory that is not the server's primary Domino Directory is considered a secondary Domino Directory. A secondary Domino Directory may be associated with another Notes domain, or it may be created manually and unaffiliated with any Notes domain.

To set up directory assistance for a secondary Domino Directory, administrators create a Directory Assistance document with "Notes" selected in the "Domain Type" field. Setting up a "Notes" Directory Assistance document allows servers to do any of the following, depending on the options chosen in the Directory Assistance document:

- Use security credentials in the directory to authenticate Web clients that use the Domino Web server
- Extend LDAP client searches to the directory
- Allow Notes users to easily address mail to users registered in the directory

One Directory Assistance document provides any of these capabilities for a specific secondary Domino Directory.

The server directory catalog and directory assistance

A server always searches its primary Domino Directory and then the directory catalog, if a directory catalog is set up on the server, before it uses directory assistance. In general, it's best to configure a secondary Domino directory in both a server directory catalog and in the directory assistance database.

The server directory catalog can often satisfy requests — for example, mail addressing requests — so that a server needs to search only the directory catalog and not individual secondary Domino Directories.

In the cases where the directory catalog doesn't store the requested information — for example, if it doesn't store all the fields required to process specific LDAP searches — a server can use information in both the directory catalog and the directory assistance database to quickly locate the complete entries in the secondary directories. Each entry in a directory catalog includes the replica ID of the Domino Directory from which the entry was derived and the unique ID (UNID) for the entry. To locate a complete secondary directory, the server matches the replica ID for the directory stored in the directory catalog with the same replica ID in the directory assistance database. To locate an entry within the directory, the server uses an entry's UNID specified in the directory catalog; this process is similar to that used to locate a document referenced by a document link. A search of a secondary directory using directory assistance alone without the directory catalog usually takes longer.

It's not necessary to configure the same secondary Domino Directories in the server directory catalog and in the directory assistance database, although as mentioned above it's often beneficial to do so. If a secondary Domino Directory is configured in the directory catalog but not in the directory assistance database and if a search occurs for a field not configured in the directory catalog, the search can't extend to the complete entries in the secondary Domino Directory. If a secondary Domino Directory is configured in the directory assistance database but not in the directory catalog, a server uses directory assistance alone to search the secondary directory after searching the directory catalog.

Web client authentication in a secondary Domino Directory

When Web clients connect to a Domino Web server, the server can look up security credentials in a secondary Domino Directory to authenticate the clients. The Directory Assistance document for the secondary Domino Directory must contain at least one naming rule that is trusted for authentication. Domino authenticates only those Web clients who have Person documents that contain a hierarchical name that corresponds to a trusted naming rule.

To authenticate Web clients registered in a secondary Domino Directory, the Web server can use either an X.509 certificate or name-and-password authentication.

A server cannot authenticate other types of Internet clients — for example IMAP, NNTP, and LDAP — registered in a secondary Domino Directory, only Web (HTTP) clients.

The server directory catalog can work with directory assistance to facilitate Web client authentication. Using the server directory catalog is beneficial because Domino can quickly locate names from secondary directories in one database — the directory catalog — rather than search for names in multiple directories.

If the Web server uses name-and-password authentication, the directory catalog can store the passwords if the administrator adds the HTTPPassword field to the directory catalog configuration. Then, although the directory assistance database is required to define which names in the directory can be authenticated, password lookups occur directly in the catalog, so there is no need for servers to use directory assistance to access a full replica of a secondary Domino Directory to look up passwords.

Even if passwords aren't stored in a server directory catalog, using a server directory catalog along with directory assistance is helpful because the Web server can use the replica ID and UNID information sorted in the directory catalog to look up the password in the secondary directory.

Although it is possible to store X.509 certificates in the server directory catalog, doing so significantly increases the size of the directory catalog; therefore, this configuration isn't recommended.

LDAP searches in a secondary Domino Directory

When an LDAP client sends a search request, the LDAP service can use directory assistance to extend the search to a secondary Domino Directory. To do this, there must be a Directory Assistance document for the directory in the directory assistance database on the server that runs the LDAP service.

Again, it's often beneficial to configure the secondary Domino Directory in the directory catalog as well as in the directory assistance database. If the server directory catalog configuration includes the fields that LDAP clients most frequently search, then in most cases the LDAP service has to search only the server directory catalog, not the full secondary Domino Directory.

When LDAP users search for fields not configured in the directory catalog, the server uses the replica ID and UNID information in the directory catalog along with replica ID specified in the Directory Assistance document to access the full entries from the secondary Domino Directory.

Notes addressing lookups in a secondary Domino Directory

When a Notes user enters a name in the To, cc, or bcc field of a Notes memo, the user's mail server or directory server can look up the address for the name in a secondary Domino Directory if there's a Directory Assistance document for the directory in the directory assistance database on the server. The Directory Assistance document also allows a Notes user to address mail by browsing and selecting names from the secondary directory.

If the secondary directory is configured in a mobile directory catalog on the client or in a server directory catalog on the user's mail server or directory server, creating the Directory Assistance document solely for mail addressing isn't necessary. Because the default directory catalog configuration includes the fields necessary for mail addressing, directory assistance isn't needed to look up this information in the full secondary Domino Directory.

Directory assistance for LDAP directories

LDAP is a standard protocol used for accessing directory services over TCP/IP. Directory assistance allows a server to access or refer to a remote, LDAP-compliant directory — for example, a third-party LDAP directory. To set up directory assistance for an LDAP directory, administrators create a Directory Assistance document with "LDAP" selected in the "Domain Type" field. Setting up an "LDAP" Directory Assistance document allows servers to do any of the following, depending on the options chosen in the Directory Assistance document:

- Use security credentials in the LDAP directory to authenticate Web clients that use the Domino Web server
- Verify membership in groups registered in one LDAP directory for Notes ACL verification
- Refer LDAP clients that connect to the Domino LDAP service to the directory
- Allow Notes users to verify mail addresses of users in the LDAP directory

Administrators use one Directory Assistance document to enable any of these options for a specific LDAP directory.

Web client authentication using a remote LDAP directory

When Web clients attempt to access a database on a Web server, the server can authenticate the clients by looking up security credentials in a remote LDAP directory. For example, if an organization registers users in a remote third-party LDAP directory, when the users connect to a Domino Web server, the server can connect to the directory server and look up the users' names and passwords or client certificates in the LDAP directory. There must be an "LDAP" Directory Assistance document for the remote directory server with a naming rule that is trusted for authentication. The Web server authenticates only those Web clients who are registered in the LDAP directory with hierarchical names that correspond to a trusted naming rule.

The Web server doesn't also have to run the Domino LDAP service; it just has to be able to connect to the LDAP directory server. Administrators are given the option, which is recommended, to use X.509 certificate authentication to authenticate the LDAP directory server. A server cannot authenticate other types of Internet clients — for example, IMAP, NNTP, and LDAP — that are registered in an LDAP directory.

ACL group verification using a remote LDAP directory

A database ACL can include the names of one or more groups from a remote LDAP directory. When a Notes or Web user attempts to access a database on a Domino server or Web server, the server can connect to the LDAP directory server to look up the members of the LDAP groups included in the database ACL to determine if the user's name is included in the groups. For this to occur, there must be an "LDAP" Directory Assistance document enabled for "Group expansion" in the directory assistance database on the server. This document must contain at least one naming rule trusted for authentication that corresponds to the names of the LDAP groups. The Domino or Web server will only look up LDAP groups with names that correspond to a trusted rule.

The Domino or Web server doesn't also have to run the Domino LDAP service; it just has to be able to connect to the LDAP directory server. Administrators are given the option, which is recommended, to use X.509 certificate authentication to authenticate the LDAP directory server.

There is a configuration option in the Directory Assistance document that lets administrators choose whether to look up nested groups (groups within groups).

Administrators can set up group lookups for the purpose of ACL verification in only one LDAP directory; this limitation doesn't apply to group lookups for mail addressing.

LDAP client referrals to a remote LDAP directory

The Domino LDAP service can refer LDAP clients to a remote LDAP directory server if an LDAP search is not successful in the primary Domino Directory, the directory catalog, or secondary Domino Directories configured in the directory assistance database. For this to occur, there must be an "LDAP" Directory Assistance document for the LDAP directory enabled for LDAP users. To return a referral, the Domino LDAP service never connects to and searches the LDAP directory. Instead, the service uses information provided in the Directory Assistance document to return a referral. The referral is compliant with LDAP v3 and includes:

- The URL host name for the LDAP directory server
- The base distinguished name configured for the directory in the Directory Assistance document
- The port the LDAP directory server uses

By default, for a given search, the LDAP service can refer an LDAP client to only one LDAP directory. Administrators can use the LDAPReferrals setting in the NOTES.INI file to increase the number of referrals that the LDAP service can return.

Notes addressing lookups in a remote LDAP directory

If a Notes user addresses mail to a person registered in an LDAP directory, the Notes user's mail server or directory server can use directory assistance to look up the person's address in a remote LDAP directory. For this to occur, there must be an "LDAP" Directory Assistance document for the LDAP directory server in the directory assistance database that's enabled for Notes use. Address lookups occur only when a Notes user presses F9 or as the Notes client sends the mail. Domino doesn't use Notes type-ahead addressing to resolve the addresses of users in LDAP directories, and Notes users cannot address mail by browsing and selecting names from the LDAP directory.

The Notes users' mail servers or directory servers must be able to establish a network connection to the LDAP directory server, but the servers don't have to run the Domino LDAP service.

A Notes Release 5 user can create LDAP directory accounts in the Personal Address Book. These accounts allow the Notes client to use LDAP to connect to and search LDAP directories. Configuring directory assistance to allow Notes users to use an LDAP directory for addressing provides more limited functionality to Notes users who don't create LDAP directory accounts or who use Notes Release 4.5 or 4.6.

Comparison of directory catalogs and directory assistance

The mobile directory catalog, server directory catalog, and directory assistance provide similar functionality. This table compares the features that each directory supports.

Feature	Mobile directory catalog	Server directory catalog	Directory assistance
Find entries in secondary Domino Directories on behalf of Notes users for mail addressing	Yes	Yes	Yes
Find entries in LDAP directories on behalf of Notes users for mail addressing	No	No	Yes
Find entries in secondary Domino Directories on behalf of LDAP clients	No	Yes	Yes
Refer LDAP clients to LDAP directories	No	No	Yes
Authenticate Web clients using entries from secondary Domino Directories	No	Yes*	Yes
Authenticate Web clients using entries from LDAP directories	No	No	Yes
"Recipient name type ahead" addressing	Yes	Yes	Yes

*Requires directory assistance to define the names that can be authenticated.

The Domino LDAP service

Domino supports the ability to set up a Domino server to provide LDAP services. LDAP is a standard protocol for directory services and is defined in RFCs 2251—2256. LDAP is a simplified version of the X.500 DAP protocol. Unlike DAP, LDAP can use the TCP/IP protocol stack and is "lightweight" enough to run on PCs.

LDAP uses a small but comprehensive set of protocol operations that allow it to perform the following tasks:

- Initiate and terminate client/server sessions
- Process complex search queries using search filters
- Search a specific scope of the directory
- Return only requested attributes
- Add, change, and delete attributes
- Add and delete entries
- Change the name of entries
- Abandon requests

Running the LDAP server task on a Domino server enables the server to provide LDAP directory services. The LDAP server task enables a listener process over port 389. Port 389 is the industry-standard port for LDAP connections over TCP/IP, but the LDAP service can use a different port. The LDAP service can also use a port for SSL connections.

The LDAP server task loads into memory a list of directory locations to search. The LDAP service always searches its primary Domino Directory first, then the directory catalog if one is set up on the server, and finally any secondary Domino Directories that are configured in the Directory Assistance database but that are not also configured in the directory catalog. If a search is unsuccessful in any of these directories, the LDAP service can refer LDAP clients to an LDAP directory on a remote server, if the Directory Assistance database is configured to do this.

The Domino LDAP service is compliant with LDAP v3 and v2.

The LDAP schema

Each entry in an LDAP directory consists of one or more attributes. Each attribute has a name and a value. For example, the standard LDAP attribute cn typically is given an entry's common name — for example, Phyllis Spera — as its value. Each attribute has associated syntax that determines the format for the attribute value. Attributes can be required or optional.

A group of related attributes that collectively define an object that a particular type of entry represents is called an object class and is defined by the special attribute, objectClass. For example, Person is a standard object class associated with entries for people. Multiple object classes can collectively represent an object class structure, with object classes lower in the structure inheriting attributes from object classes higher in the structure.

The set of objectClass attributes defined for a particular LDAP server make up the schema.

The default Domino LDAP service schema includes many standard LDAP attributes and object classes as well as some that are specific to Domino.

Domino LDAP schema attributes

LDAP attributes defined in the Domino LDAP schema usually correlate to fields in the Domino Directory. The name of a field and the name of the attribute to which the field correlates are not necessarily the same. Some attributes in the default schema don't have a field correlation and are accessible from LDAP, but not from Notes/Domino.

Domino LDAP schema object classes

LDAP object classes defined by the Domino LDAP schema usually correlate to forms in the Domino Directory. The default Domino LDAP schema defines the object class structures that are described in the following tables.

Object class structure	Schema source
top	RFC 2256
person	RFC 2256
organizationalPerson	RFC 2256
inetOrgPerson	inetOrgPerson draft
dominoPerson	Domino

Object class structure	Schema source
top	RFC 2256
$PersonGeneralInfo*	Domino

* Extends the dominoPerson object class, by default.

Object class structure	Schema source
top	RFC 2256
groupOfNames	RFC 2256
dominoGroup	Domino

Object class structure	Schema source
top	RFC 2256
locality	RFC 2256

Object class structure	Schema source
top	RFC 2256
dominoServer	Domino

Object class structure	Schema source
top	RFC 2256
organization	RFC 2256
dominoOrganization	Domino

Object class structure	Schema source
top	RFC 2256
organizationalUnit	RFC 2256
dominoOrganizationalUnit	Domino

Schema extensions

To extend the directory schema, you add attributes or object classes to the schema. to extend the standard Domino LDAP schema, you use the Domino Designer to add forms or subforms to the Domino Directory. New forms and subforms added following specific procedures create, respectively, new object classes and new auxiliary object classes, which are object classes that extend one or more structural object classes. After extending the schema, administrators can use the server command **tell ldap reloadschema** to update the schema in memory, or they can wait for the Designer server task to do this.

Schema checking

There is a NOTES.INI setting that administrators can use to enforce schema checking. Enforcing schema checking ensures that entries in the directory contain all the attributes required by the object class(es) associated with those entries and that all attributes contained in the entries are allowed by the object classes.

Schema information

There are two ways to find out about the directory schema. The administrator can use LDAP search commands to search the schema entry in the Domino Directory. In addition, the Domino LDAP Schema database (SCHEMA50.NSF) presents schema information in an easy-to-understand format. Administrators can use the server command **tell ldap exportschema** on the server that runs the LDAP service to build and update this database or they can wait for the Designer server task to do this.

LDAP service authentication and access control

Authentication

The LDAP service supports anonymous LDAP client connections, name-and-password authentication, and SSL certificate authentication. SSL certificate authentication occurs using the Simple Authentication and Security Layer (SASL) EXTERNAL method. Administrators control which combination of these client authentication methods to use.

Access control

The Domino Directory ACL and LDAP configuration options control the extent to which LDAP users can search and modify attributes in the Domino Directory. LDAP users can never access directory information that is not part of the directory schema.

Search access

Administrators use an LDAP service configuration option to define the fields in the Domino Directory that anonymous LDAP users can search. The Domino Directory ACL does not control search access for anonymous LDAP users. Anonymous LDAP users can search the fields the administrator defines for anonymous LDAP access even when the Anonymous entry in the Domino Directory ACL has “No Access.”

The Domino Directory ACL does, however, control search access for authenticated LDAP users; authenticated LDAP users can search all attributes in the directory schema if they have at least Reader access in the Domino Directory ACL.

Write access

Administrators use an LDAP service configuration option to enable LDAP write access to the Domino Directory. By default LDAP write access is not allowed. If write access is enabled, the Domino Directory ACL controls the extent to which both authenticated and anonymous LDAP users can modify entries in the Domino Directory. For example, an LDAP user with Editor access can modify all attributes defined by the schema, whereas an LDAP user with Author access and the UserModifer role can modify only attributes associated with Person documents.

By default, the Domino Directory gives anonymous users No Access, so by default, anonymous LDAP users can't make changes to the directory if write access is enabled.

LDAP searches in secondary Domino Directories

The LDAP service can use directory assistance to search secondary Domino directories in addition to the primary Domino Directory. When processing an LDAP client request, the LDAP service can then search for information in these secondary directories, even when the search is successful in the primary Domino Directory.

If a server directory catalog is configured on a server running the LDAP service, the LDAP service searches the directory catalog before using directory assistance.

LDAP referrals to other directory servers

The LDAP service can use directory assistance to refer LDAP clients that connect to the LDAP service to another LDAP directory on a remote server. A referral can occur only if a search is not successful in the Domino LDAP service's primary Domino Directory, a server directory catalog, and secondary Domino Directories configured in directory assistance. To return a referral, the Domino LDAP service doesn't connect to and search the LDAP directory. Instead, it uses information in the Directory Assistance database to return a referral, and the clients then connect themselves.

By default, for a given search, the LDAP service can refer an LDAP client to only one LDAP directory. However, you can configure the LDAP service to return more than one referral.

LDAP service and authentication of clients that use a third-party server

A third-party, LDAP-compliant server, such as a Netscape Enterprise Server, can use passwords or X.509 certificates stored in the Domino LDAP service's directory to authenticate its clients.

LDAP alternate language searches

The LDAP service can be configured so that LDAP users can use their native languages to create search queries and display the results. For any Person document in the Domino Directory, the administrator can create an Alternate Language Information document for each alternate language needed. An Alternate Language Information document includes fields that also appear in a Person document, but the administrator uses the alternate language to enter values for the fields. This feature is compliant with RFC 2596.

To derive an attribute for a field in an Alternate Language Information document, the LDAP service tags the attribute derived from the Person document with an alternate language subattribute. For example, the LDAP attribute *sn* maps to the last name in a Person document, and the attribute *sn;lang-ja* maps to a Japanese last name in a Japanese Alternate Language Information document.

Note that this feature is different from the alternate certified names feature used by Notes users. Alternate certified names allow Notes users to use their native languages to address mail and to view names in Notes databases. In contrast, the alternate language support discussed here is useful only in the context of LDAP searches.

LDAP service statistics

The server command **show stat ldap** generates a variety of LDAP service statistics, including statistics about LDAP connections, searches, operations, and sessions.

Other LDAP features Domino offers

Domino provides several LDAP features that are independent of the Domino LDAP service and that the administrator can use with any LDAP directory server.

Domino Web client authentication using a remote LDAP directory

When Web clients connect to a Domino Web server, the Web server can look up passwords or X.509 certificates in an LDAP directory on another server — for example, on a third-party LDAP directory.

Group lookups in an LDAP directory for database access control

Database ACLs can contain groups stored in an LDAP directory on another server. When Web or Notes users attempt to access these databases, the server can search for their names in these to determine their database access.

LDAPSEARCH utility

Domino and Notes come with the utility LDAPSEARCH, which is useful for searching any LDAP directory.

User migration from a foreign LDAP directory to the Domino Directory

The migration tool available in the Domino registration program can migrate LDAP directory entries stored in an LDAP Data Interchange Format (LDIF) file into the Domino Directory. The LDAPSEARCH utility can be used to create the LDIF file.

LDAP accounts for Notes users

Notes users can create accounts in their Personal Address Books for LDAP servers that they want to search or administrators can use User Setup Profiles to set up these accounts for users automatically. Then using Boolean operators, Notes users can search for information about people. In addition, Notes users can do LDAP-style searches on directories stored locally on the clients.

Directory search orders

In an environment with multiple directories, the order in which the directories are searched depends in part on the nature of the search.

- Name lookups in multiple directories for Web client authentication
- Group lookups in multiple directories for database access verification
- LDAP searches in multiple directories
- Notes mail addressing lookups in multiple directories

Name lookups in multiple directories for Web client authentication

To authenticate a Web user connecting to a Domino Web server, the server searches directories for the Web user name in the following order.

1. The primary Domino Directory on the Web server.
2. Directory catalog on the server. If the server finds the name, it refers to its Directory Assistance database to determine if the Domino Directory from which the name came is configured with a naming rule that is trusted for authentication and that matches the user name. The server won't authenticate the client if it doesn't find such a naming rule.
3. All directories* defined in the Directory Assistance database that are not Domino Directories already configured in the directory catalog and have a naming rule that is trusted for authentication and that matches the Web user name.

   If there is more than one directory assigned a trusted naming rule that matches the user name, the directory with the most specific matching rule is searched first. If directories have identical trusted naming rules that match the Web user name, search orders assigned to these directories determine the order in which the directories are searched.

* LDAP directories are searched only if their Directory Assistance documents allows Notes/Web searches.

Group lookups in multiple directories for database access verification

When a Web or Notes user attempts to access a database on a server, if the database ACL includes a group name, the server looks for the group as follows to determine if the user's name is included the group:

1. Primary Domino Directory.
2. One LDAP directory configured in the server's Directory Assistance database with:
    - The "Group Expansion" option selected
    - A trusted naming rule that matches the group name specified in the database ACL

Group lookups for ACL verification don't occur in a server's directory catalog or in any secondary Domino Directories. In addition, the lookup can occur in only one remote LDAP directory. Note that these restrictions don't apply to group lookups for mail addressing.

LDAP searches in multiple directories

A server running the LDAP service searches directories in the following order to process LDAP search requests:

1. The primary Domino Directory on the server.
2. Directory catalog on the server. If the LDAP user searches for an attribute that maps to a field that is not configured in the directory catalog, if a secondary Domino Directory is configured in the Directory Assistance database as well as in the directory catalog, the search continues to the complete entries in the secondary Domino Directory itself. For each entry, the directory catalog stores the replica ID of the Domino Directory from which the entry comes as well as the UNID for the entry. This information ensures that the LDAP service can quickly locate the complete entries.
3. Domino Directories defined in the server's Directory Assistance database that are not included in the directory catalog.

    If an LDAP user doesn't specify a search base*, the server searches all Domino Directories according to the search orders assigned to the directories. The server searches Domino Directories with no assigned search order alphabetically according to their specified domain names.

    If an LDAP user specifies a search base, only Domino Directories assigned naming rules that correspond to the search base are searched. If there is more than one directory assigned a naming rule that matches, the directory with the most specific matching rule is searched first. For example, if a user specifies the search base "ou=Sales,o=Acme," the server first searches a directory with the rule /Sales/Acme, before searching a directory with the rule */Acme. If directories have identical naming rules that match the search base specified by the user, search orders assigned to these directories determine the order in which the directories are searched.
4. If the search is not successful in any Domino Directory, the server can pick an LDAP directory enabled for LDAP clients in the Directory Assistance database to refer clients to and the clients can then connect to the directory server themselves.

    If an LDAP user doesn't specify a search base, the LDAP service uses search orders assigned to LDAP directories that are enabled for LDAP users to determine the order in which to refer clients to these directories.

    Note By default, for a given search request the LDAP service can return only one referral, but administrators can use the NOTES.INI setting LDAPReferrals to change the number allowed.

    If an LDAP user specifies a search base, the server picks an LDAP directory enabled for LDAP users with a naming rule that matches the specified search base. If there is no such directory, the server doesn't return a referral. If there is more than one such directory, the server picks the one with the most specific matching rule before picking one with a less-specific rule. If directories have identical naming rules that match the search base specified by the user, search orders assigned to these directories determine the order in which the LDAP service picks them for referrals.

* A search base is a distinguished name used to represent the location in the directory tree at which to begin a search.

Notes mail addressing lookups in multiple directories

When a Notes user enters a user or group name in an address field of a Notes memo, directories are searched in the following order. If a name is found during any step, searches only continue for the name if the "Recipient name lookup" field in the Notes user's current Location document is set to "Exhaustively check all address books."

1. The user's Personal Address Book.
2. A local mobile directory catalog.

   For searching to continue to a server, the "Mail file location" field in the active Location document must be set to "On server."
3. The primary Domino Directoryon the user's mail server or directory server.
4. Directory catalog on the server.
5. Directories defined in the server's Directory Assistance database that are not included in the directory catalog.

   If the user enters a common name rather than a hierarchical one, the server searches all Domino Directories and LDAP directories* according to the search order specified for the directories.

   If the user enters a hierarchical name, only directories assigned naming rules that correspond to the hierarchical name the user entered are searched. If there is more than one directory assigned a naming rule that matches, the directory with the most specific matching rule is searched first. For example, if a user enters the name Phyllis Spera/Sales/Acme, the server first searches a directory with the rule /Sales/Acme, before searching a directory with the rule */Acme. If directories have identical naming rules that match the name entered by the user, search orders assigned to the directories determine the order in which the directories are searched.

* LDAP directories are searched only if their Directory Assistance documents allow Notes/Web searches.

Directory servers

A directory server is a Domino server that administrators configure and dedicate to providing directory services. A directory server reduces the workload of servers — for example, mail servers — that you might ordinarily configure to also provide directory services.

Notes users can set up a directory server in their Location documents so that name lookups occur on the directory server rather than on their mail servers. Administrators can also use User Setup Profile documents to "push" a directory server configuration to Notes clients.

Typically, a directory server is configured as follows:

- Replicas of any secondary Domino Directories are created on the directory server and directory assistance points to these replicas.
- The Dircat task builds and updates source directory catalogs on the directory server.
- The LDAP service runs on a directory server.

User and group synchronization between Domino and Windows NT

Administrators can synchronize user and group information between Domino and Windows NT and then use either product to perform many user and group management tasks. Domino adds Notes-specific menu choices to the Windows NT User Manager. The Notes User Manager Extension (NUME) communicates the administrator's choices to Domino.

Domino tasks administrators can complete from Windows NT

Administrators can perform these Domino administration tasks from Windows NT:

Register Domino users from Windows NT
As administrators create new Windows NT user accounts or if they have existing Windows NT user accounts, they can also register the accounts as either Notes or Internet users in the Domino Directory. These options are available: use Windows NT full names to create Domino user first names, middle initials, and last names; share passwords between Windows NT and Domino; use the Windows NT user name and host domain as the Internet Address in Domino Person documents.

Create Domino groups from Windows NT
As administrators create new Windows NT group accounts or if they have existing Windows NT group accounts, they can also register the accounts as groups in the Domino Directory. These options are available: register members of the groups as users in the Domino Directory; customize the members of the group for the Domino Directory; if a local groups include global groups as members, add the global groups to the Domino Directory too.

Delete Domino users and groups from Windows NT
Administrators can delete a user or group account from Windows NT and simultaneously delete the corresponding Person or Group document from the Domino Directory. To delete a Person document, the Domino Administration Process deletes all other references to the user from the Domino Directory and from database ACLs. In addition, administrators can choose to delete a user's mail file.

Administrators can simultaneously delete a Windows NT user and a Domino user only if the two accounts are linking Windows NT user accounts with Domino Person documents — that is, the Windows NT user names matches the "Network account name" field in the Person document.

Windows NT tasks administrators can complete from Domino

Administrators can perform these Windows NT administration tasks from Domino:

- Create Windows NT user accounts from Domino

  As administrators register users in Domino, they can create Windows NT accounts for the users with common passwords used between Windows NT and Domino. In addition, they can add the users to a Windows NT group.

- Rename Windows NT user accounts from Domino

  If administrators change a Domino user's common name, they can add the new name as the Windows NT user's full name. The Domino and Windows NT user records must be linked.

- Delete Windows NT user accounts from Domino

  When administrators delete a Notes user, they can delete the corresponding NT account. The Domino and Windows NT user records must be linked.

- Delete Windows NT groups from Domino

  When administrators delete a Domino group, they can delete the corresponding Windows NT group account.

Linking Windows NT user accounts with Domino Person documents

To delete or rename a user account in Windows NT and have the change occur in Domino (and vice versa) the entry in the “Network account name” field in the Domino Person document must match the Windows NT user account name. When administrators register Domino users from Windows NT and when they create Windows NT user accounts as they register users in Domino, this linking occurs automatically.

If Windows NT users and Domino users already exist independently, administrators can use a synchronization option available in User Manager for Domains to link the two types of user records. Using this option causes the User Manager to attempt to unambiguously match one of a user’s Windows NT names with one of the names in a Person document. If it’s able to do this, the User Manager updates the “Network account name” field in the Person document with the Windows NT user name and adds the Windows NT full name to the list of names in the User name field in the Person document. In addition, administrators can specify one password as both the Windows NT user password and the Domino user’s Internet password. After running synchronization, administrators can rename and delete user records from either Windows NT or Domino.

If a Windows NT user account name is changed, running synchronization again updates the corresponding Person document accordingly.

Administrators can also run synchronization to specify a common Windows NT user and Domino Internet user password.

Migrating users to Notes/Domino

Migration tools in the Domino Administrator allow administrators to easily migrate information from an existing messaging and directory system to Notes/Domino. With the migration tools, administrators can import users from a foreign directory, register them as Notes users, and then convert mail from supported mail systems to Notes mail. The migration tools lets administrators migrate all users from a foreign directory or select specific users to migrate.

- Migrating cc:Mail users
- Migrating Microsoft Mail Users
- Migrating Microsoft Exchange users
- Migrating Windows NT users
- Migrating users from an LDIF file
- Migrating users from Novell GroupWise
- Migrating users from the Netscape Messaging Server

Migrating cc:Mail users

Using the cc:Mail™ to Notes migration tool in the Domino Administrator, administrators can import cc:Mail users and groups from a selected post office and register them as Notes users. The migration process:

- Imports local users and groups from the post office (users in the directory whose locations are designated as L or R) and creates entries for them in the Domino Directory
- Lets administrators create Notes IDs and mail files for imported users
- Migrates the contents of mail boxes from the cc:Mail post office server
- Converts Organizer® 2.x for cc:Mail Group Scheduling files (OR2 files) into Notes group scheduling format (NSF)

The following table shows the data that the cc:Mail to Notes migration tool migrates from a cc:Mail post office to Notes.

cc:Mail	**Notes**
Archives	Not migrated*
Bulletin boards	Discussion databases
Bulletin board messages on mobile post offices	Not migrated
Clipboard folder	Not migrated
Folders and subfolders	Folders and subfolders**
Message date	Message date
Message priority	Message priority ***
Messages and attachments in migrated folders	Messages and attachments ****
Mobile sessions log	Not migrated
Organizer data (OR2 files)	Entries in mail file, Personal Journal, and Personal Address Book
Password	Password
Post office directory	Domino Directory
Private mailing lists	Group documents in Personal Address Book +
Public mailing list	Group document in Domino Directory
Read and unread marks	Not migrated ++
Return receipts	Return receipts
Rules	Not migrated
Sender and recipient information	Sender and recipient information
Trash folder	Not migrated
Undeliverable mail reports	Not migrated

* Archives are not migrated by the migration tool in the Domino Administrator, but after you migrate users, they can run an upgrade wizard at their workstations to migrate their message archives.

** For cc:Mail Release 8 clients, the migration converts nested folders and the messages they contain.

*** The migration tool preserves message status in cc:Mail messages marked Urgent (an exclamation mark appears beside the message in the Notes Inbox or folder). However, Notes does not assign a special status to migrated cc:Mail messages that were marked Low priority.

**** During migration rich text attributes such as color, font style, font size, underlining, boldface, bullets, embedded objects, and doclinks are not preserved.

+ Private mailing lists are automatically sent to users in a Notes message attachment. Users then run an upgrade wizard at their workstations to migrate these lists to their Notes Personal Address Books. The upgrade wizard also migrates private addresses that cc:Mail Release 2.x and 6.x clients maintain locally in the file PRIVDIR.INI. The upgrade wizard does not support migrating private addresses for cc:Mail Release 8.x clients.

++ All migrated messages are marked unread.

Migrating Microsoft Mail Users

Using the Microsoft Mail to Lotus Notes migration tool in the Domino Administrator, administrators can import users and groups from a selected Microsoft Mail postoffice and add them to Notes. The migration process:

- Imports users and groups from the postoffice and creates entries for them in the Domino Directory
- Lets administrators create Notes IDs and mail files for imported users
- Migrates the contents of mailboxes (MMF files) from a central location, such as the Microsoft Mail Postoffice server

Administrators must be able to map a drive to the location of the MS Mail mailboxes from the workstation running the Domino Administrator.

Migration of mailbox information is supported for Microsoft Mail versions 3.2, 3.5, and 3.6.

A separate user upgrade wizard allows the migration of messages from an off-line Microsoft Mail mailbox.

The following table shows the data that the Microsoft Mail to Notes migration tool migrates from a Microsoft Mail postoffice to Notes.

Microsoft Mail	Notes
Custom message types, including Scheduler messages	Not migrated
Delivery failure messages	Not migrated
Public groups	Group document in Domino Directory
Interpersonal mail messages (IPM)	Not migrated
Message date and priority	Message date and priority
Messages and attachments	Messages and attachments
Outbox folder	Not migrated
Password	Password
Personal Address Book	Personal Address book*
Postoffice address list	Domino Directory
Read and unread status	Read and unread status
Sender and recipient information	Sender and recipient information
Shared and group folder	Not migrated
Wastebasket or Deleted mail folder	Not migrated

* Personal Address Books are automatically sent to users in a Notes mail message attachment. Users then run an upgrade wizard at their workstations to migrate address book information to their Notes Personal Address Books.

Migrating Microsoft Exchange users

Using the Microsoft Exchange to Notes migration tools in the Domino Administrator, administrators can import Microsoft Exchange users and distribution lists from a selected server and add them to the Domino directory as Notes users and groups. The migration process:

- Imports users and distribution lists from the Exchange server and creates entries for them in the Domino Directory
- Lets administrators create Notes IDs and mail files for imported users
- Migrates the contents of mailboxes (PST files) that are located in a central location, such as on the Microsoft Exchange server.

The following table shows the data that the Microsoft Exchange migration tool migrates from the Microsoft Exchange server to Notes:

Microsoft Exchange data	Equivalent Notes data
Calendaring and scheduling information	Not migrated
Contacts	Personal Address Book entries *
Custom message types, including Scheduler messages	Not migrated
Deleted items	Trash folder**
Delivery failure messages	Not migrated
Digital signatures	Not migrated
Distribution lists	Group document in Domino Directory
Encrypted messages	Not migrated
Document links	Not migrated
Journals	Not migrated
Message date and priority	Message date and priority
Messages and attachments	Messages and attachments
Microsoft Office documents	Messages with attached Microsoft Office document
Notes	To Do tasks
Outbox folder	Drafts folder
Exchange directory	Person documents in Domino Directory
Private Address Book	Not migrated
Public folders	Not migrated
Read and unread status	Not migrated ***
Roles and Permissions on Mailboxes and Folders	Not migrated
Sender and recipient information	Sender and recipient information
Tasks	To Do tasks

* Contacts, which are stored on the Exchange server, are available for Outlook clients only. During migration Contacts information is placed in a Notes Personal Address Book database, which is automatically mailed to the user. Users run the upgrade wizard for Microsoft Exchange to copy information from this temporary database to the Personal Address Book on the Notes client.

** Depending on the user preferences set at the Notes client, messages migrated to the Trash folder may be deleted when users close their mail files.

*** Release 5 of the Domino migration tool for Microsoft Exchange marks all messages migrated from Microsoft Exchange as *unread* in the Notes mail file.

Migrating Windows NT users

Using the Microsoft Windows NT to Notes migration tools in the Domino Administrator, administrators can import users and groups from a selected Windows NT domain and register them as Notes users. The migration process:

- Imports users and groups from the Windows NT domain list and creates entries for them in the Domino Directory
- Lets administrators create Notes IDs and mail files for imported users

The migration tool lets administrators choose whether or not to:

- Generate random passwords for users with no passwords
- Overwrite existing passwords with random passwords
- Add the Windows NT full name to the Notes User name field in a Person document
- Add the unique Windows NT user name to the Short name field in a Person document
- Allow the addition of empty groups to Notes

Migrating users from an LDIF file

LDIF is a data format standard for conveying information from an LDAP directory. By using an LDIF export tool to create an LDIF file and then using the Domino migration tool for LDIF, administrators can migrate users from any LDAP-compliant external directory and add them to the Domino Directory.

The migration tool for LDIF creates Person documents in the Domino Directory from the person entries in an LDIF file. Optionally, administrators can create Notes IDs and mail files for migrated users, and register them as Notes users.

Currently, the migration tool does not process LDIF group entries.

Migrating users from Novell GroupWise

Using the Domino Upgrade Service for Novell GroupWise in the Domino Administrator, administrators can import GroupWise 4 or GroupWise 5 users from a selected post office and register them as Notes users. The migration process:

- Imports users and groups from the post office and creates entries for them in the Domino Directory
- Lets administrators create Notes IDs and mail files for imported users
- Migrates the contents of mailboxes from the GroupWise post office

Migrating users from the Netscape Messaging Server

Using the Domino migration tool for Netscape Messaging Server in the Domino Administrator, administrators can import Netscape messaging users registered in a NetScape Directory Server and register them as Notes users. The migration process:

- Imports users and groups from the Directory server and creates entries for them in the Domino Directory
- Lets administrators create a Notes ID and mail file for imported users
- Migrates the contents of IMAP4 mailboxes from the Netscape Messaging Server

Migration requires a Netscape Mail Server 3.5 which uses the Netscape Directory Server for storing user information. The migration tool migrates only mail that is stored in the IMAP mailboxes, not the mail in POP3 mailboxes or local mail.

Chapter 8
The Administration Process

This chapter provides a detailed technical discussion of the Administration Process.

The Administration Process

This section offers an in-depth look at how the Administration Process processes administration requests. An administration request is created by the administrator or a user performing an action, by the Administration Process itself, or by a server. Each request represents an administration task and is run by the Administraton Process server task. When an administration request is generated, it appears in the Administration Requests database (ADMIN4.NSF).

The Administration Requests database is created

When you set up the first server in the domain, Domino automatically creates the Administration Requests database for the Domino Directory. This database is propagated when additional servers pull the database during their initial server setup. Replication enforces consistency among the Administration Requests databases on servers in a domain.

The next few sections explain how the Administration Process determines the requests it has to process and how it schedules those requests. The discussion also includes an explanation of how threads process administration requests as well as where and when the Extension Manager is called upon to extend the processing of core administration requests.

The Administraion Process checks the scheduling attributes

The Administration Process checks the Administration Requests database for requests that need processing according to the scheduling attribute of the request. The scheduling attribute for each request controls when a request is processed. For example, the “Create new mail file” request is processed immediately, while the “Add Internet Certificate” request is processed according to an interval listed in the Server document and defined by the administrator. The scheduling attribute is fixed and cannot be modified.

How and when the Administration Process uses formulas

The Administration Process checks the scheduling attribute first, and then, recognizing whether it is the administration server of the Domino Directory or not, it determines which formula to use for determining which requests to process. The Administration Process uses a formula to determine what to process and uses the scheduling attribute to determine when to process a request.

The administration server for the Domino Directory looks for requests in this order:

1. Requests that only the administration server processes. These are requests that affect the Domino Directory or that the administration server must sign before they can be processed. The integrity of the request is ensured if the request is signed by the administration server of the Domino Directory because of the integrity of that server.
2. Requests that every server in the domain processes.
3. Requests for which the administration server for the Domino Directory is the target.

The other servers in the domain look for requests in this order:

1. Requests that every server in the domain processes
2. Requests for which the current server is the target server.

The Administration Process checks the Response documents

To determine which requests to process, the server first looks for unprocessed requests. The server then looks for previously processed requests that the administrator has requested should be processed again. To request reprocessing, the administrator selects “Perform Request Again” in the Administration Process Log. In this way, the Administration Process prepares a table of requests that need processing.

After processing a request, the Administration Process either generates a response document called the Administration Process Log or stores the Notes ID of the request in an internal table. If the Administration Process does any work or experiences an error, it generates a response document for that request. If a request does not require any work on the part of the Administration Process, the Notes ID for that request may be stored in an internal table. The administrator can designate whether these requests are stored in the internal table of Notes IDs by entering Yes or No in the field “Store Admin Process log entries when status of no change is recorded:” in the Server document. Storing the request Notes IDs internally reduces the size of the Administration Requests database.

Worker threads process administration requests

The Administration Process is multi-threaded. The main administration process thread looks for new work and then creates a table of note IDs. Worker threads wait until a new copy of the note ID table is available. One thread then begins processing the first ID; a second thread processes the next; and so forth. After processing a request, a thread looks for another ID and the process continues until all IDs are processed. By default, three threads are assigned to the Administration Process; however, the administrator can customize the number of assigned threads. For example, if maintaining the Domino Directory is a high priority, the administrator may designate a higher number of threads.

How threads process requests

After picking up a request, a thread makes a series of decisions. First, the thread determines whether the Administration Process or a third-party product will process the request. If it is a third-party request, the Administration Process opens a process queue on the server and moves the request to the queue. The name of the queue is the name specified in the Proxy Process field of the administration request.

If it is an Administration Process request, the Administration Process checks whether the request is from another domain. If the request originated in another domain, the Administration Process checks the configuration documents in ADMIN4.NSF and the integrity of the request. If the appropriate configuration documents exist and the request has not been tampered with, processing begins immediately.

If the request originated in the current domain, the Administration Process verifies whether the creator of the request has the authority to perform the action indicated by the request and checks the integrity of the request. If the request is valid and its integrity is intact, the Administration Process processes it.

Table of Administration Process scheduling attributes

This table describes the five scheduling attributes.

Attribute	Description
Immediate	The Administration Process checks the Administration Requests database at one-minute intervals to locate and begin processing Immediate requests.
Interval	The Administration Process checks the Administration Requests database every *n* minutes. The value of *n* is set in the Interval field in the Administration Process section of the Server document.
Daily	The Administration Process checks the Administration Requests database at the time specified in the "Execute once a day requests at" field in the Administration Process section of the Server document.
Delayed	The Administration Process checks the Administration Requests database at least one time per week at a designated interval defined in these two fields in the Administration Process section of the Server document: Start executing on (day of the week) and Start executing at (time).
Set execution schedule	The Administration Process checks the Administration Requests database on a set execution schedule to locate and process the "Delete unlinked mail file" request. No other administration request uses this scheduling attribute.

To override the designated schedule for a request, use the server commands.

For more information on server commands, the Administration Process, and the Administration Requests database, see *Administering the Domino System*.

Chapter 9 Replication

This chapter describes, in technical detail, how replication works.

Replicas and replication

To make a database available to users in different locations, on different networks, or in different time zones, you can create a special type of database copy called a replica. When someone makes a change to a replica, Domino lets you duplicate that change to the other replicas you created. This process ensures that all users share the same information. All replicas of a database have an ID in common. This ID is called the replica ID.

The process of exchanging modifications between replicas is called replication. Replication occurs only between databases that have the same replica ID. These replicas can have different names, different database designs, or different documents. As long as they have the same replica ID, you can replicate them.

As users add, edit, and delete documents in different replicas of a database and as administrators change design elements, agents, and ACL entries, the contents of the various replicas are no longer identical. To make the contents identical again, you replicate between the servers that store the replicas. The most common way to replicate is to create Connection documents in the Domino Directory. Using Connection documents, you can schedule replication to occur on a regular basis — for example, once an hour or once a day. You can schedule replication more often when it is very important to keep replicas synchronized and less often when it is less important to keep replicas synchronized.

If it is important to synchronize replicas immediately when any change occurs, you can set up a cluster of servers that remain in constant contact with each other. When a change occurs in a replica on a cluster server, that change immediately replicates to replicas on the other servers in the cluster.

Keep in mind that creating a replica of a database is significantly different from making a copy of the database. Because a copy of a database does not have the same replica ID as the original database, you cannot replicate the copies to synchronize them. Therefore, the copies become increasingly different as users add, edit, and delete documents.

The benefits of replication

Using replication, you can:

- Distribute databases to locations around the world
- Reduce communication costs by letting users access local replicas
- Distribute network traffic
- Improve performance of heavily used databases
- Keep a database that you’re redesigning separate from a production version of the database
- Keep a database available even if one server becomes unavailable
- Provide a replica containing only a subset of information that is relevant to a particular work group
- Set up Domino system administration — for example, you must create replicas of the Domino Directory, the Administration Requests database, and other critical system databases

- Give users access to a database even when the server on which they usually access the database is unavailable.
- Place a replica of a master template on each server that stores a database that inherits from the master template
- Create a backup database from which you can restore information if data becomes corrupted; since corrupted data often replicates, use this only as a secondary backup method

In addition, you can use replicas and replication to prevent users from seeing your work-in-progress. For example, if you are planning to update a Web site that you administer, you can set up a Web staging area, where you design and test new pages. After you test the design changes, you can replicate the server that contains the Web staging area with the server that contains the replica of the Web site that is available to users.

The Replicator server task and replication commands

The process of replicating between servers requires a server task and a replication command. Replicating between a client and a server requires only a replication command.

The Replicator server task

By default, the Replicator server task is loaded when you start a server. The Replicator replicates with one other server at a time. If your organization has many servers, you may want to run more than one Replicator on a server so that the server can replicate with more servers simultaneously. Each Replicator handles one replication session with another server. Using multiple Replicators, you can create Connection documents that schedule a server to replicate with multiple servers simultaneously. Using multiple Replicators can shorten the time it takes to replicate with all servers, letting you schedule more replication cycles.

Note Use multiple Replicators to replicate with multiple servers only. Multiple Replicators do not share the work of replicating with a single server, even if that server has many databases in common with the calling server. If you enable multiple Replicators and the server needs to use only one, the other Replicators remain idle.

Issuing the replication commands

There are three primary ways to issue the replication command between servers:

- Set up Connection documents to schedule replication at certain times
- Replicate immediately by issuing a replication command at the server console
- Replicate immediately by issuing the Replicate command from the Domino Administrator

Using Connection documents to set up a replication schedule

You create Connection documents in the Domino Directory to schedule replication. Each Connection document controls:

- When and how often to replicate
- Which type of replication to use
- Which databases to replicate

Whenever possible, schedule replication for times when there is the least network activity — for example, before or after work. You can schedule replication to occur at specific times or during a range of times and to repeat at a specific interval. Specifying a range of times with a repeat interval guarantees that replication occurs several times during the day. After the server successfully replicates, it waits the amount of time you specify as the repeat interval before calling the other server to replicate again. If you specify a particular time and the source server cannot reach the destination server at that time, it does not try again; it skips that replication cycle.

You can also specify the replication type, which determines whether a server sends or receives changes during a replication session. There are four choices for replication type:

- Pull-push, which is the default choice, is a two-way process in which the calling server first pulls changes from the answering server and then pushes changes to the answering server. For example, if Server A calls Server B, Server A first pulls all new and modified information from Server B. Server A then pushes all of its new and modified information to Server B. The Replicator on Server A does all of the work.
- Pull-pull is a two-way process in which two servers share the work of replication. First the Replicator on the calling server pulls new and modified information from the answering server; then the Replicator on the answering server pulls new and modified information from the calling server. If the servers are connected through a modem rather than a LAN, however, both servers pull simultaneously in order to make the best use of the connection.
- Push-only is a one-way process in which the Replicator on the calling server pushes new and modified information to the answering server.
- Pull-only is a one-way process in which the Replicator on the calling server pulls new and modified information from the answering server.

Using the server console or the Domino Administrator to initiate replication immediately

You can issue commands at the server console to initiate replication immediately, rather than wait for the time scheduled in the Connection document. These commands correspond to the types of replication you can schedule in a Connection document, although there is no command to perform pull-pull replication from the console or the Domino Administrator. You can issue these commands to replicate all databases the servers have in common, or you can specify a particular database to replicate.

Initiating replication from a Notes client

If you have the correct access rights, you can replicate databases between Notes clients and Domino servers. To do this, you can use the Replicator in the Notes client or issue the Replicate command from the client.

For more information about replicating between Notes clients and Domino servers, see Notes Help.

Replication controls

You can control replication to limit the notes that are replicated, and you can control database access to prevent replication that you do not want.

Replicating only a portion of a database

By default, replicas exchange all edits, additions, and deletions if the replicating servers have the necessary access privileges. However, you can alter replication settings to customize replication. For example, you can do the following:

- Limit the contents of a replica
- Limit what a replica sends to other replicas
- Assign miscellaneous replication settings, such as replication priority, to a database

You can specify replication settings on a new replica as you create it or on an existing replica. You can also specify some replication settings for multiple replicas at once from a central source replica. You must have Manager access to a replica to set replication settings for it.

To change the replication settings for a database, you choose File - Replication - Settings. To specify replication settings for multiple replicas from a single source replica, you specify the destination and source servers in the Advanced panel of the Replication Settings dialog box. To replicate these settings to the other databases, you must select “Replication formula” in the Advanced panel of the Replication Settings dialog box.

The replication settings for each replica are saved in a special class of note — NOTE_CLASS_REPLFORMULA. When you use a single source replica to specify settings for replicas on other servers, Notes creates a separate note for each destination server you list. If you specify multiple source servers as well as multiple destination servers, Notes creates a separate note for each source-destination combination. To be sure all replicas have the same settings, replicate with each source and destination server you specified.

The replication settings

The following table describes the replication settings.

Setting	Description
Remove documents not modified in the last *x* days	When Domino purges document deletion stubs and, optionally, unmodified documents
Only replicate incoming documents saved or modified after: *date*	The cutoff date, so that a replica only receives documents created or modified since the date Which documents are scanned during the first replication after clearing the replication history
Receive summary and 40KB of rich text only	The size of documents that a replica receives
Replicate a subset of documents*	Which documents a replica receives
Replicate*	Which non-document elements this replica receives. These can be design elements, such as forms and views, agents, replication formulas, ACL changes, deletions, or fields. See the table below for more information.
Do not send deletions made in this replica to other replicas	Whether a replica sends document deletions to other replicas
Do not send changes in database title & catalog info to other replicas	Whether a replica sends changes to the database title and Database Catalog categories to other replicas
Do not send changes in local security property to other replicas	Whether a replica sends changes to the database Encryption Settings (on the Database Basics tab of the Database Properties box) to other replicas
Temporarily disable replication	Whether a replica replicates
Scheduled replication priority	The replication priority of a database used in Connection documents for scheduling replication
CD-ROM publishing date	The publishing date for a database on a CD-ROM

*You can customize this setting for multiple replicas from a single source replica.

The Replicate settings

The table below describes how you can use the Replicate settings to affect the way that non-document design elements replicate.

Setting	Description
Forms, views, etc.	If selected, allows a replica to receive design changes, such as changes to forms, views, and folders from a source replica. If deselected, prevents a replica from receiving design changes. Alternatively, you can assign source servers Editor access or lower in the ACL; however, doing so prevents agents from replicating. Do not select this option when you first create the replica because the new replica won't contain any design elements for displaying information.
Agents	If selected, allows a replica to receive agents. If deselected, prevents the replica from receiving agents, although the replica still receives changes made by the agents.

continued

Setting	Description
Replication formula	If selected, ensures that replication settings specified for multiple destination replicas from one source replica can replicate. This option is required if you use a central source replica to manage replication settings for multiple replicas.
Access control list	If selected, allows the replica to receive ACL changes from any server that has Manager access in the replica's ACL.
Deletions	If selected, allows the replica to receive document deletions. If deselected, the replica won't receive deletions through replication, but users assigned "Delete documents" access in the replica ACL can still delete documents from the replica. **Note** If "Do not send deletions made in this replica to other replicas" (on the Send panel of the Replication Settings dialog box) is selected for the source replica, this replica won't receive deletions from the source replica, regardless of this setting.
Fields	If deselected, the replica receives all fields in each document received. If selected, you select a subset of fields to receive, but you should do this only if you have a thorough knowledge of application design. If you're replicating a Domino Directory, you can also choose among minimal Address Book options. These options let mobile users replicate a small version of a Domino Directory locally. The minimal Address Book options are also available in the Space Savers panel. Note that users can also use a mobile directory catalog to have local access to names in a Domino Directory.

Replication formulas

You can also use replication formulas to specify which documents to pull into a database during replication.

Using the access control list in replication

The access level and access privileges that you give a server in a database ACL determine what the server can push to that database. The table below shows how the ACL settings affect replication.

Access level	Allows a server to push these changes	Assign to
Manager	• ACL settings • Database encryption settings • Replication settings • All elements allowed by lower access levels	Servers you want to use as a source for ACL changes. For tight database security, give this access to as few servers as possible. In a hub-and-spoke server configuration, you typically give the hub server Manager access.
Designer	• Design elements • All elements allowed by lower access levels	Servers you want to use as the source for design changes. Use Manager access instead if you want one server to control ACL and design changes.
Editor	• All new documents • All changes to documents	Servers that users use only to add and modify documents. In a hub-and-spoke configuration, you typically give the spoke servers Editor access.
Author	• New documents	No servers. You don't typically use this access for servers.
Reader	• No changes; server can only pull changes	Servers that should never make changes. Servers in the OtherDomainServers group are often given Reader access.
Depositor	• New documents. Also prevents the server from pulling changes.	No servers. You don't typically use this access for servers.
No Access	• No changes. Also prevents the server from pulling changes.	Servers to which you want to deny access. Servers in the OtherDomainServers group are sometimes given No Access.

Using the access privileges in replication

For each access level, you can select or deselect these privileges:

- Create documents
- Delete documents
- Create personal agents
- Create personal folders/views
- Create shared folders/views
- Create Java agents
- Read public documents
- Write public documents

In general, enable all the privileges that the selected access level allows. This ensures that the server has access that is as high as users might have and can replicate all user changes. However, to prevent certain changes from replicating without deselecting privileges for each user, you can deselect a particular privilege for a server entry in the ACL. For example, to prevent all document deletions made in a database on a particular server from replicating, deselect "Delete documents" in the ACL entry for the server. Then when users who have "Delete documents" access in the ACL delete documents, the deletions don't replicate.

How replication works behind the scenes

This topic describes pull-push replication, which is the default type of replication.

In the following example, the Replicator on Server A replicates between Server A and Server B. To do this, the Replicator on Server A uses Notes RPC capabilities to read from and write to replicas on both servers. No Replicator runs on Server B.

The example refers to a "source" replica and a "destination" replica. When Server A pulls from Server B, the source replica resides on Server B, and the destination replica resides on Server A. When Server A pushes to Server B, the source replica resides on Server A, and the destination replica resides on Server B.

The Replicator task remains idle until the server initiates replication. Then the following occurs:

1. When a server initiates replication with another server, the servers authenticate each other by finding a certificate in common and testing to be sure the certificates are authentic.
2. The Replicator on the initiating server (Server A) searches the directory of databases on both servers and constructs a list of the databases on each server.
3. The Replicator searches through the two lists to find databases that have identical replica IDs.

   A replica ID is an 8-byte number that Notes creates when it creates a database. The number represents the day and time that the database was created. Databases that are replicas have the same replica ID.
4. When the Replicator finds a match, it looks at the replication history in the source replica to find the last time the replicas replicated.
5. The Replicator searches the source replica for notes that have changed since the last replication.

   To find this information, the Replicator issues an NSFSearch request against the source replica. To receive the correct documents, the Replicator passes NSFSearch the time from the replication history of the source replica, as well as any replication settings or replication formulas that are in the destination replica.

   NSFSearch returns several pieces of information, including a list of the OIDs of all documents that have been created or modified since the last replication, as long as they meet the criteria of the replication settings and replication formula. (If there are no entries in the replication history, NSFSearch returns this information about all documents that meet the criteria of the replication settings and the replication formula.)

Note An OID consists of three parts: a UNID, which is a unique 16-byte document identifier that never changes; a sequence number, which indicates how many times the document has been modified; and a time stamp, which indicates the last time the document was modified. Each of these parts plays a role in replication.

6. The Replicator issues another NSFSearch request, using the UNIDs in the list it received from the source replica to search the destination replica for the corresponding notes. It receives information, including a list of the OIDs, of the corresponding notes.
7. The Replicator examines the two lists of OIDs to determine what to replicate and where conflicts exist.
   - If the UNID of a note in the source replica does not have a corresponding UNID in the destination replica, this is a new note. The Replicator pulls that note to the destination replica.
   - If a note in the source replica has the same OID as a note in the destination replica, the notes are identical; no replication is necessary.
   - For notes that have the same UNID but different OIDs, the Replicator checks the revision histories of the notes (the $Revisions field) to determine if a conflict exists.

   The revision history is a list of when changes were made to a note. Each time someone updates a note, the time from the OID is transferred to the revision history, and the current time is placed in the OID. During replication, the $Revisions field is updated so that the revision history in corresponding notes is identical.

   Here are the rules the Replicator follows to determine if there is a conflict:
   - If the revision histories are identical except that one has additional entries, no conflict exists. This shows that one note has information that can be replicated to the other note. If the changed note is in the source replica, the changes are pulled to the destination replica.
   - If the revision histories are identical to a point but then both have additional entries, both have changed and a conflict exists.
8. If a conflict exists, the Replicator checks the field $ConflictAction in the destination document. If this field contains a "1," the Replicator can copy the changes from the source document to the destination document (merge the documents) if the changes in the documents are in different fields. Otherwise, the Replicator must create a conflict document.

   A "1" in the $ConflictAction field indicates that the form for the document was designed with "Merge Replication Conflicts" enabled or that the field was set with an API program, LotusScript, or Java.
9. If $ConflictAction=1 in the destination document, the Replicator uses the following logic to determine if the changes to the documents are in different fields:
   - The Replicator looks at the sequence number in the corresponding field of each document and compares the numbers to the sequence number that existed in the note when the divergence began in the revision history. This determines whether the field changed since the point of divergence.
   - If both fields changed since the point of divergence, there is a conflict and you cannot merge the documents. If only the source field changed since the point of divergence, you can pull that field to the destination document. The Replicator copies changes from one document to another only if it can do so in all the fields that have changed. Otherwise, it creates a conflict document.
10. If there is a conflict and the Replicator cannot merge the documents, it leaves one of the documents as it is (the "winner") and turns the other documents into a conflict document (the "loser"). The Replicator uses the following logic to determine a winner and a loser:
    - The note that has the highest sequence number in its OID is the winner.
    - If the sequence numbers are the same, the note that has the most recent time stamp in its OID is the winner.

11. The losing server always creates the conflict document. While doing a pull, if the losing server is the destination server, it pulls the winner from the source server and then creates a conflict document as a response to the winner.

    If the losing server is the source server, no conflict document is created during a pull because the source server cannot pull the winner from the destination server. In this case, the conflict document is created during the push portion of replication.

12. The Replicator on Server A initiates the push portion of replication. The Replicator on Server A still does all the work, but the replica on Server A is the source replica and the replica on Server B is the destination replica.

How the Replicator handles conflicts in ACL, design element, and agent changes

During replication, the Replicator replicates changes to all notes — document, ACL, design element, and agent. The Replicator follows the same logic and procedure for all of these but does not create a conflict document for ACL, design element, and agent changes. For these notes, the winner's changes replicate, and the loser's changes are discarded.

How the Replicator handles conflicts when documents are deleted

When a document is deleted, a deletion stub is left behind. This deletion stub includes all the information needed to identify the document so that the Replicator can follow the preceding procedure to find changes and replicate them. In case of a conflict with a deletion, the Replicator follows this rule: If a document is deleted in one replica but edited in another replica, the deletion takes precedence unless the edited document was edited more than once or the editing took place after the deletion.

Preventing and resolving replication conflicts

There are ways to minimize the likelihood of having a replication conflict occur and to resolve the conflicts when they do occur.

Preventing conflicts

Although Domino has significant built-in logic to prevent replication conflicts, they still occur occasionally. Here are a few things you can do to prevent conflicts:

- Set up the ACL to minimize conflicts. For example, assign users Author access or lower instead of Editor access. Limiting the number of Editors in a database reduces the possibility that someone will accidentally edit someone else's document.
- Whenever possible, use the Administration Process to process administration requests.
- When designing forms, select the Form property "Merge replication conflicts" to automatically merge conflicts into one document if no fields conflict. Alternatively, use LotusScript, Java, or an API program to program a $ConflictAction field to merge documents automatically when no fields conflict.
- When designing forms, specify a Form property for versioning. This property makes edited documents into new documents.
- Use LotusScript to write a conflict handler that lets you resolve conflicts when they occur or that automatically resolves conflicts.
- Keep the number of replicas to a minimum.
- If the database property "Limit entries in $Revisions fields" is set to a value greater than 0, increase the limit by specifying a greater value than the existing one or specify -1 to remove the limit.

Resolving existing conflicts

When the Replicator determines that it cannot resolve a conflict, it makes the winning document the main document and makes the losing document a response document that is displayed in the view as "[Replication or Save Conflict]." To fix this problem, you copy information into one document and then delete the other document.

To save the winning document

1. (Optional) Copy any information you want from the losing document into the winning document.
2. Delete the losing document.

To save the losing document

1. Open the losing document in Edit mode.
2. (Optional) Copy any information you want from the winning document into the losing document.
3. Save the losing document.

   This makes the losing document become a main document.
4. Delete the winning document.

Clusters and replication

A Domino cluster is a group of servers that keep in constant contact with each other so that you can provide users with continual access to data, balance the workload between servers, improve server performance, and maintain performance when you increase the size of your enterprise. The servers in a cluster contain replicas of databases that you want to be readily available to users at all times. If a user tries to access a database on a cluster server that is not available, Domino opens a replica of that database on a different cluster server, if a replica is available. Through replication, Domino continuously synchronizes databases so that whichever replica a user opens, the information is always identical.

How cluster replication differs from standard replication

Cluster replication is event-driven rather than schedule-driven. When the Cluster Replicator learns of a change to a database, it immediately pushes that change to the other replicas in the cluster. If a cluster server is unavailable for replication, the Cluster Replicator on the source server stores the replication events in memory until it can push them to that server. If the source server shuts down before replication completes, the replication events in memory are lost. For this reason you should use standard replication to perform immediate replication with all members of a cluster when you restart a cluster server. It is also a good idea to schedule replication between cluster servers on a regular basis, such as once an hour, to ensure that databases remain synchronized.

Because replication formulas can use a lot of processing power, the Cluster Replicator leaves the processing of these formulas to the standard Replicator. This reduces the overhead of using cluster replication. If you use selective replication, therefore, a database may temporarily include documents that do not match the selection formula. Domino deletes these documents when you use standard replication.

In addition, cluster replication does not honor the settings in the Advanced panel in the Replication Settings dialog box. The Cluster Replicator always attempts to make all replicas identical. Keeping replicas identical ensures that users will not notice a difference if Domino opens a different replica than the one that the users attempted to open.

Note Standard replication does not remove changes to specific database elements, such as the ACL, agents, or design elements. If limiting replication of these elements is important for a database, consider using only standard replication, not cluster replication, for that database.

How cluster replication works

There are several components that work together to make clustering work correctly. The two components most important in the process of cluster replication are the Cluster Database Directory and the Cluster Replicator. The Cluster Database Directory contains a document for each database and replica in the cluster. These documents tell the Cluster Replicator which events to replicate and where to replicate them. A replica of the Cluster Database Directory resides on every server in the cluster.

The Cluster Replicator looks in the Cluster Database Directory to determine which databases have replicas on other cluster servers. The Cluster Replicator stores this information in memory. When the Cluster Replicator detects a change in the Cluster Database Directory, it updates the information in memory — for example, adding or deleting a database or disabling replication for a database. This ensures that the information cached in memory is always the same as the information in the Cluster Database Directory.

The Cluster Replicator also creates a queue to receive notifications of changes to databases in the cluster. The Cluster Replicator registers the queue with a Domino internal function that detects database changes and sends notifications to the queue each time a database change occurs. The notifications include the replica IDs and path names of the databases, as well as the note IDs of any changed notes. The Cluster Replicator looks in this queue once a second for new notifications. It then checks the information from the Cluster Database Directory to see if any of the changed databases are on another server in the cluster. It then uses basic replication to push the changes to the appropriate servers. It checks the same things that the standard Replicator checks — UNIDs, OIDs, revision histories, sequence numbers, and so on. It also honors the ACL settings.

Because cluster replication can result in so many replications, the Cluster Replicator writes to the replication history only once an hour. It stores the changes in a queue until it writes them.

Chapter 10
Clusters

This chapter describes, in technical detail, how clusters and failover work.

Domino clusters

A Domino cluster is a group of two to six servers that you use to provide users with constant access to data, balance the workload between servers, improve server performance, and maintain performance when you increase the size of your enterprise. The servers in a cluster contain replicas of databases that you want to be readily available to users at all times. If a user tries to access a database on a cluster server that is not available, Domino opens a replica of that database on a different cluster server, if a replica is available. Domino continuously synchronizes databases so that whichever replica a user opens, the information is always identical.

Each server in a cluster contains cluster components that are installed with the Lotus Domino Enterprise Server license. These components and the Administration Process perform the cluster management and monitoring tasks that run the cluster. The components keep replicas synchronized, and they continually communicate with each other to ensure that the cluster is running efficiently and smoothly. In addition, you use the components to set limits for workload balancing, track the availability of servers and databases, and add servers and databases to the cluster.

The benefits of clusters

Clustering servers provides many benefits.

High availability of important databases

When a hardware or software problem occurs, cluster servers redirect open database requests to other servers in the cluster. This process, which provides users with uninterrupted access to important databases, is called failover. Clusters provide failover for business-critical databases and servers, as well as passthru server failover to other servers in the cluster.

Workload balancing

When users try to access databases on heavily used servers, Domino redirects the user requests to other cluster servers so that workload is evenly distributed across the cluster. Workload balancing of cluster servers helps achieve optimum system performance, which leads to faster data access.

Scalability

As the number of users you support increases, you can easily add servers to a cluster to keep server performance high. You can also create multiple database replicas to maximize data availability and move users to other servers or clusters as you plan for future growth. As your enterprise grows, you can distribute user accounts across clusters and balance the additional workload to optimize system performance within a cluster.

Data synchronization

A key to effective clustering is setting up replicas on two or more cluster servers so that users have access to data when a server is unavailable or is being used heavily. Cluster replication9 ensures that all changes, whether to databases or to cluster membership, are immediately passed to other databases or servers in the cluster. Thus, databases are continuously synchronized to provide high availability of information.

Ease of changing operating systems, hardware, and versions of Domino

When you want to change the hardware, operating system, or Domino release, you can mark the cluster server as RESTRICTED so that requests to access a database on the server fail over to other servers that are in the cluster and that contain replicas. After marking a server RESTRICTED, you can perform system maintenance without interrupting the productivity of your users.

System backup

You can set up a cluster server as a backup server to protect crucial data. You can prevent users from accessing the server, but cluster replication keeps the server updated at all times.

Cluster requirements

There are special requirements for servers in a cluster and for clients that access those servers.

- All servers in a cluster must run the Domino Release 5 or Release 4.62 Enterprise Server license, or the Domino Release 4.5 or Release 4.6 Advanced Services license.
- All servers in a cluster must be connected using a high-speed local area network (LAN). You can also set up a private LAN for cluster traffic only.
- All servers in a cluster must use TCP/IP, be on the same Notes named network, and use the same set of network protocols.
- All servers in a cluster must be in the same Notes domain and share a common Domino Directory.
- In the domain that contains the cluster, you must specify an administration server for the Domino Directory. If you do not specify an administration server, the Administration Process cannot change cluster membership. The administration server does not have to be a member of a cluster or be running the Enterprise Server license.
- Each server in the cluster must have a hierarchical server ID. If any server has a flat ID, you must convert it to a hierarchical ID before you can include the server in a cluster.
- A server can be a member of only one cluster at a time.
- Notes clients must run Notes Release 4.5 or higher to take advantage of the cluster failover feature.
- Clients who access a cluster server must use TCP/IP.

Cluster components

There are several components that work together to make clustering function correctly. These include:

- The Cluster Manager
- The Cluster Database Directory
- The Cluster Database Directory Manager
- The Cluster Administrator
- The Cluster Replicator

The Cluster Manager

A Cluster Manager runs on each server in a cluster and tracks the state of all the other servers in the cluster. It keeps a list of which servers in the cluster are currently available and maintains information about the workload on each server.

When you add a server to a cluster, Domino automatically starts the Cluster Manager on that server. As long as the server is part of a cluster, the Cluster Manager starts each time you start the server.

Each Cluster Manager monitors the cluster by exchanging messages, which are known as probes, with the other servers in the cluster. Each minute, the Cluster Manager uses the NSPingServer command to probe the other servers in the cluster to determine the workload and availability of the other cluster servers. The Cluster Manager issues the NSPingServer command once every minute, unless you change this default by using the Server_Cluster_Probe_Timeout NOTES.INI setting.

When it is necessary to redirect a user request to a different replica, the Cluster Manager looks in the Cluster Database Directory to determine which cluster servers contain a replica of the requested database.

The tasks of the Cluster Manager include:

- Determining which servers belong to the cluster. The Cluster Manager periodically monitors the Domino Directory for changes to the ClusterName field in the Server document and the cluster membership list.
- Monitoring server availability and workload in the cluster.
- Informing other Cluster Managers of changes in cluster server availability.
- Redirecting database requests, based on the availability of cluster servers (failover).
- Balancing server workloads in the cluster, based on the availability of cluster servers.
- Logging failover and workload balance events in the log file (LOG.NSF).

When it starts, the Cluster Manager looks in the ClusterName field in its Server document in the Domino Directory to determine its cluster name. It then looks at the cluster membership list to determine the names of the other servers that belong to the cluster. It finds this information in the Server document of each server. It maintains this information in memory in the server's cluster name cache. The Cluster Manager uses this information to exchange probes with other Cluster Managers. The Cluster Manager also uses the cluster name cache to store the availability information it receives from these probes. This information helps the Cluster Manager perform the tasks listed above.

The cluster name cache contains the following information:

- The name of the cluster
- The names of the servers in the cluster
- The server availability index of each cluster server, sorted by the most available first
- The state of the server if it is BUSY, MAXUSERS, or RESTRICTED
- The cluster probe timeout value

The Cluster Manager checks the modified time of the Domino Directory once a minute and compares it to the last update time of the cluster name cache. If the modified time of the Domino Directory shows that the Domino Directory has changed since the time the cluster name cache was last updated, the Cluster Manager does a lookup in the Domino Directory to see if the cluster membership or the name of the cluster has changed. The Cluster Manager copies the new information to the cluster name cache.

To view the information in the cluster name cache, type **show cluster** at the server console.

The Cluster Database Directory

The Cluster Database Directory (CLDBDIR.NSF) resides on every server in a cluster. The Cluster Database Directory contains a document for each database and replica in the cluster. This document contains the database name, server, path, and replica ID, as well as other replication and access information. The cluster components use this information to perform their functions — for example, to determine failover paths, control access to databases, and determine which events to replicate to which servers.

The Cluster Database Directory Manager

The Cluster Database Directory Manager task (Cldbdir) on each server creates the Cluster Database Directory and keeps it up to date. When you first add a server to a cluster, the Cluster Database Directory Manager creates the Cluster Database Directory on that server. When you add a database to a cluster server, the Cluster Database Directory Manager creates a document in the Cluster Database Directory. This document contains information about the new database. When you delete a database from a cluster server, the Cluster Database Directory Manager deletes the document for the server. The Cluster Database Directory Manager also tracks the status — for example, Out of service, Pending, or Delete — of each database.

When there is a change to the Cluster Database Directory, the Cluster Replicator immediately replicates that change to the Cluster Database Directory on each server in the cluster. This replication ensures that each cluster member has up-to-date information about the databases in the cluster.

The Cluster Administrator

The Cluster Administrator performs many of the housekeeping tasks associated with a cluster. For example, when you add a server to a cluster, the Cluster Administrator starts the cluster tasks, such as the Cluster Database Directory Manager and the Cluster Replicator. It also adds task names (Cldbdir and Clrepl) to the ServerTasks setting in the NOTES.INI file so that these tasks start each time you start the server. The Cluster Administrator also starts the Administration Process, if it is not already running. When you remove a server from a cluster, the Cluster Administrator removes these commands from the NOTES.INI file and stops these tasks. In addition, the Cluster Administrator deletes the Cluster Database Directory on the server and cleans up records of the server in Cluster Database Directories on other servers.

The Cluster Replicator

The Cluster Replicator task (Clrepl) constantly synchronizes data among replicas in a cluster. Whenever a change occurs to a database in the cluster, the Cluster Replicator immediately pushes the change to the other replicas in the cluster. This replication ensures that each time users access a database, they see the most up-to-date version. The Cluster Replicator also replicates changes to private folders that are stored in a database. By default, each server in a cluster runs one Cluster Replicator, but you can run more cluster replicators to improve performance.

The Cluster Replicator looks in the Cluster Database Directory (CLDBDIR.NSF) to determine which databases have replicas on other cluster members. The Cluster Replicator stores this information in memory and uses it to replicate changes to other servers. When the Cluster Replicator detects changes in the Cluster Database Directory, it updates the information in memory — for example, by adding or deleting a database or disabling replication for a database.

The Cluster Replicator task pushes changes only to servers in the cluster. The standard Replicator replicates changes to and from servers outside the cluster.

Failover and workload balancing

A cluster's ability to redirect requests from one server to another is called failover. When a user tries to access a database on a server that is unavailable or overloaded, Domino connects the user to a replica of the database on another server in the cluster.

When a user tries to access a database that is not available, the Cluster Manager redirects the user request to a replica of the database on a different server in the cluster. Although the user connects to a database on a different server, failover is transparent to the user.

Workload balancing

By distributing databases throughout the cluster, you balance the workload in the cluster so that no server is overloaded. In addition, you can use several NOTES.INI settings to help balance the workload. For example, you can specify an availability threshold that limits how busy a server can get. When the server reaches the availability threshold, the Cluster Manager marks the server BUSY. When a server is BUSY, requests to open databases fail over to other servers that contain replicas of the requested databases. You can also use the Server_MaxUsers setting in the NOTES.INI file to specify the maximum number of users allowed to access a server. When the server reaches this limit, it is marked MAXUSERS, and users fail over to another server. Controlling failover with these settings keeps the workload balanced and keeps the server working at optimum performance.

When failover occurs

Failover occurs when users cannot access the server that contains a database or cannot access the database itself. The following table describes why a user may not be able to access a server or database.

Category	**Cause of failover**
Unable to access server	• The server is unavailable. • There are network connectivity problems. • The server has reached the maximum number of users allowed, as specified in the Server_Maxusers setting in the NOTES.INI file. • The administrator used the Server_Restricted setting in the NOTES.INI file to restrict the server. • The server is BUSY because it has reached the maximum load allowed (the server availability threshold).
Unable to access database	• The database is marked OUT-OF-SERVICE in the Cluster Database Directory. • The database is marked PENDING DELETE in the Cluster Database Directory.

When a server or database is not available, failover occurs when a user attempts to use Notes to perform certain actions. The following table describes the actions that trigger failover.

Category	**Action that triggers failover**
Database open operations	• Opening a database from a bookmark • Clicking a document link, a view link, or a database link • Activating a field, action, or button that contains @*command* ([FileOpenDatabase]) • Running a LotusScript routine that contains the OpenWithFailover method of the NotesDatabase class • Using Java that contains the Open Database method of the DbDirectory class • Replicating with a database on a cluster server that is not running or not reachable on the network
Mail server operations	• Composing mail • Name lookups • Type-ahead addressing • Routing mail messages • Mail pre-processing agents • Meeting invitations • Free time lookups • Server lookups
Web server operations	• Selecting the Open URL icon • Clicking a URL hotspot • Accessing a URL with a Web browser

When failover does not occur

Failover does not occur in the following cases:

- When a server becomes unavailable while a user has a database open

  If the user opens the database again, the server fails over to a different replica, if one exists in the cluster. If the user was editing a document when the server became unavailable, the user can copy the document to the replica.
- When a user chooses File - Database - Properties or File - Database - Open
- If the Router attempts to deliver mail while MailClusterFailover is set to 0
- If the template server is unavailable when a user attempts to create a new database
- When running agents, other than the mail pre-delivery agent
- When replicating with a server that is restricted by the administrator or that has reached the maximum number of users or the maximum usage level set by the administrator. Also, when replicating with a database marked "Out of Service." Replication occurs regardless of such restrictions, so there is no need for failover to occur.

Mail failover

If you create replicas of mail databases in a cluster, failover occurs by default in the following instances. Note that failover does not occur if you deactivate mail failover in the Configuration Settings document for a server.

When a user tries to open a mail database that is unavailable

Failover for mail works the same as for any database.

If a user's mail server becomes unavailable and then the user tries to send a message

If a user is composing a message when the mail server becomes unavailable, the user can still send the message if the cluster contains a replica of the user's mail database. The delivery fails over to another cluster server, where Notes deposits the message in the outgoing mailbox.

When the Router tries to deliver mail to a server that is unavailable

If the server that contains the mail database is unavailable, the Router delivers the mail to a cluster server that contains a replica mail database. The Router uses this process to find the correct mail database.

First, the Router checks if mail failover is enabled for the local server and if the user's mail server is in a cluster. If the local server is in the same cluster and has a replica of the user's mail database, the Router delivers the mail to that database. Otherwise, the Router asks an available cluster member for a server that contains a replica of the user's mail database and delivers the mail to that database. If there is no replica available, the Router tries again to deliver the mail to the user's mail server.

When the user is using shared mail

Shared mail works the same on a cluster server as it does on a non-cluster server. Just as users can access the shared mail database from their primary mail databases, they can access the shared mail database from replicas of their mail databases. When the Cluster Replicator replicates mail databases, it copies the message header to each user's mail databases and copies the message body to the shared mail database.

How calendars work in a cluster

Domino supports clustering of calendars and the Free Time database. To make calendar failover work, the scheduling system works a little differently behind the scenes in a cluster than when it is not in a cluster. However, these differences are not noticeable to users.

When not in a cluster, each server contains a Free Time database (BUSYTIME.NSF) that includes scheduling information for all users who use that server as their mail server. In a cluster, there is one Free Time database for all users whose mail servers are in the cluster. This database is named CLUBUSY.NSF, and it contains all the information that was in all the Free Time databases on all the servers in the cluster. Every server in the cluster contains a replica of this database.

When you add a server to the cluster, the Schedule Manager deletes the BUSYTIME.NSF database on the server and creates the CLUBUSY.NSF database, which then replicates with the other servers in the cluster. When a user in the cluster looks for free time, the server looks in its own CLUBUSY.NSF first to find information for every user in the cluster. For users whose mail servers are outside the cluster, a request is sent to those servers for the free time information. When a user outside the cluster makes a request for information about a user in the cluster, the request fails over to another server in the cluster if the user's mail server is unavailable. Whenever there is a change to the CLUBUSY.NSF on any server in the cluster, the Cluster Replicator replicates the change to the other servers in the cluster.

When you remove a server from a cluster, the Schedule Manager deletes CLUBUSY.NSF from that server and creates BUSYTIME.NSF on the server. The Schedule Manager on each server in the cluster removes the deleted information from its replica of CLUBUSY.NSF.

Note If there are Domino Release 4.5 or 4.6 servers in a cluster, these servers maintain their BUSYTIME.NSF databases. These databases are not converted to CLUBUSY.NSF. Using calendars on these servers works the same as in a non-clustered environment.

Limiting the workload of a server

To better balance the workload among the servers in a cluster, you can limit the workload of each server by setting the server availability threshold. When a server's workload reaches the server availability threshold, the server is marked BUSY. At that point, access requests fail over to another server in the cluster, if one is available. If no other server is available, the original server takes the access request, even though it is in a BUSY state. No request is denied access because a server is in a BUSY state. Each time an access request is redirected, Domino generates a workload balancing event in the log file (LOG.NSF).

To determine if a server is BUSY, Domino compares the server availability threshold with the server availability index, which is a measure of the current workload on a server. When the availability threshold is equal to (or greater than) the availability index, the server is BUSY.

Note The availability threshold does not affect replication. Replication occurs even when a server is in a BUSY state.

The server availability index

Each server in a cluster periodically determines its own workload by calculating the average response time of the requests the server has recently processed. The workload is expressed as a number from 0 to 100, where 0 indicates a heavily loaded server and 100 indicates a lightly loaded server. This number is called the server availability index. As response times increase, the server availability index decreases.

Although the numbers range from 0 to 100, the availability index is not a percentage. Instead, the availability index is calculated by dividing the response time for a function under the current load by the response time of the same function under a light load and then subtracting the result from 100. For example, if a database open call currently takes 3 seconds but would take only .3 seconds under optimum conditions, the availability index is 100 minus the result of 3 divided by .3. Thus the availability index is 90.

To determine the current availability index, Domino divides the current response time by the optimum response time for a representative set of transactions that occurred during the previous minute or so and then subtracts the average from 100, as shown in this formula:

```
    Availability index = 100 - (Current response time / optimum response time)
```

The availability index measures only the server response time, which is usually only a small portion of the response time that clients experience. For example, the network response time between a client and a server often accounts for a significant portion of the response time that the client experiences. Therefore, an availability index of 90 indicates that the response time of the server itself is 10 times longer than optimal, not that the response time that the client experiences is 10 times longer than optimal.

Setting the maximum number of users on a server

Another way to balance the workload in a cluster is to use the Server_MaxUsers setting in the NOTES.INI file. This setting specifies the maximum number of active users allowed on a server at one time. When the server reaches this limit, the server goes into the MAXUSERS state and rejects any additional requests until the number of active users falls below the Server_MaxUsers limit. When Domino rejects access requests because of a MAXUSERS state, the Cluster Manager attempts to redirect new user requests to other cluster servers that contain the appropriate replicas. If no other server is available, Domino rejects the access request and displays an explanatory message.

Note that the Server_MaxUsers setting does not affect replication. Replication occurs even when a server is in a MAXUSERS state.

You can use the Server_MaxUsers setting on any Domino server. However, only cluster servers redirect access requests; servers that are not in a cluster reject access requests.

Causing failover to occur

To cause failover to occur, you can use the Server_Restricted setting in the NOTES.INI file. This setting tells a server to deny new open database requests and places the server in a RESTRICTED state. This state prevents new users from accessing a server but allows users who have active connections to databases on the server to retain their connections. This setting is useful when you want to make a server unavailable so that you can perform routine maintenance on it or upgrade it. This setting is also useful when users have failed over to a server and you want them to fail back to another server.

When a server is in a RESTRICTED state, the Cluster Manager redirects new open database requests to other servers in the cluster. When an attempt to redirect is unsuccessful, the user receives an explanatory message and cannot access the server.

You can also set up a server as a backup to another server. You can set the availability threshold to 100 on the backup server so that it is BUSY at all times. That way, the backup server accepts open database requests only when the primary server is unavailable.

By setting the availability threshold on a server to 100, you put the server into a BUSY state. This is similar to a RESTRICTED state, except a BUSY server accepts new open requests if no other replica is available, while a RESTRICTED server does not.

Note that the Server_Restricted setting does not affect replication. Replication occurs even when a server is in a RESTRICTED state. In addition, you can use the Server_Restricted setting for any Domino server. This setting is not limited to clusters.

Managing database availability in a cluster

Using the three database attributes — OUT-OF-SERVICE, IN-SERVICE, and PENDING DELETE — you can specify whether a database is available for user access.

On occasion, you may want to mark a database OUT-OF-SERVICE — for example, to perform database maintenance or to force users to fail over to a different database replica because the server is reaching a high level of use.

You use the Domino Administrator to mark a database OUT-OF-SERVICE. Users cannot open an OUT-OF-SERVICE database. Instead, open database requests fail over to a replica, if one is available. If no replica is available, Domino denies users access to the database and displays an explanatory message. To make the database available again, you use the Domino Administrator to mark it IN-SERVICE.

In addition, you can use the Domino Administrator to mark a database PENDING DELETE. When a database is PENDING DELETE, the database does not accept any new database open requests. After all users close the database, Domino pushes changes to another replica and then deletes the database. Mark a database PENDING DELETE if you plan to remove it because it is obsolete or if you are copying the database to another server and want to delete the database from the original server.

Example of failover

This example describes the steps involved in failover.

1. A Notes user attempts to open a database on Server 1. To open the database, Notes calls NSFDbOpenExtended with the FailOver and Workload Balance options, or it calls another database open function, depending on the method the user used to try to open the database.
2. Notes receives a message indicating that the server is not responding. Notes calls a Name services function to attempt to locate a replica of the database on another cluster server.
3. Notes looks in its cluster cache (populated by the contents of the CLUSTER.NCF file) to see if the server is part of a cluster and to find the name of another server in the cluster.

   The names of the servers in a cluster are added to the CLUSTER.NCF file on the client the first time the client accesses any server in the cluster. When you start the client, this information is copied to the cluster cache in memory. This information is updated each time Notes opens a database if at least 15 minutes have passed since the last update. If the client has never accessed a server in the cluster prior to receiving the server not responding message, the client has no information in the cluster cache and will not be able to find another server in the cluster to fail over to.
4. Notes issues an RPC transaction to the first server listed in the cluster cache (Server 2) in order to get availability information for the servers in the cluster. As part of the transaction, Notes sends the path name of the database it is trying to open and asks Server 2 for the availability of all cluster servers that contain a replica of the database and for the replica ID of the database. (Depending on the method the user used to try to open the database, Notes may include the replica ID of the database when it sends the path name to the server.)
5. The Cluster Manager on Server 2 looks in the Cluster Database directory to locate servers that contain a replica and to obtain the replica ID of the database, if the Cluster Manager does not already have the replica ID. The Cluster Manager stores the server names in a text list in memory. If a server contains a replica that is marked OUT-OF-SERVICE or PENDING DELETE, the Cluster Manager does not include that server on the list unless that server contains an additional replica that is available.

6. The Cluster Manager on Server 2 looks in the server's cluster name cache to find the availability of each server on the list created in Step 5. The Cluster Manager deletes from the list any server that is in a MAXUSERS or RESTRICTED state. The Cluster Manager then sorts the list by availability and puts any servers that are in a BUSY state at the bottom of the list.

   The Cluster Manager sends Notes the list of server names, as well as the replica ID of the database. If one of the cluster servers contains multiple replicas, Server 2 sends a null replica ID to Notes.

   If Notes receives a null replica ID, Notes must use failover by path name rather than failover by replica ID. This type of failover can occur when you use selective replication and store multiple versions of a database on a single server. Notes uses failover by path name to be sure it chooses the correct version of the database.

   Note If you put multiple replicas on a server, be sure that all replicas in the cluster that use the same selective replication formula have the same path name. Otherwise, users may fail over to a different replica.

7. Notes looks for the database on the most available server that contains a replica. If that server is no longer available, Notes tries the next server on the list. It uses one of the following procedures to find the fully qualified path name of the replica on this server:

   If Notes received a null replica ID in Step 6, it does the following to find the fully qualified path name of the replica on this server:

   a. Notes starts with the path name of the original database the user tried to open. Notes adds the name of the new server and includes the full path to the database to makes a fully qualified path name.
   b. Notes uses NSFDbOpen on the server to be sure the database will open and then closes the database and returns the fully qualified path name back to the Name services function that Notes called in Step 2

   If Notes received a replica ID in Step 6, it does the following to find the fully qualified path name of the replica on this server:

   a. Uses NSFDbOpen to open the Notes data directory on the server.
   b. Uses a Name services function to locate the database in the Notes data directory or in one of its subdirectories. This function returns the fully qualified path name of the replica on this server.
   c. Notes uses NSFDbOpen on the server to be sure the database will open and then closes the database and returns the fully qualified path name back to the name service function that Notes called in Step 2.

8. Using the path name from Step 7, Notes issues a new command to open the database, such as the NSFDbOpenExtended command or whichever command Notes used originally, to open the database. This time Notes disables the Workload Balance option of the command so that the command will access a BUSY server if there is one on the list that Server 2 sent to Notes. Since the Cluster Manager put any BUSY servers at the bottom of the list, Notes tries to access BUSY servers only if it cannot access any other server.

If Notes does not find an available server that contains an available replica, or if the database Notes found in Step 7 is no longer available, Notes does not open the database and sends a message to the user.

Chapter 11
The Indexer

The Indexer keeps database views and full-text indexes up to date after documents are added, removed, and/or modified.

The Indexer

The Indexer is composed of two server tasks:

- The Update task
- The Updall task

These tasks use two sets of NOS services, the Notes Index Facility (NIF) and Full-Text services, to keep database views and full-text indexes up to date after documents are added, removed, and/or modified. Use of both Update and Updall is optional. You do not need to run them in order for client and server programs to see up-to-date views because a view will be updated automatically if it is out of date when a client or server program accesses it. However, it may be useful to run Update and/or Updall if a server has excess performance capacity. Running the tasks during normal hours of operation or during off-peak times enables Update and Updall to use the excess capacity to update views, thereby reducing the delays that clients and server programs experience when attempting to access the views. Both Update and Updall update out-of-date views. The difference between the two is that Updall runs once, updates all out-of-date views in all databases, and then stops; Update runs continuously, updating views in response to certain database activities as they happen.

By default, the Domino server runs Update continuously in the background and runs UpdAll every night at 2 AM to update any additional views and full-text indexes that Update did not manage to update during the day.

By default, both Update and Updall update a view only if it has an index in it. The presence of an index indicates that client and/or server programs have used the view. By default, Update and Updall ignore views that do not have an index in them. The tasks do not waste time building an index for a view that has not actually been used. This can happen because the template used to create a database may be designed to serve many needs and so may include many views, but the use of specific views in any one database created from the template may vary greatly.

The Updall task

The Updall task is a single instance of the Update task. Although Updall does not check or use the requests in $UpdateQueue, it performs the same tasks as Update. When Updall runs, by default at 2 AM, it refreshes outdated views and full-text indexes and discards expired view indexes. If you use the -r switch with Updall, by typing **load updall -r** at the server console, Updall discards and rebuilds every view in a database.

Discarding view indexes

When creating a view, a designer uses the Discard View property to define when to remove the view information, which is known as the view index, from the database. The choices are: Never, After each use, and If inactive for *n* days. The Updall task removes the view index from the database. Removing unused views saves database space and improves performance.

Even if the discard frequency is set to “After each use,” the view is not removed from the database until the Updall task runs.

The Update task

The Update task updates views by accounting for deletions and/or changes to existing documents and the creation of new documents.

Update is enabled, by default, on a Domino server. To enable one or more Update tasks, type **load update** one or more times at the server console or add Updaters=*n* to the server's NOTES.INI file, where *n* is the number of Update tasks to launch. Note that if you type **tell update quit** at the server console, all Update tasks on the server stop.

Using multiple Indexers improves the speed of indexing, but it may also cause a decrease in server performance, because more than one Update task may contend for access to the same view collection.

To configure the log file (LOG.NSF) to record the activity of each Indexer, use the Log_Update=1 setting in the NOTES.INI file.

```
Indexer Updating names.nsf view "($Users)"

Indexer Updating names.nsf view "($Users)"
```

$UpdateQueue and view changes

A request to update a view can be triggered by the Router, by replication, or by the server when it closes a database that has been modified. Update requests are stored in the $UpdateQueue, which holds a maximum of 500 requests. Each request contains the name and path of the database that needs updating. When running, the Update task polls the queue every five seconds to look for requests. If Update finds requests, it acts on them in sequential order. If the queue becomes full, some requests may be dropped, which may result in a client or server program having to synchronously update a view when it tries to access it instead of finding the view already updated by Update.

Each Update task takes a request and performs the necessary database updates. Multiple Update tasks can work on the same database, but semaphore-locking prevents two tasks from updating the same view or full-text index at the same time.

Suppression time and efficient view rebuilds

Although the Update task checks the $UpdateQueue every five seconds, it does not always immediately refresh a database view or full-text index. To increase efficiency and improve server performance, Update collects changes so that it can process many changes at once. The suppression time interval controls how often the Update task processes changes to a view or full-text index. By default, suppression time is set to five minutes. However, you can change this setting by adding Update_Suppression_Time=*x* to the NOTES.INI file, where *x* is the number of minutes.

The Domino Directory and indexing

In most cases, users, servers, and applications cannot access a view while the Indexer is updating it. However, an exception is made when accessing certain views in the Domino Directory. Because the Domino Directory is one of the most heavily used and most frequently updated databases and because the Domino Directory is needed to manage critical processes such as authentication and mail, Domino allows name lookups to access certain views of the Domino Directory that may not be quite up-to-date — ($ServerAccess), ($Users), ($Groups), ($NameFieldLookup). Although this view availability ensures quick mail routing and authentication, it also risks that the information retrieved from these views may not be current. This mechanism works because the Update and Updall tasks, which run in the background, eventually update the Domino Directory, so that the Domino Directory is always reasonably up-to-date for lookup operations.

Chapter 12
The Router

This chapter describes, in technical detail, how the Router moves a document to a recipient's mail file or transfers the document to another server on the route to the recipient's home server.

The Router

Lotus Domino provides mail services as part of its architecture. The Router, a specialized program on the server, moves a document to a recipient's mail file or transfers the document to another server on the route to the recipient's home server. Mail routing on a Domino server begins when the server receives a message from a client mailer, a Router on another server, or an application. The Router examines the message and determines whether its recipients are on the current server or on a different server. The Router then moves the message either to the recipient's mail file on the server, to the other server, or to both.

Components that the Router uses

The Router uses many components of the Domino and Notes architecture:

- Router task
- MAIL.BOX database (or multiple MAIL.BOX databases in Domino Release 5)
- Domino Directory
- Mail files
- NOTES.INI file for each server
- Domain Name Service (DNS) servers and/or hosts files
- (optional) SMTP Listener task
- (optional) Shared mail database
- (optional) Domino clusters

Mail protocols

Domino uses two protocols to route mail: Notes remote procedure calls (NRPC) and Simple Mail Transfer Protocol (SMTP). When using NRPC, Domino can route messages in Notes rich-text format (Notes RTF) and in MIME (Multipurpose Internet Mail Extensions) format. When using SMTP, Domino routes only MIME messages.

How the Router works

A Domino server's MAIL.BOX database acts as a mail repository. Mail routing begins when the Router finds a new or changed message in MAIL.BOX. The following steps describe basic mail routing:

1. The Router task initializes (for example, at server startup), builds the routing table, determines the maximum number of threads to allocate, and checks MAIL.BOX. If the Router finds new or changed messages in MAIL.BOX, its main thread builds the main message queue.
2. The main thread dispatches each queued message to a transfer queue or to a local delivery queue.

3. Transfer and delivery threads check the transfer and delivery queues, respectively, and process the messages in those queues.
4. The main thread checks the transfer and delivery queues for updates. Based on these updates, the main thread modifies the main message queue and the messages in MAIL.BOX.

Router initialization

When the Router task loads, its main thread uses a DatabaseOpen call to open each MAIL.BOX database on the local server. The Router thread performs a Search NSF call to check each mailbox for new or modified documents, such as documents that arrived since its last check or documents that the server administrator modified, for example, to correct a misspelled recipient name. The thread builds the main message queue, which contains an entry (a list item) for each new or modified message and summary information about the message, such as recipients, destination, date sent, and so forth. After building the message queue, the main thread dispatches each message in the queue.

Routing tables

Using information in the Connection documents, Domain documents, and Server documents in the Domino Directory, the Router builds the routing table and determines which servers are connected. The Router periodically checks the $Connections, $Domains, and $Servers views for edits and additions to these documents. The Router does not pick up a change until one of the views changes — for example, if you change a value in the Server document, the Router is not aware of it until the view refreshes. The Router tries to check the views before dispatching each message. If it finds changes, it reloads the routing table. The data structure of the routing table is significant to the routing algorithm. Essentially, the Router builds a graph of all servers in the domain and all servers that use Adjacent domain documents to connect to other domains in a format that ensures fast lookups.

Maximum threads

During startup, the Router evaluates the available memory on the server and determines the maximum number of transfer, concurrent transfer, and delivery threads that it can create and use. When the Router needs a thread but one is not available, it creates a new thread as long as this does not cause the total number of threads to exceed the determined limits.

Note An administrator can set the NSF_Buffer_Pool_Size setting in the NOTES.INI file to control how much physical server memory to allocate to Domino and, consequently, to specify the maximum number of threads. If NSF_Buffer_Pool_Size is not explicitly set, then the Router calculates a default value. Although an administrator can modify this setting, it's best to let Domino allocate the memory.

Message routing

To dispatch a message, the main thread checks each recipient's address for an *@domain.* If the address has an *@domain,* the thread attempts to determine whether the domain is a local domain. If the address does not have an @domain, the thread looks up the local part of the recipient's address in the Domino Directory on the server.

Determining local or external domain

If a recipient's address has an *@domain*, the Router needs to determine whether that domain is part of the local domain. If it is part of the local domain, the Router can look up the local part of the recipient's address in the Domino Directory. If it is not part of the local domain, the Router forwards the message to a server in the external domain.

To determine whether the domain listed in the recipient's address is part of the local domain, the main thread compares the domain to the Notes domain listed in the local server's Server document and to the Internet domains listed in the "Fully qualified Internet host name" field in the local server's Server document, and the "Local primary Internet domain" and "Alternate Internet domain aliases" fields in the Global Domain document in the Domino Directory. If there is a match, the Router removes the *@domain* from the recipient's address and looks up the local part of the recipient's address in the Domino Directory.

If there is no match, the Router treats the domain as an external domain.

If the domain is a Notes domain (for example, @Acme), the Router looks in the routing table for a connection to a server in that domain. If the domain is an RFC 822 / Internet domain, the Router looks in the DNS for MX or A records that correspond to that domain. After the Router determines a destination server (and, for Internet domains, MX or A records), it appends that information to the message and places the message in the transfer queue.

Note If the DNS lookup does not return any information for the receiving server or if the DNS server is down, the transfer thread marks the message DNS WAIT in the main message queue and retries the lookup later.

Looking up recipients in the Domino Directory

After the Router determines that a recipient is in the local domain, either by resolving the *@domain* portion of the address to a local domain or by determining that there is no *@domain* in the address, the Router looks up the recipient in the Domino Directory. For example, if a message is addressed to Jane Doe and Sales, the Router looks up both Jane Doe and Sales in the ($Users) view of the Domino Directory. The Router expands Jane Doe to Jane Doe/Tampa/Acme and expands Sales to John Smith/Charleston/Acme and Jeff Nguyen/Washington/Acme.

Next the Router determines the home mail server for the recipient. If the recipient's home mail server is the local server, the Router places the message in the local delivery queue. If not, Domino calculates how to route the message to the recipient's home server. The configuration of the local server and the message format determine how Domino moves the message to the server. If the message is in MIME format, the local server can send SMTP within the local Internet domain, and the home mail server can receive SMTP, the Router marks the message for SMTP and looks up the home mail server in the DNS. If the DNS returns either MX records or an A record for the home mail server, the Router appends these records to the message and moves the message to the transfer queue. In all other cases, the Router uses NRPC as the transfer protocol.

After the Router sets NRPC as the transfer protocol, the Router checks the routing path to the home mail server. The Router selects the route that has the lowest routing cost. If more than one path has the lowest cost, the Router chooses the one that has the fewest number of hops. If more than one path has the lowest cost and fewest hops, the Router selects among them alphabetically based on the name of the first server hop. For external domains — that is, Notes domains that are outside the current Notes domain — the Router employs a slight variation on this algorithm. When choosing between two routes that have the lowest cost and fewest hops, the Router selects alphabetically based on the name of the server *in the other domain*.

Note Routing cost is the total of the costs of all connections on a path to a server. A connection's cost is an arbitrary, relative integer. A fast LAN connection might have a cost of one, while a slow WAN connection might have a cost of five. If a connection is slow or fails, its routing cost increases.

Next, the main thread calculates the next server along this best route to which it should transfer the message — the "next hop server." The Router appends the server information to the message and puts the message in the transfer queue.

Group expansion

When a message is addressed to a group — for example, Sales — the Router performs group expansion — that is, it looks up the group in the Domino Directory and converts the group name into individual recipient names. Group expansion may create new message instances. A message instance is an in-memory copy of a message that the main thread can process. For example, group expansion creates a new instance for each group, which is then expanded to a list of recipient names.

Domino automatically eliminates duplicate recipients, which may occur when a message is addressed to an individual twice or when an individual is included both as a recipient and as a member of a group. To eliminate duplicates, Domino builds a vector of recipients and ensures that each entry in the array is unique. The Router performs recursive expansion (expanding groups within groups) to remove duplicate names. This process helps prevent a single recipient from receiving multiple copies of the message.

Delivery failures

If the main thread is unable to find a name in the Domino Directory or encounters a duplicate name, it generates a non-delivery failure. For example, if a message is addressed to Jane Chang but there is no Jane Chang or there are two entries (Jane I Chang and Jane L Chang) in the Domino Directory, Domino generates a non-delivery failure. Acting as the dispatcher, the main thread can insert new messages in MAIL.BOX — for example, delivery failures or new messages generated when a recipient has a forwarding address. The Router treats these as new messages.

When creating a non-delivery failure, the Router appends a copy of the message, changes the recipient name to the sender's name, adds an explanation of the failure, and moves the non-delivery failure message to MAIL.BOX.

Route calculation

Each server calculates the routing path for each recipient independently. This dynamic calculation allows Domino to adjust for changes in routing cost or routing problems while a message is in route. For example, Server A determines that a message should go to Jane Doe on Server D by transferring it from Server A to Server B to Server C and then to Server D. However, after the message reaches Server B, Server C becomes unavailable. If Server B uses the routing calculations performed by Server A, it cannot transfer the message. When Server B uses the routing algorithm to calculate the best path, the increased routing cost of a path to Server C — because of the server's inability to receive messages — causes Server B to route the message to Server E, which then routes it to Server D.

Message transfer queues

Transfer queues are in-memory queues that move a message from one server to another. Each transfer thread handles a single message for a single destination, but deals with all recipients of that message at that destination. The sending server tries to transfer as many pending messages to a destination as possible over an open connection for maximum efficiency.

For maximum efficiency, a sending server can create multiple transfer threads to a single destination. For example, suppose two messages are pending for transfer from Server A to Server B. Message 1 has a large attachment (50MB) and Message 2 is a 25KB text message. While one transfer thread is transferring Message 1 to Server B, another thread can transfer Message 2 to Server B. Using multiple transfer threads prevents large messages from creating a bottleneck in transfer between two servers. One thread, however, operates on each message — two threads would not attempt to operate on the same message.

If the transfer thread sends over NRPC, it makes a DatabaseOpen call to the MAIL.BOX database on the destination server. The thread performs a NoteUpdate to write the new message to MAIL.BOX and closes the session.

If the transfer thread sends over SMTP, it uses information in the message to determine how to connect to the destination server. The transfer queues list messages by domain and store MX record information. When handling a message, a transfer thread tries the MX record that has lowest cost and chooses randomly among records that have equal cost. if the initial connection attempt is unsuccessful, the thread retries the connection on an MX record that has higher cost (or a different record that has an equal cost). After selecting an MX record, the transfer thread attempts a TCP connection on port 25 to the receiving server at the designated IP address. If the connection is successful, the transfer thread initiates an RFC 821 conversation with the receiving server and attempts to transfer the message.

Transfer threads may have to convert messages from one format to another. For example, if the sending server is a Domino Release 5 Internet relay but the receiving server is a Domino Release 4 server, and the message being transferred is in MIME format, the transfer thread on the Release 5 server converts the MIME message to Notes RTF and then transfers it. The sending server determines whether the receiving server can handle MIME messages by checking the version of the receiving server.

Note You can see which transfer queues are operating by entering this command at the console:

```
Tell Router Show
```

The console displays the destination, number of messages pending, and state for each active transfer queue.

Message delivery

Domino treats "local delivery" as a special destination. When the Router dispatches a message, it notes that one or more recipients have mail files on the local server. It marks the message as "local delivery" for these recipients and places the message in the delivery queue. Local delivery is composed of multiple sub-destinations which are recipients' mail files. The main thread creates a queue in matrix form for each mail file. One axis of the matrix shows messages for delivery, and the other axis shows the local mail files on the server.

Delivery threads

A delivery thread handles delivery to each mail file. Delivery threads are essentially the same as transfer threads — they treat local delivery as a special destination. To deliver a message, the thread performs a DatabaseOpen call to the mail file, uses a NoteUpdate to deliver all messages pending delivery to that mail file, and closes the session. While the delivery thread is operating on the mail file, it locks the file so that there is no contention for access to the recipient's mail database. Domino allows multiple delivery threads to operate independently on multiple mail files at the same time. A single message can be delivered to multiple mail files at once, even though each delivery thread handles only a single destination and each destination is handled by only one delivery thread.

After the delivery thread delivers a message to a recipient's mail file, it marks that recipient Complete and deletes the message from the delivery queue. The Router updates the main message queue with this state information and removes that recipient and that message as work items.

Message cleanup

After a thread transfers or delivers a message, it updates the status of that message. For example, if Domino marks a message for local delivery to Jane Smith's mail file and a delivery thread successfully delivers the message to her mail file, the delivery thread marks that message as delivered in the delivery queue. The main thread checks the delivery and transfer queues for these updates and then reflects the changes in the main message queue and in MAIL.BOX. For example, after reading the change to the message addressed to Jane Smith, the main thread updates the information in the main message queue and in MAIL.BOX to reflect the delivery to her mail file.

If the Router processes all recipients of a message —meaning that the message was successfully transferred to a server along the best path to the destination, that it was delivered locally, or that the Router created delivery failure reports — the main thread deletes the message from the main message queue and then from MAIL.BOX. Deleting the message after processing prevents redelivery or multiple deliveries of a message if the server crashes. There is brief period of time between when the transfer and delivery threads update the main message queue and when the main thread updates MAIL.BOX during which a crash could cause redelivery of a message. For reliability reasons, it is better to risk redelivery than to risk non-delivery.

If a message has a delivery failure, the main thread uses the information generated by the delivery thread to update the message queue and MAIL.BOX.

After deleting a message from MAIL.BOX, the main thread checks to make sure that the message isn't being processed by other transfer threads. If not, it deletes the message from the transfer queues.

After updating messages in MAIL.BOX with new information from message processing, message cleanup ends.

Message state

Each message in the main message queue is in one of these states.

State	Meaning
NORMAL	The message is ready for processing.
DEAD	Message delivery has failed, and the server's attempt to send a non-delivery failure to the sender has failed.
HELD	The message would normally return a non-delivery failure; however, the server administrator has set the server to hold the messages so that the administrator can attempt to correct the error — for example, by entering a correct user name or address.
DNS WAIT	The server tried to use SMTP to send the message but was unable to get information from a DNS server. The server waits for an interval and retries the DNS server. If a message is in the DNS WAIT state for 24 hours, the Router generates a non-delivery failure to the sender. **Note** If you use the Domino Administrator or a Notes client to open MAIL.BOX and see messages in the queue, you cannot see messages in the DNS WAIT state. To determine the number of messages in this state, check the server statistic Mail.WaitingForDNS.

Message transfer threads

Transfer threads move messages from the transfer queues to other servers.

Number of threads

Domino can create multiple transfer threads to one or more destinations. While you can use a setting in the Configuration Settings document to control the number of threads, it's best to leave the setting blank so that the server can set the number of threads, based on available memory.

To configure the maximum number of threads, Domino uses an algorithm that evaluates the physical memory (RAM) of the server. The amount of RAM determines the Domino NSFBufferPool setting, which is 25 percent of RAM, by default. The maximum number of transfer threads is based on the size of the NSFBufferPool. The Router creates three threads plus one for every 32MB of memory in the NSFBufferPool, up to a maximum of 25 threads.

On some servers, such as ones with multiple partitioned servers, administrators should tune the size of the NSFBufferPool in the NOTES.INI file, since not all RAM is available to each server.

Processing transfer queue entries

After dispatching messages, the main Router thread starts transfer threads to process messages in the transfer queues. The main thread checks the transfer queues for destinations that have messages that have not been processed or that do not have a thread assigned to them. Essentially, the main thread is looking for work that is not yet assigned to a transfer thread.

When the main thread finds a message that is not yet assigned to a transfer thread, the main thread looks for an idle transfer thread. The main thread activates the idle thread, which checks the transfer queue and begins operating on the first unassigned message. If the main thread discovers work that can be done in parallel, such as processing multiple messages for one destination, it can activate multiple transfer threads at once. However, the main thread does not activate more transfer threads than are needed.

If the main thread determines that a destination has multiple messages pending, it does not immediately create multiple transfer threads (one thread per message). First, the main thread checks to see whether the server has successfully connected to this destination. If not, the main thread assigns a single transfer thread to the destination. This thread attempts to connect. If there has been a successful connection, the main thread can assign more threads to the messages for this destination. Regardless of protocol, the main thread does not initially assign multiple transfer threads to a single destination, because if the destination is unreachable, the threads would be busy but performing no work. Waiting to assign multiple threads until after one transfer thread successfully connects prevents inefficiency.

Transfer threads and protocols

Transfer threads are not procotol-specific; in other words, a transfer thread can use either NRPC or SMTP to transfer a message. However, destinations *are* procotol-specific. When a transfer thread selects an unassigned message to process, the message contains information telling the thread which protocol to use. A transfer thread does not choose the protocol; it uses the protocol assigned to the message that it is working on.

Multiple transfer threads to a single destination

Domino can assign multiple threads concurrently to a single destination, even if that destination uses Domino Release 4 servers. However, Domino does not assign multiple transfer threads to a single destination if the connection takes place over a slow network link. Assigning multiple transfer threads to work over a slow network link does not increase efficiency or speed. For NRPC, connections that rely on Connection documents are assumed to be slow; hence, Domino does not assign multiple concurrent transfer threads to a single destination that requires a Connection document.

Multiple MAIL.BOX databases and multiple transfer threads

Creating multiple MAIL.BOX databases greatly increases mail routing efficiency. Multiple MAIL.BOX databases are especially useful when a sending server is trying to connect with multiple transfer threads. When the transfer thread connects to MAIL.BOX, it performs a NoteUpdate that temporarily locks MAIL.BOX. The NoteUpdate transaction is lengthy by Domino standards and can present a bottleneck for mail transfer. If you create multiple MAIL.BOX databases, a transfer thread can move to an available MAIL.BOX if one MAIL.BOX is locked by another process. Multiple transfer threads and multiple MAIL.BOX databases on the receiving server result in fast processing of messages for that destination.

Note There is no relationship between the number of MAIL.BOX databases on the sending server and the number of transfer threads.

Message transfer over SMTP

To transfer a message over SMTP, the transfer thread attempts to use SMTP to connect to the destination server on port 25. If the connection is successful, the thread establishes a session with the receiving server and initiates an RFC 821 conversation. After the initial helo (or ehlo) handshake, the thread initiates a message loop for each message, sending the Mail From:, Rcpt To:, and Data: for each message, then moving through the same loop for the next message. After transferring all messages, the thread disconnects.

If the receiving server rejects a message, the thread generates a non-delivery report. If the server rejects a particular recipient, the thread creates a non-delivery failure to send to the sender. If the server rejects an entire message — for example, if the size is greater than the maximum message size allowed or if the message is attempting to relay through the mail system in a way that is not permitted — the thread generates a non-delivery failure to the sender for all recipients of that message. The Router creates the non-delivery failure as a new message in MAIL.BOX on the sending server.

Pipelining, available through theE/SMTP (Extended SMTP) feature, improves message transfer efficiency. Pipelining sends several commands — such as, Mail From: and Rcpt To: — in a single packet, rather than sending one command per packet.

Transferring to a Domino Release 5 server via SMTP

Domino includes an SMTP Listener task that listens on port 25 (by default) and receives messages sent over SMTP. The Listener task deposits the message in MAIL.BOX after itemizing the message from RFC 822 data into Notes items. Itemizing the message creates Notes items for each of the RFC 822 header items (such as From:, To:, Subject:, and so on). In most cases, RFC 822 items map directly to existing Notes items. In some cases, the name of the item changes slightly; for example, the RFC 822 To: header item maps to the Notes SendTo item.

Message delivery threads

Delivery threads function similarly to transfer threads. For performance reasons, delivery threads try to optimize the opening of messages, which is a relatively expensive transaction since it requires reading the message from MAIL.BOX into memory. Delivery threads maintain information about the mail files on the server for which the message is destined. If there is more than one recipient on the server, the first delivery thread opens the message and reads it into memory. After the thread delivers the message, it leaves the message open in memory for the delivery threads for the other recipients. The thread for the last recipient closes the message and thus removes it from memory.

This algorithm becomes more complicated when dealing with multiple message formats and format preferences. For example, a message in MIME format may have three recipients on a server. One recipient prefers MIME messages, and two recipients prefer Notes RTF messages. The first delivery thread opens the message, delivers it to the MIME recipient, and closes the message. The second delivery thread opens the message, converts it to Notes RTF, makes a copy of the message, and delivers it to the first Notes RTF recipient. The final delivery thread uses the converted copy of the message (in Notes RTF) and delivers it to the final Notes RTF recipient.

Making a copy of the message is necessary because the thread may need to alter the message — for example, by removing blind carbon copy (BCC) recipients from the message header. By operating on the copy of the message, the thread leaves the original message open and unaltered for the other delivery threads to use. This results in the creation of (*n-1*) copies of the message, where *n* is the number of recipients on the server. Instead of making a copy, the last delivery thread operates on the original message. Not making a copy for the final recipient improves performance. For example, if a message has three recipients on a server, the first two delivery threads make a copy of the message, and the final thread operates on the message itself.

Message processing and pre-delivery agents

When the delivery thread processes its copy of a message, it adds the delivery time to the message and modifies the recipient list, if necessary — for example, by removing BCC recipients. Next, if the recipient has set up an agent to run in the mail file before new mail is delivered, the delivery thread invokes the Agent Execution engine. The agent can instruct the thread to perform up to two tasks: to modify the folder into which the thread places the message, instead of placing it, by default, in the Inbox, or to delete the message. The agent cannot perform other actions on the message because the message is not yet in the mail file.

Rules processing

After running any pre-delivery agents, the thread checks to see if the user has created any rules for message processing. Rules sort or filter mail based on criteria defined by the user. If there are rules, the Router passes the rules to the NSF engine, which performs rules processing after the message is delivered to the mail file.

Then, the thread performs a NoteUpdate to deliver the message to the mail file. If the mail file's owner created rules for mail processing, the NSF engine runs those rules on the message. Rules can move or copy the message; change its importance; or delete it. If the rules require that the message be deleted, the NSF engine passes this information to the Router.

After performing the NoteUpdate, the delivery thread delivers any other messages queued for the mail file. After the delivery thread delivers and processes all messages, the thread calls the Update task, which refreshes the Inbox view in the user's mail file.

Post-delivery agents

The delivery of the mail triggers the Agent Manager, which runs any agents that are set to run after new mail arrives. After this, the delivery thread closes the session with the database.

Delivery failure

Message delivery may fail for a variety of reasons — for example, if the size of the user's mail file exceeds its database quota or if the server is out of disk space. If delivery failure occurs, the delivery thread generates a non-delivery report and places it in MAIL.BOX on the server. The delivery thread marks the message Done, and the main thread updates the status of the original message during message cleanup.

Controlling the Router task

Using the Domino Administrator or the server console, administrators can enter commands to cause the Router to perform certain tasks or actions. These commands begin with the syntax "Tell router." The commands are stored in the main message queue, which the Router checks periodically. When the Router checks the queue and finds the command, it performs the desired action.

Configuration changes

The Router updates its configuration dynamically. It reads configuration variables — for example, from the Configuration Settings document — every five minutes and updates its configuration.

Daily housekeeping performed by the Router

The Router performs maintenance tasks once a day at 4 AM. This includes compacting MAIL.BOX, which is useful but less necessary in Domino Release 5 than it was in Domino Release 4. In Release 4, deleted messages left a stub behind in MAIL.BOX; stubs were removed only during compaction. Hence, database size increased due to stubs until housekeeping compacted the database. Domino Release 5 processes messages differently — deleted messages no longer leave a stub behind.

Release 4 used a table known as the UNK table to store a reference to each type of Notes item, with information about what that item was. The Release 4 UNK table was limited to 64KB of memory. Once the table reached 64KB, the database no longer accepted messages or notes with new items. Compacting the database rebuilds the UNK table with items from notes in the database, discarding all other items. Compaction in Release 4 was necessary to keep the UNK table from reaching the 64KB limit, which prevented MAIL.BOX from accepting messages with items not already in the UNK table.

In Release 5, the UNK table can use a new property of the Release 5 ODS to allow up to 64,000 entries in the table. This removes the immediate need for compaction, since mail servers are extremely unlikely to accumulate 64,000 unique items. This property is enabled when the Router upgrades MAIL.BOX to the Release 5 ODS.

Compacting MAIL.BOX

In Domino Release 5, the Router attempts online in-place compaction. This allows MAIL.BOX to continue accepting messages during compaction and is more efficient than Release 4-style compaction. This type of compaction does not purge the UNK table, but does consolidate free space in the database into a block, improving efficiency. If the Router detects that the UNK table is approaching its limit (64,000 entries using the Release 5 ODS property; 64KB of entries otherwise), it does Release 4-style compaction to purge the table.

During Release 4-style compaction, MAIL.BOX is offline and cannot accept mail. This is an important consideration in an Release 4-infrastructure, since mail cannot be delivered to the server while MAIL.BOX is undergoing compaction. In Release 4 compaction, the Router makes a copy of MAIL.BOX under a temporary file name, compacts the copy of the database, and checks for new messages that have entered MAIL.BOX while the Router compacted the copy. If new messages have arrived, the Router copies them to the temporary copy of MAIL.BOX, deletes the old MAIL.BOX, and renames the temporary copy to MAIL.BOX. During the deletion and renaming process, there is a brief window during which locking prevents mail from entering MAIL.BOX. This could potentially result in a non-delivery failure, though the window is quite small.

To compact MAIL.BOX, the router's main thread closes all threads and deletes all message queues. The Router then calls the Compact task to compact MAIL.BOX, then reactivates the threads. The Router rebuilds the main message queue, reading in all messages from scratch.

With multiple MAIL.BOX databases in Domino Release 5, the Router performs this process on each database in turn. The Router marks each database as in use during compaction, so a request for MAIL.BOX causes the server to establish a session with one of the MAIL.BOX databases that is not in use.

Failover in clustered mail servers

During dispatching, the Router calculates the next hop for a message on the path to its destination. That calculation includes the routing cost information that is stored in the routing table. Every connection has a cost bias that reflects the bandwidth and speed of the route. If a server becomes unavailable, the Router adds to the cost bias for connections to that server. Clustering servers helps prevent transfer failures to a server — if one server in a cluster becomes unavailable, mail is re-routed to other cluster members. From there, the mail can be routed or held until the server becomes available.

Cluster algorithm

The cluster algorithm comes into play when a server is unavailable and the Router attempts to fail over to another server.

Destination server in the same cluster

If the next hop for a message is the recipient's home server and the current server is a member of the home server's cluster, the Router looks on the current server for a replica of the recipient's mail file. If there is a replica of the mail file, the Router marks the message for local delivery to that replica. If there is no local replica, the Router consults the Cluster database (CLDBDIR.NSF) to find a cluster member that has a replica. If a cluster member has a replica of the mail file, the Router transfers the message to that server. If not, the Router queues the message for the original destination (the unavailable server) and lets the transfer thread retry transfer after the server becomes available.

Destination server in a different cluster

If the next hop for a message is the recipient's home server and that server is a member of a cluster that does not include the current server, the Router transfers the message to another member of that server's cluster.

Cluster cache

When the Router performs a DatabaseOpen call to open a remote MAIL.BOX on another server, the call queries the cluster cache on the server. The cache, CLUSTER.NCF, is a file that lists the members and states of all clusters. The cache is built when the client or server asks for cluster information and is updated periodically. The Router uses the Notes API to query the cache, locate cluster members, and determine their availability. The cache selects the least-busy server in a cluster to fail over to.

Domino Release 5 cluster enhancements

Domino Release 5 allows intermediate-hop failover to provide additional reliability and ease of routing. A Domino administrator can set which hops should use failover — all hops, the last hop, or no hops. If you use clustering in Release 5, it is helpful to cluster hubs, so that a hub failure does not cause a disruption in mail routing.

Message-delivery retry algorithm

The Router includes a retry algorithm at several levels. Retrying occurs after a thread attempts to connect to a server but the connection fails. The retry algorithm is independent of protocol and can occur for delivery as well as transfer.

Transfer threads

When a transfer thread tries to connect to MAIL.BOX (via NRPC) or the SMTP listener task (via SMTP) on a destination server and the connection times out or fails, the thread records the message state as RETRY. The destination is marked RETRY, which prevents the Router from assigning other threads to that destination. Initially, the Router waits 15 minutes and then tries again to connect. After the retry interval expires, the main thread activates a transfer thread and retries a connection to the destination server. If this connection attempt fails or times out, the destination remains in RETRY and the interval increases to 30 minutes. After 30 minutes, the main thread again wakes a transfer thread, which attempts to connect. If this attempt fails or times out, the destination remains in RETRY and the interval increases to 45 minutes. Subsequent attempts follow the same pattern, but the interval remains 45 minutes. While a destination can remain in RETRY mode for as long as it is unreachable, individual messages expire after 24 hours and generate non-delivery failure reports to the sender. To see what threads and destinations are in RETRY state, an administrator can enter the command **tell router show** at the console.

Message level retry

Individual messages can also go into RETRY mode. Over SMTP, there may be a transient error that interferes during message transfer. The transfer thread aborts the attempt to transfer that message and then retries the transfer. Over NRPC, the connection may time out in trying to transfer a message, leading the thread to retry the transfer later. RETRY may happen for messages on delivery as well — if the user’s mail file is locked by another process or is being compacted, the local destination (the mail file) is marked RETRY, and the delivery thread attempts delivery later.

Glossary

A

About This Database document

The About document contains information supplied by the database designer, and is often a description of the database purpose and contents. To open this document, choose Help - About This Database.

accelerator key

A key used in conjunction with the ALT key to trigger an action. For example, ALT+F shows the File menu. In Notes menus, accelerator keys are underlined. See Extended accelerator key.

access control

A security feature that determines the tasks that each user, server, or group of users or servers in a Lotus Notes database can perform. Some can do all tasks while others may be limited to specific tasks.

access-controlled section

A defined area on a form that allows only certain users to edit the fields in the section. Besides fields, it can include objects, layout regions, and text.

ACL (access control list)

A list of database users (individual users, Lotus Domino servers, and groups of users and/or servers) created and updated by the database manager. The ACL specifies which users can access the database and what tasks they can perform.

ACL Monitor

A document created in the Statistics & Events database that causes the Event task on a server to monitor a specific database for ACL changes.

Adjacent Domain document

This document defines the name, location, and access to adjacent (connected) and non-adjacent (unconnected) Domino domains and non-Domino domains. It is stored in the Domino Directory.

Administration Process

A server task (Adminp) that automates many administrative tasks. You initiate the tasks, and the Administration Process completes them for you. Some of the tasks the Administration Process can automate are: recertifying Notes IDs, renaming and deleting references to Notes users and groups, creating replicas of databases, and moving databases.

administration server

The server that you assign to apply Administration Process updates to a primary replica.

agent

A program that performs a series of automated tasks according to a set schedule or at the request of a user. An agent consists of three components: the trigger (when it acts), the search (what documents it acts on), and the action (what it does).

Agent Builder

Where users create these types of agents: simple action, formula, LotusScript, or Java. An agent automates a task and can be run by a user or according to a set schedule.

Agent Manager

The background server program that manages and runs agents on a server. An agent performs a series of automated tasks according to a set schedule or at the request of a user. The Agent Manager runs by default on a server. You set guidelines for the Agent Manager in the Server document in the Domino Directory.

alarm

For end-users: Reminds you of an entry on your Calendar. When an alarm goes off, Notes shows a description of the Calendar entry associated with the alarm.

For administrators: A document generated in the Statistics database indicating that a server statistic has exceeded a specified threshold. For example, an alarm can notify you if disk space on server drive C drops below 10 percent. You create Statistics Monitor documents to configure alarm reporting.

alias

An additional name for a form, view, or keyword.

alternate mail

A mail system other than Notes mail.

anonymous access

Lets users and servers access a server without authentication. This level of access is useful for providing the general public access to servers and databases for which they are not certified. It is typically used for granting access to the servers and databases on a Web site.

API (application programming interface)

A set of functions that gives programmers access to another application's internal features from within their own application. Notes and Domino offer several APIs that give developers access to Notes and Domino features and functionality, enabling them to create powerful and customized client and server applications.

application proxy

A firewall configuration that examines the destination of a packet and the type of information it contains, checks whether your network allows delivery to that destination, and controls the information flow between internal and external clients and servers.

attach

To store a file with a Lotus Notes document or form. The file, or attachment, is stored with the document or form in the database until you delete one of them. If you mail the document, the attachment is mailed with it.

attachment

A file attached to a Lotus Notes document or form or to a Web page document. An attachment remains with the document until you delete the attachment or the document.

authentication

A security mechanism that verifies the identities of clients and/or servers. There are three types of authentication in Notes and Domino -- Notes/Domino, SSL, and name-and-password authentication.

- Notes/Domino

  Verifies that the user or Domino server trying to access a particular Domino server has a trusted certificate in common with it. Authentication occurs in both directions: the server authenticates the user, then the user authenticates the server.
- SSL authentication

  Used by Internet protocols over SSL. During SSL authentication, the server exchanges the server certificate with the client and, optionally, the client exchanges the client certificate with the server. This exchange determines whether the client and server have a certificate in common and verifies the identities of the server, and optionally, the client.
- Name-and-password authentication

  Used by Internet protocols over TCP/IP and SSL. During name-and-password authentication, a text password is sent by the client to the server. The server verifies the identity of the client by making sure the password provided matches the password stored in the Domino Directory for that person. Name-and-password authentication does not use certificates.

Author access

An access level that allows users to create and read documents and edit the ones they created and saved. Servers with Author access can replicate new documents and can usually delete documents marked for deletion. Access levels can be further refined using roles and access restrictions.

Authors field

A field that lists the names of people who have Author access. This field does not override the access control list. Use this field to control edit access on a document-by-document basis.

autolaunch

When a user creates or opens a document for reading or editing, an embedded object automatically opens in its native format. The user can change the embedded object and add a new object directly in the application that created the object. The Notes document automatically displays the changes.

autoregistration

A process by which external databases may be automatically added when the database type and path are supplied during connection.

B

billing

A feature that enables a Domino server to track specific Domino activities for a chargeback of server processing time. The billing server task collects this information and records the data.

binary tree server topology

Connects servers in a pyramid fashion: the top server connects to two servers below it, each of which connects to two servers below it, and so on. Information travels down the pyramid and then back up.

bookmark

A customizable, graphical link to databases, views, documents, Web pages, and newsgroups.

bookmark folder

A folder in the Bookmark Bar containing bookmarks.

broadcast meeting

Invitees are notified about a meeting but do not need to respond to the invitation. This option is useful when individual response will not affect the occurrence of the meeting.

button bar

In the Notes client, the bar that displays actions as buttons. Actions let users click to accomplish tasks, from mimicking the Notes menus to tasks defined by formulas or a LotusScript program.

C

CA (Certificate Authority)

The link that allows a server and client to communicate. A CA vouches for the identity of a server and client by issuing certificates stamped with the CA's digital signature and including the CA's trusted root certification. The digital signature assures the client and server that both the client certificate and the server certificate can be trusted. If the client and server can identify the digital signature on the certificate, then a secure SSL session can be established. Otherwise, the client and server cannot authenticate each other, and the session cannot be established. Clients and servers identify digital signatures by comparing them against the trusted root certificate.

A CA can be a third-party, commercial certifier, such as VeriSign, or a certifier that you establish at your organization using Notes and Domino. Third-party and Domino CAs create both server and client certificates.

Calendar

A view in your Notes mail database that you can use to manage your time and schedule meetings. You can add appointments, meetings, reminders, events, and anniversaries to the Calendar view. You can also display tasks in the Calendar view.

Calendar profile

A document that lets you customize your calendar and indicate the times that you are available for meetings. Notes saves this information in a database on your mail server; you define who can look up the information.

canonical format

A format for storing hierarchical names that displays the hierarchical attribute of each component of the name. For example, the canonical format for the name Reuben D. Smith/Ottawa/Acme/CA is: CN=Reuben D. Smith/OU=Ottawa/O=Acme/C=CA

where:

CN is the common name

OU is the organizational unit

O is the organization

C is the country code

category

A word, phrase, or number used to group documents in a view.

certificate

A certificate is a unique electronic stamp that identifies a user or server. Domino uses two types of certificates: Notes certificates and Internet certificates.

A Notes certificate is stored in a Notes or Domino ID file that associates a name with a public key. Certificates permit users and servers to access specific Domino servers. An ID may have many certificates.

An Internet client certificate lets a user access a server using SSL client authentication or send an S/MIME message. The client certificate is stored in either the Notes ID file if you are using a Notes client or in a file stored on the user's hard drive. An Internet server certificate lets users access a server using SSL server authentication. The server certificate is stored in a key ring file on the server's hard drive.

Internet certificates contain a public key, a name, an expiration date, and a digital signature.

certification

Process that creates special signed messages called certificates, which state that a particular public key is associated with a particular user or server name. Domino automatically issues Notes certificates for users and servers when you register them.

Certificate Authority certificate

A binary file stored on the CA server's hard drive that contains a public key, a name, and a digital signature. The CA certificate identifies the Domino or third-party CA.

certifier ID

A file that generates an electronic "stamp" which indicates a trust relationship. It is analogous to the device used to stamp passports -- it verifies that a person is trusted by that stamping authority.

CGI

Abbreviation for Common Gateway Interface (CGI). CGI is a standard that connects external programs with information servers such as Web servers or HTTP servers. CGI scripts are a common way of customizing information presentation and retrieval on the Web; they can run within databases and on a Domino server.

chain server topology

Connects servers one-to-one, end-to-end. Information travels along the chain and then back.

character set

A set of binary codes that represent specific text characters.

child document

A document that inherits values from another document (the parent document).

client certificate

An electronic stamp that contains a public key, a name, an expiration date, and a digital signature. The client certificate uniquely identifies the user and is used when accessing a server using SSL and sending encrypted and signed S/MIME messages.

The client certificate is stored in the Notes ID file if you are using a Notes client or on the user's hard drive.

CLS files

Country Language Services (CLS) files convert characters such as foreign currency symbols and accented letters to other characters when importing or exporting files. CLS files also control the order in which characters are sorted.

cluster

A group of two to six Domino servers that you set up to provide users with constant access to data, balance the workload among servers, improve server performance, and maintain performance when you increase the size of your enterprise.

collapse

An action that hides documents under categories or hides response documents under main documents in a view.

@command

A special @function that performs an immediate action in the user interface.

command key

A key that directly triggers an action and usually makes use of the CTRL (Windows) or COMMAND (Macintosh) keys. For example, to print press CTRL+P (COMMAND+P on Macintosh).

compact

To compress a database, in order to reclaim space freed by the deletion of documents and attachments.

computed field

On a form, a field whose value is determined by a formula that you write.

Connection document

In the Domino Directory, a Connection document enables communication between two servers and specifies how and when the information exchange occurs. In the Personal Address Book, it describes how a client accesses a certain server.

create access list

A list that restricts a form, so that only specified users can create documents using the form.

criteria

Data you specify so that your application can select records during a query. You can use matching criteria, in which records must match the criteria you set, or formula criteria, which use logical formulas and @functions to test records.

cross-certificate

Domino uses two types of cross-certificates: Notes and Internet. Notes cross-certificates allow users in different hierarchically-certified organizations to access servers and to receive signed mail messages. Internet cross-certificates allow Notes users to secure S/MIME messages and verify the identity of a server using SSL.

Cross-certificates are stored in the Domino Directory or Personal Address Book.

D

data directory

Directory that contains local databases, local database templates, country language services (CLS) files, DESKTOP.DSK files, and if you're using OS/2 or UNIX, your NOTES.INI file.

data note

A document in a Notes database.

data type

The type of information that one field in a database can store, for example, text, rich text, numbers, keywords, and time.

database

A collection of documents and their forms, views, and folders, stored under one name. Notes databases can be part of a Web site or part of a Notes application.

database cache

A section of memory on a Domino server where databases are stored for quick access. You can display cache statistics, change the number of databases that a server can hold in its cache, close all databases in the cache, and disable the cache.

Database Catalog

A database containing information about databases stored on a single Domino server, a group of servers, or all the servers in a domain. Database Catalogs are commonly used to let users add the databases in them to the users' desktops.

database header

An internal structure that stores database-wide information such as, a time stamp that indicates when a database was first created or when the Fixup task last ran on it.

DBID (database ID)

The time stamp that is located in the database header and that indicates when a database was first created or when the Fixup task last ran on it.

DBIID (database instance ID)

A value that is located in the database header and that associates the database with specific entries in the transaction log.

database library

A database that provides information about databases including descriptions, replica IDs, and manager names. Database libraries can be local and describe databases on a workstation or can be on a server and describe shared databases. Database libraries are commonly used to let users add the databases in them to the users' desktops. Related databases can be grouped together for easier access.

database manager

A person with Manager access to a database whose responsibilities include setting up and maintaining access to the database and monitoring database replication, usage, and size.

database replica

A special copy of a database that, because it shares a replica ID with the original database, can exchange information with it through replication.

DDE (Dynamic Data Exchange)

DDE is a method for displaying data created with other Windows and Presentation Manager applications, such as graphics or spreadsheet ranges, within Notes documents. DDE objects can be reactivated and updated to reflect the current state of changing data.

default value formula

The formula that lets you set an initial value for an editable field.

default view

The view displayed the first time you open a database.

deletion stub

A truncated document that is left in a database in place of the original document to indicate to the Replication task that the document should, in fact, be deleted from all other replicas.

Depositor access

An access level where users can create documents but can't read any of the documents in the database.

design pane

The workspace area that displays design options, as well as areas to enter design information.

design template

A database design that lets you share design elements among databases and store design elements with a template. You can enable the template so that when it changes, the change automatically occurs in all databases created with that template.

designer

The person who creates and develops a database or an application, pilot tests it, refines it as necessary, and delivers it to the database manager.

Designer access

An access level where users can compose, read, and edit any documents, plus modify the database icon, About and Using documents, and all design elements. Servers can replicate all of the above and, if they have delete access, deletions.

DESKTOP.DSK

File that contains information about your workspace.

detach

To make a local copy of a file that is attached to a Lotus Notes document.

dialup

A connection type, usually a port, that is not on a local area network and must be accessed by modem and telephone lines.

dialog box

A box that appears when an application needs additional information to complete a task. A dialog box can contain check boxes, command buttons, option buttons, list boxes, information boxes, scroll buttons, drop-down boxes, and text boxes.

digital signature

The electronic equivalent of a handwritten signature, a digital signature is a unique block of text that verifies a user's identity and is appended to a message. The signature can be used to confirm the identify of the sender and the integrity of the message. The block of text is encrypted and decrypted using public and private keys.

digital speech synthesizer

A device used with screen readers to portray what is on screen through voice.

DIIOP (Domino Internet Inter-ORB Protocol)

A server task that runs on the server and works with the Domino Object Request Broker to allow communication between Java applets created with the Notes Java classes and the Domino server. Browser users and Domino servers use IIOP to communicate and to exchange object data.

Directory Assistance

A feature that allows you to extend client authentication, name lookups, and LDAP searches to secondary Domino directories and to LDAP directories. You use the Directory Assistance database to set up directory assistance.

Directory Assistance database

A database used by directory assistance that serves as a directory of secondary Domino directories and of LDAP directories.

Directory Catalog

A database that contains Person, Group, Mail-In Database, and Scheduling Resource entries from one or more Domino Directories. Servers use Directory Catalogs for quick name lookups, and Notes users use a Mobile Directory Catalog to easily address mail to people throughout an organization, even when disconnected from the network.

DNS (Domain Name Service)

An Internet service that translates domain names into IP addresses.

document

A Notes database entry that users create by using a form on the Create menu. Documents consist of fields, text, numbers, graphics, and so on. Information may be entered by a user, automatically calculated by formulas, imported from other applications, or linked to another application and dynamically updated.

domain

A Domino domain is a collection of Domino servers and users that share a common Domino Directory. The primary function is mail routing. Users' domains are determined by the location of their server-based mail files.

For a Domino server to communicate with a server in a different domain, you create a Domain document in the Domino Directory to define the name, location, and access to adjacent and non-adjacent Domino domains and non-Domino domains. Other domains are:

- Foreign domain

  A Domino domain and an external mail system such as SMTP or cc:Mail. It specifies which outbound addresses are Internet addresses and where the Notes Mail Router sends those messages.
- Global domain

  A group of Domino domains, such as Sales1, Sales2, and Marketing, under a single Internet domain, such as acme.com. All outbound SMTP mail, whether it originates from the Sales1 or Marketing domains, has the return address acme.com.

Domino Directory

A directory of users, servers, groups, and other entities -- for example, printers. It is a tool that administrators use to manage the Domino system, for example, to connect servers for replication or mail routing, to schedule server tasks, and so on. In previous releases the Domino Directory was called the Public Address Book.

Domino server

A computer that runs the Domino Server program and stores Notes databases.

Domino Server program

The program that supports the connection between clients and the serer and also manages a set of server tasks, which are programs that either perform schedule-driven database chores -- such as, routing messages to mailboxes and updating user accounts -- or connect various types of clients -- Notes clients, Web browsers, CORBA clients -- to the server.

E

ECL (Execution Control List)

An ECL is a feature accessed through the User Preferences dialog box that enhances security of your workstation data. The ECL lets you control which formulas and scripts created by another user can run on your workstation.

Edit mode

The state in which you can create or modify a document.

editable field

On a form, a field whose value is determined by a formula that you write to supply a default value, edit the user's entry, and validate the entry to make sure it meets specific requirements.

Editor access

An access level that allows users to create, read, and edit any documents. Servers can replicate new documents, change existing documents, and, if they have delete access, make deletions.

electronic signature

A stamp added to mail messages, fields, or sections that verifies that the person who originated the message is the author and that no one has tampered with the data.

encryption key

Security feature that ensures that only the intended recipient can read encrypted text. Every Notes user ID contains two: a public key for sending and encrypting and a private key for receiving and decrypting. Users may also have a public and private key for S/MIME encryption and signatures.

event

In LotusScript, an action or occurrence to which an application responds. That action can be a user-generated one, such as a mouse click; a system-generated one, such as the elapsing of a set amount of time on the computer's clock; or an application-generated one, such as the saving of a document via the product's autosave feature. Each LotusObject can respond to a predefined set of events, those defined for the class that the object is an instance of. Events are the primary way to initiate the execution of scripts: when a script is attached to an object event, it is executed when the event occurs.

In the Calendar, an entry with a duration of at least one day. For example, an all-day meeting or a vacation is an event.

event script

A script attached to a particular event. Examples in LotusScript are Initialize, Queryopen, and Postopen. When the event occurs, the script runs.

export

Save a Notes document or view in a non-Notes format.

extended accelerator key

Additional accelerator keys, used for bookmarks and task buttons. To view the extended accelerator keys, press and hold down the ALT key.

extranet

An intranet with extended access, generally behind a firewall. For example, a company may give the public access to certain parts of its intranet and restrict access to others. This can be done by using firewall programs or routers, via a proxy, or by specialized software.

F

failover

A cluster's ability to redirect requests from one server to another. Failover occurs when a user tries to access a database on an unavailable server or one in heavy use, and the user instead connects to a replica of the database on another (available) server in the cluster. Failover is transparent to the user.

field

On a form, a named area containing a single type of information. The field's data type determines the contents -- text, rich text (including styled text, graphics, and multimedia), numbers, or time-date.

firewall

A firewall is a system that is designed to control access to applications on a network. Typically, a firewall controls unauthorized access to a private network from the public Internet.

folder pane

The workspace area that shows the folders and views available in the opened database.

form

Forms control how you edit, display, and print documents. A form can contain fields, static text, graphics, and special objects. A database can have any number of forms.

formula

An expression that has program-like attributes; for example, you can assign values to variables and use a limited control logic. Formulas are best used for working within the object that the user is currently processing. The formula language interface to Notes and Domino is through calls to @functions.

You can write formulas that return a value to a field, determine selection criteria for a view, create specific fields in a form, determine the documents a replica receives, help users fill out a document, increase database performance, and create buttons or hotspots.

FTP (File Transfer Protocol)

A protocol used to transfer files from one computer to another. FTP also refers to the actual application used to move files using the FTP protocol.

full-text index

A collection of files that indexes the text in a database to allow Notes to process users' search queries.

full-text search

Search option that lets you search a database for words and phrases, as well as perform more complex searches using wildcards and logical operators.

@function

A built-in formula that performs a specialized calculation automatically.

G

group

A named list of users and/or servers. It can be used in Domino Directories, Personal Address Books, access control lists, and so on.

groupware

Applications that enhance communication, collaboration, and coordination among groups of people.

H

hierarchical naming

A system of naming associated with Notes IDs that reflects the relationship of names to the certifiers in an organization. Hierarchical naming helps distinguish users with the same common name for added security and allows for decentralized management of certification. The format of a hierarchical name is: common name/organizational unit/organization/country code -- for example, Pam Tort/Fargo/Acme/CA.

hierarchical view

A view that distinguishes between main documents and response documents. Each main document has its response documents indented under it.

hop

An intermediate stop on the path along which mail is routed when the sender's server and recipient's server are not directly connected.

hotspot

Text or a picture in a rich text field that a user can click to perform an action, run a formula or script, or follow a link.

HTTP (Hypertext Transfer Protocol)

An Internet protocol used to transfer files from one computer to another.

hub-spoke server topology

Establishes one central server as the hub and other servers as the spokes. The spokes update the hub server by replication and mail routing, and the hub in turn updates each spoke. Hub servers replicate with each other or with master hub servers in organizations with more than one hub.

hunt group

A group of servers that are assigned one phone number. Clients dial the one phone number and connect to any available server. Hunt groups balance the load on servers.

I

ICAP (Internet Calendar Access Protocol)

Network protocol that lets a client access, manipulate, and store Calendar information on a server. ICAP can be used either as a set of capability extensions to IMAP4 to create a server that supports both messaging and Calendar functions, or as a stand-alone protocol for a server dedicated only to the Calendar.

IIOP (Internet Inter-ORB Protocol)

An Internet protocol that implements CORBA solutions over the Web. IIOP lets browsers and servers exchange complex objects, unlike HTTP, which only supports transmission of text.

IIS (Internet Information Server)

The Microsoft Internet Information Server is a Web server that lets you browse HTML and Active Server pages. Domino includes an IIS product extension that lets you browse Domino databases using IIS.

IMAP (Internet Message Access Protocol)

Mail protocol that allows clients running it to retrieve mail from a host mail server also running the protocol. IMAP is similar to POP3 but has additional features. For example, it supports three modes of mailbox access. You can enable IMAP on a Domino server.

input-translation formula

In an editable field, the formula that converts or translates entered information into a specified value or format.

input-validation formula

In an editable field, the formula that verifies that the entered information meets the specified criteria.

intranet

A computer network with restricted access. Companies use intranets to share information internally. Increasingly, intranets are built as private Internets: a TCP/IP network based on Web standards like HTML, SMTP, or POP3. The difference is access -- anyone can access the Internet with the appropriate software, but only employees can access an intranet. See extranet.

ISAPI (Internet server application programming interface)

The Internet server application programming interface supported by IIS. Developers use this interface to create programs, called extensions, that extend the capabilities of IIS.

ISDN (integrated services digital network)

An international communications standard for sending voice, video, and data over digital telephone lines.

item descriptor

Stored in an array of fixed-size structures in a note header, each item descriptor describes one note item. Each structure has information describing the item name, type, value, size, and so on.

ISP (Internet Service Provider)

A company that provides access to the Internet.

K

key ring file

A binary file that is protected by a password and stores one or more certificates on the server hard drives. Domino uses two types of key ring files: server and CA. You do not use a key ring file for client certificates.

keyboard shortcut

A key combination that can be pressed instead of using a command from a pull-down menu. CTRL+letter and SHIFT+letter are the most common keyboard shortcuts. Some products let users define their own keyboard shortcuts; these shortcuts may be single keys or key combinations.

keywords field

A multiple-choice field that lets users make selections by clicking, rather than typing, an entry. Keywords fields can display in several formats, including a drop-down list box, a check box, and a radio button.

L

layout region

On a form or subform, a fixed-length design area in which related elements can be dragged and moved easily and can be displayed in ways not possible on regular forms and subforms.

LDAP (Lightweight Directory Access Protocol)

A set of protocols for accessing information directories. LDAP is based on the X.500 protocol, but supports TCP/IP, which is necessary for Internet access. Because it's a simpler version of X.500, LDAP is sometimes called X.500-lite. You can enable LDAP on a Domino server to allow LDAP clients to access information in the Domino Directory, for example, e-mail addresses.

LDAP directory

A hierarchical directory of names that can reflect an organization's structure or geography and that is accessed via the LDAP protocol.

Running LDAP on a Domino server enables the Domino Directory to serve as an LDAP directory. Two popular public LDAP directories are Bigfoot and Four11.

letterhead

The particular way that your name, the date, and the time appear at the top of the mail messages you create. You can choose from several letterhead styles.

library

A database containing lists of other databases.

license

Determines which databases, templates, and functions users have access to and the extent to which they can perform design and administrative tasks.

LICS (Lotus International Character Set)

A character set supported by Notes.

link

An icon that gives you direct access from one Notes document, view, or database (the source object) to any other document, view, or database (the target object). Notes opens the target object without closing the source object you branched from.

local database

A database is local if it can be accessed only by programs running on the same computer.

LMBCS (Lotus Multibyte Character Set)

The format in which Notes stores all internal text, except file attachments and objects. As a result, any user can edit, forward, and mail documents and work with databases in any language.

All text leaving the system -- that is, displayed, printed, and exported -- is translated from LMBCS to the appropriate character set. LMBCS supports Western and Eastern European, North American, and Asian languages.

LN:DO

Lotus Notes:Data Object is an LSX-compliant module that allows the use of LotusScript scripts for external data access applications.

local database

A Notes database stored on your computer's hard disk drive, on a disk, or on a networked file server.

Location document

A document in your Personal Address Book that contains communication and other location-specific settings you use when you work with Notes in a specific place. You can create as many Location documents as you need.

LotusObject

Any object that is an instance of a Lotus-product class. LotusObjects can be manipulated using LotusScript. LotusObjects share a common design. Many are implemented either the same way across products, or almost the same way, with slight variations from product to product.

LotusScript

A version of Basic that offers not only standard capabilities of structured programming languages, but a powerful set of language extensions that enable object-oriented development within and across products. Its interface to Notes is through predefined object classes.

LS:DO

The ODBCConnection, ODBCQuery, and ODBCResultSet classes, collectively called the LotusScript Data Object (LS:DO), provide properties and methods for accessing and updating tables in external databases through the ODBC (Open Database Connectivity) Version 2.0 standard.

M

macro

A program that performs a series of automated tasks on behalf of the user. A macro consists of three components: the trigger (when it acts), the search (what documents it acts on), and the action (what it does). Also called an agent.

Manager access

An access level that allows users to compose, read, and edit any documents; modify the access control list, database icon, About and Using documents, and all design elements; define replication settings; and delete the database. Servers can replicate all the above and, if they have delete access, deletions.

MIME (Multipurpose Internet Mail Extensions)

Software that allows you to attach non-text files to Internet mail messages. Non-text files include graphics, spreadsheets, formatted word-processor documents, and sound files.

MSAA (Microsoft Active Accessibility)

An enabling technology, used to make software more accessible for people who use devices such as screen readers. It helps to distinguish user interface elements, items in documents, and the organization of documents.

MTA (message transfer agent)

A program that translates messages between mail formats. Also called a gateway.

N

Name & Address Book

Now called the Domino Directory or Personal Address Book.

named element

A specific design element in a database -- for example, a view or folder.

named-object table

The named-object table maps names to associated notes and objects. For example, this table manages per-user unread lists.

named style

A collection of styles that you can apply to other data in a file. Styles stored in a named style can include number format, typeface, type size, underlining, bold, italics, lines, colors, and alignment.

navigation pane

The pane that either displays icons for all views, folders, and agents in a database, or displays the current navigator.

navigation buttons

Browser-like buttons in Notes used to navigate among open pages of databases or Web pages. Button functions include back, forward, stop, refresh, search, and go.

navigator

Programmed graphics in the user interface that direct users to specific parts of a database without having to open views. Navigators usually include hotspots, and can do simple actions such as opening a database, document, URL, view or folder, or even another navigator.

negotiated session key

Encryption key that is created at the beginning of the SSL handshake, which determines the key used when encrypting information over an SSL connection. The negotiated session key changes each time a new session is initiated.

newsgroup

An online discussion group that users with newsreaders can participate in. A Domino NNTP server can store USENET newsgroups, public newsgroups distributed on the Internet, and private newsgroups.

newsfeed

The periodic transfer of newly posted newsgroup articles from one NNTP server to another using the NNTP protocol. If you enable the NNTP protocol on a Domino server, you can set up a newsfeed to transfer both USENET and private newsgroup articles.

newsreader

A client application that runs the NNTP protocol and is used to select, view, create, sort, and print USENET and private newsgroup articles.

NNTP (Network News Transfer Protocol)

Protocol that supports reading newsgroups, posting new articles, and transferring articles between news servers. When enabled on a Domino server, allows NNTP clients to access newsgroups on the server and allows the Domino server to exchange news with other NNTP servers.

No Access

An access level where users have no access to a database; they cannot even add the database icon to their workspaces.

note

A note is a simple data structure that stores database design elements (forms, views, and so on), user-created data (documents), and administrative information, such as the database access control list.

note header

A note header is a structure that contains, among other things, the note's originator ID (OID), which includes the note's universal ID (UNID); the note ID; the note's parent note, if one exists; the number of items in the note; and the list of the note's item descriptors.

note ID

A 4-byte value that is assigned to a note when the note is first created. Note IDs are stored in the record relocation vector table, which maps a note's note ID to the position with the database file. A note ID is unique within a database but not across replicas of the database, meaning that the same note in two replicas can have different note IDs, even though the replicas have identical UNIDs.

Notes application

A Notes application is the design of a Notes database. A complex Notes application may consist of several individual database designs that work together to perform a specific task. A typical Notes application consists of a set of design elements that specify, among other things, the type of documents in the database, the way that documents can be indexed and viewed, and the application's logic, which is written in the Notes Formula Language, LotusScript, Java, or JavaScript.

Notes client

Client software that allows you to access Notes databases on a Domino server, send mail, browse the Web.

Notes database

A Notes database is a single file that physically contains both a set of documents and a copy of the application design elements that control the creation and modification of those documents. A database can be shared, local, or remote.

Notes domain

A Notes domain is a network of client and server computers whose users, servers, connections, and access control information is described in a single database called the Domino Directory.

Notes mail database

A Notes database in which you send and receive mail. Your mail database is stored on your home server.

NOS (Notes Object Services)

The Notes Object Services are a set of portable C/C++ functions that create and access information in databases and files, compile and interpret formulas and scripts, and interface to operating systems in a consistent, portable way.

Notes program

A Notes program is written in C or C++, compiled into machine code, and then distributed as an executable (EXE) file. Examples of Notes programs include the Notes Client, the Domino Designer, the Domino Administrator, the Domino Server program, and Domino server tasks.

Notes program component

A Notes program component is written in C or C++, compiled into machine code, and then distributed as a dynamic link library (DLL) file. Program components contain reusable code and/or resources -- for example, text strings -- that can be used by one or more running programs. An example of a Notes program component is the Notes Object Services (NOS).

NRPC (Notes remote procedure call)

This is the architectural layer of Notes used for all Notes-to-Notes communication. You can set up either the HTTP or the SOCKS proxy to work with RPC.

Notes/FX

Notes/FX (Field Exchange) is a technology that lets desktop applications and Notes share data fields.

NOTES.INI

A settings file that includes installation choices, server console commands, and setup selections.

NotesNIC

The administrator of the NET domain, a way to communicate with other Notes organizations on the Internet.

NSF

The file extension for a Notes database file. A database file contains the data for an application. Its structure is composed of forms, fields, folders, views, and other presentation features, such as a navigator and a database icon.

Notes Storage Facility

Part of the Notes Object Services, the Notes Storage Facility is a library of C functions that implement the most basic database-creation and database-management operations.

NTF

The file extension for a Notes template file. A template file contains the structure for the database -- that is, forms, folders, and views -- but does not contain documents. Domino Designer comes with a collection of templates that you can use to create system and application databases.

O

ODBC (Open Database Connectivity)

A standard developed by Microsoft for accessing external data. ODBC has four components: the ODBC-enabled application, the ODBC Driver Manager, ODBC drivers, and data sources. Lotus Notes is an ODBC-enabled application.

ODS (on-disk structure)

The common, portable format used to store information in a Notes database. In Domino Release 5, the ODS version of a database is listed on the Info tab of the Database Properties box.

OID (originator ID)

A 28-byte identifier that contains a note's unique universal ID (UNID), which is essential for replication. The OID contains a UNID, which uniquely identifies the note and all replicas of the note. The OID also contains a sequence number and a time stamp that together indicate how often the note has been modified and when it was last modified. Replication uses all three OID values to synchronize changes between replicas of a note.

outgoing mail database

A file (MAIL.BOX) that temporarily stores outgoing mail that users create when not connected to a mail server.

P

pane

An area of a workspace that shows a specific part of an opened database; for example, available folders and views, the current view, or the contents of the highlighted document.

parent document

A document whose values are inherited by another document (the child document).

partitioned server

A feature that lets you run a maximum of six Domino servers on a single computer. With partitioned servers, you can increase the number of servers in your organization without additional investment in hardware.

passthru server

An intermediary server that lets a client access a target server to which the client is not connected. A mobile user can access multiple servers through a single phone connection; a LAN client can connect to servers running network protocols different from its own.

peer-peer server topology

Connects every server in your organization to every other server. For organizations with only a few servers, this allows for rapid updates.

permanent pen

An editing feature that allows users to edit documents in a second font.

Personal Address Book

A database that contains the names and addresses of users and user groups that you enter yourself.

Personal Web Navigator

A feature that retrieves, displays, searches for, and stores Web pages in a local Personal Web Navigator database. Because this database is stored locally, you are the only person who can access the Web pages stored in it.

PKCS (Public Key Cryptography Standards)

Industry-standard format for certificate requests. You see this acronym in both the Domino Certificate Authority and Server Certificate Administration applications. It means that if the CA server understands how to read PKCS format, it will understand your certificate request. This is important when you submit server certificate requests to an external CA, as the external CA must understand PKCS format.

platform

A platform is a specific operating system running on a specific computer.

POP3 (Post Office Protocol Version 3)

A mail protocol that allows clients running it to retrieve mail from a host mail server also running the protocol. You can enable POP3 on a Domino server.

preview pane

The preview pane lets you read the content of the document that is selected in the view pane. If Notes is set to preview document links, you can also view documents linked to the selected document.

primary replica

The replica designated to be the only recipient of updates by the Administration Process. By updating a primary replica and then replicating that database to other replicas on other servers, you avoid creating replication conflicts.

private folder

A folder that users design and save for their own use with a database.

private key

A secret encryption key that is stored in a Notes ID file and that is used to sign and decrypt messages and to authenticate as the owner of the key.

For SSL-encrypted transactions, public and private keys are a unique pair of mathematically related keys used to initiate the transaction that are stored in the Notes ID file, Internet client hard disk drive, or server key ring file.

private view

A view that users design and save for their own use with a database.

proxy server

A server that intercepts all requests made to another server and determines if it can fulfill the requests itself. If not, the proxy server forwards the request to the other server.

public key

An encryption key associated with a Notes ID that is used to verify an electronic signature, encrypt a message, or identify an authenticating user. A public key is part of each user ID and a copy of the key is stored in the Domino Directory. Certificates on IDs ensure that public keys are valid.

For SSL-encrypted transactions, public and private keys are a unique pair of mathematically related keys used to initiate the transaction that are stored in the Domino Directory.

public key certificate

A unique electronic stamp stored in a Notes or Domino ID file that associates a name with a public key. Certificates permit users and servers to access specific Domino servers. An ID may have many certificates.

public key encryption

Public key encryption provides a user with a key pair -- private and public. The public key is distributed to everyone with whom the user wants to communicate. In Domino, the public key is published in the Domino Directory. Public/private key encryption is used for two purposes: to communicate securely and to generate electronic signatures.

R

read access list

A list that restricts a form so that only specified users can read documents created from the form. Use the Readers field to control access on a document-by-document basis.

read-only mode

A document state that allows a user to read but not modify a document. To modify a document, a user must have Editor access (or higher) to the database or be the document’s author.

Reader access

An access level where users can only read documents.

Readers field

A list of names (user names, group names, and access roles) that indicates who can read a given document. This field does not override the access control list.

referral

An LDAP directory URL returned to an LDAP client. The Domino LDAP server can return a referral if an LDAP client query is not successful in a Domino Directory and an entry in the Master Address Book suggests that the query may be successful in another LDAP directory.

remote database

When a program is running on one computer accesses a shared database on another computer, the shared database is considered to be a remote database, with respect to the program accessing it.

replica

A special copy of a database that, because it shares a replica ID with the original database, can exchange information with it through replication.

replica ID

The replica ID, which is stored in the database header, is a unique number that is generated when you first create a database. The replica ID never changes. When you make a replica of the database, the replica inherits the replica ID. For two databases to replicate, they must share the same replica ID.

replicate

Update database replicas that are on different servers or on your workstation and a server. You can replicate the entire database so that over time all database replicas are essentially identical, or select specific items or areas to replicate.

replication

The process of exchanging modifications between replicas. Through replication, Notes makes all of the replicas essentially identical over time.

replication conflict

A condition that occurs when two or more users edit the same document in different replicas of a database between replications.

Replication Monitor

A document created in the Statistics & Events database that causes the Event task on a server to monitor a specific database to make sure it is replicating.

Replicator

The part of the workspace where Notes displays all replica databases and lets you manage the replication process. Also the name of the server task that replicates databases between servers.

response document

A document created using a Response form, a typical component of a discussion database. In a view, response documents are usually indented underneath the document to which they respond.

rich-text field

A rich-text field can contain text, objects, file attachments, and pictures. You can tell you are in a rich-text field if the status bar at the bottom of your screen tells you what font size and font name you are using.

ring server topology

Connects servers one-to-one in a circle with the ends connected. It is similar to chain server topology, which connects servers one-to-one but with the ends unconnected.

role

Database-specific groups created to simplify the maintenance of restricted fields, forms, and views. You can apply a role to Authors fields, Readers fields, and read and create access lists in forms and views.

RRV (record relocation vector) table

Each database contains an RRV table that maps a note's note ID to the position of the note in the database.

S

SASL (Simple Authentication and Security Layer)

Internet protocol that allows LDAP clients to authenticate with an LDAP server and provides security for the data transmitted with this protocol.

save conflict

A save conflict occurs when two or more users edit the same document in a database on a server at the same time. The document saved first becomes the main document; subsequent users are prompted to save their changes as responses titled "[Replication or Save Conflict]."

screen reader

A device that reads what is displayed on the computer screen. See digital speech synthesizer.

secondary Domino Directory

A secondary Domino Directory inherits its design from PUBNAMES.NTF and contains directory information about an additional Notes domain.

section

A defined area on a form that can include fields, objects, layout regions, and text. You can set section properties to expand automatically at certain points.

server certificate

An electronic stamp stored in the server's key ring file that contains a public key, a name, an expiration date, and a digital signature. The server certificate uniquely identifies the server.

server command

Command that lets you perform a task, such as shutting down or restarting a server. You can enter commands manually at the console or remote console or use a Program document in the Domino Directory to run commands automatically.

server connection

A document in the Domino Directory or your Personal Address Book that defines a connection to a server. There are four types of server connection documents: dialup, network, passthru, and remote LAN.

server program

A program that automates an administration task, such as compacting all databases on a server. You can schedule server programs to run at a particular time, or you can run them as the need arises.

server task

A program provided with the Domino server that runs only when specifically loaded. Server tasks serve various purposes; the Administration Process, HHTP Server, and Reporter are just a few examples of server tasks.

shared field

A field that is used in more than one form. For example, many forms have a creation date field, so you can define the field once and reuse it.

shared mail

A feature that stores messages addressed to more than one user on a mail server in a central database, called the shared mail database. Message headers are stored in user mail files. When users double-click the headers, links to the corresponding content in the shared mail database are activated. This is a space-saving feature. The shared mail database is also known as the Single Copy Object Store (SCOS).

shared view

A view that is public to more than one user.

sibling document

In a view or folder, a document at the same level as another document.

sign

To attach a unique electronic signature, derived from the sender's user ID, to a document or field when a document is mailed. Signing mail ensures that if an unauthorized user creates a new copy of a user's ID, the unauthorized user cannot forge signatures with it. In addition, the signature verifies that no one has tampered with the data while the message was in transit.

single copy object store (SCOS)

The feature that allows mail addressed to multiple users to be stored in a central database, called the shared mail database.

site certificate

A certificate obtained for an individual site. A site certificate is different from a trusted root certificate in that a site certificate lets you access only a specific site. A trusted root certificate lets you access any servers with certificates issued from that trusted root Certificate Authority.

SLIP/PPP

A dialup version of TCP/IP.

S/MIME (Secure/MIME)

A secure version of the MIME protocol that allows users to send encrypted and electronically signed mail messages, even if users have different mail programs.

SMTP (Simple Mail Transfer Protocol)

The Internet's standard host-to-host mail transport protocol. It traditionally operates over TCP, using port 25. SMTP does not provide any mailbox facility, nor any special features beyond basic mail transport.

SOCKS (SOCK-et-S)

A mechanism by which a secure proxy data channel can be established between two computers. It is generally used as a firewall.

SSL (Secure Sockets Layer)

A security protocol for the Internet and intranets that provides communications privacy and authentication for Domino server tasks that operate over TCP.

stacked icon

A Notes database icon that represents a database and all of its associated replicas that are currently added to the workspace.

static text

Text that remains constant on every document created with a particular form, as opposed to fields in which you type or in which Notes calculates information.

stub

A replica or database copy that has not yet been filled with documents. The database is no longer a stub after the first replication takes place.

subform

A form-building shortcut that lets you store regularly used fields, sections, actions, and other form elements together. You can place subforms on a form either permanently, or as computed subforms that display on documents as dictated by a formula.

symmetric encryption

Often referred to as secret key encryption, symmetric encryption uses a common key and the same mathematical algorithm to encrypt and decrypt a message. For two people to communicate securely with each other, both need to agree on the same mathematical algorithm to use for encrypting and decrypting data. They also need to have a common key: the secret key.

T

TCP/IP (Transmission Control Protocol/Internet Protocol)

Network protocols that define the Internet. Originally designed for UNIX, TCP/IP software is now available for every major computer operating system.

template

A design that you can use as a starting point for a new database. If it is a design template, it will update database design elements created from the template.

temporary field

A field used during calculations. It is not stored.

trusted root

A Certificate Authority's certificate merged into the Domino Directory, client's browser, or the server's key ring file, which allows clients and servers to communicate with any client or server that has that Certificate Authority's certificate marked as trusted.

U

UBM (Unified Buffer Manager)

The component of the Notes Storage Facility that caches information about open databases.

UNID (universal ID)

The UNID is a 16-byte value that is assigned to a note when the note is first created. The UNID uniquely identifies a note. UNIDs are used when replicating database notes and when replacing or refreshing database design notes.

UNID table

The UNID table maps a note's UNID to its note ID, which, in turn, can be mapped through the database's RRV table to the note's position within the database file.

Unread Journal log

This log keeps unread lists synchronized between various replicas of a database and records when a document's status changes from read to unread and vice versa.

URL (uniform resource locator)

The Internet address for a document, file, or other resource. It describes the protocol required to access the resource, the host where it can be found, and a path to the resource on that host.

user ID

A file assigned to every user and server that uniquely identifies them to Lotus Notes and Domino.

Using This Database document

A document that explains how the database works. Specifically, it provides users with instructions on using various forms, views, and navigators in the database.

W

Welcome page

The customizable default opening screen in the Notes client that includes major tasks such as sending mail, creating appointments, and making a to do list. The page also contains a search bar, information on what's new in Notes, and a tour of Notes.

window tab

A button that represents an open window in Notes. Window tabs are convenient for switching back and forth between windows.

Index