Performance Considerations for Domino Applications

Søren Peter Nielsen, Alessandro Toscano Mariscal, Umasuthan Ramakrishnan, Walt Simons, Chris Thornton

International Technical Support Organization
www.redbooks.ibm.com
Performance Considerations for Domino Applications

March 2000
Contents

Preface ................................................... ix
The Team That Wrote This Redbook .......... x
Comments Welcome .............................. xii

1 Introduction to Performance in Domino Applications .................. 1
What is Performance? ............................... 2
  What Determines Performance in Your Domino Application? ....... 2
What Do You Measure? .............................. 3
Which Parts of Your Application Affect Performance Most? ............ 4
Where to Look for Information .................. 5
Summary ................................................ 6

2 Design Considerations ......................... 7
The Best Solution is the Simplest ................ 8
When Does it Make Sense to Split Up the Application in Several Databases? .... 8
  Supporting More Users and Getting Better Response Time .......... 9
Extending Capacity and Overcoming Limits .......................... 9
Sharing of Data and/or Functionality Between Several Applications .... 10
Access Control Requirements .................... 10
Support of Mobile Users ............................ 10
How to Split Up Your Application in Several Databases ............. 11
  Load Distribution on Several Servers Through Clustering ......... 11
  Load Distribution Through Replication .................. 12
  Keeping Database Size Down Through Archiving .................. 13
Load Distribution by Splitting the Application into Several Databases .... 13
Keeping Database Size Down by Using Databases with Subsets of Data .... 15
Using Special Databases and a Subset of the Application for Mobile Users .... 16
How Application Deployment on Several Servers and Several Locations Can Affect Your Design .................. 17
Think about Caching ................................ 31
User Perception of Performance ................... 32
Give the Users Feedback ............................ 33
Use Deferred Processing to Let the Users Continue with Other Tasks .... 33
Techniques for Multi-database Applications .......................... 35
  Using Keys to Relate Data in Different Databases .................. 35
  Look up Data Dynamically or Use Redundant Data ................ 37
  Switching Seamlessly Between Databases .................. 38
  Signaling That Data or Design Needs to be Updated ............... 39
A Bit of Methodology Discussion ................ 39
Using Scenarios to Design Your Application .................. 40
Software Performance Engineering .................. 44
Validating Your Design Through Proof-of-Concept Prototypes ........ 45
  Similar Environment ................................ 46
  Similar Data ...................................... 46
3 Programming Considerations

Choosing a Language ........................................ 51
The Lotus Formula Language ............................. 52
LotusScript .................................................. 52
LotusScript or Formulas: Which is Faster? ........... 53
Java ......................................................... 53
JavaScript ................................................... 54
Use the Language You Are Most Familiar With ... 54
And the Winner is... ....................................... 54

Language Performance Aspects ......................... 55
@Formulas .................................................. 55
LotusScript .................................................. 60
Java ......................................................... 68
JavaScript ................................................... 69

The Domino Object Model ................................. 69
Maximize Resource Usage ................................ 70
Getting the Most from Your Domino Objects ...... 73

Code Reuse .................................................. 79
What Can Be Reused? ...................................... 80
Script Libraries ............................................. 81
Agents ....................................................... 82
Best of Both Worlds ....................................... 82
%INCLUDE Files .......................................... 83
External Programs ......................................... 84
Dynamic Script Library Loading ....................... 84
Sorting ....................................................... 84
Asynchronous Processing ................................. 85
Using Asynchronous Processing ..................... 85

Summary ...................................................... 49

4 Performance Aspects of Domino Design Elements

The Basics ................................................... 87

Designing for Maximum Performance Using Forms
Using Graphical Objects ................................ 88
Using Subforms ............................................ 89
Using Tables ............................................... 91
Using Sections ............................................. 92
Using Layout Regions .................................... 92
Using Progressive Disclosure ......................... 93
Caching Field and Hotspot Formulas ................. 93
Using JavaScript ........................................... 94
Time-Date and Numerical Calculations ............... 94
Keep Hide When Conditions Simple .................. 94
Fine Tune Your Lookups .................................. 95
Using Fields Wisely ....................................... 98
Automatic Field Refresh ................................ 104
Combine Field Formulas into Events ................. 104
Working With Pop-ups ................................... 105
Allow Domino to Automatically Redirect URLs ... 105
Defragmenting Complex Forms ......................... 106

Profile Documents ......................................... 106
Usefulness of Profile Documents ..................... 106
Profile Information for Web Browsers ............... 107
Linking an Identity to a Web Browser ............... 108
Browser Caching .......................................... 108

Views ......................................................... 109
Some Internal Information About Views .......... 109

View Design Considerations ......................... 110
Type of View .............................................. 110
Selection Formula ......................................... 120
Three-Tiered Distributed Data Architecture — MQ on Server  ...  207
Three-Tiered Distributed Data Architecture — MQ on client  ...  208
Four-Tiered Distributed Data Architecture — MQ on Staging Server  209
The Tiered, Distributed Data Architecture  ...  211
What is the Tiered Architecture?  ...  212
Data Distribution and Scalability  ...  215
Data Flow Across the Tiers  ...  217
Minimizing Application-Induced Replication Conflicts  ...  221
Performance Characteristics  ...  222
Overall Performance Test — Part 1  ...  222
Overall Performance Test — Part 2  ...  224
Future Applicability  ...  225
Appendix A-2 High Performance Reference Databases  ...  227
Appendix A-3 Third-Party Tools for Performance Measurements  ...  231
Monitoring Tools  ...  231
  Candle Pinnacle Performance Manager  ...  231
  Candle ETEWatch  ...  232
  Tivoli Manager for Domino  ...  232
  Coast WebMaster  ...  232
Stress Testing and Load Tools  ...  233
  Websiszr  ...  233
  Groupsiszr  ...  233
  Proactive Assistant  ...  233
  Proactive Load  ...  234
Appendix B-1 Time/Date Views  ...  235
Background  ...  235
Creating Time/Date Views Using Selection Formulas  ...  236
Use @Today in the Selection Formula  ...  236
Use @Environment in the Selection Formula  ...  236
Use @Text in the Selection Formula  ...  237
Creating Time/Date Views Using Agents  ...  239
Save a DisplayInView Indicator in the Document  ...  239
Put Documents into a Folder  ...  240
Appendix B-2 Dynamic Script Library Loading  ...  243
Background  ...  243
Problem  ...  243
Solution  ...  245
Usage Example  ...  248
Evaluation  ...  249
Conclusion  ...  250
Appendix B-3 Displaying Processing Status  ...  251
Status Line Messages  ...  251
Status Line Progress Bar  ...  251
How Many Documents Are in That View?  ...  253
Progress Bar Using Undocumented API Calls  ...  256
Appendix B-4 Sorting  ...  257
Sorting Scope and Limitations  ...  257
Sorting Algorithms  ...  257
Where Can You Find Sorting Routines?  ...  258
Bubble Sort  ...  258
Selection Sort  ...  260
Insertion Sort  ...  261
Quick Sort  ...  263
Heap Sort  ...  264
Shell Sort  ...  267
Pros and Cons  ...  268
Bubble Sort  ...  268

vi Performance Considerations for Domino Applications
Moving the Cursor from One Editable Field to Another
...
Toggling from Edit Mode to Read Mode with Document Open (and with No Changes)
...
Toggling from Edit Mode to Read Mode with Document Open (with Changes Made and Saving)
...
Toggling from Edit Mode to Read Mode with Document Open (with Changes Made but without Save)
...
Moving the Cursor from One Editable Field to Another

Appendix C-4 LotusScript versus Formulas

Appendix C-5 Execution Order of Notes Form Events and Formulas

Appendix C-6 How Domino Handles Document URL Requests

Appendix C-7 Ways to Control Caching of Pages Served by Domino

Pressing F9 to Refresh Fields While in Edit Mode

Special Notices

Additional Web Material

Related Publications

Index
This document describes performance considerations when developing applications for Domino R5.0 or R4.6.

We discuss the following issues in detail:

• How to consider performance and capacity when designing and developing a Domino application

• How to analyze a Domino application that does not meet its performance and capacity goals

While both platform setup (Domino server and operating system) and application architecture need to be considered when looking at performance and capacity, this book focuses on the application perspective.

This redbook was written for Domino designers and programmers, customers, IBM Business Partners, and members of the IBM and Lotus community who need a good technical understanding of how to develop applications that scale and perform using Lotus Domino R5.0 and R4.6.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Center at Lotus in Cambridge, Massachusetts, USA.

Søren Peter Nielsen works for the International Technical Support Organization at Lotus Development, Cambridge, Massachusetts. He manages projects to produce redbooks on all areas of Lotus products. Before joining the ITSO in 1998, Søren worked as an IT Architect for IBM Global Services in Denmark on solutions for a wide range of industries. Søren is a Certified Lotus Professional at the Principal level in Application Development and System Administration.

Alessandro Toscano Mariscal has worked with Lotus Notes for five years, specializing in application development. In Brazil he received a degree in Electrical Engineering. Alessandro works as a consultant at a Lotus premium business partner, Notion Informatica in Rio de Janeiro, where he is involved in developing several Notes applications. He is also a Principal Certified Lotus Professional in application development for R4.x and a Certified Lotus Professional for R5.
Umasuthan Ramakrishnan is a software engineer for IBM Global Services in India. He has two years of experience with Lotus Notes and Domino.

Walt Simons is a Senior Architect with IBM Global Services in Sydney, Australia. He is part of the IBM Notes/Web Application Center of Competence, and also works with various development teams in Australia producing global enterprise applications. He specializes in LotusScript, the C API and the ODS and has developed productivity tools for the development community. Walt has over 24 years of experience in a variety of areas in the IT industry.

Chris Thornton is a Field Support Engineer for Lotus Professional Services in England. He has worked for Lotus for two years and has six years of experience with Lotus Notes and Domino. Chris has particular expertise in infrastructure planning, server administration, application design and troubleshooting.

A number of people have provided support and guidance and we have also harvested information from a number of resources. Refer to the related publications appendix for these resources. We would like to thank the following people:

- Jeff Blaney — Lotus
- Chris Cole — Lotus
- Michael Confoy — Lotus
- Teresa Deane — Iris
- Sunsan Florio — formerly Iris
- Dean Garyet — IBM Netherlands
- Garry Greenwood — G2 Associates, Inc.
- James Grigsby — formerly Iris
- Brendan Hoar — Price Waterhouse Coopers
- Ralph Hueske — IBM USA
- Ryan Jansen — Iris
- Dave Karasic — Lotus
- Brian Kiser — Lotus
- William A. Littlewood — Lotus
- Steve Krantz — IBM USA
- Bill Masek — Lotus

x Performance Considerations for Domino Applications
• Salvatore Mazzotta — Lotus
• Boris Minnaert — IBM Netherlands
• Steve Mullen — Lotus
• Collin Murray — Lotus
• Michele Pennell — Lotus
• Juliet Porter — Lotus
• Raphael Savir — Lotus
• Jasper Schroder — IBM Netherlands
• Richard Schwartz — RHS Consulting Incorporated
• Pramod Singh — IBM USA
• Jane Stanhope — Lotus
• William H. Tetzlaff — IBM Watson lab
• Scott Vrusho — Lotus
• Kevin Waterman — Lotus
• Andrew Wharton — Iris
• James Willard — IBM USA
• Robert Yates — Lotus
• Wai Yip — Iris
• Carol Zimmet — Iris
• The team making Lessons Learned available at the intranet site for IBM Lotus Notes/Web Applications Center Of Competence.
• The Performance Team at Iris
• Lotus Enterprise Center Of Competence
• Alison Chandler, ITSO Poughkeepsie
• Graphic Services, Lotus North Reading
Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found at the back of this book to the fax number shown on the form.
- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an Internet note to redbook@us.ibm.com
Chapter 1
Introduction to Performance in Domino Applications

With the adoption of Domino worldwide in every industry and for all kinds of solutions, there is a need among solution architects and developers to better understand the performance and capacity aspects of Domino applications.

There are many examples of Domino being used for mission-critical applications that require performance and scalability. With Domino R5.0, the potential for superb performance and capacity in Domino applications is even better than was the case with previous releases.

However, as with any other powerful tool, problems can arise if you don’t know how to use Domino properly for large applications. That is why we wrote this book for you as a designer and developer.

If you are an administrator you may also find some helpful information here, but we focus only on the application aspects of performance in this book. While we may touch on some general server tuning topics, we will not cover how to configure your Domino server or the underlying platform operating system for optimum performance. Check the Related Publications section for related publications that contain information about performance in the Domino server.

If you are interested in development methodology, this may not be the place to go either. We do talk a bit about methodology, but the content in this book is based on experience and best practices we have gathered from a number of sources and it can be used with any development method.

Finally, if we haven’t scared you off yet — this is not a book that will teach you the basics of how to program Domino. We assume that you have experience with Domino development. If you are new to Domino, you may want to read one of these redbooks first:

- *Lotus Domino R5.0: A Developer’s Handbook*, IBM form number SG24-5331, Lotus part number CC7EDNA
- *Developing Web Applications Using Lotus Notes Designer for Domino 4.6*, IBM form number SG24-2183, Lotus part number 12974
In this chapter we will discuss performance; what it is, how you measure it, and which parts of Domino applications affect performance. We will then describe what information you can find in the rest of the book.

**Note** Designing Domino applications for performance is not an exact science. Given the broad diversity of business solutions and application types, there are few rules that are valid in all circumstances. We will discuss experiences with specific applications. The guidelines we derive from these experiences may not apply in all cases, but should be of help to you if you are doing something similar. To find out how a specific application you are designing or developing will behave, you must create a proof-of-concept application and test out its performance aspects.

---

**What is Performance?**

When talking about performance, people often have different perceptions of what it is. Here are two ways to look at performance:

- **Time aspect — Responsiveness**
  Performance is a measure of time spent on an individual operation or job.

- **Capacity aspect — Throughput**
  Capacity or throughput can be another measure of performance.

  Performance can be measured in terms of data transfer rates, generally expressed as transactions per hour. It can also be measured in terms of resource utilizations such as CPU utilization or disk storage utilization.

It does not make sense to talk about performance without mentioning throughput capacity or scalability. An application may perform very well during a pilot deployment, where only a subset of the targeted user population works with it — and then fail badly when rolled out to all users.

---

**What Determines Performance in Your Domino Application?**

The responsiveness and throughput you can reach in your Domino application is a product of the following:

- Application design and implementation
- Potential interaction with external systems
- Domino Server configuration
- Network configuration
• Server and client operating system configurations
• Server and client hardware configurations

Often the main factor in a long response time or missing throughput is found in some of the layers below the application. However, to the user it is the application that doesn’t perform as expected, and the problem is the application owner’s until proven otherwise. We have included some tools and techniques in Chapter 5, “Troubleshooting an Existing Application” to help you determine whether performance problems may be attributed to some of the layers below your application.

What Do You Measure?

When measuring performance we most often look at the following:

• Responsiveness of the server as load increases
  Load can be the number of documents in a database, the number of users using the application, the number of user operations per time unit, and so on.

• Response time for a user performing certain operations
  In most cases the response time perceived by the user is the most important factor.

The requirements for an application may not always include tangible performance requirements beyond statements such as ‘the application must be as fast as the application it replaces’ or ‘the application must be able to support 500 users on one server.’

An application that replaces another seldom has the same user interface or functionality. The users of the new application will likely perform their work in another way than in the old application. This makes it difficult to compare response time between the two applications. Instead, the requirement should state within which time zone certain user operations such as ‘Create new order’ must be completed.

It does not make sense to establish a goal for the number of users your application must support without stating what the users will be doing. Your application may support several thousand users on one server if all they do is read one document and create one document per day. To have the number of users as a useful requirement, you also need to know what the users will be doing and when they will do it during the day.
You may also set response time goals for when the application responds to a single request. From the user’s point of view, the following rules of thumb apply:

- A user is more productive when the time between recessing an action and getting a reply is less than one second.
- 10 seconds is the limit for keeping the user’s attention at the interaction with the application. If it takes more time to fulfill a request the user will switch to other tasks.

There will be many cases where you will not be able to get sub-second response times, but you should not go over 10 seconds, as this will seriously hamper the user’s productivity.

In each of the above examples, the key to establishing and achieving measurable goals for performance in your application is to have a good description of what the users will be doing with your application and what their usage pattern is likely to be. We will discuss a method for creating such a description in Chapter 2 under “The Use Case Model — Using Scenarios in the Design of Your Application.”

**Which Parts of Your Application Affect Performance Most?**

The number of concurrent users in your application and what they are doing will affect performance. The parts of an application where you need to pay the most attention to performance are as follows:

- Application factors affecting server load
  - Indexing of views
  - Running of agents
- Application factors affecting user response time
  - Open database
  - Open View
  - Open document, recalculate document
  - Perform lookup against another database
  - Controlling what is being cached on the server and the client

These are some of the areas to focus on in your design and development. There are other factors as well, and we discuss many of them in this book.
Where to Look for Information

It is important that you address performance from day one of your work with the application. This book focuses on what you as a designer or developer can do to design and implement your application for optimal performance. This section describes the topics covered and how the book is organized.

Design
Chapter 2 discusses the overall performance and capacity considerations you should go through when designing your application. We discuss when it makes sense to use several databases for your application and how to do so. For a distributed application you also need to take performance and capacity restrictions in the infrastructure into account, and we will discuss different infrastructure scenarios. We will also discuss techniques for implementing multi-database applications such as seamless switching between databases, cascaded deletes and so on. Knowing the way that users work with your application is very important to predict peaks of load and to identify the most important areas on which to focus. A method to get this information (a Use Case Model) will be briefly described before we discuss how you test that your design is valid with regard to performance and capacity.

Languages and the Domino Object Model
Chapter 3 is about some of the programming languages that you can use to develop Domino applications: formula language, LotusScript, Java and JavaScript. You can also write API programs using C and C++, but we will not cover API programming specifically in this book. We discuss considerations for choosing a language and performance aspects of the different languages, especially the formula language and LotusScript. We then discuss the Domino Object Model and performance issues when you access the Domino objects. This information is relevant for LotusScript, Java and COM-programmers. Finally, we describe different techniques for code reuse and discuss their pros and cons relative to performance.

Domino Design Elements
In Chapter 4, we examine performance aspects of different Domino Design elements. We talk about the use of graphics, subforms, tables, sections, hide-when formulas, different field types and more. We discuss Views in great detail because they are so critical to application performance, but we also cover many other design elements.
Troubleshooting
Often you do not have the luxury of being allowed to develop a brand new application from scratch. Instead, you need to improve performance in an existing application, and in Chapter 5 we discuss how to troubleshoot an existing application that does not perform adequately. First we give you some tools and techniques for making sure that the performance issues are not somewhere in the underlying server, network or client configuration, and then we discuss how to find performance bottlenecks in Domino applications.

Appendices
We have quite a few appendices in this book with specific examples, tools and techniques, and reference information. The appendices have been split into three groups:

- Appendices Group A: Application examples and tool descriptions.
- Appendices Group B: Programming Techniques (time views, seamless switching between databases, cascaded deletes and so on)
- Appendices Group C: How some of the Domino stuff works and some limits to be aware of.

Note As with many other things in life, the devil is in the details when looking at performance and capacity in Domino applications. We have tried to structure the book so you can locate your areas of interest, but often there may be information in several places relating to a particular issue you are pondering. We encourage you to read the entire book the first time in order to get the full benefit.

Summary
In this chapter, we have defined what we mean by performance — both the responsiveness and capacity of an application. We have discussed ways to measure performance and identified which parts of a Domino application affect performance the most. Finally, we have given an overview of what is covered in the rest of this book.
Chapter 2
Design Considerations

In this chapter, we discuss aspects of designing for performance and capacity. The focus is not as much on how do you optimize certain functions, but more about how you structure your data and application functionality to avoid potential bottlenecks in capacity and response time. You have to think about where to put the data, how to structure the data for access and how to distribute the load. This includes thinking about how to avoid costly opening of databases, lookups, heavy switching between views, and so on, when the user uses your application for their normal daily work.

It is important that you make your design scalable. If you design a good application it may well grow beyond its original scope. We discuss when you should consider splitting your application into several Domino databases as well as how to do it.

You also have to consider the infrastructure your application will be deployed in. We discuss the impact of having an application distributed over several servers, several locations, supporting mobile users, and connecting to back-end systems.

We hope the discussion in this chapter will help stimulate your thinking in this area and give you ideas for solutions. If you are looking for specific details regarding how to optimize certain language constructs or the performance impact of certain design elements, this is not the place where you will find it. Instead you should go right to the following chapters.

In general, we do not distinguish between applications accessed by Web browsers or by Notes clients except where noted. Most of the considerations are valid whether you use one or the other.

**Note**  With Lotus’ introduction of Domino OffLine Services (DOLS), a Web browser can now support more “Notes-like” functionality. For example, DOLS gives a Web client the ability to replicate data to a local store and work with them off-line without compromising Domino security.
The Best Solution is the Simplest

In general, whenever you add complexity to a solution you also add to the cost of the solution, so our ideal Domino application is one that is contained in one Domino database and resides on one server only. Centralization reduces cost and improves reliability. Fewer servers mean less complex server topologies and fewer server-to-server activities, less redundant data, and less replication.

From a programming standpoint, you don’t have to concern yourself with a lot of setup information about names of other databases and views if all the data you are interested in resides in the same database.

With the performance and capacity improvements in the Domino R5.0 server and database structure, you can get quite a lot further along with a single database application.

However, there will still be situations where the limits are reached and there may also be other reasons for using several databases for an application. We discuss these circumstances in the following section.

When Does it Make Sense to Split Up the Application in Several Databases?

How far can we go with the simplest solution? Besides performance and capacity considerations, some of the business requirements may demand that the application spans several databases. Here we discuss the factors that can prompt you to split up your application into several databases. Before we discuss different ways to do the actual splitting up of the application we will discuss how you have to take the infrastructure and types of users into account before deciding how to split up the application.

You can split up an application in several databases by:

- Replicating the application database to other servers
- Creating several databases to hold the application
- A combination of these two methods

Reasons for splitting up the application can include:

- Supporting more users and getting better response time
- Extending storage capacity and overcoming limits
- Sharing data or functionality between several applications
- Access control requirements
- Support of mobile users
Supporting More Users and Getting Better Response Time

The number of concurrent users of the database may grow to a point where the database response time starts degrading because of the user load. The size and complexity of the views in the database, or the sheer database size, can also degrade response time.

One way to alleviate this problem is to distribute data and functionality in the application over several databases on the same server. You may argue that this does not seem to buy much. However, as long as the server performance isn’t a bottleneck this will work, because with several databases in the application the load by concurrent users on one out of several databases will be less than with a one-database solution. Also, the individual databases in the multi-database solution will be smaller than the sole database in the one-database solution, and smaller databases are faster than larger databases.

Extending Capacity and Overcoming Limits

You need several databases to store all your data if the amount of material in the database reaches the physical limitation of the database size. You may also want to store the data in a back-end enterprise database and access the data in real-time from your Domino application.

If the network capacity is not sufficient to give remote users acceptable response time, it is necessary to replicate the application to a server close to the users.

Another time when you might want to do this is when the disk capacity on the Domino servers does not allow all application data to be stored on all servers where the application is deployed. In this case the application can be split into several databases, of which only some are replicated to all servers; or you can replicate only subsets of data in a single database using selective replication, or readers’ fields to control which part of the data goes to a specific server.

Even though all of your data will fit into one database, you may encounter other limits that require you to split it up into several databases. In Domino R4.x the size of a view cannot go beyond 512MB. You can, of course, create several ‘sub-views’ in the same database to remedy that, but sometimes it may make more sense to split the data behind the view into several databases. In Domino R5.0 the view can be as large as the database size limit that is defined when the database was created.

There is also a Unique Key (UNK) table limit if you use many fields in your database. The UNK table is an internal table that stores all of the unique fields in a Domino Database. The UNK table has a limit of 64KB in Domino R4.x. It is derived from a list of the unique field names in a database. This list
contains the field names, data types and other related information. Because the lengths of field names can vary, it is not possible to pin down an exact number of fields that a database can contain before maxing out the UNK limit. On average, most databases can store around 3,000 field names before hitting the limit. Using shorter field names is one way to get more fields into the database, but if your application requires lots of fields, this limit may also prompt you to split it up in several databases. In Domino R5.0 the UNK table limit has been raised to 65,000 entries, no matter their size, but Domino Designer still works with the 64KB limit. To get more information about specific limits in Domino, see Appendix C-1, “Limits for Application Development Features.”

Sharing of Data and/or Functionality Between Several Applications

If you use an overall architecture in the development of your application, you may have identified common sets of data and functionality that can be used by several applications. For example, you might have a Customer Service application (Helpdesk) and a Sales Force Automation application that share the same Domino database containing customer information. To share data and functionality between several applications in an extensible architecture, the shared data must reside in its own database.

Access Control Requirements

Some data and functionality in your application may be accessible to all users, while access to other parts of the data or functionality must be restricted. A simple example of this is a Web PR application where all users are allowed to see all released news and announcements, while only the authors and reviewers are allowed to see the documents that are being worked on. Another example is a customer relationship management application where all users are allowed to see all base data for a customer, but only the department that ‘owns’ the customers is allowed to see correspondence, current orders and so on. In both of these examples, the access restrictions each may be implemented with fields, but this adds complexity to the application. It is often more efficient to put the data all users are allowed to see in one database and the restricted data in another database, and then place controls at the database level on who can access what data.

Support of Mobile Users

Mobile users should ideally only have the data and functionality that they require while on the road.

Having all data in a large application reside locally extends the replication time, while a lot of data irrelevant to the user replicates. In addition to wasting time, this requires more disk space on the mobile user’s PC.
There is also the security aspect to consider. A mobile user’s PC is much more vulnerable to loss or theft than a PC located in a controlled company environment. Limiting the amount of data stored on the portable machine minimizes the exposure to your company and your customers in case of such occurrences.

Before starting development of your application, make sure you know what kind of users this application will have now and in the future, the security requirements, and how your application must be able to interact with other existing and future applications.

How to Split Up Your Application in Several Databases

You can split up your application to achieve better load distribution, to stay within the database capacity limits, or for other reasons like support of mobile users. We will discuss the following different goals you can achieve by splitting up your application in several databases:

- Load distribution on several servers through clustering
- Load distribution through replication
- Keeping database size down through archiving
- Load distribution by splitting the application into several databases
- Keeping database size down by having databases with subsets of data
- Using special databases and a subset of the application for mobile users

We applied the following rules of thumb to the design concepts discussed here:

- In most cases, the number of concurrent users will be an issue before the storage capacity of the database is an issue.
- Small databases are faster than large databases. Therefore, an application consisting of several smaller databases can support more users than a similar one that is based on a single, larger database.

Load Distribution on Several Servers Through Clustering

You can perform load balancing by deploying your application on several clustered servers:

- For Notes users, deploy your application on Domino clustered servers.
- For Web users, use the Internet Cluster Manager (ICM) for Domino R5.0 or similar technology, such as IP Sprayer for Domino 4.x.

A solution where servers are clustered should not affect your design considerations compared to having your application on a single server.
Load Distribution Through Replication

Even with sufficient server capacity, you may experience declining performance when many users access your application concurrently. In addition to clustering, you can achieve load balancing by moving part of your user population to a new server and then replicating the application between the servers. In some cases you can do this without any design implications. You must make sure that the requirements for up-to-date data are met. Count replication cycles and estimate how much data can be replicated during the assigned time slot, taking into consideration that other applications may also replicate data during that same time period.

You must also consider whether the potential for people updating the same data at the same time in replica databases on different servers is something you want to avoid. If your application doesn’t have any requirements that will hinder it, you should enable field-level replication on the forms for your documents. As long as the users on different servers update different fields in the same documents, these changes will be merged into one document during replication and no replication conflicts will occur. Further, the enabling of field-level replication may shorten the replication time because only the data in the changed fields is transferred. Without field-level replication enabled, the content of the full document has to be transferred during replication in case anything in the document is updated.

If your documents can be assigned ownership so that only users accessing one server are allowed to update documents, while everybody is allowed to read the documents, you can use Authors fields to implement this.

In case you really need to control document access through a check out/check in mechanism for documents in replicated databases, one thing you can consider is to put the different servers together in a cluster. This enables event-driven replication where as soon as a document is marked as checked out it is replicated to all other servers in the cluster. You can also go a step further and add a server task that communicates with server tasks on the other servers holding replica copies of your document to prevent a full check out until it has been verified that the document isn’t already checked out. Lotus’ document management system Domino.Doc uses a technique along those lines.

Another solution is to use your application in conjunction with Lotus Sametime. If your application is Sametime-enabled, then users on all servers can see if a user is working with a specific document. It will not lock other users out from modifying the document, but most users will not change a document if they are aware that another user is working on it.
You can also replicate your application or parts of it to your users’ local PCs and then have them replicate with the server every 10–15 minutes. This will utilize the power of the PC and take load off the server. One drawback is that unlike server-to-server replication, a client-server replication always has to be initiated by the client; the server cannot “push” any data to the client. If the user inadvertently turns off the client background replication, they can end up working on old data. You may consider implementing some data age check in your code that can prompt the user to replicate in case no updates have been received over a certain span of time.

**Keeping Database Size Down Through Archiving**

For databases with many views, moving seldom-used data to an archive database can significantly improve response time. Keeping database size down can also help mobile users who have the application on their PC by avoiding the need to specify selective replication formulas.

One challenge with archiving is knowing what you are “allowed” to archive. A simple way to archive is based on when a document was created or last modified. This will work well for a simple application such as a news feed, but in other cases it will not be sufficient. You may have some reference data (for example, basic customer data) that seldom changes, but that still needs to be in the application.

Another way to archive is based on status, where only documents with a “ready to archive” status actually are archived. This is good for case-handling types of applications, where you archive a case once it is closed, but it requires that the user or the application change the status on each document before it is archived.

When archiving you must also consider whether there may be doclinks in other documents pointing to the ones being archived, or other references such as a keyed lookup. In such cases you should leave a “stub” document in the application with a doclink or a reference to the archived documents in the archive database.

Domino R5.0 has some built-in functionality for archiving which we discuss, together a more advanced method, in “Database Archiving” in Chapter 4.

**Load Distribution by Splitting the Application into Several Databases**

We actually introduced this topic in the previous section about archiving, where we looked at the specific case of splitting an application into the main application database and an archiving database. There are many ways to split up the data in your application. The two major approaches are described in this section.
Hiding the Database Structure and Adding Databases as Required
The simplest example is a variant of archiving, where you create a new database and move your users to that database whenever a certain period of time, for example six months, has elapsed.

Another example is to add another database whenever the current database reaches a certain size, say 2GB. The Lotus document management solution Domino.Doc uses this concept where the database structure is hidden from the user.

Splitting up Data and Functionality into a Fixed Set of Databases
If you can keep the database sizes below the capacity limits, either through archiving or because the amount of data doesn’t grow above a certain size, you can split up your application into a fixed set of databases and let your users access those databases directly.

Some examples of such applications are as follows:

- Customer Relationship Management application with databases for the following:
  - Customer information
  - Case handling (for example, processing of loan applications)
  - Correspondence
  - Archive
  - Application setup

- Ordering application with databases for the following:
  - Customer profile
  - Orders
  - Account Management (tracking queries and so on)

- Quality Management and Process Documentation application with databases for the following:
  - Valid public documents
  - Document revision, review and signoff
  - Document archive

Splitting up databases in this way may allow mobile users to have only a subset of the application on their PC and still be fully functional, whereas the applications where the database structure is hidden cannot easily support a mobile user.

Applications with hidden database structures will be dependent on the overall application navigation hierarchy and search in finding documents. For applications with data split up according to function, region, and so on,
users will be able to take advantage of views in the individual databases for navigation. For applications with a hidden database structure, you may need to spend more time on creating a good user interface, while you get some of this functionality “for free” in applications where the users access the individual databases directly. On the other hand, applications with hidden database structures give you more flexibility in extending the capacity of your application.

Keeping Database Size Down by Using Databases with Subsets of Data

If users on different servers only require access to a part of the data in a database, you can keep the database size down by only storing the data relevant for the users on that server. This can be accomplished through the following methods:

- Selective replication formulas
  To use selective replication you must be able to identify which documents to replicate through a formula or through the placement of documents in views or folders.

- Readers fields
  You can accomplish the same result as with selective replication, plus you get added security. Here the requirement is that the name of the receiving server must be listed in the readers field for all documents that are to be replicated. It also requires the addition of a role to the readers field and the assignment to that role of all the users who should be able to see the documents on the server.

- Non-replica copies of the same database
  In cases where you can determine that data created on one server does not need to be available on other servers, you can simply create non-replica copies of your database on each server. These databases can still archive to a common database, but they are in effect individual applications. The reason for mentioning them is that they sometimes can be used as parts of a larger application with replicating databases. For example, consider a case-handling application where all locations support all customers, but where one location handles an entire case once they have initiated it. In this situation you need a database with the customer information that is required by the case-handling application. The customer database must be replicated to all servers in full because all locations must be able to support all customers, while the case-handling database does not need to replicate because only users on the same server need access to it.
The use of selective replication is dependent on having an experienced administrator, or a design that makes it easy to set up selective replication. Incorrect use of selective replication formulas may result in the wrong documents replicating; this can be a major effort to clean up.

In some situations the use of readers fields may affect user response time. For more details see Appendix C-3, “Fast Access to Reader Names Protected Documents.” However, if all users have access to all documents in the subset that is replicated to their server, this should not be a problem.

**Using Special Databases and a Subset of the Application for Mobile Users**

As discussed previously, the goal should be to minimize the number of databases mobile users have to replicate, the size of those databases, and the number of updates that are irrelevant to the individual user.

There are two main things you can do to accomplish this:

- Define a subset of the application, with databases that it can work with independent of the rest of the application
- Create special compact mobile versions of big databases

When you design the application you have to identify which data needs to be available to the mobile user, and place this data in as few databases as possible. A simple example is to not have a separate help database for your application; instead, include the help in the database that the mobile users replicate anyway. In this case they only have to replicate one database instead of two.

You must also make sure that the mobile subset of databases can work without direct access to the other databases in an application. For example, if you have a database to which all confirmed orders are moved from the order entry database, before being transferred to a back-end fulfillment system, you must make sure that the application does not break if the database for confirmed orders is not online. One way to do this is to define the database for confirmed orders as a mail-in database and simply send confirmed orders from the order entry database. For disconnected users the order will be placed in their local outgoing mail box, and the next time they connect to the server the order will be transferred, together with any other outgoing mail. This solution will also allow you keep strict security on the database through which all confirmed orders passes. For instance, you can set the default access to depositor because only the router needs to access the database to deliver the confirmed orders.

Sometimes mobile users requires access to large amounts of data, like all company customers or all parts being sold by he company. If the application does not require the mobile user to be able to update the data, but only use it
for reference or copy it to another database for further work, you can create a special compact database like Lotus Mobile Enterprise Directory, where each document holds information about several records. For example, a version for all people in IBM, Iris, Lotus and Tivoli holds information about 468,856 entries (persons, groups and so on) in only 2674 documents. While the uncompressed data size is 3896MB, the compressed version only requires 70MB plus 68MB for the Full Text Search Index. Refer to Appendix A-2, “High Performance Reference Databases” for a description of how to create a similar database and how to use it from other databases. Before you begin to create such a database, remember that you must create the logic that can create and update the compact database. The data must be read-only, so for cases where the mobile users need to create new records or update existing ones, you can put in a special form where the new data is entered and the document is sent to a mail-in database without being saved in the compressed database. A central administrator can monitor the mail-in database and apply any required changes in the original data, or you can automate this on your own.

It may be inconvenient if there are a lot of updates to the data because this will cause extended replication time. In the example where 2674 documents hold information about 468,856 entries, updated information for one percent of the entries can cause updates to 100 percent of the documents in the database in the worst case. You need to establish what the requirement is for up-to-date data, and then if possible only update the compressed database when needed. In this way you may have a better chance of several updates hitting the same documents.

**How Application Deployment on Several Servers and Several Locations Can Affect Your Design**

When designing, you need to take into account the infrastructure on which your application is to be deployed. For example, an application requirement may be that all users can see any update as soon as it has been saved to the database. If the existing infrastructure has users spread out on several replicating application servers, you will not be able to fulfill that requirement; you will need to get a dedicated server for your application that all users can access. Alternatively, you can go back and see whether the requirement really is valid, or if it can be modified to something less restrictive, such as “all data must be available to all users within four hours of the update.”

You must also consider how the current infrastructure can grow and how it will affect your application. You can have situations where more servers, more users, and more locations are added to the infrastructure through ordinary growth, through mergers and alliances, and you can have situations where an application is deployed in part of the infrastructure
and then is later deployed more widely. For example, a departmental solution that is deployed on one server may suddenly be elevated to the status of an enterprise application and be deployed over several servers. Your application may have an archive database and your Domino administrator may wish to not replicate this database to all servers because it takes up a lot of disk space and has little user traffic. In this situation, your application must be able to look up the archive database even though it is not located on the same server, and your archiving code must be able to handle it if the server with the archive database is unavailable by deferring the archiving until the next time the server is available.

When discussing these considerations, we do not mean to imply that you should design even the smallest workgroup solution to have enterprise strength. The important thing is that you think about how far the use of your application may go, and either design it for that ultimate use or so it can be easily extended should the need arise. The important thing is that you identify which parts of your application will need modifications, or the limitations if it is to be deployed beyond its original scope, so this doesn’t come as a surprise after the fact.

In the following sections, we discuss how the infrastructure can affect your application. We will start simply, with one server and one location, and then add gradual complexity as we go along.

**Application on One Server in One Location**
The application in this scenario can be anything from a small workgroup solution to an enterprise application deployed on a central server. Here we do not need to be concerned about the implications of replication and redundant data, low bandwidth links and so on, but there are still some performance considerations for this scenario.

What are the access control requirements for the application? In some cases you can avoid using readers fields (which can impact performance) by using several databases, and then restrict access to the content in some of the databases through the ACL.

Further, in many situations most users only require read access. With a solution where all “read-only” data is stored in one database and the content is created in another, access to the “read-only” database can be accelerated by setting the default access to reader and having no other entries in the ACL. This will keep the Domino server from going through the ACL list when a user tries to open the database.

**Local Replicas**

With this configuration, you can perform load distribution if the clients are Notes or Web clients using Domino OffLine Services (DOLS). The users can have local replicas of the busiest database on their own PC and replicate periodically in the background with the database on the server. There are several questions to ask before choosing this path, as follows:

- Is there sufficient disk space on the users’ PCs for the local replica? Often a user will only need a subset of the data that is in the database on the server. To avoid replicating all data out to each client, you can either use selective replication or Readers fields.

  Selective replication cannot be set up or changed programmatically, so except for ‘easy’ cases, such as where the user just needs to select to replicate a folder, you must plan to have assistance ready to help the users set up selective replication.

  While Readers fields add complexity to the application and can also have significant impact on databases that users access directly on a server, they should not have any performance impact when replicating in the background.

- Are the PCs connected to the network at all times? If yes, you only need to have the busiest databases replicated to the PCs and let the users access other, less busy databases directly on the server.

  To allow administrators flexibility in determining whether all or only some of the databases should be replicated to the users’ PCs, make sure that the way you look up the different databases can be configured to allow for some databases to reside locally and some databases to reside on a server.
• Does the application contain sensitive data and are the PCs physically secured?
  Consider using encryption on the local databases if they contain sensitive data.

• Do different users work on the same data? What are the concurrency requirements for such data?
  You must consider each of the following situations: two users update the same data at the same time, causing a replication conflict; and one user has updated a document, but another user is working with the “old” version of the document because the update has not been replicated from the first user’s workstation to the server and then to the second user’s workstation.
  If your application requires that any update must be available to all as soon as it has happened, you should not use local replicas of your application databases. The issue of several people updating the same document at the same time is discussed under “Load Distribution Through Replication” earlier in this chapter.

Note  Use of Roles is not supported in local databases unless you use ‘Enforce Consistent ACL’. An example of using roles to control your application is if you only show an ‘Escalate’ action button to help desk personnel. This can be done through a hide-when formula that hides the action button unless the current user is attached to a role such as [HelpdeskTeam]. However, in a local replica the button will always be hidden unless Enforce Consistent ACL is used, or you add your own programming to support roles locally. For an example of the latter, see Appendix B-6, “Supporting Roles Locally Without Enforcing Consistent ACL.”

Some advantages of using local databases in your application are as follows:

• Utilization of workstation power
• Support more users with the same hardware
• Better response time
  Often, very little of the processing power of user PCs is used. Moving parts of your application to the local PC will take load off the server and allow you to use the CPU of the local PC. Assuming that most of your transactions can be completed locally, this also will give you a better response time in most average installations. When you take load off the server, you can allow more users to connect to the same server.

20  Performance Considerations for Domino Applications
Some disadvantages of using local databases in your application are as follows:

- **The solution is dependent on workstation replication.**
  You cannot push anything to the workstation. If the user accidentally turns off the background replication they may be working on outdated data. To avoid this, you need to put checks for outdated data and perhaps outdated design in your code.

- **Data is only updated according to your replication cycle.**
  This is generally not a problem if the users have their application set up to replicate every 10–15 minutes. In most cases this update frequency is good enough. However, users that need to monitor databases for documents created by others and act on them as soon as possible (for example, Help desk personnel) should watch server-based databases.

- **Security exposure**
  A server is normally located in a more secure environment than the clients, so there is a larger possibility that a workstation can be stolen than a server. To address this, local databases with sensitive data should be encrypted.

- **More administration**
  Having parts of the application split out on all users’ workstations requires more administration work helping users to set up background replication, selective replication if needed, database encryption, trouble shooting replication problems, and so on.

**Application with Mobile Users**
We have already introduced the topic of supporting mobile users in your application above, where we discussed local replicas of application databases. As with the use of local replicas in a permanent environment, mobile applications require a fair amount of expertise on the user side to handle the initial replication of databases, potentially specifying selective replication formulas, enabling background replication, making sure updates also get replicated back to the server, and so on. In addition, there are two points to consider:

- Mobile users are not connected to the network all of the time.
- When mobile users are connected, the bandwidth can be low.

For constantly connected users, you can have the busiest databases accessed as local replicas and keep the less frequently used databases on the server. For mobile users, all databases required by the application must reside locally.

Refer to “Using Special Databases and a Subset of the Application for Mobile Users” earlier in this chapter for some suggestions on how to split up your application when supporting mobile users.

Security is an even bigger consideration for mobile users than with databases residing locally on PCs in a controlled environment like an office. Databases on mobile PCs should always be encrypted, even though there is a minor performance impact associated with encrypted databases.

Use of Enforce Consistent ACL is also an option, but this is not as secure as encryption.

Application with Back-end Connectivity
There are two main scenarios to examine with regard to Domino and enterprise integration:

- Storing core data on enterprise systems and accessing them in real time from your Domino application
- Transferring data from the external database to Domino as a scheduled one-way transfer (read-only) or as a two-way data replication between the external system and Domino

There are many good reasons for storing some or all of the application data on a back-end system. Among them are the following:

- Larger capacity
- Better security
- Data available for other (non-Domino) applications

If you choose a design that transfers data to Domino (mirroring) before accessing the data in your application, the situation is almost equal to what we discussed under “Application on One Server in One Location” from a performance design perspective.

If you need to manipulate the data that is transferred from the host before you make it available to users, you may want to dedicate a special database to this process and then move the data over to the database where it is available to the users. You could also introduce another server (called a staging server) where you put a replica copy of the user database and then manipulate all the data in that database on the staging server before replicating with the production server. Using a dedicated staging server to transfer data and process it using agents or API programs will also help performance-wise by not adding the load of transfer and processing to the production server that the users access.

Mirroring or staging may give better response time, but it cannot be used in some situations, such as when immediate updates are required in the back-end system, when the size of all the back-end data makes it impractical, or when the data must be stored in only one place for security reasons. If you use mirroring or staging, calculate how much time the transfer and synchronization of data will take and whether there will be any conflict with the availability requirements of the applications if it is unavailable during that operation.
There are several products and techniques that you can use for real-time access. Among them are the following:

- **Domino Enterprise Connection Services (DECS)**
  DECS is bundled with the Domino Application Server. The service runs on the Domino server and acts on Domino document events, allowing you to query, create, update and delete data in the back-end system. The requirement is that you identify the back-end document through a simple keyed query. You specify keys and field mapping between the back-end tables or views and the fields in a Domino document using a form in the DECS administration database. No programming is required.

  Normally only the key field(s) for a record is stored in the Domino document between queries — the other fields are removed as soon as the document is closed, as they are only valid on the back-end system. Even though the data only resides in the back-end system, Domino can index the back-end data and searches can be conducted against the index of the Domino database. This way you can search through a huge amount of back-end data very quickly without accessing the back-end system until you want to work with one of the records in your search result set.

- **Lotus Enterprise Integrator (LEI)**
  Besides functionality for scheduling large data transfers, LEI also supports real-time data access. With LEI you can extend your declarative definition of the transfer (specifying keys and field mapping) with scripting.

- **Lotus Connector classes**
  If you need programmatic flexibility during your real-time access to back-end data, you can use the Lotus Connector classes for LotusScript or for Java. To use the connector classes, you need a Lotus Connector for the back-end system to which you want to connect, and software to connect to the back-end system must be installed on all clients.

  In some instances you can avoid installing the communication software on all clients by having an agent on the server do the query and return the result to you. You need to use documents to pass arguments to the agent before starting it from LotusScript with the runOnServer(NoteID) method (introduced in Domino R5.0.2). The agent must then write the returned data to the same document, which your code can read once the agent has run. In most situations, the additional steps of writing and reading documents and kicking off a server agent will not give as good a response time as a solution where the communication software resides on the client. However, for low volume queries this is a very flexible and inexpensive method to access back-end data in real-time.
• Enterprise Solution Builder  
Lotus Enterprise Solution Builder (ESB) is a high-performance enterprise integration runtime and development environment for Basic developers. Applications which run on ESB can be executed remotely from Notes, Web or Windows 32 clients. ESB consists of a server-based runtime service and an integrated development tool. The runtime service, ESB Runtime, can run on the Domino server or on an isolated server machine. The developer tool, ESB Developer, is a productive development environment for building and debugging application logic in LotusScript. In addition to being based on LotusScript, ESB exploits all Lotus Domino Connectors via the Lotus Connector LSX (LC LSX).

ESB Runtime allows Domino developers to deploy high-volume and interactive solutions with sophisticated functionality, integrated with enterprise applications and systems that previously were difficult to build in the Domino environment. Applications such as business intelligence and functionality such as data analysis require continual queries and updates to enterprise data. ESB provides Domino with the services needed to support this functionality, and allows Domino solutions to be considered for a much broader set of applications than before.

Use ESB for high-volume real-time transfers. While ESB can also be programmed to do scheduled data movements, like those LEI performs, it does not have any pre-built modules to facilitate this. In many respects, ESB offers the same functionality that you can get using Java servlets. However, you only need LotusScript or Basic skills to develop with ESB.

• Servlets  
If your clients are Web browsers and you can program in Java, you can also use servlets to access back-end data. Servlets are often several times faster than agents. To learn more about how to use servlets with Domino, see Connecting Domino to the Enterprise Using Java, IBM form number SG24-5425, Lotus part number CT6EMNA.

• Add-In Task  
Finally, you can also pull down your C++ compiler and write a Domino Server add-in task that accesses the back-end system directly. This may or may not be faster than servlets, but it will allow you to control all aspects of the code that accesses the back-end data, including memory management. In addition, you can also write add-in tasks to enable a Notes client to access the back-end data in real-time.

Note  There are also many good business partner solutions for accessing back-end data from Domino. You may want to check out these solutions before developing your own.
The order in which we presented the products and techniques above also represents increasing complexity for implementing a solution, as well as increasing the performance and scalability potential of the solution.

Which product or technique you should choose depends on the response time requirements for your application and how complex your query is. If all you need is to find and update data through a simple keyed lookup, DECS will work in many cases. When accessing back-end data through Domino you have a three-tier application; and while predicting performance for Domino applications where users only access data in Domino databases (a two-tiered application) is difficult, it is even more difficult for a three-tiered application. To get an idea of the scalability and response time you can get, you need to create a proof-of-concept prototype accessing the real-world systems.

You should also note that an application originally requested as a standalone may be required to connect and synchronize with back-end data in the future. For example, an employee skills planning application may start as an independent solution. After deployment you may be required to get the basic employee information from a Human Resources (HR) system and to store the approved employee skill plans under their records in that system. Therefore, if the data required for your application also exists in back-end systems, you may want to structure it in a similar fashion even though back-end connection is not a stated requirement at the time.

**Application on Several Servers**
In this scenario, we examine the deployment of the application on several
servers in the same Local Area Network (LAN). When the servers are on the
same LAN they are relatively close, often in the same building or the same
compound. This is in contrast to servers on a Wide Area Network (WAN),
which we will discuss in a later section.

There are several reasons why you may want to have your application
deployed on several servers within the same LAN. They include the
following:

- **When one large server is not enough**
  Your application may be so CPU or I/O intensive that you need to
  spread the user load over several servers.

- **When the current infrastructure requires it**
  For example, users may only be allowed access to dedicated servers, or
  several servers are already deployed in the existing infrastructure and
  there is no money to acquire a server large enough to host the
  application for all users.

Deployment on several servers offers the following advantages:

- **Load balancing using replicated databases**
- **Load balancing where the same application is deployed on several
  servers without replicating data**

The application requirements will determine whether it is feasible not to give
all users access to all data. If having subsets of data available only to certain
users is feasible, there are at least two ways to do so:

- **You can still deploy replica databases on the servers and then control
  which data goes to an individual server through selective replication or
  readers fields.**
- **Create non-replica databases on each server from the same templates.**
  An example of this is a Customer Relationship Management application
  where the basic data for all customers is replicated to all servers in one
database, while another database that is a case handling application
exists as a non-replica copy on each server and is used to store data that
is relevant only to the region that accesses each server.

There are some additional considerations when the application data resides
on several servers:

- **Requirements for up-to-date data**
  This was discussed under “Load Distribution Through Replication”
earlier in this chapter.
Whether to replicate all databases in the application

For data-intensive applications you may end up with a lot of redundant data on each server. One way to save disk space is to replicate only the databases in your application that are most heavily accessed, and keep less frequently used databases on one server. An example of this is an archive database. Often this is the largest database in the application, and one that users seldom access.

**Note** If you deploy an application where some of the databases are replicated while others only reside on one server, make sure that the way you locate databases in your setup takes into account that the database you want may not be located on the same server. If your application uses a simple setup solution that assumes that the other databases in an application reside on the same server or even in the same directory, this solution may not work once you start deploying some but not all of your application databases on several servers.

**Application with Users in Several Locations**

If your application must support a remote office without any servers, you must consider how users can access your application over a WAN. Ability to work on a WAN may be part of the application requirement, but it may also be a requirement that suddenly appears after the application has been deployed because of a merger or business expansion.

For applications on a WAN, the network may be the bottleneck before anything else. Domino uses Notes Remote Procedure Calls (NRPC) to communicate between the server and the Notes client, and one user action normally results in many NRPCs going back and forth between the client.
and the server. If the client is a Web browser, the communication uses the HTTP protocol. While you can do simple processing like input validation on the Web browser client using JavaScript, you will still need more round trips on average between the Domino server and the Web browser, than if you were using a Notes client for the same operation, so more bandwidth is required for Web browsers.

How good a bandwidth you can get depends both on cost and infrastructure. In some parts of the world WAN connections are still quite expensive, and in some places you are not able to get very much bandwidth no matter how much you are ready to pay for it.

When looking at how many users you can support remotely on a dedicated line, the following guidelines suggest how much bandwidth a user requires:

<table>
<thead>
<tr>
<th>Client</th>
<th>Bandwidth requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes</td>
<td>3.6 Kbs</td>
</tr>
<tr>
<td>Web</td>
<td>5.0 Kbs</td>
</tr>
</tbody>
</table>

These rules are mostly derived from mail users. Remember, the usage pattern for your application may be very different from mail usage. In addition, it is very rare for you to get a line to a remote location dedicated to your application; most likely you will need to share it with other applications.

What if you have more remote users than the current bandwidth can support?

- Have the network upgraded to support a larger bandwidth.
  This comes of the thesis that it is better to have the application on as few servers as possible when looking at the administration cost.

- If the remote users run Notes, you can replicate some of the application databases to their local PCs. See the section titled “Application on One Server in One Location” for pros and cons of this approach.

- Put a Domino server in the remote location to support all users there.
  This option will be further discussed in the next section.
If your application will be deployed on servers connected through a WAN, you still need to consider the same things as we discussed under “Application on Several Servers.”

In addition, you need to consider that replication over a WAN takes more time. Try to calculate how long it will take before an update at any server is replicated out to all other servers. You should also estimate the amount of data to replicate in each replication cycle. Is there any danger that all data cannot be replicated within the assigned time slot?

Make sure that field-level replication is being enabled for all documents in the application unless there is a specific requirement that prohibits it. An example where you cannot enable field-level replication is when each version of a document must be assigned to a specific author for legal reasons. With field-level replication enabled, you may end up with a document that is a merger of changes made by several authors at the same time, independently. However, in most cases you can and should enable field-level replication. Moving the content of only fields that have changed over the network, instead of all fields in a document, makes replication much faster.

**If Your Application Is Ever to Be Deployed, Take the Infrastructure Into Account**

In our walkthrough of different infrastructure scenarios we have seen different ways to achieve load distribution, but also how a given infrastructure can put some restrictions on your application. You are advised to take the infrastructure into account right from the beginning, when you start to design the application.

30 Performance Considerations for Domino Applications
Think about Caching

One of the best ways to speed up your application is to move the application and data close to the user. We have already discussed forms of data distribution where we put replica copies of databases on servers close to the user or on the user’s local workstation. Under enterprise integration we discussed how data can be transferred to a staging server that clients access instead of going all the way to the original source. We call this distribution of data “caching.” However, there is also another area where you can employ caching: the application design. Caching is related to design in the following ways:

- You can cache application design (and data) in your code.
- Notes caches design elements on the workstation.
- Domino caches design elements, lookups and profile documents on the server.

Optimizing the caching of application design is very important. In some cases, you can control some of the caching. In other cases you are dependent on how Notes and Domino cache design information, but with knowledge of how the caching is done you can still design your application to utilize caching in the best way.

You will find discussion about caching in several places where it is relevant to other topics in this book. For example, in the section “Maximize Resource Usage” in Chapter 3, “Programming Considerations” there is an example of how you can store design and data in global variables for reuse with LotusScript. The way the Notes client stores design elements on the workstation is described in the section “CACHE.DSK” in Chapter 4, “Performance Aspects of Domino Design Elements.” You can read about “Profile Documents” for fast data access in the same chapter. In Appendix C-7, “Ways to Control Caching of Pages Served by Domino,” you can read about the rules the Domino Web server applies when caching Web pages, which @functions you can use and still have the result cached, and so on.

On the server side, items such as name lookups from Domino Directory, view information, database properties, Web pages, and so on have their own caches. Some settings of the database server cache can be controlled by your Domino administrator through the NOTES.INI parameters:

- NSF_Buffer_Pool_Size=value sets the maximum size of the NSF buffer pool.
- NSF_DbCache_Maxentries=value sets the number of databases that a server can hold in its database cache at one time.
When using caching from a development perspective (either directly or by utilizing the caching supplied by Notes/Domino) you must consider the following:

- Which items are most important to cache?
  Whenever something is cached it initially needs to be moved to the cache. The exercise here is to only move the data and design elements that the users access often to the cache. An example is using selective replication for local database replicas, so that only data relevant for the user is replicated. Another example is structuring code in script libraries in such a way that only necessary script libraries are loaded when a user opens a form.

- When should I cache an item?
  Different users have different patterns of usage. You probably can not predict when a user needs a certain item in the application. The easy way out is to cache everything at application startup, but it will give you a very long application loading time and unnecessary use of memory to store the cached items. We already mentioned that if you can structure your script libraries without many interdependencies, you can get faster loading of items referencing script libraries. Another example discussed in Chapter 3 is lazy initialization, where you do not initialize Domino objects until needed.

- What are the update requirements for a cached item?
  As a rule of thumb, only cache items that change during the course of a user session. If you need to cache items that may be updated by other systems or users, you need to develop a mechanism for signaling that a cached item needs to updated.

---

User Perception of Performance

While you can measure how long a certain transaction takes, it is often more important how long the user perceives it to take. There are two main things that you can do to enhance the user experience during transactions that take more than a few seconds. They are giving user feedback and using deferred (background) processing that lets the user continue with a new task right away. We discuss each method in this section.

32 Performance Considerations for Domino Applications
Give the Users Feedback

Tell the user what is going on. One of the most stress-producing situations is when you feel that you have lost control. It is very frustrating if a user starts a transaction, for example by pressing a button, and then nothing happens in the user interface for many seconds. One might argue that as soon as the users have done it a few times they will know how much of a delay to expect. This is a bad excuse for not being user-friendly, and furthermore, the actual delay may be dependent on server load, back-end connectivity and so on, so the delay will not be the same every time.

For the Notes client there are several ways to update the user. The most simple is to use the status line to write what is going on. From LotusScript you use the Print statement to write to the status line in the Notes client. If the transaction is processing many documents, you can write the number of documents being processed, or the percentage of the total number of processes. If it is a transaction where different types of data are being processed, you may simply write which type of data is currently being processed so the user can get a feel for how far along the processing is without showing details like number of documents or percentage completed. As long as the feedback message changes without too much delay it will make the user more comfortable and avoid the feeling of being locked out and uncertain about when the transaction is done.

See Appendix B-3, “Displaying Processing Status,” for some techniques to give feedback in the Notes client.

In a Web client you don’t have the same capabilities for feedback. You can load an animated GIF file or you can create progress indicator applets, but considering the overhead of loading the applets and perhaps the communications with the server to get the actual progress information, it may not be worthwhile to do. For Web clients you should focus on how much deferred processing you can do.

Use Deferred Processing to Let the Users Continue with Other Tasks

You may have situations where a transaction is made up of a user entering or validating some data, followed by some processing of that data where the processing doesn’t require user interaction but takes up quite some time. In such situations you should see whether you can do this processing in the background and let the user continue with their other work.

An example is the deletion of a document that has a lot of related documents in other databases that must be deleted together with this document to enforce referential integrity. Instead of doing all the deletions before returning control to the user, you could instead mark the main document for deletion and then have a scheduled agent do the actual deletions, including in related documents, in the background. With this approach the user will
not get immediate feedback regarding the success of the operation. To remedy this you can create a document with the status of the operation after the agent has run, and create a special administrative view where all status documents can be seen, or you can choose to create a status document only in a case where something unexpected happens. However, in the spirit of “trust but verify,” we recommend that you log the status of a deletion operation of business documents. If the main document you delete has a field with an owner name you can also mail the status of the operation to the owner directly. There are several way to do this; the important thing is to have a log of background operations you may need to track. For example, with several people being authorized to delete, you may want to log which user actually requested the deletions.

For deferred operations, there may be situations where the operation cannot be completed because of business rules or because verification is needed from the person requesting the operation. For example, if a customer document is scheduled to be deleted together with all associated documents, and it then turns out that there are open orders for this customer, you may not be allowed to delete anything. If your business rules say that such a customer must never be deleted, you could simply create a status document or mail the document owner a message saying that the deletion cannot be completed because of open orders for this customer. But if your rules say that it is OK to delete all other documents and just leave the order documents in the application, you could go along and delete everything else. However, if the user had known about the orders in process, they may not have requested the deletion in the first place. If possible in such a situation, you should check for open orders before marking the main document for deletion, and if there are any open orders, inquire of the user whether they still want to have all other documents deleted. However, the open orders may reside in an external system and the response time for such a check may be too long. In this case you can consider always asking the user whether they still want to delete all other documents if there are open orders for this customer, before marking the main document for deletion.

If you use deferred processing, remember to do the following:

- Validate all user input before returning control to the user if this can be done in a timely manner.
- Anticipate exceptions that require additional information from the user. If possible, you should check for exceptions and get additional information to handle them before returning control to the user, or you may consider always asking for this information so that the deferred processing can complete successfully.
• Make sure to log the results of the deferred processing using a single log document, status documents in a view, mail to the user or a similar technique.

See also “Synchronous Processing” in Chapter 3 for some implementation techniques.

Techniques for Multi-database Applications

This section is about techniques that you can use once you have chosen to split up your application into several databases for any of the reasons discussed previously.

We discuss these techniques even though they are not directly performance-related, because the use of several databases in an application many times is prompted by a goal of better performance.

The techniques are mostly focused on applications where data and functionality are split up in a fixed set of databases, but some are also relevant for applications where databases or subsets are replicated to the users’ local PCs.

We will discuss the following:

• Using keys to relate data in different databases
• Whether to look up data dynamically or use redundant data
• Switching seamlessly between databases
• Signaling that data or design needs to be updated

Using Keys to Relate Data in Different Databases

Once you have your application split over several databases, you must have a way to jump from related data in one database to another. There are several ways to do this. Two widely used methods are the following:

• Use the same key for all documents relating to a specific entity.
• Arrange related documents in a tree structure, where any document has references to its parent plus any children it may have.

When you use a key to look up related data in other databases, you must make sure that:

• The key is unique and everlasting
• You are able to supply a key for all records

Therefore, you must select the keys very carefully. For example, consider an application where personnel data is looked up based on the name of the person. Obviously, the application will stop giving valid results when several people have the same name. Also, people may change their last name.
when getting married or for other reasons, and while you may want to change their last name in most places in your application anyway, a unique key should not change once it has been assigned. You may have archived documents that you cannot change, or files that must not be modified for legal reasons, and so forth. For similar reasons, you shouldn’t use a telephone number as the key either. While in many you cases want to have a view where you can locate people by their telephone number, and most people are able to remember their phone number, you should not use a telephone number as a unique key because this number is not a lifelong property and it may be transferred to another person or another company.

You can create your own unique numbering in its most simple incarnation through the @Unique function in Domino, or you can use the DocumentUniqueID generated by Domino for each document. However, if your application will work together with other systems, you must make sure that the other systems can handle the format of the unique key that you create. It is preferable to have a unique key for your data that is universal to that data, rather than one just created by your application.

For person information in many countries, such a key could be the social security number that is assigned to you when you are born or when you immigrate. Using a universal key will make it easier to transfer data with external applications at other companies, government agencies and so on. For person data, it will also make it more likely that people can remember their key. On the other hand, you must make sure that you can get the universal key for all your data. In the case of social security numbers, some people may be hesitant to supply their social security number to anybody — even to your foolproof application! To dwell a bit more on this example, you may also want to create records for persons from foreign countries who do not hold a social security number, or extend your application to include company as well as personnel data. In most countries, companies do not have a social security number. In some countries there is something similar for companies, but you need to make sure that all companies have such a number (or easily can get one), and be sure not to make any assumptions about the format of the social security number key in case the number for companies uses a different format. While making assumptions about the format of the key can help you validate that it is a valid key that is being entered, you must also consider whether other copies of your application will be deployed in other countries, where a different format for the unique key is used.

To support an application that can work with several types of main records (for example, people, organizations and companies) in several countries, and exchange data with systems external both to the application and the company, you really need to consider which key to use. Most applications don’t have to support as many different requirements as we just discussed.
The important thing is that you clearly document how far your application can go before problems may arise with the key you are using. For example, for customer relationship management solutions you will get along fine with a system where you create customer numbers for the people and companies you deal with in your application, or for a Human Resources application the best key to use is the employee number. Often this selection of a key will come automatically because the source data is created in another system with a key assigned already, and the data in Domino is then created on the basis of that pre-existing key.

The second method, where a document refers to one parent and a number of children, requires you to follow the tree structure when you navigate through your application. This “locked” navigation scheme may enable the user to move between the different documents a bit faster, while simply using the same key for all related documents affords the user more flexibility in navigation.

Using the same key for all related documents also allows the user to specify a selective replication formula that replicates all related documents for a given key.

To make replicating subsets of data to different servers easier, you should identify a secondary key that will be the same for all documents (or at least a large group of them) going to a particular server. The selective replication formula then simply selects documents to replicate based on the value of the secondary key. The value of a secondary key can be the name or number of the regional office that ‘owns’ the data, postal zip number, the first five digits in a ten-digit customer number, and so on. The important thing is that this secondary key must also be present in all documents.

**Look up Data Dynamically or Use Redundant Data**

Domino is not a relational database. Sometimes it can be expensive to pull data together from several documents in several databases to give the users a comprehensive view of the data they are working with. For example, in a skills/resource application where resource coordinators can look for people with certain skills, you may want to include information such as cost rate and manager in any employee’s skills description even though this information exists in another Domino database or in an external system. You can choose to store a copy of this data in the skills database to cut down on the number of required lookups.
If you use redundant data (copies of data), you must have a mechanism for keeping the data in synch. You can use background agents for this. Here it is important to check what the business requirements are for having data synchronized. For example, in the skills application with cost rate and manager as redundant data, it may work only to synchronize data once a week, as cost rates and managers do not change that often. The reason for mentioning this is that the background agents that synchronize redundant data may put a heavy load on the server.

Switching Seamlessly Between Databases

Once you have split up your application into several databases, you have the challenge of making it easy for the user to navigate between the databases. For Web users this can be done relatively easily by using frames, including one frame with a navigator where the different options take the user to different databases. With the support for frames in Notes R5.0, you have the same possibilities in Notes, but for Notes R4.x you must use a different method to make the switch between databases seamless for the user.

The challenge is not to switch from one database to another, but to keep on working with the relevant data. If your documents are tied together by keys in a tree structure, you will get this automatically, just for the cost of following the tree structure in your navigation. If you use the same key for all related data you must store the key you are working with, switch to the new database and then go to the data for the key you just stored. In Appendix B-5, “Switching Seamlessly Between Databases,” we show an example of how this can be done for Notes 4.x.

It is important to design your logic for switching between databases in a generic manner. Your application may have additional functionality and databases added later on. You may have situations where all databases do not reside on the same server (for example, an archive database only resides on one server while the rest of the application databases are deployed on several servers), so your code must not assume that all databases are on the same server. Also, for mobile users with only a subset of databases, you should give them their own version of the seamless interface so they don’t try to switch to a database they don’t have. Alternatively, if you stick to the same interface that server-based users have, your code should inform mobile users politely that they don’t have connection to the database to which they want to switch.
Signaling That Data or Design Needs to be Updated

If you deploy your application or parts of it as local replicas on the users’ PCs, you must make sure that the users replicate with the server to keep data and design up-to-date. For mobile users replicating is a normal way of life, but users that continuously network and access other applications directly on servers may forget to replicate, or they may have been asked to turn off background replication to preserve server resources. If you use redundant data in your application that is not centrally updated, you must also check its validity. To ensure that the user does not work with outdated data or with a design with outdated business rules, you can put in a check for this when the user opens the database. Following is an overview of one way to perform these checks in the database PostOpen event:

1. Check that the database is a local replica using the NotesDatabase.Server method. If the method returns an empty string “” the database is local.
2. Get the refresh intervals for external data and database replication from the database profile document, and check them against the dates for the last updates in the user’s private profile document. For example, you may have a business rule that external data should be updated at least every 14 days and the database must have been replicated within the last seven days.
3. If an update of external data is required, tell the user. Give the option of canceling the update to make the user feel in control. Should the user not wish to update the external data, your business rules must decide whether the user still can be allowed to work with the application. If the user accepts the update, refresh the external data and write the date and time for the update in the user’s profile document.
4. If replication of the database is required, you can again give the user the option to cancel the operation. If the user accepts, go on and replicate the database using the NotesDatabase.Replicate( serverName$ ) method. Write the date and time for the replication in the user’s profile document.
5. Continue with any additional processing you may have in the database postOpen event.

A Bit of Methodology Discussion

Methodology discussions deserve their own dedicated books. Designing a Domino application does not require any special methodology. Any good methodology for designing applications will do.
However, for those of you who are starting to look into the use of a methodology we will point you to a very important technique in designing applications. For those of you who already have incorporated a methodology in your design work, we will point you to a method that focuses on constructing applications to meet performance objectives.

In this section we discuss the following:

- The Use Case Model: using scenarios to design your application
- Software Performance Engineering (SPE) methods

**Using Scenarios to Design Your Application**

To design your application you need information about data size, number of users, their geographical location, the configuration of their PCs, and so on. This information can often be found in the requirements document, or can easily be gathered.

You also need information about the business process your application will support. This should be documented in the requirements document, but sometimes it needs to be better defined during the design phase.

Finally, you need to anticipate how users will use your applications. This is where the *Use Case Model* or the more loosely defined word *scenarios* comes to play.

We will explain a bit about use cases below, but first we describe the advantages of using them. The information gained from this methodology can help you to design for performance and capacity by identifying the following:

- **Performance or Capacity Bottlenecks**
  
  For example, your application may be required to accept orders from the Web and then periodically transfer them to a back-end system for processing. You may have anticipated the orders coming in spread out through the whole day. The use case for ordering from the Web uncovers that the ordering process is triggered when a daily news alert about pricing of raw materials sent by a third party hits the customer. This news alert arrives at all the customers at the same time and thus you have to design for the Web orders arriving in peaks instead of being spread out over the full day.

- **Ways to split data in databases**
  
  For example, your application will store different types of data, like customer contact information and customer orders. Through use cases you may find that some users mostly work with customer contact information and others mostly with customer orders, according to where they are in the organization. In this case it makes sense to put customer...
contact information and customer orders in different databases. On the other hand, you may find that users work both with customer contact information and customer orders for customers that they are assigned to. In this case it makes sense to keep related customer contact information and customer orders in the same database and then split different groups of customers in different databases.

A Use Case Model describes how the application will work from the user’s point of view. It does not describe the internal structure of the application. Do not confuse use cases with user interface design. Use cases describe the system at a higher level. They will be part of the basis for defining the user interface later on, but use cases make no assumptions about the user interface.

A Use Case Model is described by:

- **Actors (external agents)**
  
  *Actors* can be:
  - Users (Customer, Sales Person, System Administrator, and so on)
  - External hardware, like a scanner
  - Another system, like an ERP system with which your application interfaces

- **Use cases**
  
  A use case describes a particular usage of the application, like order entry from the Web.

- **Links between the actors and the use cases**

A Use Case Model is the central part of a requirements document. It states what the proposed system is to do. This is in contrast to the nonfunctional requirements which impose constraints on the system, like performance, reliability, availability, and so on.
The following figure is an example of Actors in a system. The figure comes from the use cases prepared for IBM’s Electronic Response Management System (ERMS), a solution that categorizes and distributes service requests from the Web to different IBM call centers.

---

42 Performance Considerations for Domino Applications
The figure lists the different actors that will be involved with the system. Different symbols are used for human actors and other systems and processes. You might also encounter use cases where the same symbol is used for all kinds of actors, but distinguishing between human actors and other actors makes it easier for the non-expert to relate to the use cases. Note that the use case model can also be used to state functions that will not be supported by the system. For instance, in the figure we see that submitting service requests directly by e-mail (without a Web form) will not be supported.

For each actor there is also a textual description (not included here) with their name, a brief description, relationship to other actors, association to use cases, and whether this actor inherits capabilities from other actors or passes capabilities on to other actors (used in object-oriented design).

The next figure illustrates a subset of the use cases in the ERMS Use Case Model that deal with different ways to submit a service request.

There is a generic use case that describes what is common for all service requests being submitted, and then there are use cases when the request is submitted by telephone or the Web. Submitting service requests from the Web utilizes general functionality described in use cases, Input/Edit Web form fields, Web site-related attributes, and Send e-mail.

The textual description of a use case includes a use case title, subject area (for grouping use cases like submitting service requests), the business event that triggers the use case scenario, actors involved, a brief use case overview, preconditions, outcome, brief description of events/conversations likely to occur in the use cases, and related use cases.

The main purpose of the Use Case Model is to establish the boundary of your application and fully state the functionality that must be delivered to the users.
Other purposes are as follows:

- The Use Case Model gives both the user and the developer a common understanding of the system because it concentrates on what can be done with the application and does not allow information about internal structure, mechanisms, or algorithms to interfere.
- If you use object-oriented design you can identify objects, object functionality, interaction, and interfaces.
- The Use Case Model provides the basis for defining the following:
  - User interface requirements
  - Test cases
  - User documentation
  - Acceptance testing

A Use Case Model is also a very efficient way to discover other systems with which your application interfaces and processes that are not described.

If you do not have a Use Case Model for your application and the application is not simple, we recommend that you get started building one right away. A good reference for learning more about Use Cases is Applying Use Cases: A Practical Guide, by G. Schneider and J. P. Winters, Addison Wesley, MA, USA, 1998, ISBN 0-201-30981-5.

As with anything that has to do with methodology, defining a Use Case Model can be somewhat elaborate and complicated when taken to the fullest extent. You can get very efficient results by using a simple form where you draw on paper and discuss with the users of the application.

It is important to note that you should not try to describe every potential use of your application in a Use Case Model, nor should you go into too much detail. Together with the process owner, user representatives, and the system administrator, you should identify the most important operations that your application must perform as a basis for use cases.

**Software Performance Engineering**

Software Performance Engineering (SPE) is a method for constructing software systems to meet performance objectives, as described in Performance Engineering of Software Systems by C. U. Smith, Addison-Wesley, Reading, MA, 1990.
In this method, performance refers to the response time or throughput as seen by the users. The SPE process begins early in the software life cycle and uses quantitative methods to identify a satisfactory architecture and to eliminate those that are likely to have unacceptable performance. SPE continues throughout the development process to predict and manage the performance of the evolving software, monitor actual performance against specifications, and report problems as they are identified. SPE begins with deliberately simple models that are matched to the current level of knowledge about the emerging software. These models become progressively more detailed and sophisticated as more details about the software become known. SPE methods also cover performance data collection, quantitative analysis techniques, prediction strategies, management of uncertainties, data presentation and tracking, model verification and validation, critical success factors, and performance design principles.

Below are two Web-based references where you can read more about Software Performance Engineering:


- Software Performance Engineering for Object-Oriented Systems: A Use Case Approach by C. U. Smith, January, 1998 (with L.G. Williams) submitted for publication

---

**Validating Your Design Through Proof-of-Concept Prototypes**

As soon as you have a good draft of your design ready, you should start to create a proof-of-concept prototype to validate that the performance and capacity goals for the application can be reached using your suggested design.

Create a prototype with the most used and biggest/most complex views, and documents that can store all data required by the application, to test your design concept.
Ideally, your proof-of-concept must relate to the final application in the following ways. It should:

- Run in a similar environment (WAN, Processor type)
- Use similar data with regard to:
  - Volume
  - Distribution
- Use similar clients with regard to:
  - Number of clients accessing the server
  - Usage patterns

### Similar Environment

There may be some real-world obstacles that will not allow you to do an ideal proof-of-concept prototype. The easiest way to run in a similar environment is to run in the actual production environment. However, most administrators never allow any testing in their production environment. While you may be allowed to run your tests through the night shift, remember that this is still not similar to the environment as it behaves during the daytime with regard to network activity, load on servers, and so on.

If you need to build your own prototype environment, try to get a server as close as possible to the servers where the application will be deployed. Simulating a slow WAN link may be more difficult, depending on which network systems and protocols you have on your test machines. With TCP/IP as the protocol you may be able to flood the system with dummy data using the ping command (see more about ping in Chapter 5, “Troubleshooting an Existing Application”). However, you will not be able to limit the available bandwidth to a certain specified size so it is not really of much use. The best way to see how your design will behave on low bandwidth is to establish a 28.8K dial-up connection to a server with the proof-of-concept prototype and run through your scenarios over that connection.

### Similar Data

It is important that the data be as similar to the real-world data as possible. The best solution is to get actual data from an existing application, if one exists. However, Domino applications often hold sensitive data that is protected by company rules and perhaps also by law, so in some cases you may not be able to get real-world data for your proof-of-concept prototype. If you construct the data yourself it is important that it is diversified (to spread over a lot of categories for categorized views, and so on). It is not enough to create five different documents and then copy these x number of times, nor is it enough to fill out the first ten fields of a document if the document in average will have 20 fields filled out when being deployed.
You can write a program to create simulation data by combining a set of different values for each field for most or all fields on a form. See Appendix B-9, “Method to Populate a Database with Simulation Data,” for further discussion of how to create simulation data using the output from a third-party analysis tool.

**Do Not Create All Test Data with the Same User ID**

For the capacity studies, you should also consider using several different user IDs when filling your application with simulation data. The size of the $UpdatedBy and $Revisions fields grows as different users modify an existing document, and can often take up a lot of space. To prevent this potential pitfall, you should make sure that you create the test data using different user IDs to allow these fields to be populated with a list of revisors and when they made the revisions.

**Similar Number of Users**

It is difficult to obtain a number of users for a proof-of-concept prototype that is similar to how many will access the application. Furthermore, to get these users to exhibit a usage pattern that is similar to what they will exhibit once the application is deployed is even more difficult. You cannot expect to get this number of users and usage pattern before going into pilot production. In the mean time, you need to simulate the numbers of users and their actions.

To produce load on the server you can write agents that create documents, update documents, perform searches, and so on, depending on which scenarios you have identified as the most common or the most critical. These agents can be executed at different clients while you run through some of the scenarios on another client and record the response time.

Writing and maintaining these agents can become a major task. There are several third-party tools that can relieve you of most of that work except writing the scripts to define the workload. See Appendix A-3, “Third Party Tools for Performance Measurement,” for a discussion about some of the available tools.

One thing that is very difficult to simulate is the performance of applications that use readers fields. See more about the impact of readers fields on performance in Appendix C-3, “Fast Access to Readers Names Protected Documents.” You must use as many different clients as possible to create load on the server and make sure that each client runs with a different Notes.ID.
If your clients are Web browsers, you don’t have the same options as you would with agents on different clients creating load on the server. To simulate Web client access to Domino, you need to consult Web test tools, such as those listed in Appendix A-3.

You can also get a tool from Lotus named Server.load to produce load on the server. Read more about this tool in Chapter 5, “Troubleshooting an Existing Application.”

**How to Measure the Results**

During your proof-of-concept prototype testing you should notice how much of the server CPU your application/Domino consumes, and what response time the user gets. Refer to Chapter 5, “Troubleshooting an Existing Application,” for an explanation of some of the tools on the different platforms that you can use to monitor CPU load on a server.

To measure response time, you can get along for quite a while using a stopwatch. You can also monitor the Notes Remote Procedure Calls (NRPC) because they include response time information. For more information on using the Notes Remote Procedure Calls Console, see Appendix B-8, “Using NRPC Output.”

Finally, if you prefer to get output data in a more digested format, you may want to consider buying a third-party tool that allows you to capture and format response time information for Domino applications.

**Performance Testing Never Stops**

While it is very important that you start with proof-of-concept prototype testing early in the design cycle, it is also important to continue testing during the development cycle. A simple change to the design may change the performance characteristics in a major way — and it is important to identify such changes as soon as possible.
In this chapter, we discussed performance aspects of your Domino application that you should already be considering during the design phase, such as the following:

- When you need to split up your application into several databases
- How can you split up your application

The infrastructure where your application is to be deployed will also influence what you can and cannot do; we discussed different scenarios.

We also discussed how you can enhance the user experience by:

- Providing feedback
- Using deferred processing

Once you have decided to split up your application into several databases, you need to make the different databases appear to the user as one application; we have discussed different techniques for doing this.

Some methodology has also been covered with our description of the Use Case Model and Software Performance Engineering methods.

Finally, we discussed how to do a proof-of-concept prototype early on to test that your design performs and scales as required.
Chapter 3
Programming Considerations

This chapter describes the performance aspects of the various programming techniques available to you when designing a Domino application. The chapter includes information about the languages to use and about the Domino objects you will need to manipulate. Techniques for sorting and searching are also covered in this chapter. In many places we include sections of sample code for illustration.

After reading this chapter you should have a better understanding of the many possibilities available to you when designing a Domino application. You will understand the performance implications of various language constructs you might use. You will better understand the relationships among the various Domino objects. In all, you will have a good footing to start designing your application.

Choosing a Language

As Domino has evolved, the choice of languages available to build applications has expanded. Originally only the Lotus Formula language was available. In Notes R4.0, LotusScript was added; in Domino R4.5, JavaScript and Java were both added to give you four choices of language. To add to this, Domino R4.5 also allows the use of HTML in some parts of forms. Languages external to the Domino database can also be used. Among the supported languages are C, C++, Perl, CGI, and so on.

So which one should you use? The answer depends on two factors. The first is whether you can even write the code in the language you choose. In some cases, only one of the languages will accomplish the task. For example, when coding the QueryOpen event in a form, you have a choice of using LotusScript or the Formula language, but for the database Initialize event you can only use LotusScript. If your application is being used on the Web, you also have JavaScript and Java as options in certain situations.

The second factor is performance. It turns out that under certain conditions, accomplishing the same task can take radically different amounts of time depending on the language you use.

Unfortunately, as with many performance considerations, there are no hard and fast rules. You need to assess each situation in context.
The Lotus Formula Language

The Lotus Formula language has been around for a long time and there are many professionals adept at using it. However, that doesn’t make it the correct choice in all cases.

The Formula language is built close to the core of Notes, thus there is little overhead involved when invoking Formulas. This means that simple tasks will execute very quickly. However, the Formula language has very limited capability when it comes to complex logic. There are no loops or subroutines in Formula, so writing complex logic becomes not only difficult, but inefficient.

The Formula language is very focused on the user interface. The current view, selected document or set of unprocessed documents play a very prominent role in Formulas. Most of the menu functions are available in the Formula language via @commands. You can use Formulas to access documents other than the current one, or access documents in another database, but the functionality in this area is limited.

Formulas can be very compact. It will usually take fewer lines of Formula than LotusScript to code simple functions. Also, Formulas are stored very efficiently in the database. When the same piece of code can be written in both languages with no performance impact, the Formula language is preferable to LotusScript because the code will take up less space. As we shall see throughout this book, database size is always a factor when it comes to application performance.

LotusScript

LotusScript is a full-fledged object-oriented programming language and has a much greater scope than the Formula language. One of the main advantages LotusScript offers over the Formula language is that it provides the means to do sophisticated iteration and branching. Not only can you include complex logic in your design, but you can use LotusScript to create libraries of subroutines that contain common code. These subroutines can be called from any other place in your application that uses LotusScript.

LotusScript lets you work with the object-oriented programming model. Most things in a Domino database can be accessed via the built-in Domino classes. These Domino classes are pre-loaded objects that can be manipulated through their methods, properties, and events.

Your application may be required to reach beyond the Domino sphere (such as accessing Word Pro, 1-2-3, a relational database, or the file system). LotusScript has the ability to do this using features which include LotusScript Extentions (LSX), OLE-Automation, Domino Enterprise Connectors, and the C API.
LotusScript or Formulas: Which is Faster?

Formulas that need only a single @function, such as @Command([FilePrint]), are more efficient and perform better than scripts that do the same thing. Formulas also perform better than scripts when you want to update more than 15 percent of the documents in the current database and the logic is simple enough to produce with @functions.

Scripts are often more efficient than complex formulas that use many @functions. LotusScript accesses documents more efficiently and quickly than a formula that uses @functions to “touch” many databases or documents.

The LotusScript user interface classes (the front-end classes) use the same underlying code as their equivalent @commands, so LotusScript won’t perform better than the formula language when you use these classes. However, the database classes (back-end classes) use different code and perform more quickly than the equivalent @functions.

Not everything can be done in either language. Sometimes one language has some functionality that the other cannot provide (for example, LotusScript can indicate if a document is marked as read or not when it is saved). This can affect your choice of language.

We cover more performance details in the section titled “Language Performance Aspects” later in this chapter.

Java

In terms of performance and functionality, Java and LotusScript are comparable when executing code that is written in the same way. There are some differences between the two languages in terms of architecture that may affect the performance of your application, depending on its design. Using Java with Domino gives you

- Multithreading
- Built-in networking support
- Data access though Java Database Connectivity (JDBC)
- Direct access to the Domino Object Model from a Web browser in Domino R5.0 using CORBA/IIOP
- Servlets

See the section “Agents and Other Processing” in Chapter 4, “Performance Aspects of Domino Design Elements,” for more discussion of the Java functionality in Domino.
If your program can utilize the multithreading support in Java, such as processing a large number of documents, it will perform faster than LotusScript which does not support multithreading.

Also, Java is a pure object oriented language: you have to structure your code into classes, which can be a big help for complex programs. While you can do object oriented programming with classes in LotusScript you are not forced to do it, and for complex programs you may end up with a lot of global variables (and more maintenance) to get your code to work.

**JavaScript**

JavaScript was introduced as an alternative in Domino R4.5 for use on the Web client. As of Domino R5.0, it can be used in Notes client events as well.

JavaScript is a good choice for field validations that need to be done on the Web. The script is embedded as part of the HTML page served to the client, and the code executes on the client as required. To code the same functionality in LotusScript or Formula requires a round trip to the server.

JavaScript can also be used to allow multiple buttons to exist on your Web page.

In general, a number of very useful Domino features are not available when the application runs on the Web. JavaScript can be used to fill in the gap by giving you more control over the appearance and functionality of your Web user interface.

**Use the Language You Are Most Familiar With**

This is a good choice when you are doing rapid development, and is often associated with the prototyping phase of the project. Often in prototyping, the full application logic is not included in the code, but only the visual details and the navigation are coded. Whenever you have a choice of language, use the one that can get the job done quickly. This will speed the production of the prototype.

**And the Winner is...**

The winner is Domino! With Domino, you will be able to use any combination of the available languages required to create an application that maximizes performance. You are not faced with a difficult choice at the start of the project, but instead, you can assess each situation as it arises. This way you have all the facts at your disposal so your choice will yield the best results.
Language Performance Aspects

This section details specific situations and language constructs relating to performance. Many times you will be able to get the job done in several ways. Knowing the alternatives, especially relating to performance, will enable you to design better applications.

@Formulas

The following general rules apply when programming with the Lotus Formula language. Remember that there are no absolutes — everything is a trade-off.

Use @ClientType instead of @UserRoles

From Domino R4.6 on, we have the ability to determine client type using the @ClientType function. This is a simpler calculation, and thus faster, than using @UserRoles.

Use Simple Hide When Conditions

As a designer, you have several options for writing Hide When conditions:

- Hide depending on client type
- Hide depending on Edit mode
- Hide depending on a formula

The first two are much more efficient than the last one.

If you need to hide depending on client type, use the Hide When properties, “Hide from: Notes R4.6 or later” and “Hide from: Web browsers,” instead of writing a formula based on @ClientType.
Likewise, if certain parts of the design need to always be hidden, don’t use a formula like “@True.” Use the Edit mode check boxes to indicate this condition. They are always faster.

![Hide paragraph from](image)

**Use Column Numbers Instead of Field Names**

When using @DbColumn or @DbLookup, using column numbers is faster than using field names. When you use a field name, Domino must compare the string you request against the list of all field names in the documents in the view. This is slower than going directly to the column specified by the column number.

The trade-off in referring to column numbers vs. field names is that it is harder to maintain. Any change to the order of columns in a view may require changes to @Db functions that use that view.

**Allow @Db Lookup Functions to Cache Data**

Use the Cache argument whenever possible when using the lookup formulas @DbColumn, @DbCommand, and @DbLookup. Cached data is served faster than non-cached data. Unless you have a real need to get a new copy of the lookup information each time it is used, allow Domino to cache the data.

Cached data is not related to one specific @Db lookup, but is related to the database. That means if one lookup retrieves data from the server (and is therefore saved in the cache), then another lookup in the same database which resolves to the same formula will use the cached data. The cache is cleared when the database closes. Conversely, two identical lookups in different databases will not share cached data.

**Note** When using @Db lookups in LotusScript via the Evaluate statement, no caching occurs, regardless of the cache option in effect.
Tip If you need to access the same set of lookup data from Formula and LotusScript, put the lookup information into a profile document.

Use Hidden Views for Lookups
When you need to access a large view for lookups, it may be better to create a hidden view with the minimal columns required to support the lookup.

Smaller views are faster views.

Two ways to minimize the data in the view are to use sub-strings, or to concatenate strings in one column.

Combine Data in One Column
If you need to access several columns in a view to populate several fields on a document, each lookup will take time. Combine values into a single hidden column in the view and use that column as a single lookup. Write a formula to parse the returned data into the appropriate fields.

Cache the Results of @Db Lookups
When the same set of data is required several times in a formula, don’t do the lookup each time it is needed. Instead, create a temporary variable for the lookup results. Set the temporary variable once, and then refer to the variable for the rest of the formula. Using fewer calculations provides better performance.

Use Alternate Search Methods
When processing small groups of documents (for example, less than 15 percent of all documents in the database), LotusScript can give better performance. Try using GetView, Search, and FTSearch methods in LotusScript instead of formulas.

Use Computed for Display Fields to Avoid Unnecessary Computations
Knowing what calculations are being done as your design executes will help you design code that performs better. There are a number of common situations where performance can suffer because unnecessary calculations are being done.
The following table lists what calculations occur for different events in a form:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Create</th>
<th>Open, empty</th>
<th>Open, data</th>
<th>Show dialog</th>
<th>Save</th>
<th>Refresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit (normal)</td>
<td>DV</td>
<td></td>
<td></td>
<td></td>
<td>IT, IV</td>
<td>IT, IV</td>
</tr>
<tr>
<td>Edit K-UF-DB</td>
<td>DV</td>
<td></td>
<td></td>
<td></td>
<td>IT, IV</td>
<td>IT, IV</td>
</tr>
<tr>
<td>Edit K-UF-Not DB</td>
<td>KF, DV</td>
<td>KF</td>
<td>KF</td>
<td></td>
<td>IT, IV</td>
<td>IT, IV</td>
</tr>
<tr>
<td>Computed</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computed for display</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computed when composed</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The legend for the above table is:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Normal edit field (not one of the two special ones that follow)</td>
</tr>
<tr>
<td>K-UF-DB</td>
<td>Keyword, Use formula for choices, dialog box</td>
</tr>
<tr>
<td>K-UF-Not DB</td>
<td>Keyword, Use formula for choices, check boxes or radio buttons</td>
</tr>
<tr>
<td>DV</td>
<td>Default value formula executes</td>
</tr>
<tr>
<td>IT</td>
<td>Input translation formula executes</td>
</tr>
<tr>
<td>IV</td>
<td>Input validation formula executes</td>
</tr>
<tr>
<td>KF</td>
<td>Keyword formula executes</td>
</tr>
<tr>
<td>V</td>
<td>Value formula executes</td>
</tr>
<tr>
<td>Create</td>
<td>Field is being created (either because the document is being created or an existing document is opened that does not contain the field)</td>
</tr>
<tr>
<td>Open</td>
<td>Opening an existing document — in either Read or Edit mode</td>
</tr>
<tr>
<td>empty</td>
<td>Field is empty (exists in document but has no contents)</td>
</tr>
<tr>
<td>data</td>
<td>Field has data</td>
</tr>
<tr>
<td>Show dialog</td>
<td>User presses the icon to display the dialog box</td>
</tr>
</tbody>
</table>

The table is much as expected; however, there are two interesting things that can seriously affect your design:

1. The keyword formula for an editable field which uses check boxes or radio buttons always runs when you open the document. This seems reasonable because all choices need to be displayed, but at the same time many users only want to view document contents, not change the data.
2. The keyword formula for an editable field which has data and uses a dialog box is run when a document opens. This one is less reasonable. For the many users who only want to view document contents, the data is already set so why run the keyword formula for choices? These users are not going to display the dialog box.
The running of the keyword formula at inappropriate times can be a serious performance issue when that formula does extensive work. This is particularly so if the formula does @Db lookups.

For the first situation described, you might think “Just hide the selections and display a read-only field, a computed for display version of the actual data.” Unfortunately, hiding the keyword field does not prevent the keyword formula from running.

The following solution, for the second situation, can be adapted to the first case also. This solution is to use an extra (hidden) multi-valued text field on the form, which holds the original keyword lookup formula. This field is used as the formula for the actual keyword field. The new field is computed for display, so the entire list is not stored with the document. Also, a computed for display field’s Value formula runs on document refresh so you have some control over when the formula runs.

To prevent unnecessary computations, the hidden computed for display field, kwDataHidden, contains the following Value formula. The code ensures that the @Db lookup only runs when the document is being created, is opened for edit, or is being refreshed:

```
REM "Do not run formula during doc save";
@if (@IsDocBeingSaved; @Return(""); "")
REM "Only run if doc is in edit mode";
@if (@IsDocBeingEdited; @DbColumn(""; ""; "ViewName"; 1); "")
```

In the PostModeChange event, you also need to trigger a document refresh. This means that when you change to Edit mode, the refresh sets the value in the kwDataHidden field.

```
@if (@IsDocBeingEdited; @Command([ViewRefreshFields]); "")
```

The formula for the keyword field is the name of the hidden field:

```
kwDataHidden
```
Finally, you need to select the “Refresh choices on document refresh” option in the keyword properties. This is required to refresh the keyword from the hidden field when the hidden field data is set during refresh:

![Image of keyword properties]

Using this technique, documents opened in Read mode suffer no keyword formula overhead until they are put into Edit mode.

**Use Computed When Composed Fields**
When fields are only updated at the back-end (via LotusScript), make these fields Computed When Composed. This speeds up any updates of the UI, since these fields will not recalculate.

**LotusScript**
LotusScript is very powerful and can be used to solve complex tasks. With power comes responsibility: you can easily write code that solves a problem but does so in a very inefficient manner. The following general rules apply when programming with LotusScript. These suggested techniques, which may not be obvious from reading the user guide, will help you create the most efficient code possible.

**Use For Loops Instead of Do Loops or While Loops**
For loops are faster than Do loops and While loops because the condition that bounds the Do or While loop must be evaluated at each iteration of the loop.

60 Performance Considerations for Domino Applications
In the following example, the first fragment runs 60 percent faster than the second fragment.

Fragment A

```
For y = 1 to 15000
    z = z+1
Next
```

Fragment B

```
Do
    z = z+1
    y = y+1
Loop While y <= 15000
```

The performance savings occur because the looping overhead is minimized in the For case. If your loop does a lot of work, the actual performance improvement will be less than the 60 percent mentioned since the looping overhead becomes a smaller fraction of the total loop time.

**Use Forall Instead of For in Referencing Array Elements**

Forall loops are faster in handling arrays than For loops.

Forall is 75 percent faster than For when referencing all elements in a 1-dimensional numeric array, and 50 percent faster when referencing elements in a 2-dimensional array. Similar performance improvements occur in arrays with higher numbers of dimensions.

String arrays achieve somewhat less performance improvement than numeric arrays, but these are still significant.

**Simplify If Statements**

LotusScript needs to evaluate the entire If statement before action is taken. Unlike some languages (like C) where evaluation stops as soon as a result is known, LotusScript will evaluate each condition.

This statement always results in both conditions being evaluated.

```
If a=x And b=y Then
```

The following statement runs 40 percent faster than the previous example because the second condition evaluates only if the first is true.

```
If a=x Then
    If b=y Then
```
Avoid Mixing Variable Types
Casting, or matching variable types in operations, slows down LotusScript performance. Where possible, avoid casting by performing operations with like variables. If this is impossible, use explicit casting rather than implied casting to improve performance.

The following example shows implied casting, where LotusScript has to convert data types to perform the operation. The implied casting slows this code down by 40 percent.

```
Sub initialize
    Dim a As Single
    Dim z As Long
    z = 3
    a = 1.0
    z = z + a
End Sub
```

Use Integer Division for Integer Results
Regular division, using the forward slash (/), always returns a float, even if the operands are integers. Use a backslash (\) for integer division where you do not need or want a floating point result.

In the following example, code fragment A is 60 percent faster than fragment B.

Fragment A
```
z& = x& \ y&
```

Fragment B
```
z& = x& / y&
```

Use Variant Types Only When Required
Because a variant can contain many different types of data, the LotusScript engine needs to interpret the type of data before operating on a variant variable. If you define your variables with explicit types, you will create faster code.

The effect of explicit type casting is particularly noticeable when performing integer operations, where an iteration with a declared integer can perform up to 200 percent faster than the same operation with a declared variant. The difference is less dramatic with string operations.
Avoid Reading Files One Line at a Time
File operations are faster if you read an entire document, then operate on individual records, rather than if you read the file one line at a time. In Domino R5.0, the LotusScript string variable limit has been increased to 2GB. The string limit in previous releases was 64K of memory (32K 2-byte characters).

In the following examples, code fragment A runs about 500 percent faster than code fragment B. The percentage improvement increases as file size increases.

Fragment A

```lsp
Open fName$ For Input As #fNum
buff$ = Input$(Lof(fNum), fNum)
stPos = 1
lineNo = 1
eoFile = False
While Not eoFile
    nlPos = Instr(stPos, buff$, Chr$(13))
    If (nlPos > 0) Then
        fData$(lineNo) = Mid$(buff$, stPos, nlPos-stPos)
        stPos = nlPos + 1
    Else
        fData$(lineNo) = Mid$(buff$, stPos)
        eoFile = True
    End If
    lineNo = lineNo + 1
Wend
Close #fNum
```
For Domino R4.6 and earlier, there is a 64K limit on strings. Reading the entire file at once can only be done if you know your files are 32,000 characters or less (the 64K limit applies to bytes, and there are 2 bytes per character).

If your files are larger than 32,000 characters, you can still use this performance advantage by processing the file in large chunks instead of line by line. Input chunks of 32,000 characters at a time. This does involve handling a line split across blocks, so it is a bit more code to maintain, but a 500 percent performance gain is worth it for large files.

Avoid Using Unnecessary Arrays

Storing data in arrays rather than in non-array variables can degrade performance up to 50 percent. If possible, avoid using arrays to store temporary or intermediate data.

In the following examples, code fragment A runs twice as fast as code fragment B.

Fragment A

```
For i = 1 to 100
    sum = sum + x(i)
Next

t(1) = sum
```

Fragment B

```
For i = 1 to 100
    t(1) = t(1) + x(i)
Next
```
**Note**  Domino R5.0 contains new array functions that may simplify array operations: ArrayAppend, ArrayGetIndex, ArrayReplace, and Fulltrim. Using these functions in place of multiple lines of manual code to perform the same purpose will be faster.

**Tip**  An alternative to using an array of strings is to create a single string from all of the strings in the array, using a delimiter like a carriage return. Then use Instr to search for a particular string.

**Optimize String Handling**
Excess string handling can be expensive. It is common to see string build-up such as:

```
x$ = x$ & "a"
```

This is expensive in terms of performance. Likewise, reducing strings by stripping the beginning or end is also expensive:

```
x$ = Right$(x$, currentLength% - lengthStripedOff%)
```

Instead, try to create strings from known pieces. In the following examples, fragment A is 25 percent faster than fragment B:

**Fragment A**

```
str$ = stringA$ & stringB$
```

**Fragment B**

```
str$ = stringA$
str$ = str$ & stringB$
```

When processing a string in pieces, extract the desired data from the middle of the string and remember your position. This is faster than reducing the original string each time. In the following example, fragment A is 85 percent faster than fragment B.

**Fragment A**

```
strPiece$ = Mid$(str$, startPos%, pieceLength%)
startPos% = startPos% + pieceLength%
```

**Fragment B**

```
strPiece$ = Left$(str$, pieceLength%)
strLength% = strLength% - pieceLength%
str$ = Right$(str$, strLength%)
```
Minimize the Use of Redim
Compute array size before declaration if possible. Don’t Redim in the loop that calculates the array size, but after the loop instead. Sometimes this means you need to process a set of data twice in this case, Redim in blocks (see the following section).

Re-dimension Arrays in Blocks
If you need to alter the size of an array repeatedly in a loop, alter it in blocks. In the following code fragments, fragment A is 20% faster than fragment B.

Fragment A

```
For i = 1 to 10000
    If (i > iMax) Then
        iMax = iMax + 100
        Redim Preserve sArray(1 To iMax)
    End If
    sArray(i) = ""
Next
```

Fragment B

```
For i = 1 to 10000
    Redim Preserve sArray(1 To i)
    sArray(i) = ""
Next
```

Note  In fragment A, when the loop is finished, you may want to Redim the final array to its actual size. Sometimes that is important in an application.

Use the Correct Sequence When Traversing n-Dimensional Arrays
Index the first bound of an n-dimensional array with the innermost loop and index the last bound with the outermost loop. In the following examples, code fragment A runs 400 percent faster than code fragment B.

Fragment A

```
For y = 0 to 2
    For q = 0 to 5000
        z = z+x(q,y)
    Next
Next
```
Fragment B

For q = 0 to 5000
    For y = 0 to 2
        z = z + x(q, y)
    Next
Next

Compare Integers Instead of Strings
Comparing strings is slower than comparing integers. Code fragment A is 50 percent faster than code fragment B.

Fragment A

    If (Asc(x$) = Asc("A")) Then

Fragment B

    If (Left$ (x$, 1) = "A") Then

Code fragment C is 30 percent faster than code fragment D.

Fragment C

    If (Asc(Mid$(x$, 1, 1) = "A")) Then

Fragment D

    If (Mid$(x, 1, 1) = "A") Then

In checking for a null string, code fragment E is 30 percent faster than code fragment F.

Fragment E

    If (Len (x$) = 0) Then

Fragment F

    If (x$ = ") Then

Use Automatic Conversion When Converting Numbers to Strings
The automatic conversion that happens when a text is concatenated with a number is, in fact, slightly more efficient than doing the manual conversion. The difference is small, so this is not something to change code for. It is mentioned here to allow designers the freedom to use either technique, knowing they are not affecting performance if they prefer one style over another. Fragment A is slightly faster than fragment B.

Fragment A

    s2$ = "Text " & ii
Fragment B

\[ s2$ = "Text $ & Cstr(ii\%)
\]

**Note** In fragment A, when using automatic conversion, you cannot add a
% suffix to indicate integer type (since the syntax wants string types).

**Avoid Overuse of Environment Variables**

Each access to an environment variable causes I/O to the NOTES.INI file.
This can have a performance impact if used to excess.

As an alternative, save persistent data in profile documents.

When environment variables must be used, combine all your information
into a single environment variable instead of using many variables. Use
predetermined delimiters to separate your data. The cost of the I/O itself
when accessing the NOTES.INI file is far greater than the amount of data
retrieved during that I/O.

During testing, it was determined that performance savings are proportional
to the number of variables you combine into one. Saving five environment
variables individually took five times longer than saving the five values
together as a single variable. Likewise, saving two took twice as long, and
saving 10 took 10 times as long.

**Java**

Java has an inherent language overhead caused by the need to load the Java
virtual machine (JVM). This is a piece of software used to drive Java code; it
is loaded the first time Java is run on your machine. Once loaded, the JVM
stays resident for the process that triggered it.

On the server, the JVM is loaded with the HTTP task. As such, the overhead
occurs when the Domino server starts up. For Notes clients, the JVM also
loads with the Notes client. Again, the client opens a bit slower, but the
JVM is available from then on. Lastly, on the Web browser, the JVM is only
loaded when Java is first required, but once loaded, the JVM is available for
the rest of the browser session.

Although there is nothing you can do about the JVM loading overhead, there
are some things you can do to maximize the performance of your Java code:

**Put Your Java Classes Near the Top of the List**

If you put the Java classes that are loaded frequently towards the top of the
class list (as soon as possible after the base class), you will improve
performance.

This technique allows the virtual machine to find your classes quicker.

**Note** The base class must appear first in the list of classes.
Call the Recycle Function Periodically
This function will free memory used by unreferenced Domino objects. The LotusScript interface performs this task automatically but Java does not.

An unreferenced object is any object that does not have a variable or another object that refers to it. Unreferenced objects can result when an object variable goes out of scope or is assigned to another object.

In LotusScript, objects that become unreferenced are immediately destroyed. While Java as a language does have its own “garbage collection,” this method does not clear the memory used by the underlying Domino objects.

JavaScript
JavaScript is in a league of its own. You cannot easily compare it to LotusScript or Java since they don’t operate in the same space. But that’s not going to stop us from providing some details on using JavaScript to help performance.

JavaScript is coded into its own section on an HTML document, between <SCRIPT> tags. Therefore, the size of the data sent to the Web client is a bit larger when using JavaScript. However, with JavaScript you gain access to elements of the UI that are not exposed to LotusScript (such as the background color of the screen and background images). These can be used in the Notes client as well (for Domino R5.0).

Two key areas in which JavaScript can excel are field validation and field input translation. For documents that are seen on the Web, field validation and translation pose a problem for the client. They require a round trip to the server to be executed. This is where JavaScript can really help.

If JavaScript is used for areas in a form where otherwise a round trip to the server would be required, then the reduction of network access will immediately be seen by the user in a positive way. The actual time the CPU is busy may increase a tiny bit, but the network wait time is drastically reduced, giving an overall performance benefit.

The Domino Object Model
This section details a number of things to consider during your design when dealing with Domino objects. They are stated in a “rule” format and cover known areas of performance limitations.

Chapter 3: Programming Considerations 69
In many cases a rule might say something like “Don’t do X.” You may find yourself, however, in a situation where you have no other choice. That’s okay because we cannot cover every possibility in a book like this and your situation may be unusual. Remember that there are no absolutes — everything is a trade-off.

Although the examples here will be written using LotusScript, the calls to the Domino object classes use the Domino object interface (DOI). This is the underlying code that is called when manipulating Domino objects, such as databases, views, and documents. Thus if you are using Java or JavaScript to access Domino classes through the DOI, the performance implications will be comparable.

Maximize Resource Usage

The techniques of lazy initialization and object reuse are two key ways to maximize resource usage. They are effective when dealing with complex objects, not only the Domino objects mentioned in this section, but also objects you create yourself.

A basic concept that was at the forefront of programmers’ minds in the earlier days of application design concerned resource constraint. Now the emphasis may be on different resources, but the rules haven’t changed. Maximizing performance means not wasting resources if we don’t need to. When dealing with Domino objects, this boils down to two rules:

1. Don’t create objects unnecessarily.
2. Once you do have an object, reuse it in other parts of the code — don’t recreate it.

Lazy Initialization

Many times in your design it will be necessary to have access to a complex object (for example, a database or view). That database or view may not be required each time the function is called; its use may depend on various conditions. In such situations, use lazy initialization to defer the overhead of opening a database or getting a view until these objects are needed.

If the code is contained within a single module, this can be easily achieved by not defining the object until the appropriate place in the logic.

If, on the other hand, you have structured code with multiple functions and are using global variables for shared objects, then the initialization of these objects is often done by a common initialize function. Usually this means that all the objects are initialized, even if some of them are never needed.
The following example shows one way to implement lazy initialization for a view that is only required to validate one type of customer:

```vba
'*** Globals
'*** Declare the global data we share
Public gDb As NotesDatabase
Public gVwCustomerType As NotesView

'*** Functions
Sub Initialize
'*** Set up the database (NOT lazy)
Dim ss As New NotesSession
Set gDb = ss.CurrentDatabase
End Sub

'*** Get the customer type view (lazy initialization)
Function GetCustomerTypeView() As NotesView
    '*** Only get it the 1st time it is used
    If (gVwCustomerType Is Nothing) Then
        Set gVwCustomerType = gDb.GetView(“Customer Type”)
    End If
    Set GetCustomerTypeView = gVwCustomerType
End Function

'*** Do account validation
Function ValidateAccount(custName As String, accountType As String) As Integer
    '*** Return True if account is valid
    Dim vw As NotesView
    Dim doc As NotesDocument
    Dim aKeys(0 To 1) As String
    Select Case accountType
        Case “New”
            '*** New accounts are always valid
```

Chapter 3: Programming Considerations  71
ValidateAccount% = True

Case Else

'*** Is customer allowed the specified account type?

Set vw = GetCustomerTypeView()
aKeys(0) = custName$
aKeys(1) = accountType$

Set doc = vw.GetDocumentByKey(aKeys, True)

If (doc Is Nothing) Then
    ValidateAccount% = False
Else
    ValidateAccount% = True
End If

End Select

End Function

The ValidateAccount function is called whenever required to ensure data is valid. When we need to validate the customer type, instead of getting the required view via the standard “GetView” method, we call the GetCustomerTypeView function. It is this function’s job to return the Customer Type view.

Now we have one location where we can put in some logic about fetching the view the first time it is accessed and then saving this view in a global variable for later reuse. Subsequent calls to the function simply return the global data.

Object Reuse
In the prior example, you may also notice that once the view has been retrieved from the database, it becomes available for future requests for that view. The view is saved in a global variable and is available for later reuse. This is true within the context of the running code (for example, within the current document). Once the document closes (if that is your context), then the objects are destroyed.
Getting the Most from Your Domino Objects

The principals discussed here highlight some effective ways to use Domino objects efficiently. Don’t memorize the rules and then use them inflexibly; instead, try to understand the underlying reason why performance is affected. For example, if opening a document has performance implications because the design of the form must be retrieved from the server, then other areas where design information is required from the server may also be suspect.

Here are some basic rules to follow when using the Domino object model.

Minimize the Number of Times You Open a Database

When opening another database, the Domino Object Interface has to do several things. These include the following tasks:

1. Create an object in memory for the database. This involves memory allocation and initialization.
2. Locate the database. This may involve directory searches or access to another server.
3. Initialize some object properties. Things like the ACL will have to be read from the database to determine your access.

Since this entire operation can be costly, you need to minimize the number of times a database is opened. If the database is needed by several functions in the code, pass the database between functions as a parameter or keep it as a global variable, instead of having each function open the database separately.

Use Existing Objects

If you already have a reference to an object, don’t create a new copy of the object unnecessarily. In many situations, a related object is available in the properties of an object. For example, fragment A is 10 percent faster than fragment B.

Fragment A

\[ \text{dbTitle$} = \text{doc.ParentDatabase.Title} \]

Fragment B

\[ \text{Set session = New NotesSession} \]
\[ \text{Set db = session.CurrentDatabase} \]
\[ \text{dbTitle$} = \text{db.Title} \]

Use Back-end Classes instead of Front-end Classes

Back-end classes do not need to manage or process UI effects, and thus are faster than front-end classes.
For example, avoid using the front-end NotesUIDocument class to do many field updates. The back-end NotesDocument class is much faster and allows you to assign data types (including rich text) and to add new (hidden) fields. The front-end class allows you to update only fields that already exist on the form and it allows you to insert only text in the field.

**Disable AutoReload When Using the NotesUIDocument Class**

The AutoReload option, which is on by default, will update a NotesUIDocument object every time a change is made to the corresponding back-end NotesDocument object. If you are making many changes to a NotesDocument object, turn AutoReload off to eliminate the extra processing required to update the corresponding NotesUIDocument object.

Before your script does any updates, use the following statement to disable AutoReload:

```log
source.AutoReload = False
```

After you disable AutoReload, it remains disabled until your script enables it again using this syntax. To display changes, you also have to reload the changes to the UI:

```log
source.AutoReload = True
source.Reload
```

**Use ColumnValues, Not Field Names**

When documents are accessed from a view, the ColumnValues property of the document contains the information that is displayed in the view.

Using the ColumnValues property is about 10 percent faster than getting a handle to a document and using the extended class syntax. This is because column information is available directly in the view’s index, but document fields can only be accessed by opening the document.

The ColumnValues property returns an array of that document’s column values in the document’s parent view. Code fragment A is about 10 percent faster than fragment B.

**Fragment A**

```log
var = doc.ColumnValues(columnNumber%)
x = var(0)
```

**Fragment B**

```log
x = doc.fieldname
```

**Note** Be sure to allow for columns that can have multiple values in them. In this case the data type of doc.ColumnValues(0) will be an array, not a scalar.
Note Not all columns participate in column numbering. For example, if a column contains the document size (using the Size (bytes) simple function), then that column is not included in the ColumnValues array.

Use GetNextDocument For Document Collections
GetNthDocument is designed to retrieve single documents from a known position in a collection. When accessing item ‘n’ in a document collection, Domino needs to start from the beginning of the collection each time. If you are iterating through a collection one by one, GetNextDocument is more efficient.

Fragment A is significantly faster than fragment B.

Fragment A

```vbnet
Set doc = collection.GetFirstDocument
While Not doc Is Nothing
    Set doc = collection.GetNextDocument(doc)
Wend
```

Fragment B

```vbnet
For x = 1 to collection.Count
    Set doc = collection.GetNthDocument(x)
Next x
```

Note This does not apply to GetNthDocument in a view. In that case, both methods are equally fast.

Use Search or FTSearch instead of GetView
The NotesDatabase Search and FTSearch methods can be very fast compared to GetView when accessing a small number of documents in a database. This “small number” is on the order of 1 to 15 percent of documents in a database. When the number of documents affected becomes greater than 15 percent, the GetView technique will usually be a better performer.

Instantiating a view object and opening a document from a view keeps summary information resident in memory, which is overhead that can decrease performance when using GetView.

You can use the existing view selection formulas from your views as the formula parameter for the Search method.

When choosing Search over GetView, you need to consider the environment that your application is deployed in. Search will put a load on your server while the client waits. GetView puts more work on the client, but needs to send design summary information to the client before the client can do the work. For low bandwidth situations, this can have a detrimental effect on overall performance.
Minimize the Use of GetView
The GetView method can generate a lot of network traffic.

The DOI uses Note ID as the design element identifier. When the Notes client requests a view by name, the DOI first needs to determine the Note ID of the view before it can request that view from the server. The DOI determines the Note ID by scanning the design summary for the view (the design summary contains the design element names, Note IDs, and other information). Unfortunately, the design summary is not cached at the client. Thus each time a GetView is issued by the client, the DOI has to request the design summary first, scan the summary for the view’s Note ID, and then request the view by Note ID.

For databases with a complex design, the design summary can be 100KB or more. Combine this with a low bandwidth environment and you have a performance problem.

Use GetDocumentByUNID
The NotesDatabase GetDocumentByUNID method is the most efficient way to access documents. In most situations where you need to access a document, you know something about the document in order to find it. This information comes from some common data between related documents.

Design your system such that the document unique ID is used as common data. If this is available when it is time to locate another document, then finding that document becomes very easy — and very fast.

Note There are limitations to using document unique ID as a link between related documents. In particular, when archiving data to another database, there is no guarantee that unique IDs are maintained during the copy. Integrity checks will be required to ensure links in the archive database are maintained.

Extended Syntax Comes at a Cost
The facility to access document fields via the extended syntax makes it easy for the designer to get at document contents. However, this does not come for free. Each time the extended syntax is used, Domino must locate that field among the other fields in the document before the item becomes available. If you reference a field more than once, this will impact performance since the work of finding the field needs to be done each time.

When accessing a field more than once, get the item once and cache it in a NotesItem variable. Then reference the cached item in your logic.

Fragment A runs 10 percent faster than fragment B.
Fragment A

```vba
Set itm = doc.GetFirstItem("fieldA")
If (itm.Value > 0) then
    pSum = pSum + itm.Value
else
    nSum = nSum - itm.Value
End If
```

Fragment B

```vba
If (doc.fieldA(0) > 0) then
    pSum = pSum + doc.fieldA(0)
else
    nSum = nSum - doc.fieldA(0)
End If
```

**Use Available Information to Locate Documents**

When you are trying to locate a document and you have the selection criteria, don’t just scan through a view of all documents to find the one you’re after. Instead, use GetDocumentByKey or GetAllDocumentsByKey to locate the document straight away. If there is no existing view with the correct selection formula, create a hidden view with an appropriate key.

Sometimes a view to solve your problem cannot be designed. In that case use Search or FTSearch to locate the document.

**Avoid Using AppendToTextList to Collect Data**

For multi-valued fields, each new entry in the list takes incrementally longer to add than the previous entry. As the list grows longer, these additions take longer and longer. Access to items farther down in the field also take longer.

If possible, use multiple fields to store the data instead of a single field. The complexity of accessing the data is increased, but in situations where there is a lot of access to this data, the performance improvements can be enormous.

Code fragment A runs much faster than code fragment B.
Fragment A

```
fldNo = 0
fldData = “CN=Walt G Simons/OU=Australia/O=IBM”
For ii = 0 To loopMax-1
  ‘*** Create a new field every 10 appends
  If ((ii Mod 10) = 0) Then
    fldNo = fldNo + 1
    fldName = “field” & Format(fldNo, “000”)  
    Set itm = doc.AppendItemValue(fldName, fldData)
  Else
    Call itm.AppendToTextList(fldData)
  End If
Next
```

Fragment B

```
fldData = “CN=Walt G Simons/OU=Australia/O=IBM”
For ii = 0 To loopMax-1
  ‘*** Create a single field
  If (ii = 0) Then
    fldName = “field001”
    Set itm = doc.AppendItemValue(fldName, fldData)
  Else
    Call itm.AppendToTextList(fldData)
  End If
Next
```

The following table shows the savings with different numbers of items in the list:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.109 sec</td>
<td>0.476 sec</td>
<td>3,766</td>
<td>3,748</td>
</tr>
<tr>
<td>200</td>
<td>0.188 sec</td>
<td>2.396 sec</td>
<td>7,486</td>
<td>7,448</td>
</tr>
<tr>
<td>400</td>
<td>0.387 sec</td>
<td>17.028 sec</td>
<td>14,926</td>
<td>14,848</td>
</tr>
</tbody>
</table>

As you can see, with the ‘A’ method execution times go up linearly with the number of items in the list. With the ‘B’ method there is an exponential growth of time to add each entry. Although the ‘A’ method makes documents that are larger, the size difference is almost insignificant.
Use the Append Item Option For New Documents
When you want to copy all fields from one document to another and you know the destination document has no fields matching the source, set the second parameter to False. This appends to existing fields. Replacing fields is more time consuming since the extra step of finding and removing the fields must be done (even if the field does not exist, it must be searched for). Fragment A is 15 percent faster than fragment B.

Fragment A

```csharp
Call docSource.CopyAllItems(docDest, False)
```

Fragment B

```csharp
Call docSource.CopyAllItems(docDest, True)
```

Be Careful When Using Reader Names Fields
Reader names fields are a very good way to place security on your documents. Domino will protect the privacy of documents with reader names fields so that only authorized users can access the data.

However, reader names can play havoc with performance in certain situations. See Appendix C-3, “Fast Access to Reader Names Protected Documents” for a detailed discussion.

UnprocessedDocuments
The UnprocessedDocuments property can be effectively used to select a subset of documents for an agent. Documents which are processed by one agent do not affect their status as being “unprocessed” for another agent. Using this property will eliminate the need for other selection criteria for the agent (for example, it may eliminate the need for a hidden view). Do not forget to mark the document as processed for your agent by calling the UpdateProcessedDoc method.

Code Reuse
The reuse of code is not generally seen as a performance improving technique. Certainly, if you reuse an existing sort routine that is known to be fast, then your application will not suffer from slow sorts. But in general, reusing code usually involves a very slight performance degradation. This degradation occurs because access to the existing code is via an interface between the application and the reused code. You don’t have the option of accessing the function directly.

Advantages of reusing code are usually seen in reduced cycle time, not faster performing applications. However, in this section, we will show how reusing code can improve Domino application performance.
What Can Be Reused?

When using the Formula language there is minimal scope for code reuse. The only techniques for reusing code are to use subforms, shared fields, agents and, for Domino R5.0, shared actions. The various formulas in these design elements can be maintained in one location and used by many forms or views.

When a complex formula is required in multiple forms, using a subform to allow central control of the code can prove beneficial. Likewise, if similar actions are required from multiple views, agents can be used to centralize the action formula.

Unfortunately, these methods of reuse, while reducing cycle time, do not always improve the performance of the application.

Reuse Via Subforms

Depending on your situation, using subforms for code reuse can also provide performance gains.

In general, using subforms results in performance degradation. This is because Domino needs to resolve the form first and then locate and resolve the subform. Accessing two design items for a large database will always take longer than accessing only one. However, in a limited bandwidth environment, we have to consider the messages sent between the server and the client as well.

For Notes clients, subforms are cached at the client. If the amount of design in the subform is nontrivial, and the subform is used in several forms, then gains made by sending less form information for subsequent forms can outweigh the extra design item resolution. In this case we have a trade-off between server work and network time.

Note  A portion of the subform is duplicated in the form design (automatically, when you save the form). Depending on the complexity of your subform, this duplication can range from 20 percent to 40 percent of the subform size.

Reuse Via Agents

When complex formulas are behind action buttons in views, this code can be put in an agent. That agent is then called from the view action button using:

```
@Command([ToolsRunMacro]; "Agent Name")
```

The only potential performance gain here is when there are multiple complex actions in the view. In this case, coding the formula in the view means the entire suite of actions is sent to the client when any action is taken. When agents are used, only the formula for the single action taken is loaded to the client.
Reuse Via LotusScript
There is a lot more scope for reuse when using LotusScript. Not only can you achieve reduced cycle times, but LotusScript reuse can improve performance in certain circumstances.

When using LotusScript, there are several ways to write code so that it can be used in more than one situation. These include:

- Script libraries
- Agents
- %INCLUDE files
- External programs (DLLs)

Script Libraries
Ever since Domino R4.5, script libraries have been available to help a developer in a variety of ways. By putting common code into script libraries, it is possible to use this code in forms, view actions, or agents without the need to duplicate the code. Script libraries can also help improve performance under some conditions.

Script Libraries Are Cached
There are several design items that are cached at the Notes client. Forms, subforms, and script libraries are cached at the Notes client in the CACHE.DSK file. The size of the cache can be seen in the Workspace Properties box on the second tab. If you find that the value is too low, change it (5MB is the default).

Since script libraries are cached, you gain performance improvements whenever several related forms are used that share one or more common script libraries. The initial use of a script library involves the uploading of the script library executable code from the server to the client; however, once at the client, the script library code is cached for future use. All subsequent uses of the same form will reuse information from the cache, not only for the form information, but also for the script library information.

If another form that uses the same script library is required, the new form must be uploaded to the client, but the script library does not need to be uploaded since it is already in the cache.

There is a performance overhead in using script libraries, but this is outweighed by the gains in network traffic, especially if your network is slow.
Agents

Many developers think of agents as things that run on the server. In fact, much discussion about agents never mentions any other way for them to run. The truth is that running agents on the client can be a novel way to improve application performance.

Agents are a useful way of deferring code loading until required. In many situations, complex code for validation or status changes is not required except in a few situations. Why, then, does the form load all this complex logic if it most likely will not be used? The way Notes works, all code related to a form is loaded when the form loads. This includes subforms, shared fields, and script libraries. Although the code is cached for future usage of the form, it still needs to be transmitted from server to client the first time it is used. For applications where a document is read much more often than it is updated, this data download to the client is an unnecessary waste of network resources.

One way around this it to use agents to defer loading parts of the code. When a user clicks an action or saves the document, it is possible to put code in the QuerySave event or behind the action button which runs an agent on the client as follows:

```plaintext
@Command([ToolsRunMacro]; "Agent Name")
```

The code in this agent does not get loaded to the client until the action button is clicked by the user. At that point the agent code is downloaded to the client and executed.

Agents Are Not Cached

One negative thing about using agents is that they are not cached at the client. If the user performs an action on several documents, the form code is cached but the agent code must be retrieved from the server for each action.

Best of Both Worlds

If your application has network constraints, then reloading agents for each action or document save is not what you want. However, agents can use script libraries.

Write your code in the agent as a single call to a function in a script library. The amount of code in the agent is reduced to two lines — the ‘Use’ statement and the call to the function (and your comments, of course).

The advantages in this approach are that the script library contains the code and it is cached once it’s been loaded to the client. In fact, if the same function in a script library is used by another agent, the code is already at the client.
If you design a form that offloads most of the work to agents, and you write these agents to call functions in script libraries, then you’ve minimized the amount of design information required at the client. Any document opened with that form will have a smaller network delay for design download than if all the code was in the form itself. Many users (who only view the document) will never experience any more network delay for design download. Those few users who need to process the data will encounter another delay when an action is performed. However, the user will not perceive this design download as significant. Since the user expects a delay when ‘processing’ data, the extra network delay is disguised as data processing.

The cost to design your forms with this kind of script library access is not negligible, but it is also not prohibitive. There is really only the extra agent layer involved. Instead of calling script library functions directly, the form must call an agent which in turn calls the script library. On the other hand, you do get the benefit of structured code. Your functions are well defined in the script library, and coding of forms independent to script libraries can help spread the load during development.

%INCLUDE Files

%INCLUDE files are another way to write and maintain code in one place but use it in many places. Since the introduction of script libraries and the removal of the 64KB limit in many areas, the need to use include files is not as great as it once was. However, one may ask the question, “Is it quicker to load a form that includes %INCLUDE statements for some of the code, or a form with all its code ‘hard-coded’?” Another question you may ask is, “Should I convert all my old applications that use include files and put that code directly into the form (or into a script library)?”

The answer is that you should not convert your %INCLUDE files to different techniques if you have no other reason to. You may decide to convert so that the code is all in one place, or for some other reason, but if you are happy with the current situation — don’t change. The loading happens only during initial code usage. The savings are small since there is no optimizing of a repetitive loop, where many performance gains are made.

There is only one situation where there may be a performance impact. If the code is large and you have a slow network, then using %INCLUDE files sends less information on the network than either hard coding in the form or using a script library. The amount sent relates directly to the size of the code in the %INCLUDE files, but you need to be aware of this if you are thinking of changing.
External Programs

The use of external programs that interface with a Domino application is another possible way to tackle a performance problem. Performance is not the reason most DLLs are written for Domino applications: usually the application is in need of functionality that just cannot be achieved any other way. However, having an external program that has full access to the operating system can sometimes be a good solution to an otherwise poorly performing application.

Dynamic Script Library Loading

Script libraries are a good way of building up code libraries for use by many parts of your application. These libraries have also made it much easier to enable object-oriented designs.

Without boring you with a lesson in object orientation, suffice it to say that it is very common for classes in an object-oriented design to be written in script libraries. For ease of development, it is also common that one script library contains one class.

The technique of using one script library per class quickly becomes a headache for the development team, however, when your object model becomes complex. The team starts to notice lengthy compile times when saving their scripts. If they get as far as testing, they notice longer load times for script libraries as well. By far the worst scenario, however, is circular references among classes. When this happens, the script libraries will not even compile.

Although not a perfect solution, there is a technique of dynamic loading of script libraries that can help. It helps the development team with quicker compiles and the elimination of circular references, but better than that, it improves the performance of the application as well. This technique is described in detail in Appendix B-2, “Dynamic Script Library Loading.”

Sorting

Sorting is not a common task for Domino applications. Most often a view is used to maintain the correct sequence for the data. However, when you are faced with a situation where you need to sort data and a view cannot be used, how should you go about it? Neither Formula nor LotusScript supplies convenient sorting routines for the designer’s use. You will need to code one yourself.
The main problem with sorting is understanding your data. If you know you will always have 10 items or less, then you can use a simple sort via brute force. Many times, however, you will not be able to guarantee how your application will be used, so the data set being sorted will not always have limits. The actual volume of sort data may be wildly outside your expectations; the size of each piece of data may be quite a bit different than planned. Since you don’t want your application to break under any circumstances, it is usually wise to insert a robust sort that will scale as your application data grows.

Appendix B-4, “Sorting,” is dedicated to several sorting options and routines. Consult it when dealing with any sorting problems.

Asynchronous Processing

Another technique to improve performance is asynchronous processing. In particular, when dealing with complex processing or multi-database relations, the user’s perceived response time can be greatly improved when asynchronous processing is employed.

Using Asynchronous Processing

A common term that is applied to asynchronous processing is Batch Processing. Although this is not altogether wrong, it is not exactly correct either. Regardless of the words used, the concept is the same: Processing is deferred until a batch of data becomes available.

How Do You Do It?

In Domino applications, the most common technique employed for asynchronous processing is to use scheduled agents. The part of the application which runs on the user client does just enough processing to capture the required information and then allows the client to continue with other work (the mouse stops being busy). Further processing is still required, but this is left to the scheduled agent.

The agent can use several techniques to determine which documents need to be processed:

- The database Unprocessed documents can be processed.
- A database search can be done and the results processed.
- A hidden view can be processed.

The actual method used is dependent on the design of the application.
Design Asynchronous Processing Early
Decide early that you will use asynchronous processing in your application. When you get as far as testing and find that performance is bad, a switch to using asynchronous processing is not a trivial task. Splitting up an application design so that the user experience and the application effectiveness are both maximized takes planning.

Reap the Benefits
A well-designed application that uses asynchronous processing can solve many problems which commonly affect ‘normal’ applications. The user benefits by having a responsive user interface. The owner benefits because the users are more productive. The server administrators benefit because their server loads can be better distributed. And lastly, the company benefits because their hardware investment is smaller.

Summary
After reading this chapter, you should have a better understanding of the many possibilities available to you when designing a Domino application. You understand the performance implications of various language constructs you might use and the relationships among the various Domino objects. In all, you now should have a good footing to start designing your application.

The next chapter deals in greater detail with the individual design elements and how you can get the most from each one.
Chapter 4
Performance Aspects of Domino Design Elements

This chapter describes the performance aspects of the various design elements at your disposal when designing a Domino application. The chapter includes information about the Notes client and the Web, with a separate section on Domino R5.0 enhancements. In many places we include sections of sample code for illustration.

Topics covered in this section include:

- Forms
- Profile Documents
- Profile Information for Web Browsers
- Views
- Pages
- Framesets
- Outlines
- Resources
- Headlines
- Agents
- Database Performance Techniques

After reading this chapter you should have a good foundation from which to build better performing Domino applications. You will understand the tradeoffs you must consider when selecting one component over another.

The Basics

Ever since Domino R4.5, the scope of concerns for designers of Domino applications has been increasing. You can no longer just worry about how an application performs in your nice little Notes environment. The Web is out there and your application should be available to those users too. Add to that your users who are still on Notes R4.1, and your design choices become complex. You have to always consider users of your application who have platforms with limited capability, who are using the Web to access the application, or who have limited network bandwidth.
Ask most Domino designers about the basic design elements for designing a Domino application and their reply will be “Forms and Views.” The realization that Outlines, Frames and Pages are now with us is still filtering down to the grass roots. These new design features can give us a lot of wonderful functionality — especially for Web based applications. Unfortunately, performance-related issues generally become known by fixing applications that are in trouble. Many aspects of the new Domino R5.0 design elements are not well understood when it comes to their performance in a real user community. For these cases, we have done some limited tests and we provide some advice. Most of this advice has not been field tested; we are relying on your experiences in that area!

---

**Designing for Maximum Performance Using Forms**

Forms are the workhorse of any Domino application. There may be many navigational aides that allow you to locate the data you need, but it is through forms that you can finally see the data and make appropriate changes.

Forms provide the ability to programmatically calculate content at many different levels, right down to individual paragraphs and fields. You can visually stimulate the user in a variety of ways, through use of graphics, fields, sections, tables, layout regions and more. The suggestions below embody some general themes that will help guide you in designing your forms for maximum performance:

- Every calculation required (through formulas, subforms and so on) reduces performance
- The more bytes that must go from server to client, the slower the page
- Cached data is served faster than non-cached data

In this section we discuss what can be done when designing forms to maximize your chances that the form will perform.

**Using Graphical Objects**

Limit the use of graphical banners, hotspots, images, logos, graphical separators and the like. Forms take longer to open when Notes is required to display these graphical objects. Not only does the UI need to display a complex image, but the image itself must be sent from server to client before anything can happen.

---

88 Performance Considerations for Domino Applications
Alternatives:

- Substitute equivalents to graphical objects (for example, use dashed lines instead of a graphical separator bar)
- Make use of caching facilities (by making the graphic a shared resource, in Domino R5.0)
- Store images on the file system for HTTP (this will enable the modified date to be used and will permit Web Client Caching)

**Bitmap Scaling (Aspect Ratios)**

A bitmap on a form has a property called Scaling (or Aspect Ratio before Domino R5.0), which determines how much the bitmap should be enlarged or reduced, horizontally or vertically. If you use a scaling factor of other than 100%, you’ll see a noticeable effect because this will force Notes to recalculate the image every time a document is opened. This performance impact is especially noticeable for larger bitmaps. To prevent this performance impact, save the bitmap at the exact size required in your graphics tool.

The first tab in the Picture Properties box should be used to ensure your scaling is 100%:

![Picture Properties Box](image)

**Using Subforms**

Subforms are very useful when you are developing an application, in terms of reuse, consistent look and feel, and design task separation. Loading subforms, however, requires calculations which slow the opening of documents to some degree.
**Extra Design Information**

When a form uses a subform, Domino must not only locate the form when displaying the document to a user, but must also locate the subform. Finding two design elements takes longer than finding a single element.

The form and the subform must be sent to the client. This involves two server-client transactions instead of one. Likewise, the design size of the form plus the subform is bigger than if you had coded it all into the form. This size increase is not just an extra design element overhead, but instead the visual details of the subform (the $Body details) are duplicated in the form visual details (Domino does this automatically when you save the form). You can achieve some savings by using computed subforms. In this case, the duplicate subform information is not stored in the form.

**Subforms Are Cached**

Since subforms are cached, there is potential for performance *improvements* as well with subforms. The first form that uses a subform will be hit with some degradation as mentioned before, but subsequent forms may load quicker since the subform is already at the client.

**Merging Subforms Back Into Forms**

Subforms are great for development, but to maximize performance you should embed subforms into their forms when promoting to production.

If we compare using a non-computed subform with putting the same elements in a form directly, there is usually a clear performance impact. Using computed subforms has an even bigger performance impact, so you should try and avoid them.

To merge the subform back into the form, follow these steps:

1. Edit the subform.
2. Select the entire subform (Select all) and copy to the clipboard.
3. Edit the form.
4. Delete the subform from your form.
5. Paste in the clipboard version.
6. Save the form.
7. Repeat Steps 3 to 6 for other forms that use the subform.
8. Remove the subform from the design.

If you decide to use this technique, make sure to retain the “master” version of your design before you merge the subforms. Then, when there is further development, this master version can utilize the subforms. You will need to perform the merge activity each time a production copy of the application is required.
Using Tables

In addition to their common usage for displaying tabular data, tables are used quite frequently for their alignment and Hide When capabilities.

Size and Number of Tables

Avoid using large tables with many fields unless you are sure that in your environment the performance is not impacted.

The use of many tables or tables containing many cells can affect the time it takes Notes to build the form for viewing, editing, or creating. The performance impact is the same whether or not the table cells are hidden.

Where performance is to be optimized, tables add to the overhead needed for Notes to build the form.

Performance-enhancing alternatives:

- Limit the use of tables to function rather than aesthetics
- Format rich text areas using tabs to align data (Notes client only)
- Consider using a computed subform to bring in more tables, only when needed
- Use the technique of putting a table into a rich text field. New rows are then added as needed, only when the user tabs in the last cell.

Note Depending on your user’s skill set, this may not be appropriate since a higher level of table skills is required.

The following figure shows some performance measurements to consider when you are creating a table.

Table Considerations

As suggested by this data, you should avoid large tables or, when it is possible, split a large table into several smaller ones.

Note: 5 10x30 tables take 50 seconds to compose. 150x30 table takes 115 seconds.

Chapter 4: Performance Aspects of Domino Design Elements 91
**Domino R5.0 Table Enhancements**

Domino R5.0 allows extra table features such as tabbed tables and timed tables. We did a variety of tests and examined the Remote Procedure Calls (RPCs). For Notes clients, we found that all the design information is sent to the client when you open the document, regardless of whether it is a tabbed table or a timed table. The performance impact is negligible.

For Web clients, however, the use of a tabbed table did provide a significant performance improvement, as only one tab of the table was sent at a time.

**Using Sections**

Sections can be used to display just the major parts of a document, leaving the details to be examined upon user request. Since a lot less is being displayed when the document first opens, does this mean we can improve performance using sections?

After some tests using RPCs, we found that it does not matter if a section is expanded or collapsed for Notes clients. In other words, all of the design information is sent to the client when the document is opened, regardless of whether a section is expanded or collapsed.

However, as with tabbed tables, when a Web client uses sections, only the information that is being displayed is downloaded. Using sections will improve the performance for Web browsers.

**Using Layout Regions**

Layout regions provide a graphical alternative to standard Notes forms. Graphical backgrounds and images can be implemented and data objects can be overlaid. However, because of the graphical nature of layout regions, there may be performance problems resulting from their use. The graphical objects used in layout regions will cause forms which contain them to generate more slowly.

With complex layout regions containing many graphic and data objects, the performance impact can be considerable. Applications that include the limited use of simple layout regions may perform acceptably. Performance-enhancing alternatives:

- Substitute standard forms without graphical elements for layout regions.
- Reduce the layout region to only the essential elements that cannot be implemented any other way (such as the date and time picker).
Using Progressive Disclosure

When documents contain many fields, there is usually some sort of grouping, or hierarchy of the data. Users don’t always want to see every single field.

You can get performance gains by displaying data to users “on request.” That is, display the main data fields to the user, but use alternatives to display the many subsections of the data.

Two techniques for this progressive disclosure of the data are:

1. Use sections to hide various groups.
   The main data is displayed but additional details are in collapsed sections. This does not improve performance for Notes clients but can be very effective for Web users.

2. Use dialog boxes to display data.
   The main data is on the form, but to see details, the user needs to take some action (as they do in opening a section). This action displays the subsection of data in a dialog box.

Since displaying the dialog box is a separate activity, and displays only a relatively small part of the data, you can be a bit more flexible in your choice of UI components. A complex display of all the data at document open may be unacceptable, but a small delay at dialog box display may be fine. The total processing time may be greater, but since it is split between several actions, the user perceives quicker response. There is a saying “Perception is reality” which accurately reflects a user perspective.

Caching Field and Hotspot Formulas

For Web clients, each formula Domino must calculate when opening a page slows your application down. Also, when an @function exists on a page (that is, on a form, document, view, or navigator), by default Domino doesn’t cache the commands used to display the page.

To force Domino to cache commands for Web clients, you can turn the Domino Formula Analyzer on. For more information about Formula Analyzer and Caching Commands, see Appendix C-7, “Ways to Control Caching of Pages Served by Domino.”

An alternative to caching @functions for Web browsers, of course, is to use JavaScript where possible. JavaScript runs on the browser, reducing server load.

Chapter 4: Performance Aspects of Domino Design Elements 93
**Using JavaScript**

For Web clients, JavaScript is evaluated on the client, as opposed to formulas or scripts that are evaluated on the server. This lightens the server load and can increase performance.

In a Web application, JavaScript is commonly used for field input validation (for example, “Did the user enter the date in the appropriate format?”). In this situation, JavaScript is more efficient than using a formula in the field input validation or input translation events. Using @functions requires a round trip to the server since the Web browser does not understand LotusScript or the Formula language.

We can also use JavaScript to access properties that are not directly exposed any other way, such as changing the form color on the fly without the requirement to switch forms.

**Time-Date and Numerical Calculations**

When optimum performance is required, limit the use and complexity of time-date and numerical calculations.

Considerable processing overhead is incurred when Domino applications are developed with complex numerical and time-date calculations in the form. This can result in slow opening, saving, and refreshing of documents.

Frequent or complex calculations also can result in unacceptable delays when the user creates, views, or edits documents.

Performance-enhancing alternatives:

- Keep calculations to a minimum.
- Use Notes @functions such as @Sum instead of writing extensive script or code to perform summations.
- Consider utilizing a spreadsheet to process the required calculations, then link and embed the results into a Notes document.
- Reevaluate the use of Domino as the correct platform for the application.

**Keep Hide When Conditions Simple**

Keep formulas for Hide When conditions in a form as simple as possible.

All Hide When formulas evaluate whenever a document is opened for viewing, creating, or editing. The extensive or complex use of Hide When formulas can impact the time it takes to open a form.
Performance-enhancing alternatives:

- Utilize hide-when-editing or hide-when-reading check box options on the item properties where appropriate.

- Make use of computed subforms where possible. Many times the Hide When formulas are there to give two alternate versions of a field: one for users who can edit the field and another for users who cannot. When a group of fields are involved, using a separate subform for each mode can achieve the same objective but with better performance.

- Make use of controlled sections. As in the prior technique, you can group the required data into a section. One group of users can edit the section while others only have read access to the section.

- Create two different versions of a form. Again, to display data to two groups of users with differing access capabilities, creating a separate form for each may give much better performance than using many Hide When formulas.

Tip  If you have Web-only or Notes-only forms (or other design elements like views and navigators), use the Design Properties box to hide them from the inappropriate audience. Give both design elements the same alias name so you can use that alias name in formulas. For example, if you create views named NotesMain and WebMain and assign them each the alias “Main,” you can then use the alias name in a formula:

```plaintext
@Command([OpenView]; “Main”)
```

With this formula Domino presents the NotesMain view to a Notes client, and presents the WebMain view to a browser client. This technique gives better performance than multiple Hide When conditions or computed subforms.

**Fine Tune Your Lookups**

The improper use of lookups within a form can sometimes create the most significant performance problem in an application.

Lookup functions like @DbLookup and @DbColumn carry the overhead of searching and potentially reindexing internal/external views before returning the required result. It is common for Notes applications to make heavy use of these functions.

There are many ways to reduce the number of lookups that an application has to perform and to design more efficient lookups. Following are some of these techniques:
• Utilize the “Cache” option within lookups wherever possible. Using this option keeps the lookup results cached at the client. Subsequent lookups for the same view/column will use the cached data, thus eliminating the need for a client-server interaction.

• Minimize the unnecessary use of @Db functions. For example, if you are using a set of keywords in your application and then access these via @Db lookups, hard-coding the keywords would improve performance. This only applies to keywords that remain relatively static, such as the use of “High,” “Medium,” “Low” or “Yes,” “No.”

  Note Other tools, like “Notes Global Designer” or “Domino Global Workbench” can be use to achieve National Language Support (NLS) enablement for this scenario.

• Have @Db functions refer to simple, efficient views which are dedicated to lookup use. When @Db lookups are required, using a concise view will yield better performance than using an existing view that carries many columns not needed for the lookup.

• Use LotusScript to perform lookups. Using the Search or FTSearch methods in LotusScript to provide lookup data can reduce the overhead of extra dedicated lookup views.

• Lookup against column numbers rather than field names. When you lookup against column names, Domino opens each document to find the data related to the name. If you use the column number, that information is already available in the view index.

• Make use of the @IsDocBeingEdited and @IsDocumentBeingSaved functions to improve keyword lookup formulas. See “Using Computed For Display Fields to Avoid Computations,” in Chapter 3 for a discussion of this technique.

• Keyword fields and pop-ups utilizing lookups all evaluate when a document is being opened. To reduce document opening time, place lookups behind buttons. Button formulas only evaluate when selected.

• Use temporary variables in your formula to save @Db lookup results. If you use the same @Db lookup in many places in one formula, replace those lookups with a temporary variable. This variable is set at the beginning of the formula to the result of a single @Db lookup.

• Use temporary fields (computed for display) in your form to save @Db lookup results. If you use the same @Db lookup in many field formulas in your form, replace those lookups with a temporary field. Put this (hidden) field in your form and set it to the result of the @Db lookup.
For example, instead of:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test_1</td>
<td>Text - Computed</td>
<td>@DbColumn(&quot;&quot;; Server:Database; View; Column)</td>
</tr>
<tr>
<td>Test_2</td>
<td>Text - Computed</td>
<td>@DbColumn(&quot;&quot;; Server:Database; View; Column)</td>
</tr>
</tbody>
</table>

You should use:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test_disp</td>
<td>Text - Computed</td>
<td>@DbColumn(&quot;&quot;; Server:Database; View; Column) for display (hidden)</td>
</tr>
<tr>
<td>Test_1</td>
<td>Text - Computed</td>
<td>Test_disp</td>
</tr>
<tr>
<td>Test_2</td>
<td>Text - Computed</td>
<td>Test_disp</td>
</tr>
</tbody>
</table>

- Combine required data into one column. When you need more than one column of data from the same @Db lookup, create an extra (hidden) column in the view that contains all the data you will need. Save the results in a temporary variable and then parse this variable whenever lookup data is needed. Use the @Subset function to parse the results.

For example: a help desk application contains two databases. The first one has the requester file (DB 1) and the second one (DB 2) performs a lookup on DB 1 to fill up a section of a form:

**DB 1 Form:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Names - Editable</td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td>Text - Editable</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Text - Editable</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>Text - Editable</td>
<td></td>
</tr>
<tr>
<td>Lista</td>
<td>Text (hidden - allow multiple values)</td>
<td>Translation formula: Phone:Address:Other</td>
</tr>
</tbody>
</table>

**DB 1 Lookup view:**

<table>
<thead>
<tr>
<th>Column 1 Formula (Sorted)</th>
<th>Column 2 Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field - Name</td>
<td>Field - Lista (Phone:Address:Other)</td>
</tr>
</tbody>
</table>
DB 2 Form:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Names - Editable</td>
<td>Name</td>
</tr>
<tr>
<td>Phone</td>
<td>Text - Computed when composed</td>
<td>Phone</td>
</tr>
<tr>
<td>Address</td>
<td>Text - Computed when composed</td>
<td>Address</td>
</tr>
<tr>
<td>Other</td>
<td>Text - Computed when composed</td>
<td>Other</td>
</tr>
</tbody>
</table>

Description: Rich Text

To retrieve the data from DB 1 you can use an action button or hotspot button with the following formula:

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lista := @DbLookup(“Notes”::“Cache”; Server:DB1; “DB 1 lookup view”; Name; 2);</td>
<td>Stores the lookup result in a variable (Lista)</td>
<td>Phone:Address:Other</td>
</tr>
<tr>
<td>@SetField(“Phone”; @Subset(Lista; 1))</td>
<td>Set the field Phone with the first element of the list (Phone:Address:Other)</td>
<td>Phone</td>
</tr>
<tr>
<td>@SetField(“Address”; @Subset(@Subset(Lista; 2); -1))</td>
<td>Set the field Address with the last element of the list (Phone:Address)</td>
<td>Address</td>
</tr>
<tr>
<td>@SetField(“Other”; @Subset(Lista; -1))</td>
<td>Set the field Other with the last element of the list (Phone:Address:Other)</td>
<td>Other</td>
</tr>
</tbody>
</table>

Note: In actual production, you would do an error check (@IsError) in the @DbLookup function to avoid error messages like “Entry not found in the index.”

Tip: Use profile documents to store specific information across a database. For example, since Domino R4.5, the mail templates use profile documents to collect calendar and schedule information, such as when a user will be out of the office and who can access the user’s mail file. Profile documents are useful for storing user-specific information in a database.

Using Fields Wisely

There are many things you can do when using fields that affect performance. Knowing more about how fields fit into the Domino architecture will help you create applications that perform.
Number of Fields on a Form

The number of fields on your form is arguably the single most important thing that affects performance. Domino must keep track of every field in your database. Views keep summary information for every field that is involved in the view formulas (select and column). Documents must process every field in relation to the form used to open the document.

Complex applications will, by definition, have many fields. This usually translates to complex forms, views and logic. Getting the most out of every field is one step in the direction of good performance.

The single rule here is “Remove all unnecessary fields.”

Complex data still needs to be displayed. Where possible, split the fields among several forms. The number of fields in a database relates to complexity. The number on any one form, however, relates to performance.

Tip After your design is completed, remove any blank or unused fields.

Tip Use progressive disclosure for complex forms.

Field Basics

Understanding how the various field types work can suggest ways to significantly improve performance. In the worst case, each field is calculated when a document is opened, refreshed, or saved.

By understanding what calculations are performed and when calculations are evaluated, forms and views can be tuned to maximum performance. This will impact how long it takes to open, save and work with documents.

Some field basics:

- Fields within a form are evaluated from left to right and top to bottom.
- Formulas within pop-ups and keyword fields are evaluated at document open.
- Do not use “Computed” fields unless they do need to always be evaluated at save time. For more information, see “Effects of a Computed Field” later in this chapter.
- Use “Computed when composed” fields when evaluations are needed upon demand. For example, if a button is defined to store the results of a formula, the results should be placed into a “Computed when composed” field.
- Consider using one event script to evaluate many fields within a form rather than having each field evaluate results individually.
- By default, fields have a “Summary” attribute (except rich text fields). Summary data about fields is stored in view indexes and other places in the database.
Also see “Use Computed for Display Fields to Avoid Unnecessary Computations” in Chapter 3.

**Shared Fields**

There is overhead required as Domino resolves the format and function of shared fields within an application. Be sure to define shared fields only when complex formulas are being shared throughout the database.

In your database design, a shared field and a form are two different design elements. Domino must make an extra internal function call (a NoteOpen) to get the design information for the shared field once it is found to exist in a form. In all, Domino does one NoteOpen call to get the document, one to get the form as specified in the document, and finally one to get each of the shared fields specified in the form.

**Effects of a Computed Field**

Computed fields require additional processing and can slow down performance.

In the following examples, we compare computed fields against editable fields:

![Graph showing performance comparison between computed and editable fields for composing and saving documents.](image)

100 Performance Considerations for Domino Applications
As shown by this data, computed fields decrease application performance more than do editable fields, especially when large numbers of fields are involved.

The following test examples were performed by accessing copies of Domino Directories using @DbLookups. Caching was enabled and the Forms had only computed fields.

The server and network performance in this example cannot be considered “typical.” Do not use the times recorded here for anything but comparisons against one another.

As you can see, the time it takes to compose a document is much more seriously impacted by the number of matches of each @DbLookup than by the number of @DbLookup formulas. For example, 20 lookup formulas that match 3 documents each works faster than 10 lookups that returns 15 matches each.

### Control Usage of Rich Text

Displaying text in a rich text field requires more overhead than displaying it in a regular text field. With a large number of rich text fields, there may be a degradation in performance.

In situations where the application functionality is not affected by using plain text vs. rich text, then choose plain text. Plain text also has the advantage of being displayable in a view where rich text is not (but this will affect the view index size).

Rich text has the capability of containing attachments. Database size could grow beyond planned sizes when rich text is used in a manner contrary to the application design. Minimizing the use of rich text lowers this risk.

### Impact of Attachments and OLE Objects

Attachments and OLE objects are not stored as part of the document they belong to but as separate objects in the database. They are only being loaded as required. As long as the user does not work with the attachment/OLE object it is not being transferred to the client. However, if you embed an OLE
object and select not to display it as an icon, a bitmap representation of the OLE object will be generated and stored in the document. This bitmap will always be transferred together with the object and may add considerably to the size of the document.

There are situations where most users do not need to work with an embedded OLE object, but still need to see some of the data stored in the object. An example can be where an economic overview for a customer is stored in an embedded spreadsheet and users need to be able to see the key figures in the spreadsheet. In low-bandwidth environments you may consider putting fields to display the key figures in your Domino form and transfer the data from the embedded spreadsheet using Notes/FX. This would allow you to display the spreadsheet as an icon and thus avoid the overhead of a big bitmap being transferred together with the document, but still allow the users to see the key data in the spreadsheet without launching it.

**Security Via Readers Fields**
Defining a field with the type “Readers” will create documents that have good protection against unauthorized access. The performance aspects of opening a document for editing or reading that contains Readers fields is minimal. However, a poorly designed view of documents with Readers fields can lead to severe performance issues.

For more information, see “View Performance Guidelines” later in this chapter and also Appendix C-3, “Fast Access to Reader Names Protected Documents.”

**Using Authors Field**
If Readers fields can cause performance issues, how about fields of type “Authors?”

After several tests and analysis of the RPCs generated, we determined that Authors fields are not a performance issue. The difference in the amount of data that is sent between client and server using normal fields versus Author fields is insignificant.

**Use the HTML Field**
For Domino R4.x Web applications, using the HTML field will provide enhanced performance. If you are doing development using Domino R5.0, you would generally use Pages in this situation.

When Domino encounters a form that contains a field named “HTML,” Domino reads the value of the HTML field and sends the data to the browser without performing any computations. Therefore, it is the fastest way to display Web pages. The HTML field produces results that are similar to the form property “For Web Access: Treat documents as HTML,” but gives better performance.
You must generate values for the fields that Domino would otherwise calculate, such as author and creation date, because Domino passes the document to the browser without any precomputing or other Domino-generated data.

**Avoid Fields With an HTTP Prefix**

For Web use, you can capture CGI variables that are preceded by HTTP or HTTPS by defining a field with the variable name. For example, cookies are sent to the server by the browser as HTTP_Cookie. A field with a name of HTTP_Cookie will capture this data.

However, this means that whenever Domino sees a field name that begins with HTTP, it tries to resolve it as a CGI variable first. You can prevent this additional server workload by avoiding the use of application field names that begin with HTTP.

**Utilize Computed for Display Fields to Prevent Duplicate Data**

Avoid using Computed and Computed when Composed fields to format fields for display purposes. These field types will store the results during a document save, resulting in duplicate data within the document. As databases grow larger, the additional data will have a significant impact on view performance.

Use the “Computed for Display” type for fields which are only used to format data for the UI. These field types are not stored in the database during a document save operation.

**Caution** When using the LotusScript back-end classes to save a document which is currently on the UI, Computed for Display fields do get stored in the database. Make sure to remove these fields before saving in the back end.

**Note** In Domino R4.5 and R4.6, if a hidden field was used, this would not be passed to the Web browser and so could not be referred to. Domino R5.0 allows you to send all of the fields on a form to the browser, which will allow you to refer to these hidden fields to collect information. If you need to use this functionality in Domino R4.x, consider using a hidden frame (a frame with no dimensions that cannot be resized) and refer to the value using JavaScript.
**Automatic Field Refresh**

The “Automatically refresh fields” check box, on the Defaults tab of form properties, allows you to set the form to do automatic refreshes.

If you enable this option, computed fields, computed for display fields, and the translation and validation formulas of editable fields within the form will reevaluate every time the user moves the cursor to another field. This may result in a significant delay as the user moves from field to field.

This property may be effectively used with simple forms requiring minimum calculations, but for complex forms it can have a considerable impact on performance.

The following alternatives can be effective:

- For keyword changes, use the “Refresh fields on keyword change” option.
- Use a LotusScript field event to refresh desired fields under full program control.
- Use the JavaScript onBlur event as you would the LotusScript field event for Domino R5.0 and Web browser access.
- Provide selected updates via a defined button.

**Combine Field Formulas into Events**

When many fields in a form contain formulas that evaluate on the same event, it is recommended that a script event be created to replace all the field formulas.

In many instances, a form will contain multiple fields with a combination of input translation formulas and field validation formulas. It is possible to improve performance by performing this logic in a single script event vs. evaluating many field formulas.

Performance improves as more fields evaluating an event are substituted for by the script event. The complexity of the event formula for each field is an indicator of the performance improvement.

As an example, use the QuerySave event at document save time to populate many fields with data based on the state of key values. The same event (QuerySave) can be used to validate field content.

Combining event code can provide a more efficient method than having each field duplicate code to test the state of those same key values or validate against the same keywords.
**Working With Pop-ups**

Minimize the use of complex pop-ups within a form. Forms containing a large number of pop-ups can cause delays when the form is opened because pop-ups are evaluated at form open time.

Simple forms with textual pop-ups should not present any problems.

As an alternative, try using buttons or actions to display prompts or dialog boxes that require complex processing. In this case, the complex work is not done until it is required.

**Allow Domino to Automatically Redirect URIs**

When specifying a result page in a $$Return field or a WebQuerySave agent, and if the page is on the same server, use double square brackets around the URL for faster performance.

When a $$Return field or a WebQuerySave agent generates a URL to display the next page, the sequence of events is:

1. The browser submits the HTML form to the Domino server.
2. The server evaluates the $$Return formula or runs the WebQuerySave agent to produce the redirection URL — for example: [mydb.nsf/view?OpenView].
3. The server sends the redirection URL back to the browser.
4. The browser accepts the server data, detects that it is a URL, and requests the URL from the server.
5. The server gets the URL and sends the result of that request back to the browser.

The process described involves two round trips from the browser to the server. If the redirection URL references a database on the same Domino server, you can avoid the extra trip back to the server (and therefore increase response time) by using double square brackets around the URL. In the above example, this would be:

```
[[mydb.nsf/view?OpenView]]
```

The server recognizes this as an internal redirection URL and the transaction sequence becomes:

1. The browser submits the HTML form to the Domino server.
2. The server evaluates the $$Return formula or runs the WebQuerySave agent to produce an internal redirection URL — for example: [[mydb.nsf/view?OpenView]]
3. Before sending the URL back to the browser, the server recognizes the URL as an internal redirection URL, evaluates the request, and sends the results of that request back to the browser.

**Tip**  Alternatively, you could use the JavaScript onSubmit event and the History.go() or History.back commands to return to items in the browser’s history more efficiently.

**Defragmenting Complex Forms**

Forms are Rich Text Objects. Over time during development a form will become fragmented as you work with it adding, moving and deleting fields and table rows and columns. If you have a real complex form with many field and tables fragmentation can add much to the size of the form. Compressing the database will not defragment a form. However, if you select all on your fields and tables on your form and cuts them to the clipboard and then pastes them back to a new blank form you can shrink the size of the form. How big a size gain you get depends on the complexity of the form and how many times it has been revised.

**Note** Any code you have in different form events and script libraries are not copied together with the tables and fields on the form. You must also copy these manually to the new form.

---

**Profile Documents**

Profile documents were introduced in Domino R4.5 and are another area to consider when thinking about performance.

**Usefulness of Profile Documents**

Profile documents can be used to store user-specific data or database-specific information. The database may contains multiple profile documents, one for each user to store user-specific information, or a single profile document to store the database-specific information. Whether to use a single profile document or multiple profile documents depends on the design of the application.

Profile documents are useful to share information between LotusScript and the Formula language. Prior to profile documents, only environment variables could do this easily. In Chapter 3 we saw that using environment variables, which means writing to the NOTES.INI file, can be expensive.
One of the main advantages of the profile documents is that the information content of the profile documents gets cached. The profile documents can be loaded at the time of database opening, using PostOpen database event. Once loaded, profile document data is cached for the duration of the user session. Upon starting a new session, profile documents can be read once again (and again are cached).

Profile documents can eliminate the use of environment variables in a Domino application. There are several limitations for the use of the NOTES.INI file:

- The number of user-defined environment variables is limited.
- The NOTES.INI file is specific to the computer. You will lose all the information stored in environment variables if you change computers.

Profile documents do not have such limitations since they are stored in the databases.

The performance of the application can also be improved by using the profile documents instead of environment variables. While using LotusScript, accessing the fields in a profile document is similar to accessing fields in any other document. This is much faster than accessing the environment variables using the GetEnvironmentValue method.

One last area of potential performance gains using profile documents is for @Db lookups. If your application uses a set of data which is accessed via @Db lookups, and you use the caching options for these lookups, then that cached data is not available from LotusScript. Even if you code an Evaluate statement in LotusScript which executes the exact same formula as in the Formula lookups, the LotusScript code will not access the cache but will go to the database each time.

---

**Profile Information for Web Browsers**

The use of Profile information for Web browsers requires further consideration. Clearly, with the Web we do not have the same caching ability (we do not have a CACHE.DSK file), and authentication is not as paramount to the overall design. In addition to this, we do not have a NOTES.INI file, or anything similar, in which to store our environment variables. How, then, do we record this information in order to enable two important features:

1. Linking our browser to specific information.
2. Caching information at the browser for high-speed access.
Linking an Identity to a Web Browser

There are two basic methods that can be used to do this. The first is to enable a form of authentication to our application or Domino Web environment. Fortunately we are using Domino as our Web Server. Domino has a very granular security model. We can enable authentication for our entire server, an individual database, a specific URL and so on. We also have a number of methods of authentication, including simple authentication (Username and Password) and SSL V3 Certificate Authentication.

Once we have a Username assigned to the browser user, we can store information in the database using a Domino document and associate it to the Username given. An example of this is when we store a document in a hidden view (for example "($Profiles)") and retrieve the information from the document using the GetDocumentByKey method.

The Web design has also given us another feature that we can use to link the browser to an identity more easily. This is done using a specific Web feature called “cookies.” Cookies are small files which are downloaded to the machine that the browser is running on. A cookie cannot be used as a security mechanism on its own; however, it can be used to allow fast access to specific information. An example of using a cookie is when the server holds a document containing your user preferences. When you connect to the server, you send your cookie. The cookie contains the document ID of your specific preference document, which can then be accessed without the need for authentication with the server.

Browser Caching

Different Web browsers cache information in different ways. Most browsers are designed to hold a list of temporary files, such as images. This will enable the reloading of the graphical pages to be improved. Cookies can only hold a limited amount of information. However, there is a technique that can be used to enable us to store application-related information on the Web browser and not have to store this temporary information on the server. For example, if we have a large list of temporary information, such as project codes that we are going to be referring to in our Web application, we probably want to send these to our browser only once. In a Notes client we would often choose to store these as a hidden field in a document or use one of the caching techniques already discussed. Prior to Domino R5.0, any hidden fields on a form were not converted to HTML and were not sent to the client to be referenced. Therefore, the ability to store some information in a hidden fashion on a Web client was very tricky.

A better method to do this is to use the Frames feature. Here we can partition the window of the user’s Web browser into different areas. We have the ability to specify the width or height of a particular frame and also
determine if the user is able to change the size of this frame. Using these two features of a frameset we are able to create a frame with no width that the user may not re-size: in effect it is a hidden frame. Into this frame we can then load a form with a number of fields that we can use to write information. Using JavaScript, it is possible to simply pass information between our “live” user frame and the “hidden” frame that we are using to store temporary information.

Views

One of the great features of Notes is its ability to display information in views: categorized, sorted, totaled, and always available. Views are a feature of Notes that make it stand out from the rest.

Using views can provide significant user productivity, but you need to be aware of the potential impacts when views are not designed with performance in mind.

Some Internal Information About Views

The internal structure of a Domino database is very complex. In general it can be divided into several sections:

- Database information
  Database information includes things like the database title, replica ID, whether background agents are enabled and so on.

- ACL information
  The access control list for the database.

- Design information
  This includes the forms, views, agents, script libraries and all other items that you design when building a database.

- Document data
  This is the user-entered information stored in the database. When a form is used to enter data, the actual data keyed in by the user (and any other hidden fields computed by the form) is stored as a document in the database. The form itself is generally not stored as part of the data, but is retained separately as a design element.
• View index data
Each view has at least one index that contains a list of pointers to the
documents contained in the view. The sequence of the pointers in this
list matches the sequence of the documents in the view. If a view has
alternate sort options, each sort sequence requires another index to store
the alternate pointer sequence.
In addition to pointers to the documents, the index stores summary
information for the data stored in each column in the view for that
document.
In many cases, view indexes take up more disk space than the
documents themselves. To see what size the view indexes are, use the
LOG.NSF database and look in the Database\Sizes view.
• Replication information
  Replication settings and history.
As you can see, views are related to three of these sections:
• Design information (where the view design is stored)
• View index data (where view indexes are stored)
• Document data (the actual data to which the indexes relate)
All three can play an important role in view performance, as we shall see.

View Design Considerations
This section examines the various properties of a view and relates them to
view performance. Although we go into a bit more detail than absolutely
necessary here, having a thorough understanding of how views work gives
insight into views as a whole.

Type of View
When we design a view, this is the first decision we must make. Is the view
shared, private on first use or private? Many times the answer is
predetermined, but let us examine the performance implications of our
choice.
We can safely rule out private views since as designers we are creating an
application for somebody else’s use. Private views can only be used by their
creator. That only leaves shared and private on first use view types to
consider.
For the applications you will be designing it is highly likely that most views
will be shared by all users. However, there are situations where private on
first use views are desired.
In Domino Designer R5.0 we select the view type from a pull-down list:

As you can see, there are five types of shared views: standard, two special types, and two private on first use types. The two special view types — “Shared, contains documents not in any folder” and “Shared, contains deleted documents” — were added in Domino R5.0 and will not be discussed here since they are not generally used as application views.

Prior to Domino R5.0, view type selection was done via check boxes:

Chapter 4: Performance Aspects of Domino Design Elements  111
The following table relates the Domino R4.x check boxes to the Domino R5.0 view types:

<table>
<thead>
<tr>
<th>Domino R5.0 View Type</th>
<th>Shared check box</th>
<th>Personal on first use check box</th>
<th>Store in desktop check box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>Ticked</td>
<td>Not ticked</td>
<td>Not ticked</td>
</tr>
<tr>
<td>Shared, contains documents not in any folder</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Shared, contains deleted documents</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Shared, private on first use</td>
<td>Ticked</td>
<td>Ticked</td>
<td>Not ticked</td>
</tr>
<tr>
<td>Shared, desktop private on first use</td>
<td>Ticked</td>
<td>Ticked</td>
<td>Ticked</td>
</tr>
<tr>
<td>Private</td>
<td>Not ticked</td>
<td>Not ticked</td>
<td>Not ticked</td>
</tr>
</tbody>
</table>

**Private On First Use Views**

Depending on the release level of Domino, these can also be referred to as Personal On First Use views. As the name implies, the view is personalized when a user uses it the first time. There is a common design for the view but the user has a unique view index (which is built the first time a user uses the view).

In general, this type of view is used when the view index needs to be different for each user (but there is an exception — see point 3). The following situations may call for a private on first use view instead of a shared one:

1. Use a private on first use view when the selection formula contains a reference to the user’s identity.

   When @UserName is part of the selection formula, a private on first use view is required. This is because the user name that will be used in the formula will be from the ID in effect when the view index is created (or updated). This will most likely be the server’s ID (but can be another user’s ID if that user rebuilds the index). To get the documents required by the design, the selection formula must execute using the user’s ID for each user. The only way we can do that is by using the private on first use view.
From a performance perspective, this is good since these views are fast to display to the user after they have been opened the first time. The first time a user accesses a private on first use view, however, can be quite slow since the private view index must be created.

Another problem with private on first use views is that their design is not easily replaced. A discussion on this is included later in this section.

2. Use a private on first use view to speed up views that contain documents with reader names fields.

If your data is full of documents that contain reader names fields and you have access to a very small percentage of the documents, the server can spend a lot of time finding enough documents in the view to fill your screen. This can not only affect your perceived response time (since you are waiting), but it can also affect other users of the server (since the server is filtering lots of data for you). By using private on first use views in this situation, you can improve performance at both sides of the client-server combination.

As in the prior case, remember that the initial open time can be long and the design is not replaced easily.

3. Use a private on first use view to offload indexing to the client.

There is a limitation, which can occur when many databases need to be indexed at the same time, which is overcome by using private on first use views. Prior to Domino R5.0, there was a limitation to the size of the indexing queue. In particular, if mail was received at a mail server for many users on that server, each of the user’s databases needed to have their view indexes (and full text indexes) updated. This could exceed the view index queue limit. By offloading this view indexing to the client, this limitation was overcome.
If you are using Domino R5.0, there are two alternatives to private views in certain situations. These conditions generally affect views that contain many documents, but in which only a few can be seen by any one user. In this case, the following options may be a good design choice:

1. Use a single category view for Web users. When designing forms or pages with embedded views, you can specify that the embedded view have the property Show single category:

When your view is categorized such that the user has access to data in a single category, this can limit the amount of data that must be sent to the Web client.
2. Use the view property Don’t show categories having zero documents:

When this property is used in conjunction with the view property Collapse all when database is first opened, the amount of work that Domino must do can be reduced, in particular for views of documents with reader names fields. The view must be categorized and the data must be organized such that users have access by category. The categories that the user cannot access are eliminated from the view processing.

Where Is the View Index Stored?
If a private on first use view is your choice, you also need to choose whether the view index is stored on the server or in the user’s desktop.dsk. This choice only comes into play if the database ACL allows users to store data on the server. The ACL property Create shared folders/views must be selected for the user to have the ability to store their view index on the server. If this ACL property is not selected, the index will be stored in the user’s desktop.dsk.

Another consideration is access to the view via the Web. Web users will not be able to access views unless the indexes are stored on the server.

If you have a choice of index location (the ACL allows it and the client is Notes), you must do it when the view is being created (you cannot change it later). Prior to Domino R5.0, this is done via the Store in desktop check box. Since Domino R5.0, select this in the View Type pull-down list.
The location of the view index is important because:

1. Storing many user view indexes on a server may adversely affect not only server performance, but also server storage capacity.

2. When an application design is updated, techniques to update each user’s private version of the private on first use view are different depending on where the index is stored.

3. Destroying private on first use views that are stored on the server must be done consciously. If they are stored in the DESKTOP.DSK, they are deleted when the icon is removed from the workspace.

**Design Changes**

A major problem with private on first use views is that you cannot change the design of the view easily. If the design changes, it is only the server version of the design that is altered. Since the user’s version of the view is private to the user, it is not updated. This private view must be deleted so that when the user next opens the view, the design is refreshed from the server version.

There are several ways to delete this private view, none of which is ideal.

**Private View in User’s desktop.dsk**

If the view index is stored in the user’s DESKTOP.DSK, you need to get the user to manually delete the database icon. When the icon is removed from the workspace, all private views are also deleted.

For versions of Domino prior to R5.0, deleting a specified database icon can be done by sending the user mail containing the following code. The code can be triggered by a button.

```plaintext
@Command([WindowWorkspace]);
@Command([FileOpenDatabase]; "server":"dbName.nsf");
@Command([FileCloseWindow]);
@PostedCommand([EditClear]);
@PostedCommand([FileOpenDatabase];"server":"dbName.nsf")
```

**Note** This code will remove not only the user’s private on first use views, but all the user’s private views for this database (that are stored in the desktop.dsk).

Since this icon deletion is a UI activity, the user will be prompted via two message boxes. The first prompt asks the user to confirm that the icon will be deleted. The second asks the user to confirm that their private views will be deleted. Make sure your message to the user tells them to accept both messages.
You may notice that the previous code not only removes the database icon (to delete the view indexes), but it puts the icon back as well. Your message to the user should indicate that the location of the replaced icon on the workspace page may not be the same as that from which it was removed.

**Note** We have no code sample for the Domino R5.0 situation. The icon must be deleted manually. For Domino R5.0 we are also concerned with bookmarks. Bookmarks are not affected when a database icon is removed from the workspace. Thus, after the icon is removed manually, a bookmark can be used to replace it.

**Private View is Stored on the Server**
For view indexes stored on the server, simply removing the icon from the workspace will not remove the view. In this case, however, we can write code to find the private views (which we cannot do if they are stored in the desktop.dsk).

The following code can be used to remove the private versions of views when a design change occurs. Again, it can be coded behind a button and sent to the user in a mail message (but the user still needs to push the button).

```vbscript
Sub Click(Source As Button)
    Dim ss As New NotesSession
    Dim db As NotesDatabase
    Dim masterView As NotesView
    Dim count As Integer
    Dim userMsg As String

    Set db = ss.GetDatabase("server", "dbName.nsf")
    count = 0
    userMsg = ""
    For all v In db.Views
        count = count + 1
        If isPrivateView(v) Then
            '*** Get the master view on which this
            '*** personal view is based
            Set masterView = getMasterView(v)
            If Not(masterView Is Nothing) Then
            ' *** Replace the private view
            end sub
```

Chapter 4: Performance Aspects of Domino Design Elements  117
'*** If private view is older - delete it
If v.Created < masterView.LastModified Then
    userMsg = userMsg & Chr(10) & v.name
    v.remove
End If
End If
End If
End Forall

If userMsg = "" Then
    Messagebox "No out-of-date private views were found"
Else
    Messagebox "The following out-of-date private " & _
    "views were removed:"
    userMsg & Chr(10) & Chr(10) & _
    "Please reopen this database for these views " & _
    "to be updated."
End If
End Sub

Function isPrivateView(v As Variant) As Integer
    '*** Return True if view is private
    Dim vdoc As NotesDocument
    Dim itm As NotesItem
    isPrivateView = False

    If Not (vdoc Is Nothing) Then
        If vdoc.HasItem("$Flags") Then
            '*** Private views have a 'V' flag
            Set itm = vdoc.GetFirstItem("$Flags")
        End If
    End If
End Function

118 Performance Considerations for Domino Applications
If Instr(itm.Values(0), "V") Then
    isPrivateView = True
    Exit Function
End If
End If
End If
End Function

Function getMasterView(pView As Variant) As NotesView
    '*** Return the view on which this private one was based
    Dim db As NotesDatabase

    Set db = pView.parent
    Dim view As NotesView
    Forall v In db.views
        '*** Master view has same name
        If v.Name = pView.Name Then
            '*** Master view is not private
            If Not(isPrivateView(v)) Then
                Set getmasterView = v
                Exit Function
            End If
        End If
    End Forall
End Function

Note This LotusScript code removes only private views that are out-of-date with respect to the master view. This can be easily modified to behave like the formula-based code listed earlier, which removes all private views.

Note This code works for Domino R5.0 as well as prior releases of Domino.
Selection Formula

The selection formula determines which documents appear in the view. There is at least one entry in a view’s index for every document selected, thus index size is directly affected by the selection formula. Any database activity that affects views will necessarily take longer if the index is larger.

Selection formulas can include time-relative constructs (for example, @Today or @Now). If either of these @functions is included in the selection formula, the view index cannot be marked as up-to-date. Because of this, the entire view index gets rebuilt every time the view is accessed by a user. Time-relative @functions should only be used in views that contain small numbers of documents (less than 1000). If a time-relative view is required in a database which contains many documents, there are techniques for the design of these views that give good performance. For a discussion on building time-relative views for large databases, see Appendix B-1, “Time-Date Views.”

Selection formulas can contain information about the user identity (@UserName). This @function should only be used for private or private on first use views. If used on a shared view, the results are unpredictable.

Document Types

The types of documents in a view are another thing to be considered in your view selection. Are the documents Main documents, Response documents or a combination of both? Do the documents have special fields, such as readers, authors, encrypted, or signed fields?

As it turns out, the document type does not play a significant role in determining the size or the efficiency of a given database. Reader names and encrypted fields, however, should be managed carefully.

Column Design

Many aspects of the view column design have no impact on performance. These include things like column width, column heading, heading fonts, data fonts, alternating line color and so on.

There are several things, however, that do have performance implications. These are categorization and sorting, features often used in views.

Categorization

When a column is defined as a category, Notes needs to store a lot more information in the view’s index to manage the category. Every aspect of view index maintenance will be affected since the index is more complex and bigger. Bigger indexes means a bigger database. This means we have performance hits, not only in index maintenance, but in other areas where database size is a factor.
Sorting
In many cases, you want the data in a view to be displayed in a particular sequence. This can only be accomplished by sorting the column. For smaller databases, there is minimal impact, but as the database size grows, so does the impact of sorting. Sorted columns take up more index space (but not as much as categorized columns). Rebuilding the index for a sorted view also takes more time (but again, not as long as categorization). When documents are added to the database, the index must also be updated. This is an incremental change and often is not noticed as a performance issue.

Dynamic Sorting
Using dynamic sorting for a view column can help to reduce the overall number of views required. However, dynamic sorting has a large impact on view index size. Every column in a view has a sequence, even if that sequence is “unsorted.” The amount of space used by the index for a view that has every column sorted compared to that used for a view that has no sorted columns is very similar. The extra time required to find the correct insert point in the index is also a small factor (insertions only happen for new documents, and they are incremental). The size of the view index, however, and thus the database size, has a far more noticeable impact on performance.

When a view has columns that can be dynamically sorted, the view must not only have an index for the normal view sequence, but also an alternate index for the other sort sequence. If the dynamic sorting includes both ascending and descending, then a third index is required. Both of these additional indexes take up space, and thus time, during the many index operations that occur during normal operations in a database. If dynamic sorting is included on several columns, an index for each unique sequence is required. Before you know it, a view can suddenly grow to the point where index size far outweighs the data size. It is not uncommon for a database to be three to ten times larger with all indexes built compared to the size with no indexes.

If you do need dynamic sorting, only use ascending or descending. Don’t allow both types of dynamic sort.

Note A view index contains more information than just the sequence of the documents. Summary information is also included in the index. When alternate sequences are required to satisfy dynamic column sorting, only the alternate sequence needs to be duplicated. The summary data does not need to be duplicated. See Appendix C-2, “View Index Comparisons,” for some test results on view index sizes.
Column Data
The number of columns and the data displayed in a column can have an adverse effect on performance.

First, how many columns are being displayed. Each column takes up room in the index for its key and summary information.

Second, what data is displayed. If the data being displayed is a simple field value, performance is maximized. On the other hand, if a complex formula is being done to determine column data, this needs to be calculated for each document in the view. Column data can include a variety of @functions including document lookups and data manipulations.

Calculated data is not stored as part of the summary information, but instead is calculated each time the view is accessed. Notice, however, that Domino does not evaluate column data for all documents in the view first, and then display the view. This would be very time-consuming indeed. Instead, it does just enough work for the user to see at least one screen full of data, and then it waits for the user to scroll forward before processing the next batch of data. Who knows — the user may decide it’s not the view they were after and just exit the view straight away. All those calculations would be a waste of effort.

Tip. When you do have to do complex calculations to display in a column, it is often better to add an extra field to the form that contains the result of the calculations. The calculations happen only when documents are changed and the view suffers no performance impact.

View Actions
View actions can affect performance, especially in a low-bandwidth network environment. Unlike forms, the design information behind action buttons is not sent from server to client when a view is opened by the user. Instead, the design information is sent to the client when an action button is activated by the user pressing on it. Then we not only get the performance hit of doing the work of the action, but before we can even start, a potentially large piece of the view design may need to make its way from server to client.

Tip. If there are a number of actions that occur on several views, this common code might be stored in a script library. If your environment is not a high-speed LAN, you need to be careful that you don’t make the script library so big that performance is affected. Break your code up into several smaller script libraries so that only the required code is loaded.
View Indexing Properties

As designers, we have some control over how often a view index is built and updated and when it is discarded. If we know that data in a view is static, or just about static, we can save some resources by preventing the index form continually being processed during index update. Likewise, if a view is so dynamic that the update task would be hogging CPU resources, we can prevent too frequent index updates. The indexing task (UPDATE) is often the heaviest user of server CPU time.

Refresh Index Options

The View Properties box is used to select the index refresh options:

The following refresh index options are available for a view:

1. Automatic

   Automatic views will be forced to be up to date when opened. When a user opens a database, and documents have been modified since the view was last updated, the view will be updated (refreshed), and the client will have to wait until the view is updated before the database will open.

   When UPDATE or UPDALL is run on this type of view, the view will always be brought up to date.

2. Automatic, after first use

   Automatic, after first use views are very similar to Automatic views. The only difference is that if the view has not been accessed, no indexes are built.
When UPDATE or UPDALL is run on this type of view, the views will only be brought up to date if an index already exists. Views that have not been accessed have no index so update processing does not affect them.

3. Automatic, at most every XX hours

Automatic, at most once every XX hours views, when opened by a user, will be brought up to date only if they have not been refreshed in the specified interval. If the view has been refreshed in the last XX hours, it will open immediately. This refresh frequency is particularly good for large databases that are often modified. When a user opens a view, they won’t have to wait for the view to be refreshed, it will open immediately, but will have the refresh icon at the top left of the view.

When UPDATE or UPDALL is run on this type of view, the view will always be brought up to date.


Manual views will never be refreshed by the user opening the view. The view will always open immediately. The only way a user can cause the view to be refreshed is to press the Shift+F9 keys (refresh) while in the view.

When UPDALL is run on this type of view, the view will always be brought up to date.

Discard Index Options
The following discard index options are available for a view:

1. Never

This is the most common option and should be used unless you have a very good reason not to. Indexes, once created, are maintained in an incremental fashion and unless you have server space constraints, should not need to be discarded.

Note View indexes are still deleted by the server after a long period of inactivity. By default this is 45 days, but it can be changed using the “Default_Index_Lifetime_Days” setting in the server’s NOTES.INI file.

2. After each use

The index is flagged for deletion after the view’s database is closed. The UPDALL task does the actual index deletion, so in fact the index will be valid for the rest of the day (the UPDALL task usually runs at 2:00 in the morning).
3. If inactive for XX days
   
   If a view has not been accessed for XX days, the index will be deleted, causing a rebuild when next accessed. Again, the UPDALL task does the deletion.

View Indexing Tasks

We will briefly describe the two indexing tasks already mentioned.

The UPDATE Task

The UPDATE task refreshes the views in a single database, and is a task that runs continuously on the server. It maintains a work queue, and regularly polls this queue to see if there are update requests to perform. A request is put into the update queue for the following reasons:

- A user closes a database after they have modified a document in that database
- A database is replicated
- The router adds a note to a database

When the UPDATE task processes the request to refresh the views in a database, it will update all views that have been previously opened in the database, regardless of the selected View Refresh option. This is done so that the user who next opens the database will not see stale information. The UPDATE task performs incremental updates, and these updates are generally very speedy.

Setting LOG_UPDATE=2 in the server NOTES.INI file will allow you to observe each view being updated in the database.

You can run multiple UPDATE tasks on the server to improve performance, but this is only effective if the server has multiple CPUs.

The UPDALL Task

UPDALL is a task which (by default) is scheduled to run at 2:00 each morning against all databases on a Notes server. It makes a list of all the databases on the server, and then updates the views in each of them. When these databases have all been updated, UPDALL terminates.

The UPDALL task will update all views that have been previously opened in a database, regardless of the selected View Refresh option.

UPDALL also has options to rebuild the view indexes instead of just updating them. This can be handy when you suspect view index corruption has occurred. You can run UPDALL specifying a single view and the rebuild option. This will rebuild the entire index for the specified view.
Set the NOTES.INI variable to LOG_UPDATE=2 to observe each view being updated in the database.

**Note**  Since UPDALL can also be initiated via the Server Console, a Remote Server Console and operating system commands, it can be used as an unscheduled way to update indexes when desired.

**The COMPACT Task**
Although not really a view index task, the COMPACT task has an option to do database compaction without doing view indexes. This can be used to remove all view indexes from a database.

---

**Designing for Maximum Performance Using Views**

Information about views is in two distinct parts: The view design and the view indexes. View indexes are the main concern when dealing with the performance aspect of views.

Since view indexes contain information about the documents in the view, these indexes need to be maintained whenever a document is updated in the database. Depending on the change, the view index might need to be expanded, the summary information may need to change, or the position of the document in the index may need to be adjusted. If the document appears in multiple views, each of those view indexes may be affected.

Likewise, since the index stores information about each document in a view (at the column level), the index size is directly related to the number of documents in the view, the number of columns in the view and the size of the data in the columns.

Many of the following performance suggestions revolve around the key areas of view indexing and view content.

**View Performance Guidelines**

The following guidelines will help you design the best possible performing views. They are not listed in any particular order since the importance of each will change with the application requirements.

**Keep the Number of Views to a Minimum**
Don’t be tempted to leave in designer views. All those temporary views that you used during testing take up space and indexing time.

The number of views is one of the key factors that increases database size. A 10MB database can grow to 100MB after all indexes have been built.

The two key index tasks (UPDATE and UPDALL) are CPU-intensive and are directly affected by the number of views.
Simplify Column Formulas
Each formula Domino must calculate when opening a view slows your application down. Try to use formulas which are not very complex.

It is not very efficient to perform calculations on field values in a column formula. Instead, place the formula on a form, and store its result in a field to be displayed in the column.

Note If agents process the documents, this extra field needs to be maintained by the agent code.

Remove All Unnecessary Columns
Each column is calculated when a view is opened or refreshed. Each column takes up index space. Smaller views are faster views.

After your design is complete, be sure to remove any unused or unnecessary columns. For hidden views which are used internally, and not by the user, remove all columns except the ones actually used. These hidden views should also have no categorization and no extra sorts.

Use of Unread Marks
Unread marks have a performance impact on views. Try to use unread marks only in those views in which you really need them. If you do have to use them, try to use the option “Unread Documents Only” instead of “Standard (compute in hierarchy)” if the application usability allows it. The latter option can degrade performance, so by just changing the behavior of the unread options, you can improve performance of your view.

Use of Categorized Views
Categorization takes more effort and space. Index sizes for categorized views can be 100% larger than those for views that are not categorized. The time to index view changes is also increased when categories are involved.

Categorized views on the Web require the screen page to be redrawn on each mouse click (unless the view applet is being used, in Domino R5.0).

Tip When categories are required, combining categories into a single column using the backslash character (“\”) is more efficient than having a separate column for each. See Appendix C-2, “View Indexing Comparisons” for details.

Do Not Use Form Formulas
Due to the architecture of Domino, computed subforms are faster. Therefore, use computed subforms instead of form formulas.
**Avoid Time-Sensitive Formulas**

Time-sensitive formulas are those that use @Today or @Now in the SELECT statement or in column formulas. These types of formulas recalculate each time a user works in the view. For example, when a user opens, scrolls, collapses, or expands the view, the index needs to be recalculated.

If you do have to use @Today or @Now, try to set the Index rebuild frequency so that it is not automatic.

There are a number of techniques available to reduce the impact of time-sensitive formulas in views. See Appendix B-1, “Time/Date Views,” for details.

**Do not allow documents to appear in multiple categories.**

When a document appears in multiple categories, the size of the view index increases, causing the view to open and refresh more slowly.

**Remove Unnecessary Fields In Documents**

When designing forms, you should keep in mind that the total number of fields on a form affects the size of the view index.

Most fields on a form contain summary data. This affects performance since the summary data for a document must be loaded as part of the view index collection. When generating fields using back-end classes, you can choose to flag the field as having no summary data to prevent it affecting view indexes.

Even with this in mind, however, adding extra fields to prevent complex calculations in view columns is the better option.

**Be Careful When Restricting Access via Reader Names**

Restricting readership of individual documents via reader names fields also has an impact on view performance (restricting access to the entire view has no impact).

Depending on how many documents are visible to the user, reader names can severely affect the response times for a view to be displayed. For shared views there is a single index for the view. Since this index is shared by all users, an additional filter is used to display to the user only documents for which their name appears in the reader name field.

Remember, the server will serve to the client only enough data to fill the one and a bit screens. If the server must process many thousands of documents via the filter before enough data to fill one screen is found, then response will be slow.
So there are a few things to consider. If the view is small, there is no problem. The server can filter the entire view quickly and display it fast. If the view is big, however, the user needs access to a reasonable percentage of the documents in the view to get adequate response times. What’s reasonable? If we assume that 1000 server filter operations is reasonable, then we need to find approximately 50 documents that the user has access to out of every 1000 documents in the view. That says the user needs access to 5 percent of the data in the view for reasonable response.

Of course there are exceptions to rules. Data distribution is the major one in this case. If the user has access to a group of data and the view has a similar grouping, group distribution will affect view responsiveness. The user may get good response in certain parts of the view, but very poor response in others. When the user is in a part of the view that he has no access to, the server may need to read through a large quantity of documents before reaching a group that can be seen by the user.

Use categorized views if possible to reduce the data sent to the client. If views are designed so that documents seen by a user are restricted to one or more categories, and if the view is flagged as collapsed on open, only the category information will be sent to the client when the user opens the view. No reader name filters apply and the response is fast. If you are using Notes Release 5, you can hide the categories that contain no data for an even better display to the user. When the user opens a category, they have access to all the documents in the category so there is no delay with reader names filtering.

For Web usage you also have the option to use a single category view. When designing forms with embedded views, you can specify the embedded view to have the property Show Single Category. When your view is categorized such that the user has access to data in a single category, then this can limit the amount of data that needs to be sent to the Web client.

A last alternative is to use the new Domino R5.0 view property Don’t Show Categories Having Zero Documents. When this property is used in conjunction with the view property Collapse All When Database Is First Opened, the amount of work that Domino must do can be reduced, in particular for views of documents with read names fields. The view must be categorized and the data must be organized such that users have access by category. The categories that the user cannot access are eliminated from the view processing.

See Appendix C-3, “Fast Access to Reader Names Protected Documents,” for additional information on ways to maximize performance while still using readers names fields.
Restrict the Number of Lines Displayed to Web Users
The number of lines in a view that gets served to the client can be restricted by the server document. You can override this number by using the optional “Count=n” argument in the URL.

Make Use of the View Applet
Domino R5.0 allows the use of a view applet on the Web client. This applet overcomes many of the limitations of using views on the Web that were found in Domino R4.6.

In particular, when a view is displayed on the Web, every time a user clicks in the view, there is traffic between client and server to display the new page. In some situations, using the view applet will make more sense. Although the applet will take a bit of time to load initially, once loaded it is cached (some browsers do this better than others).

Another thing to consider when using a view applet is that it allows you to take advantage of some new features, such as scrolling, column width changing, and column sorting, to name a few.

Pages
Pages have been added to Domino R5.0 to allow more functionality and flexibility when designing not only Web applications, but applications for Notes clients as well. Previously, when static information was required to be published in a Domino database, developers would use a combination of forms with either fixed text or text stored in fields, and two help documents (Help Using This Database and Help About This Database). Developers often created documents that were only used for Web clients, either for storing HTML or for embedding Web elements.

Pages are Domino design elements and are thus stored as part of the design, not as data. This means pages are not included in views, and therefore the indexing of the database is not affected. The performance impact of using pages has not been determined, but there seems to be no reason that a negative performance would result from using pages. In fact, as experience with pages increases, there may be situations where pages provide performance improvements. Watch this space!
Framesets

Framesets have been very popular in the Web environment for a number of years. Their popularity is based on the fact that framesets can be used to split the client’s screen into a number of regions called frames. This has given developers the ability to have static areas on the screen that can be used to aid in navigation or help text, and have other areas that display scrollable information.

Since Notes R4.0 we have had partial access to frame properties by using views and navigators. When Domino was used to develop Web applications, framesets were always available, though application development using them has always been more complex than other techniques, and generally required knowledge of HTML.

From a Web user’s perspective, even though we often find framesets to be a bit cumbersome (they take time to load and initialize, and when loaded you bookmark the frameset and not the frame that you are interested in), we understand their power from a development and application perspective.

Framesets are a native feature in Domino R5.0, which means that you can develop them pretty seamlessly for both Notes clients and Web browsers. More significantly, you don’t need to develop a different one for each type of user.

So how is this great new feature going to speed an application up? Well, simply put, it now gives the developer the ability to organize the screen into many frames for the user’s information. Let us now look at two examples of how this can help.

Object Reuse

There has always been a need to compromise between what looks good and the time it takes to display information. Users like their applications to look good; this has been a major factor in moving a lot of processing away from the old “Green screens” that used to dominate corporate computing. We can now give the users some graphics and helpful tools in different frames, which do not need to be reloaded every time the user changes pages or navigates within an application.
Application Navigation

Framesets also allow us to split our application more seamlessly than previously. With large Notes and Domino applications, development will often require that a number of databases be used. This is often for purposes of capacity, performance, or application-specific reasons (such as the requirement for an access control matrix rather than a single access control list). Collecting the information from these databases was previously a task where the developer had the following choices:

- Accessing the information and finding a way of displaying it to the user in the current context (often by using @Db commands or LotusScript to pull the information into the current document)
- Switching the user into the other database, which was often very problematic due to the number of windows that Notes could display and the possibility of losing the current context

Appendix B-5, “Switching Seamlessly Between Databases,” shows an example of such switching between databases using Notes 4.x.

In the example in Appendix B-5, we switch between three databases: Customers, Accounts, and Transactions. Let us see how we could have done it differently using frames. Before, when we switched between databases, we used a combination of agents and environment variables to get the correct information. Frames allow you to switch between databases without users knowing it. Using frames, you can open a customer document and see the details of what accounts the customer has and what transactions have been made by the customer without knowing which databases are being used. What’s more, you can easily drill down in detail to see more information about the current transaction if you need to.
The following figure shows what users would see when they enter the specific customer document in our Samples database:

![Image of a customer entry screen showing name, social security number, and date of birth.

How Was This Achieved?
Here are the steps we took to make this happen:

1. We modified the Customer form. In the QueryOpen event we wrote some LotusScript to create a profile document in the database that stored the customer number.

2. In the Account database we created a new form. This had one hidden, editable field with no value and an embedded view, pointing to the view in the database that already existed.

3. We modified the views to be a Show Single Category view. This was done simply by entering a formula into the designer’s pane under the Show Single Category section. The formula simply referred to the hidden field that was created.

4. A PostOpen event was added to the form that read the profile document created in the customer database.

5. The Transaction database had similar alterations.

6. Two framesets were created in the Customer database. The first frameset consisted of two horizontal frames (say A & B). Frame A did not have any content defined. Frame B opened our second frameset.
7. The second frameset also consisted of two frames (say C & D). These both opened a named element in the other databases. The element was the embedded view that we just created in the Accounts database and Transactions database.

8. The final step was to tell the form in the customer database to load the first frameset. This was done by setting the launch properties in the form. In the next figure you can see that we ask the form to load the first of the defined framesets. We also needed to give the destination frame for the form to go into. That top frame (frame A) was called Doc.

Does This Actually Improve Performance?
As we have already stated, performance is a complex measurement. Although we can hold a stopwatch to something and press all of the buttons and find that things go faster, that doesn’t mean it will be faster for someone to use. In the above example, we are opening a number of documents at once in a number of databases. This may not be the best approach. However, we are giving the user access to all of the databases that they require in one simple movement. The individual does not even know what database they are currently in, nor do they need to.

Without using framesets, achieving something similar would require us to read information from another database into our customer form as part of the opening. We would need to format this somehow (probably in the form design) and if we needed further information or needed to amend, say, the transactions, then we would either need to access the transactions database in another action or have another agent that would allow us to do this. With framesets, we have direct access to the information that we require in the other application.

This is only intended as an example of how we could use framesets to alter our approach to a multi-database problem that we have already discussed.
Outlines

Outlines are another new feature of Domino R5.0. They allow you to easily create a hierarchical site map for both Notes and Web clients. The key advantage of this from our perspective is that it allows us to bind databases together as if they are one. For example, we can very quickly create an outline that links three databases. The outline could look similar to the figure below:

![Outline Diagram]

Here we see that each database appears simply, like a list of views. We can split the application by content but allow the user to move between the information stored with ease. Another example of how this could be useful is if we split the databases further. For example, each year we might have a different database to store the transactions for that year. The outline then might appear like the following:

![Outline Diagram]

Performance Considerations

Though the use of outlines itself will not directly improve the performance of our application, the use of multiple databases can improve performance. By using outlines we can now connect multiple databases easily in a manner that is seamless to the user. By being able to split these databases, but still give the user of the application the illusion of being in only one Domino database, we aid the developer in building larger, scalable applications that can use different databases for different content.
Resources

Many of the features that were available in previous versions of Domino, but in different areas, have been combined into the Resources area. These include the following:

- Shared Fields
- Script Libraries
- Subforms
- Database Scripts
- Help About and Help Using
- Database Icons

In addition, Domino R5.0 has added a number of new resources, including:

- Images
- Applets
- Shared Actions

The rest of this section describes these new resources briefly, and discusses how they affect application performance.

Images

One of the most appealing concepts of the original World Wide Web (WWW) was the ability to add graphics and other types of multi-media to pages that contain text. If it were not for this, it is unlikely that the WWW would have become so popular: after all, there was already a widely used protocol for retrieving text documents (Gopher). Due to this origin, graphics have become very important in providing applications to users of Web browsers, who have come to expect highly graphical applications. The same is true with Domino applications. Desktop computers and the development of a Graphical User Interface (GUI) have given most computer users the ability to view elaborate graphics, which give applications a very professional look and feel.

It has been possible to use graphics in Domino applications for a long time. Prior to Domino R5.0, when you wanted to use a graphic in a form or in a Navigator, you had to import it into the form, navigator, or subform. If the page was to be displayed to the Web, then you also had the option to reference an image stored elsewhere. This was done using the URL format and could point to the Domino server file system, a Domino database or even to a different Web site.
The Image resource in Domino 5.0, allows you to store images as part of the database design. Generally, these images are used multiple times in your application. The images can be in several formats (for example, JPG, GIF or BMP). When you wish to use an image as part of your application, select Create - Image Resource when you are in the design element where you wish to use it.

The Notes client will cache this new Image resource as part of the CACHE.DSK file. This means that if the same image is used in multiple locations, the version in the cache will be reused rather than downloading the image again.

Web clients cache images as well. They store them in a browser-dependent manner. The browser checks the modified date of the image and if the date of the image on the server is newer than the image on the client, then the image will be downloaded. Prior to Domino R5.0, when an image was stored in a Domino database, Domino would not send the modified date to the client. This resulted in the client downloading the image from the server every time. The alternative to this was to use an image stored on the operating system since this would send the correct date to the client. With the Image resource, the current date is stored and sent to the client so that these images are cached correctly. This improves the performance of pages that are accessed often on your site.

Tip   You can reference image resources that are stored in a Domino R5.0 database from any existing HTML document. The format to refer to the resource is:

\[\text{<img src=http://server/db/resourcename?OpenImageResource>}\]

For example, to refer to an image called “logo.gif” stored in our “home.nsf” database on the Domino R5.0 server called “freja,” we would enter the following HTML:


Applets

The integration of Java into Domino has enabled us to develop many more customized applications for use from both the Notes client and the Web client. Prior to Domino R5.0, an applet was either embedded into each form that used the applet, or it was stored elsewhere and referred to with HTML.

Domino R5.0 allows us to store the files that comprise the applet in a Resource. We can then include this resource in forms and pages. However, unlike the use of Image resources, the applet actually becomes a part of the form. This means that if the applet is changed, then it needs to be put back into the form. In addition, applets are not cached as well as images (though some browsers cache applets better than others).
**Linking to Java Applets**

A better approach to accessing applets is to link to them using a URL. This allows you to access a Java applet any number of times without having to store it in multiple places. This does rely on the fact that the applet must be accessible to all users of the application.

**Shared Actions**

Shared actions are found under Resources - Other. They allow us to create actions (Simple, Formula, LotusScript or JavaScript) centrally that can be used in a number of places without having to copy them to other views or forms. Previously, subforms were often used to include a common set of action buttons on forms.

Shared actions do not appear to enhance the performance of an application; however, they do not appear to hinder it either. When tests were done which examined the remote procedure calls, we found that the amount of information downloaded to the client is the same regardless of whether it is for a shared action or a standard action.

As in other areas in Domino where the design is distributed (for example, using subforms, or shared fields), using shared actions causes a small performance overhead at the server since it must deal with more than one design item.

---

**CACHE.DSK**

The CACHE.DSK file contains design elements (forms, subforms, and LotusScript libraries) of databases located on a Domino server, in addition to cross-replica journaling information. Storing these items in CACHE.DSK helps improve client performance when accessing databases located on a Domino server and helps the process of read-mark synchronization.

As user or administrator you can control the following parameters for CACHE.DSK:

- Size
- Location

As a developer you have no way of tweaking CACHE.DSK directly. However, with knowledge of how CACHE.DSK works there are a few things you can do indirectly. We discuss this after showing how to control the size and location of CACHE.DSK.
Controlling the Size of CACHE.DSK

You can specify the maximum size of CACHE.DSK either through a NOTES.INI setting or through workspace properties. Design elements are removed from the cache based on age when the maximum size is reached. In general, do not go below the default size of 5MB for CACHE.DSK. If you set the size too small, design elements that the user accesses regularly may be swapped out of the cache. This will result in more network traffic because design elements that otherwise would be loaded from the cache will have to be fetched from the server again.

InitialCacheQuota in NOTES.INI
To set the maximum size for CACHE.DSK in NOTES.INI, use:

InitialCacheQuota=xxxx

where xxxx is the file size in bytes (specified in increments of 1024 bytes, for example InitialCacheQuota= 6144.)

Note  In some earlier Notes versions this NOTES.INI file setting was not working correctly. It is working in R4.5.5 and R4.6.2 and a fix is being researched for Notes R5.x.

Workspace Properties
To set the maximum size for CACHE.DSK via the workspace, double-click any workspace tab. The workspace properties dialog box appears. On the Beanie (advanced) tab (the Information tab in Notes R4.x), you can specify the maximum size of CACHE.DSK in MB (default value is 5MB.) However, this setting is stored in the CACHE.DSK itself. If this file is deleted, the file will be recreated the next time Notes starts, with the default size limit of 5MB.

Compacting CACHE.DSK
The CACHE.DSK is actually a Domino database with no built-in forms or views. Therefore, this file can be compacted to recover unused space. The Compact option for the CACHE.DSK is found on the same workspace properties tab where the maximum size is specified. You can click a button to see “% used.” If the percentage is under 85%, click the button named “Compact.”

Controlling the Location of CACHE.DSK
By default CACHE.DSK is placed in the data directory of the Notes client. However, for installations where users have their data directory on a network drive, the use of CACHE.DSK may actually affect performance in a negative way. Using the CACHE= parameter in NOTES.INI file you can specify the location of CACHE.DSK to be somewhere else than in the Notes data directory. In the example below, CACHE.DSK is specified to reside in the TEMP directory on the users C-drive.

CACHE=C:\TEMP\CACHE.DSK
The Developers' Perspective of CACHE.DSK

As a developer you cannot control which design elements are cached in CACHE.DSK or when this happens. However, with knowledge of “what” and “when” you may be able to structure your application to get the best out of CACHE.DSK.

For example, when a form is loaded, all referenced script libraries are transferred to CACHE.DSK as well. If you have designed your application to have most of the logic in a few large script libraries, the result may be a larger-than-needed response time the first time the user opens a form. This is because code for logic used elsewhere in the application (on other forms and so on) is transferred to the workstation when the form is opened. If you can split your code into several script libraries without interdependencies, you may be able to cut down the initial load time for your form because only the script libraries with actually required code are transferred. Still, you may have seldom-used functions on your form, for which you would like not to have the code cached until actually required. If possible, isolate the code you want to defer loading until needed in its own libraries. There is a trick to do this: place the code in an agent because agents are not cached. The user can invoke the agent from a button if needed and the code will only be transferred to the workstation if actually necessary. Caching of script libraries and agents is discussed in Chapter 3, “Programming Considerations.”

In this example we saw how performance might be improved with the knowledge that all script libraries referenced on a form are loaded together with the form, and that agents can be used for seldom-invoked functions because they are not cached.

Caution You should only look a restructuring your design and code to fit the behavior of CACHE.DSK if performance with the existing implementation is an issue or if you suspect it will be. Splitting your code into “cache-able” sections may make general code maintenance much harder. It is only you who can decide whether it is worth the effort.

Finding out What is Cached and When

The implementation of what is cached and when is a black box and may change in any future release of Notes and Domino. If you have a scenario where you would like to know what is cached first, check the information in this book about the functions and design elements involved. If you want to know more, you can trace the Notes Remote Procedure Calls (NRPCs) issued from the client during your scenario. For example, Appendix B-6, “Supporting Roles Locally Without Enforcing Consistent ACL,” explains how tracing of NRPCs revealed that information retrieved by the @UserRoles function is not cached. More information about how to set up tracing NRPCs is in Appendix B-8, “Using NRPC Output.”

140 Performance Considerations for Domino Applications
If you are curious about what is cached at a given time, remember that CACHE.DSK is actually a Domino database. You can check which design elements are cached by opening CACHE.DSK in Domino Designer or inspect it with NotesPeek (see Chapter 5, “Troubleshooting an Existing Application” for information about NotesPeek).

---

**Headlines**

Domino R5.0 includes a new feature called Headlines. This is an exciting feature because it allows a user to “Subscribe” to a database and receive notification when there are updates to that database. This is an end user feature, but it is important to understand the implications of it from an application perspective.

When Headlines is enabled, the Notes client will monitor the databases to which it has subscriptions at a regular interval. This interval is set in the Mail and News section of the User Preferences dialog. The area at the bottom of this screen, which says “Check for new mail every xx minutes” also controls how often the headlines are scanned.

**Note**  Choose File - Preferences - User Preferences to get to the User Preference dialog.

If your application is one that has a large number of new documents being created, allowing the monitoring of headlines in it may be placing a considerable extra load on the client’s computer. The feature is part of the new On Disk Structure (ODS) that Domino uses and, as such, it can be disabled under the Database properties Prevent Headline Monitoring (covered later in this chapter). We recommend that you consider whether it is appropriate to allow users to monitor the database with the headlines feature prior to deploying the application.

---

**Laying Out Web Pages for Maximum Performance**

In this section we show some techniques that can help you to lay out Web pages for maximum performance.

Domino provides many ways to lay out pages, using both HTML and Domino database constructs such as pages, forms and fields. This section focuses on approaches to laying out the appearance of a page using static elements.
Domino Graphics

You can use several techniques to improve application performance when dealing with graphics in a Domino database.

Use 256 Color Bitmaps

It is best if you keep image file sizes as small as possible. The more colors you use, the larger the size of the graphic file, and the more time required to transmit the page. By choosing 256 colors you will be improving performance.

Tip  Limit bitmap graphics to 16 colors and use low-resolution (72 dpi) images for smaller files and better performance across multiple platforms. If you use 256-color bitmaps, use no more than one per form and use the Lotus color palette to create the bitmap.

Store Graphics in Their Native Format

Domino R5.0 allows you to store graphics in their native image format (for example, GIF and JPEG). This greatly improves performance when dealing with these images. In previous releases, images were converted to bitmaps when stored in a Domino database. This required conversion back to GIF format when those bitmaps were delivered to the Web. In Domino R5.0, no conversion occurs when a graphic is already stored in its native format, which greatly reduces the overhead.

Note  If you are upgrading a Domino 4.x application, you will have to reinsert the graphics in order for them to be stored in their native format. However, keep in mind that those graphics will not be viewable from a Domino R4.x client.

Domino Images

Graphic images can also be stored apart from the design items that use them. This is another way we can maximize performance for Web clients.

Use the Server's File System

When images are stored on the file system of the operating system, Domino does not need to convert them prior to sending them to the Web client. This uses server disk space more efficiently than using imported or pasted images does because an image can be used more than once. In addition, since the Modified date of the image is sent to the Web browser, the browser can compare this date to the date in its cache and download the image only if it is newer.

Use passthru HTML to access the image using syntax such as:

<IMG SRC="banner.gif">
The downside is that images must be maintained on every server that uses the images. This can be achieved by using an application to manage the distribution and maintenance of these files.

**Use a Graphics Library Database**

You can store images in a graphics library database as attachments to documents. This configuration causes Domino to cache images. Caching makes performance better than using imported or pasted images for frequently used pages.

Before the image can be passed on to the Web client, however, the file attachment must be detached into the cache directory on the Domino server. This cached image is then passed to the client. This obviously will be slower than storing images directly on the operating system for the initial access, but will improve performance for subsequent requests.

**Note** If the image is stored in a Domino database, Domino does not pass a modified date to the Web client. It will therefore always download the image. Compared to imported or pasted images, this configuration uses server disk space more efficiently because an image can be used more than once.

Here is how to access the attachments in the library:

1. Create a view to list the images.
2. Use passthru HTML and URL commands to open a document from the view by the key name.
3. Open the image, which is an attachment in that document.

   The following tag displays the image in the “Animation” view using the key “gifs” in a different database on the same server:

   `<IMGSRC="/webimage.nsf/animation/gifs/$file/flag.gif">`

---

**Agents and Other Processing**

A number of tasks need to be accomplished outside the user experience. For example, the user does not want to be involved with monthly processing or housekeeping; they just want it to happen.

Agents of many sorts are employed to handle this “off-line” type of processing.
Domino R5.0 Agent Enhancements

There have been a number of enhancements made to agents with the release of Domino R5.0. Some of these can help your application with performance.

**Enhanced Agent Scheduling**

Agent scheduling has been enhanced in two areas:

1. Agents can be scheduled to run as frequently as every 5 minutes.
2. Agents can be scheduled to start at any time.

“So how does this improve my application performance?” you might ask. Well, since you can run the agent more frequently, you can better balance the workload. The amount of data required to be processed per agent run will be reduced, thus spreading the load.

An even better way to spread the load, however, is by scheduling your agents to start at different times within the hour. Instead of all agents starting at the same time (rounded to the nearest 30 minutes), they can now be started at any time. Again we get a good load spreading which improves overall application performance.

**Note**  There is an overhead associated with running an agent (the Agent Manager must be loaded). Make sure the gains made by load spreading are not overshadowed by the performance hit of running the Agent Manager more often.

**Java Agents vs. LotusScript Agents**

In Domino R5.0, Java agents and LotusScript agents are coded directly within the Domino integrated development environment. Both types of agents can carry out actions of arbitrary complexity, including the calling of other agents, before passing control back to the user at the interface level.

These agents also:

- Employ the same Domino user interface paradigm for their creation and management
- Pass parameters via fields within documents
  
  **Note**  Java applets, by contrast, pass parameters within the header that launches the applet itself, or use JavaScript and LiveConnect.
- Enable users to specify public sharing of agents
LotusScript or Java?
Which type of agent is best to use? The answer depends on what needs to be done.

1. If possible, use a simple action agent. Why maintain code when simple actions are built into Domino?
2. If you know only LotusScript and it can do the job, then use LotusScript.
3. If you know only Java and Java can do the job, use Java.
4. If your agent has to access a network, or would benefit from multithreading, use Java. An example where multithreading really can speed up your code is an agent that processes a huge collection of documents.
5. Use Java if you require multiple instances of the same agent.

LotusScript programming is very well supported in Domino Designer. You can access the Domino back end as well as the front-end classes from LotusScript. Java only gives access to the Domino back end classes. You can argue that while LotusScript is very good for pure Domino solutions, Java offers more support for solutions that involves other systems through its built-in networking support and Java Database Connectivity (JDBC) support. For example, with the built-in networking support in Java you can grab and parse HTML or XML data from a non-Domino server and use the data in your agent. Accessing relational data via JDBC from Java agents in Domino may scale better than through the ODBC interface in the LotusScript Data Object (LS:DO).

Tip Keep Scheduled Agents, Selection Formulas and Replication Formulas as simple as possible. When writing the code, consider how frequently it will be executed. The more complex the code is, the more time it will take to execute it and the higher the load is on the server.

Tip For complex databases or for large data sets, use views with selection formulas to select the information that you wish the agent to run on. The indexing tasks on the server will select the documents for you and thus the Agent will not need to perform a selection of its own, which could take considerable time.

Tip When you’re using Java agents you can share the libraries to save database space (javaUserClasses = <path>).

Tip You can execute an Agent to run on the server by using the RunOnServer method (since Domino R4.61). This will start an agent on the server to perform the tasks and hence there will not be the same network traffic.

Note Besides these tips, refer to “Language Performance Aspects” in Chapter 3 before writing your code.
Applets

There are some advantages to using Java applets. Applets are very useful to provide a richer UI for the end user. Applets are automatically downloaded from the server and run on the client’s machine. Applets will work in either the Web browser or on the Notes client.

Although the load time for a Java applet can be noticeable (depending on your network connection and size of applet), they process on the client side, reducing the server load. For an application with many users, this can improve overall performance.

Applets are cached at the client, so multiple uses of the applet will not incur download overhead.

The applet must communicate with the server to request reads or writes to the Domino database. The server side of the application can also be written in Java and can use the Java Classes for Notes Object Interface/Domino to read and write to the Notes databases.

Note By default, applets do not have access to the Domino back-end classes. If this is required, then you need to use the CORBA interface.

Servlets

Java servlets, as their name suggests, only run on the server. A servlet is invoked by a client request and will respond directly to the client. Every time a Domino agent is run it must be loaded, executed and then removed from memory. A servlet, once loaded, remains in memory. Typically, a servlet will be used to provide a high-performance link to a back-end system and format the results back to the client as an HTML document. However, servlets are not restricted to serving just HTTP requests and may, in fact, converse directly with any suitable client application (usually an applet).

• Servlets are not secure or functional as agents.
• Servlets must live in a file system.
• Servlets run with server privileges.
• Servlets can access Domino objects, but have no idea who the Domino user is.

Access to the servlet can be controlled by file-protection documents in the Domino Directory. If the servlet accesses Domino through the CORBA interface, it can specify a Domino user name and Internet password. Domino security applies to all CORBA operations.

Note In controlled tests, there have been instances when servlets were up to six times faster than agents.
Other Options

In some situations, you will not be able to solve your problem using the confines of Domino. In other cases, using Domino may not provide the performance required, even after your best efforts at maximizing for performance. In these cases, you may be forced to look beyond Domino.

If you are experienced with the C or C++ language, you can take advantage of accessing Domino from another layer.

DSAPI
The Domino Web Server Application Programming Interface (DSAPI) is a C API for writing your own extensions to the Domino Web Server. A DSAPI extension, or filter, is a program you create that is notified when certain events occur on the Web server, such as when a URL request is received or when a user is about to be authenticated.

For example, you might choose to write a program that performs custom authentication, which is often one part of implementing single signon within a corporation. In this scenario, the DSAPI program could be notified when Domino is about to authenticate a user. The DSAPI program could then parse the user name, check the user name and password against a legacy mainframe system, and if successful, notify the Domino Web server that it has handled the event (authentication). It then could pass Domino the distinguished name of the user.

For more information, see “DSAPI” in Chapter 6 of Lotus Notes and Domino R5.0 Security Infrastructure Revealed, order number SG24-5341.

Extension Manager
The Domino extension manager is a facility that allows an executable program library (for example, a DLL) in an application to work in concert with Domino. The DLL can register callback routines with Domino for certain events. When the registered event occurs, the DLL will be called by Domino (either before the event, after or both).

This facility can be used to allow API applications to perform database shadowing or add additional access controls, just to name two possibilities.

Database Performance Techniques

This section discusses the various database settings we can use to optimize database performance. A fair amount of the information in this chapter is new to Notes Release 5, particularly many of the database properties, so details of actual field usage are scarce.
This section also describes some techniques that can be used to improve performance in your applications.

**Database Archiving**

As mentioned in Chapter 2, “Design Considerations,” database archiving is something that must be considered so as to maximize database performance after an application has been deployed for some time. In all likelihood, your application will start to fill up with data that is no longer useful, is out of date or at least not essential. Archiving such data can speed up your application. The Notes Release 5 facilities for database archiving can help in this situation.
Use the Archive Settings: Basics dialog, shown in the previous figure, to define which documents are to be archived and the location of the archive database.

The Archive Settings: Advanced dialog allows you to set archiving as manual or automatic, and also gives more options on archive logging and archive document deletions.

Maintaining Links to the Archived Documents
The built-in archiving in Domino R5.0 uses a "copy and delete" approach. If you want to keep the size of your application database down through archiving and at the same time keep links in the application database to the archived documents, you have to develop your own archiving mechanism. There are many ways to do this. Here is a conceptual description of one of these ways:

1. Create separate archive database or databases to store copies of the documents from the production application.
2. Add a new form called "ArchivedDocument" to the production database with single rich text field called "ArchiveLink." Set the form property named "Auto Launch" to "First Document Link."
3. Create an agent that finds documents that are to be archived based on a criteria you define and copies them to the destination archive database.
4. Remove all items (fields and so on) from the document in the source database, but do not delete the document (if you require access to the document from views then you may still leave some items available for view indexing).

5. Create a new Rich Text item on the document in the source database called “ArchiveLink” with document link to the archived document.

6. Change the Form field to “ArchivedDocument.”

Now when an archived document is accessed, the user will seamlessly be taken to the document in the Archive database. However, the limitation with this approach is that the document is never removed from the production database, but as its size will now be very small and it can easily be excluded from most views, the impact will be minimal. It also has the advantage of archive database rollover, where the destination archive database can be changed depending on a defined criteria such as size or date.

Database Encryption

Depending on your security requirements and the environment in which your application will be deployed, database encryption is one option that you may consider.

For Domino databases, there are four levels of data security available. The following table outlines the security level and performance implications for each level.

<table>
<thead>
<tr>
<th>Encryption Level</th>
<th>Description</th>
<th>Compression</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No data encryption. Can be used when the data has no security restriction or if the environment is already secure.</td>
<td>Yes</td>
<td>No effect</td>
</tr>
<tr>
<td>Simple</td>
<td>Minimum level encryption is performed. Use simple encryption if you want to take advantage of disk compression and when security needs are not great.</td>
<td>Yes</td>
<td>Very small effect</td>
</tr>
<tr>
<td>Medium</td>
<td>Medium encryption is probably the best choice for most users because it balances security, strength and fast database access.</td>
<td>No</td>
<td>Small effect</td>
</tr>
<tr>
<td>Strong</td>
<td>Use strong encryption when security requirements are paramount, and the resulting database access performance is acceptable. Database access is slower with strong encryption than with the other encryption options.</td>
<td>No</td>
<td>Noticeable effect</td>
</tr>
</tbody>
</table>
**Note**  When using medium or strong encryption, you cannot use file compression for the database. This can have performance implications in certain environments.

**The Domino R5.0 Database Internals**

A number of improvements have been made to the format of the database (the On Disk Structure - or ODS), as well as to internal operations of the Domino code, which will help performance. Here we define most of these features, although you as a designer have little control over them.

**On Disk Structure (ODS)**

The format of the database has been changed to optimize I/O and CPU usage for mail and other application workloads. When you add mail, for example, there will be fewer things to write to disk. When writing to disk, a smaller number of writing operations takes place in Domino R5.0. The result is approximately half the I/O rate of Domino R4.x databases for the same set of operations. In some cases, it’s actually much less than that. The new ODS is also what allows in-place compaction and transactional logging recovery to take place (we’ll talk about those later).

The performance enhancements include:

- Bucket-based allocations and storage management
- Page-oriented handling of database metadata
- Reduced I/O for DB open/close (from 168K to 40K)
- Key data is not stored redundantly in view
- Document Bitmap Table
- Faster design information retrieval (uses internal table)
- Better memory management
- Universal Buffer Manager (more caching)
- Soft note delete

**Database Compaction**

With the new ODS, Domino can now do online, in-place compaction.

Prior to Release 5, when a database was compacted, two restrictions applied:

- The database went offline. That is, no one could use the database while it was being compacted.
- The compaction was done by copying the database to a new version, removing all the unnecessary space as it went along. After this operation it deleted the original and renamed the copy.
With the new ODS, the server can compact a database online (meaning users can continue to work in the database) and in-place (meaning that your server doesn’t require extra disk space).

**View Rebuild Via External Sort**
The UPDALL task is used to rebuild view indexes. This process has been improved by allowing the use of an external B-Tree sort. A new NOTES.INI file parameter is used to specify the location of the external sort data. Therefore, the sorting workload can be maximized by spreading it across multiple disk drives.

Specify the rebuild directory via the new INI file variable:

```
VIEW_REBUILD_DIR="C:\TEMP\VIEWREBUILD"
```

**Transaction Logging**
This new Domino R5.0 feature allows database transactions to be logged as they occur. Many of you are now thinking “Oh, more overhead, worse performance.” Well, in fact, just the opposite is true. Logging means that Domino can be a lot less conservative about database writes. Prior to Release 5, Domino had to be very careful about writing to disk to ensure no loss of data. With transaction logging, Domino is ensured of being able to recover from a crash in the middle of I/O, so it doesn’t have to “flush to disk” as often.

The end result is far fewer disk I/Os for the Domino code (even though we add some I/O for the actual logging). In performance measurements, it was estimated that there is a 10 percent performance gain when you turn on performance logging.

**Note**  It goes without saying the logging should be to a separate disk.

**Domino R5.0 Scaling Enhancements**
The following enhancements have been made to improve the scalability of Domino. Knowing these new Domino R5.0 features may influence your design decisions.

**The Domino Directory Catalog**
The Domino Directory Catalog is a compressed version of one or more Domino Directories. This improves the speed of name lookups and name resolution for all organizations. For example, if a user addresses a message to “Julia Smith,” Domino can quickly fill in her correct address.

You can also use this same Directory Catalog for mobile users to access locally for type-ahead, name lookups, and local LDAP searches.

The Directory Catalog is highly scalable, but has a very small footprint. As an example, the combined Iris/Lotus/IBM Directory Catalog takes up less than 2 percent of the space required by the equivalent Domino Directory.
To keep a Directory Catalog small, entries contain only a subset of the full record, with the default being the minimum set of attributes necessary for mail routing.

**Messaging**
The way Domino works today, there is one database called MAIL.BOX, where all mail is deposited. This can become a bottleneck, especially if you’re running a lot of users or if you want to compact the database. With Notes Release 5 you are allowed to have any number of MAIL.BOX databases. This allows you to spread the load and the contention over as many databases as you wish.

**Database Size**
The new database architecture includes a change in how pointers are stored inside the database. This will allow individual databases of over one terabyte in size in the near future. For Release 5.0, databases are certified up to 64GB, versus the Release 4.0 limit of 4GB.

**Users per Server**
The average customer today has between 100 and 1,000 active users on a single server. With Domino R5.0, the current analysis indicates that the server will be able to support five times that many users and still maintain the current service levels. This number is expected to increase with tuning.

Of course, the real measure of scalability is not just the number of users per server, but the total infrastructure needs of a large organization. That means multiple directories, distributed administration, flexible authentication/encryption, and replication/routing technologies that remain bulletproof over any network topology limitations. These are equally important things that people should be looking at when they make comparisons.

**Domino Enterprise Connection (DECS)**
Domino R5.0 includes the new Domino Enterprise Connection Services (DECS) for building live links between Domino pages and forms and data from relational databases. To set up the links, you simply use the DECS template to identify the forms and fields in your application that will contain external source data, and to define the real-time connection settings. You can set up connections for DB2, Oracle, Sybase, EDA/SQL, and more. A Domino server add-in task passes the real-time connection instructions to the Domino Extension Manager, which monitors the server for your user-initiated events. When events are intercepted (such as a query for data), the Extension Manager transfers the query to the external source, which performs the query on behalf of the end user. Results are presented to the user in real time, as if the data were stored natively in Domino.
Clusters
Domino R5.0 supports clusters for Web clients, which means that you can provide increased availability and scalability for your Web applications. Not only will Web browser requests fail over to another server in the cluster when one server goes down, but Domino clustering provides load balancing across all servers in the cluster. The Internet Cluster Manager (ICM) is the new Domino server task that acts as an intermediary between the Web client and the servers in the Web cluster. Web clients direct requests for databases to the ICM, and then the ICM determines the best server to receive the request based on the server availability and workload. Since cluster replication keeps all database replicas synchronized, the replicas appear to the user as a single, highly available database.

Database Properties that Optimize Performance
This section describes the various database properties which can be used to optimize database performance. If all (or most) databases on a server have their properties set for maximum performance, the server itself has better performance. Although some properties are server related, many are under the control of the database designer.

Database size is one area for performance gains. We noticed this with views, forms and many other areas of Domino applications: smaller is better. Even with Domino R5.0, smaller databases perform better than larger ones.

Database Basics Tab
Choose File - Database - Properties to access the Database Properties box. The box opens to the Database Basics tab by default:
Prevent the Use of Stored Forms
To ensure that a document always displays correctly, you can store the form with the document. For example, if a document is sent from one database to another and the second database does not contain the required form, the stored form is available to correctly display the document.

Storing a form with every document, however, uses system memory and may require as much as 20 times more disk space than not doing so. Design your application so stored forms are not required.

To prevent the use of stored forms, deselect the Database Basics property Allow use of stored forms in this database.

**Note** Before preventing the use of stored forms in an existing application, make sure you understand how this design feature works and if the database uses it.

**Tip** If stored forms are used for circular mailing purposes, as is common in workflow authorization, you can write an agent to remove the stored form when it returns to the originating database. To do this, use the following steps:

1. Create a shared agent.
2. In the When should this agent run? selection box, select the option After New Mail Has Arrived (for Domino R4.x this option is called If New Mail Has Arrived).
3. Enter the formula below:
   ```
   SELECT $TITLE="Form Name";
   FIELD $TITLE:=@DeleteField;
   FIELD $INFO:=@DeleteField;
   FIELD $WINDOWTITLE:=@DeleteField;
   FIELD $BODY:=@DeleteField;
   FIELD FORM:="Form Name";
   ```
   where the “Form Name” is the name of your form.
Display Images After Loading
To quickly display documents that contain images, select the Database Basics property Display images after loading. When this is selected, Notes users can read the text while the images load.

If you don’t load images after text, Notes loads images in the order in which they appear in a document; if an image appears first, Notes loads it before displaying text. With large images or slow connections, loading images in order may slow the display of the document.

This setting applies only when using the Notes client to view databases. Web browser settings control the display of images to Web browser users.

Tip Users also can specify Load images: On request in the Advanced section of a Location document to display images only when users click them. For more information, see “Displaying Web Pages containing images” in the Domino R5.0 online help.

Advanced Tab
Most of the database performance options are accessed by selecting the Advanced tab of the Database Properties box:

Note Most of these setting are turned off by default.

Don’t Maintain Unread Marks
Maintaining unread marks in a database requires system resources and can significantly slow database performance. For some databases, unread marks aren’t useful. For example, reference databases such as the help databases provided with Domino, administration databases such as the Domino Directory, or databases such as the log file (LOG.NSF) that are continually updated have no use of unread marks.
In these types of databases, consider disabling unread marks. To disable unread marks, select the Advanced database property Don’t maintain unread marks.

If you select or deselect the Don’t maintain unread marks property, you must compact the database before the setting takes effect. Compacting in this case makes a temporary copy of the database, so your system must have the disk space to make the copy.

**Note** Designing views that do not display unread marks does not improve database performance.

**Use Document Table Maps**

When updating a view, Domino refers to tables of document information. These tables are stored internally in the database. By default, during view updates and rebuilds, Domino searches each table for documents that appear in the view being updated.

To update views more efficiently, select the Advanced database property Document table bitmap optimization. This property associates tables with the forms used by the documents the tables contain. Then during a view update, Domino searches only the tables associated with the forms used by documents in the view being updated. This significantly improves the performance of view updates, especially updates of small views within large databases.

**Note** This property only works for views that use Form= as part of the selection criteria.

There is a slight performance cost to maintaining the table/form association; however, when updating small views in large databases, the benefits offset the cost.

If you select or deselect the Document table bitmap optimization property, you must compact the database so that the setting takes effect. Compacting in this case makes a temporary copy of the database, so your system must have the disk space to make the copy.

**Prevent Overwriting of Deleted Data**

When data is deleted from databases, Domino, by default, overwrites the deleted data on disk with a pattern. This pattern prevents an unauthorized user from using a utility to access the data. This overwriting causes disk I/O and can affect database performance.
Preventing the overwriting of deleted data is appropriate in these circumstances:

- The data is already secure.  
  For example, the database is on a server in a locked room.
- Deleted space in the database is constantly reallocated.  
  For example, in a system database such as MAIL.BOX.
- Data security isn’t an issue.  
  For example, in an informal discussion database.

To prevent the overwriting of deleted data, select the Advanced database property “Don’t overwrite free space.”

**Don’t Maintain “Accessed in this file” Information**

All documents in a database have a property “Accessed (In this file).” This property shows the date a document was last modified or read in the database.

The database property Maintain LastAccessed property controls whether the document last accessed property is updated if the last access to the document was a read. Maintaining this document property causes disk I/O that wouldn’t otherwise occur.

By default, the database property Maintain LastAccessed property is not selected, meaning the document Accessed (In this file) property is not updated when the last document access was a read. Only when the last access was a document modification will the property be updated.

You can change the default behavior by selecting Maintain LastAccessed property in the database properties.

**Note** If you are planning to use the document archiving facility, you must maintain last access information. The document archiving tool deletes documents from the database based on days of inactivity. Document archiving settings are available in the Database Properties box.

**Disable Specialized Response Hierarchy Information**

By default, every document stores information that associates it with a parent document or a response document. Only the @functions @AllChildren and @AllDescendants, which are often used in view selection and replication formulas, use this stored information. Maintaining this information has a significant, negative effect on database performance.
To improve database performance, disable the response hierarchy information in databases that do not use these @functions. This is done by selecting the Advanced database property Don’t support specialized response hierarchy.

**Note** Disabling the response hierarchy information has no effect on views and replication formulas that display information hierarchically without using @AllChildren and @AllDescendants.

If you select or deselect the Don’t support specialized response hierarchy property, you must compact the database so that the setting takes effect. Compacting in this case makes a temporary copy of the database, so your system must have the disk space to make the copy.

**Prevent Headline Monitoring**
Users can set up headline monitoring to automatically monitor databases for information that interests them. Enabling a database this way affects performance, especially if many users do this.

To prevent users from monitoring a database, select the Advanced database property Don’t allow headline monitoring. Administrators can also use the Security section of a Server document in the Domino Directory to control headline monitoring at the server level.

**Limit the Size of $UpdatedBy Fields**
Every document includes an $UpdatedBy field that stores, by default, the name of the user or server associated with each document editing session. Storing a complete edit history consumes disk space and slows view updates and replication.

To conserve disk space and improve database performance, use the Advanced database property Limit entries in $UpdatedBy fields to specify the number of entries that the $UpdatedBy field can contain. When the $UpdatedBy field reaches this limit, the oldest entry is removed to make room for the newest entry.

**Limit the Size of $Revisions Fields**
Every document includes a $Revisions field that stores, by default, the date and time of each document editing session. Domino uses this field to resolve replication or save conflicts that occur when two users simultaneously edit the same document on one replica or edit the same document on different replicas between replications.
By default, the $Revisions field stores a history of up to 500 edit sessions, each of which requires 8 bytes of disk space. Over time, $Revisions fields can grow large, taking up disk space and slowing view updates and replication. To conserve disk space and improve database performance, use the Advanced database property Limit entries in $Revisions fields to specify the number of entries that the $Revisions field can contain. When the $Revisions field reaches this limit, the oldest entry is removed to make room for the newest entry.

Consider limiting the entries in $Revisions fields on a database with all of the following characteristics:

- The database contains many documents.
- The database replicates often or has no replicas.
- The database contains documents that are not often edited.

A suggested limit is 10 entries. If you set the limit lower than 10, you run the risk of increased replication or save conflicts.

**Allowing More Fields in a Database**

For a Notes Release 5 database, you can select the advanced database property Allow more fields in database.

Without going into the pros and cons about using this property, you must keep in mind that more fields will likely impact database performance in a negative way.

---

**Summary**

There is a lot of choice when designing a Domino application. Many design elements can be used in a variety of ways to build your solution. It is wise to have a good understanding of all the facets of the design elements, both for Notes clients and Web browsers, before committing the details of the design.

Performance is a major concern for application owners and as such, needs to be considered right from the beginning. The next chapter will help you if you are in the unfortunate position where the application is already deployed but performance is not up to standard.
Chapter 5
Troubleshooting an Existing Application

This chapter describes methods for investigating the common causes of degraded application performance. First, we discuss the various external factors that may mislead you to believe that the performance of the application is to blame. If it is determined that the application is, in fact, the cause of the degraded performance, we then discuss how to find the performance bottlenecks.

After reading this chapter, you should be able to diagnose specific bottlenecks in performance. If the problem is with the application, you will know how to recognize the problem and have some possible ways to fix it.

What Factors Affect Performance?

There are a number of factors that contribute to the overall performance of an application, including the following:

- Network
- Server
- Client
- Usage patterns
- Application

It is important that you first discover from where the poor performance is originating. You will usually need to gather information from a number of sources in order to pinpoint the location of the bottleneck. Often there will not be a single area, but a number of areas that work together to produce poor performance.
Network Performance
There are many different standards for network bandwidths around the world. They vary from high speed Local Area Networks (LANs) to slow dial-up Wide Area Networks (WANs), such as the Internet. Often the development and testing of an application is done in a confined environment, where the application behaves differently than when it is opened up to the rest of the user population. For example, the time needed to download an Applet when testing in an office environment may seem insignificant, but when a user attempts to download it from a slow dial-up connection they may give up and choose not to use the application. In short, a WAN magnifies network-related problems and can highlight areas that may have been overlooked during prototyping and development.

Server Operating System Performance
The specification and configuration of your server plays a large role in the performance of Notes and Domino applications. A server can be tuned to help maximize the performance of an application, or it can be analyzed to help identify issues in the performance of an application.

The system administration and architecture of Domino is a large topic in its own right. It is not the purpose of this section to identify and resolve all server-related issues, but we will point out some common problems that can easily be identified.

Each operating system on which Domino runs has its own platform-specific tools and techniques that can be used to identify performance indicators.

Domino Server Performance
In addition to performance information that can be provided by the operating system, Lotus Domino also provides a number of other resources for information about Domino server performance.

Client Performance Issues
As applications become more complicated and as users become more competent, extra load is being placed on the client. It is important to identify whether the client (where the performance problem is experienced) could itself be the cause of the problems.

Application Usage Performance Issues
Sometimes the problem might originate with the user, rather than the client. While testing an application to see if it performs adequately, the tester will generally adopt specific styles of using that application. When a different person uses the application, it is possible that the way in which they use the application will also be different.

When people use applications in different ways it is possible to discover problems that would not otherwise be apparent.
An example of the user’s habits affecting performance is when a button that performs a `@DBLookup` is placed on a form. The tester would know that since it is a button it only needs to be pressed once, but a user who is new to computers might believe that it needs to be double-clicked. Double-clicking the button may cause the formula to be executed twice. An additional danger is that if the user does not receive a quick response back from the server they will often repeat the procedure. By this operation they may not only be impacting themselves, but other users of the system may experience degraded performance as well.

Another example is where it is expected that the application will be used at a specific point in time for one step, then at another point in time for another step. If the actual use is different, the clash of use could cause a number of performance issues. This was experienced by one organization in their employee appraisal application, where it was expected that the use of the application would be as follows:

- Employee enters self-evaluation in one session.
- Employee sends evaluation to manager for approval.
- Manager receives notification that evaluation is ready for approval and files it so that they may review a number in one session.
- At a later time, manager accesses the system and approve all evaluations in a single session.

In fact the actual usage pattern was as follows:

- Employee spends a number of sessions creating and reviewing their evaluation.
- Employee sends evaluation for approval.
- Manager receives a message that they have an evaluation from a worker and immediately accesses it.
- The manager may approve it right away, or wait for a few to arrive to compare them or gather more information.

This difference in the way the application was used placed considerable extra load on the system.

Application usage performance issues can often be rectified with adequate user training, as in the first example. However, when the performance problems are due to the way that the application is used, the application design and system architecture must be reviewed. It is important to identify which type of issue is causing the problem.

To isolate application usage performance issues, we can establish the application usage patterns by viewing the Notes logs, database properties, or the Billing records.
Application Performance Issues
Often applications can be recognized as the key factor in performance problems. This is generally verified by reviewing a number of information sources, such as the Server performance monitor tools and Notes logs.

Once an application has been identified as the cause of a performance problem, there are a number of ways to establish which area of the application is causing the problem (if there is a specific area). In some instances it is the application as a whole that is a problem; for example, when there are a lot of people accessing the application at the same time. The actual nature of the problem can be determined by some basic troubleshooting.

The common ways in which an application is identified as having a performance problem are as follows:

- User reports slow performance.
- Admin notices that the Indexer is constantly updating views in the database.
- The `show dbs` console command in Domino R5.0 reports that there are a large number of people “waiting” for the database.

Tools and Techniques for Troubleshooting
Troubleshooting of any application is a very logical process. However, as with most processes, knowledge of common problem areas saves considerable time and frustration. When using tools, it is important to recognize when there are possible performance problems and when they are pointing you to look at something else.

In this section we will discuss some of the tools that are available, how they can be used for evaluating different aspects of performance, and how they can help to determine whether or not the application itself is the source of a performance problem.

Network Performance
A number of very basic tools can be used to test network response time. These tools are very effective, and also have the advantage of being available on a number of platforms.

Ping
The ping utility is used to test whether TCP/IP connection can be established between two computers. This is a generic tool that works on most platforms; however, the flags below apply specifically to Windows NT.
A number of common flags can be given to the ping command to make it more useful, including the following:

- `-l length` is used to send a packet. By default the ping command only sends small packets, but it is often useful to send larger packets, since this is often a cause of network problems.

- `-f` is used to ensure that the TCP/IP packet is not fragmented on route. Gateways will often fragment packets to optimize the routing performance, but this may be a problem for some applications, particularly if tunnelling or an encrypted channel is used.

An example of the ping command is:

```
ping www.lotus.com -l 1000
```

This command might yield the following information:

```
Pinging www.lotus.com [198.114.66.50] with 1000 bytes of data:

Reply from 198.114.66.50: bytes=1000 time=8ms TTL=250
Reply from 198.114.66.50: bytes=1000 time=7ms TTL=250
Reply from 198.114.66.50: bytes=1000 time=7ms TTL=250
Reply from 198.114.66.50: bytes=1000 time=7ms TTL=250

Ping statistics for 198.114.66.50:
  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
  Approximate round trip times in milliseconds:
    Minimum = 7ms, Maximum = 8ms, Average = 7ms
```
It is important to check the time taken. A long period of time could indicate that there is a very slow connection; a large difference between the minimum time and the maximum time could indicate an intermittent network problem. The length of time that is taken is dependent on the type of network between you and the destination that you are pinging. For example, times of up to 10ms are very good for a local area network and times of around 250ms are adequate for a dial-up connection to the Internet. More importantly, if the times vary significantly between different instances when you run the ping tool and even during a single ping session, you need to investigate the network performance further.

You can also try ping with increased packet sizes. If you see a degradation in response time very quickly as you increase packet size you may have a bad network cable or NIC (network card).

Entries such as:

*Request timed out.*

indicate that the destination could not be reached, so there is a network problem. Very often, the problem is the client network setup.

**Netstat**

Netstat is a Windows 32-bit utility used to display statistics about the current network connections and what state they are in. It can be set in a loop to check for connections every n seconds. This can be very useful in checking the status of a connection to a server, or on the server it can be used to see who has a connection to it.

The format is as follows:

```
netstat 5
```

where 5 is the number of seconds between polls. If the number is omitted it will not repeat.

An example of the output that you would expect from the netstat command is shown below:

![Netstat Output](image)

166 Performance Considerations for Domino Applications
This output shows all of the connections that are active from the requesting computer when the command is run. This can be useful in seeing where the computer is connecting to gather information and what the status of that connection is.

**Tracert**

The tracert command is a Windows 32-bit command that is used to show the route that a TCP/IP packet takes to reach its destination. This is often very important in troubleshooting, since two clients will often take different routes to the server and each may have its own characteristics. For example, if the number of hops required in the live environment is more than in the test environment, it is possible that the client will time out and not process a request. Another potential problem is that the route the client attempts may not allow the type of traffic that you are sending over it.

The format is as follows:

```
tracert www.lotus.com
```

where `www.lotus.com` is the destination computer. The output would look similar to the following:

```
Tracing route to www.lotus.com [198.114.66.50]
over a maximum of 30 hops:

1  1 ms  1 ms  1 ms  Cl.test.com [92.9.32.9]
2  2 ms  2 ms  1 ms  R4.test.com [92.5.4.11]
3  *     *     *     Request timed out.
4  *     *     *     Request timed out.
5  2 ms  3 ms  2 ms  www.lotus.com [198.114.66.50]
```

Trace complete.

Here we can see the route that has been taken from the client to the destination and the time taken for each hop. This will often provide useful information if there is a significantly different response time either between two servers or between two different clients accessing the same server. If the routes are different, this could indicate that the difference in the network is a cause of the problem.
Use of Dial-up Networking
The use of dial-up networking (DUN) and a modem can provide some very interesting information. It is useful for testing a worst case scenario, as we can specify the maximum speed at which the modem will connect and this will generally be slower than the speeds that we could expect from a Wide Area Network. It can be used to simulate what a user may encounter when attempting to access the application using a very slow network or unreliable connection.

The throughput counters that are part of DUN can provide useful information for both planning applications and judging whether they are causing network overhead problems.

There are a number of additional ways to test the network, but often they require special tools or specific network knowledge which is not within the scope of this book. The above commands are useful for the initial diagnosis of network issues, but are not inclusive. For further information on troubleshooting network problems, refer to the appropriate network-specific documentation.

Server Operating System Performance
Domino is a multi-platform product. The platforms available for recent Domino releases are as follows:

<table>
<thead>
<tr>
<th>Version</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domino R4.5</td>
<td>Windows 32, OS/2, NetWare NLM, AIX, Solaris, HP-UX, AS/400, S/390</td>
</tr>
<tr>
<td>Domino R4.6</td>
<td>Windows 32, OS/2, NetWare NLM, AIX, Solaris, HP-UX, AS/400, S/390</td>
</tr>
<tr>
<td>Domino R5.0</td>
<td>Windows 32, OS/2, AIX, Solaris, HP-UX, AS/400, S/390</td>
</tr>
</tbody>
</table>

Each operating system has a different set of tools that need to be used to monitor its performance and limits. A similar set of factors are evaluated on each platform. These include the following:

- Memory — Has the server got enough spare memory? Have specific Domino tasks claimed a lot of memory?
- Swap/Virtual Memory — Virtual memory is slower than physical memory. If a lot of virtual memory is being used it is possible that the server does not have sufficient memory.
- Disk — Is there enough disk space? Is there a large amount of Disk I/O activity and a lot of cache being used?
- CPU — If a process is consuming a lot of CPU time, then this is an indicator that there could be a server issue.
These factors may indicate an issue that could either be related to server performance or to a poorly designed application. For example, if the indexer is constantly running this could be a server issue (not enough indexers running on the server), or an application issue (a poorly designed view that requires a lot of indexing).

Some tools available for each operating system are described in the following section.

**Windows NT**

Window NT comes with a number of standard tools that can be used to identify performance problems with the server. Some of these also identify Domino-specific performance issues. The tools include the following:

- **Performance Monitor (perfmon)**

  Used to monitor the usage of different elements of the computer running Windows NT. It presents the information in either a graphical format or as a report. You may also set alerts if specific thresholds are reached.

  In addition to the standard resources that are monitored, when Domino is installed it is also possible to install Domino Performance Monitors, which give you the ability to monitor Domino statistics through the NT Performance Monitor. This can be used to provide a graphical comparison between your system and standard Domino statistics, or between Domino statistics and any of the other statistics that Windows NT monitors. For more information on Domino statistics, see the *Domino Administrator’s Guide* (available online with Domino Administrator or as part of R5 Domino Administration Doc Pack, Lotus part number AE7NRNA). The figure below shows a comparison between Domino statistics and Windows NT statistics.

![Performance Monitor](image)

Chapter 5: Troubleshooting an Existing Application 169
• Task Manager

The NT Task manager can also provide very useful information. It allows you to monitor specific tasks that are running on the server, how much CPU and Memory they are consuming, and the amount of CPU time needed.

The following figure is sample output from the Task Manager utility. All of the various Domino tasks are shown in this dialog box.

Domino tasks for the Windows 32-bit operating system always start with an ‘n’ so the update task can be seen in the Task Manager as ‘nupdate.exe’. From the Task Manager output shown, you can see the percentage of CPU time that is currently being used by each task, the total amount of CPU time each task has used, and the amount of memory allocated to it.

You can also use the Task Manager if you suspect that memory leaks are occurring. Choose View - Select Columns in the Task Manager menu. Add a check for including column for the virtual memory size in the processes view. Check the view over time. If a task uses more and more virtual memory it may have a memory leak problem.
OS/2
The following tool is useful for detecting how well a computer running OS/2 is performing:
- `pstat`
  This tool provides information on the status of the different processes running in the operating system, including thread and semaphore information.

AS/400
The AS/400 platform has its own specific set of performance monitoring tools. They are packaged as *Performance Tools/400*, and are comprised of two different tools:
- **Performance Monitor**
  This component collects information relating to the performance of the AS/400.
- **Performance Explorer**
  This tool can be used to help identify causes of performance problems.
More information on Monitoring Domino for AS/400 performance can be found in the redbook *Domino for AS/400: Performance, Tuning, and Capacity Planning*, SG24-5162.

AIX
Many of the system commands that are available on the AIX platform can be used to identify server performance issues. Standard commands that can be used are as follows:
- **vmstat**
  Collects information on kernel threads, virtual memory, disks, CPU activity, and more.
- **sar**
  Collects system activity information.
- **ps**
  Reports on process activity.
- **tprof**
  Reports on CPU usage for individual programs or for the system as a whole.
- **svmon**
  Provides information about virtual memory available to the user who is running the command.

Chapter 5: Troubleshooting an Existing Application 171
• iostat
  Provides information about the status of the I/O.

• lsof
  Collects information about the number of open files. This can be helpful in determining what files are being used on a Domino server.

**HPUX**
• lsof
  Collects information about the number of open files. This can be helpful in determining what files are being used on a Domino server.

**Solaris**
The available Solaris commands include the following:

• vmstat
  Collects information on kernel threads, virtual memory, disks, CPU activity, and more.

• iostat-x
  Provides information about the status of the I/O.

• swap
  Checks the amount of virtual memory, both used and available.

• netstat
  Monitors the status of the network.

• lsof
  Collects information about the number of open files. This can be helpful in determining what files are being used on a Domino server.

**S/390**
The system 390 platform uses both the Resource Management Facility (RMF) and the System Management Facility (SMF) to gather information that relates to system performance. More information can be found in the redbook *Lotus Domino for S/390 Performance Tuning and Capacity Planning*, SG24-5149.

**Domino Server Performance**
There are a number of methods, tools and resources available to monitor the performance and “health” of a Domino server. These tools are generally not dependent on the operating system on which they are running, since they are built in as part of the Domino core services.
The performance and tuning of a Domino server are considered to be Server Administrator functions, and as such are beyond the scope of this redbook. However, as a developer or troubleshooter, it is useful to find information that could assist you in understanding specific application issues. Some of the key areas to examine are listed in this section.

**Notes Log**
The Notes Log is the system log for the Domino server. It contains a large number of server-related events. This includes session information for each user, database size information, and mail routing and replication information. The logging of Notes is an automatic function and cannot be disabled, but some of the statistics, such as the database size information, are collected daily by a server task called `statlog`. By default the statlog task runs at 5 am on the server. However, as it is an individual task, it can be disabled on some servers. If the Database\Sizes view is empty or the documents are out of date (the date that the information was collected is recorded at the top of the document), this means that the task is not running. Your server administrator will be able to confirm this.

Information held in the Database\Sizes view can provide information on database view indexes. If a database has a very large view index, this could indicate that the view is complex and may take time to open.

The Database\Usage view can be used to see the overall activity for a database, displaying the number of read, write and usage events that have occurred within the past 24 hours, week, month, and for the entire period that the database has been monitored. The view also displays individual session information for the database, such as information about individual user activity against that database.

The two views under Usage (Usage\By Date and Usage\By User) can be used to collect information about individual usage patterns. These views show what the user did on a database level during a session.

**Statistics and Events**
The ability to record Domino statistics and events is a standard feature of Domino, but the tasks must be configured and started on the server. The statistics recorded can be used to identify specific issues with the server. The items recorded include the number of users, peak activity times, amounts of memory used for Domino-specific tasks, and more. Events can be used as triggers for specific actions, such as mailing the notification to an administrator.
The information that is held in the statrep.nsf database is server capacity related, including items such as the size of various different buffers and caches. This information can be used to determine whether a Domino server has enough capacity for the number of users that it is hosting or the type of activity that it is performing.

**Domino Log**

When Domino is used with the HTTP task, logging of HTTP transactions can be enabled. The information that is collected in the Domino log, which can be a series of text files or a Notes database, or both, can be useful in establishing which HTTP transactions are occurring.

All HTTP transactions can be logged and then later reviewed. This can be useful to the designer in determining what actions are being performed on the databases and how long these actions are taking. The log also records the IP address of the remote computer connecting to it and, if the user has Authenticated, will also display the user’s name.

The information that is held in the Domino Web Server Log requires some manipulation and analysis to be useful. The domlog.nsf database is very simple in design: it contains just one form and one view. If the Domino server is heavily used, a lot of activity will be logged into this database. Therefore it is advisable to archive the information that is gathered in domlog.nsf to a separate database on a periodic basis and use this as your working data.

When Domino logging is enabled to text files, five files are created on a daily basis for recording different types of information (access, agent, CGI, errors and references). These files contain the basic text relating to which HTTP activity is occurring on the server, but their format is not particularly useful due to the amount of information that is being recorded. The following figure shows a sample Domino log:

![Sample Domino Log](image)

174 Performance Considerations for Domino Applications
A number of third-party tools that display this information in a report format are available.

**Server Console**
The Domino server console can be used to see whether there are performance issues with the server. By using the server console you can establish what the server is currently doing (such as processing agents or updating views in a specific database), what it has done in the past, and what it has been scheduled to run in the future. This is very important in diagnosing server-related problems.

Commands that can be issued from the server console include the following:

- show server
- show tasks
- show database
- show transactions
- show statistic
- show dbs *(R5 only)*
- tell amgr status

**Billing Task**
The Billing Task is provided as part of the Domino Enterprise Server. It functions as a separate task loaded onto the Domino Server and provides information about Domino-specific activities, called *Billing Classes*. The Billing Classes are:

- Session
- Replication
- Document
- Mail
- Agents
- HTTP

These classes can provide additional information on the server activities. They can be used in conjunction with the information that is collected in the Notes Log. The HTTP session will provide the same information as that provided in the Domino Log database. Agent Log provides useful information about Agent Run time and who initiated the agent. Document billing requires the addition of extra fields into specific documents; it records the cost of working with a document (including reading it).
In addition to this, the Billing APIs can be used to provide custom billing tasks that can be used to capture special events.

All tasks that are loaded onto a Domino server above the core tasks will consume extra server resources. It is important that this is considered and discussed prior to implementation.

**Third Party Tools**

There are a large number of third-party tools available to assist you in monitoring server activity and performance. They can enhance your ability to diagnose performance problems when used together with the standard Domino applications. See product-specific documentation for more information.

**Server.Load**

Server.Load has been available for Domino since 4.62. This tool was primarily developed to stress test Domino servers and provide the customer with a guide to the number of users and performance level to expect from a specific hardware configuration. In the 4.6x version of Server.Load, the basic task of the application was to simulate Notes user sessions performing operations on the server. These operations can either be one of three predefined tasks or you can create a custom script to test the server. Server.Load for R5.01 has been further developed to incorporate other protocols that Domino supports, including HTTP, SMTP, POP3, IMAP and LDAP. In addition to these new protocols, the existing NRPC protocol has been given added functionality.

The 4.6 version of Server.Load provides very little functionality to test individual application performance, but it still provides some valuable information. It can test for standard response times from the server before your application is released, or before an upgrade. Then, once your application is being used, it can test the response times again and check the server loading. In the 4.6x version of Server.Load, you can customize scripts that open databases and views then navigate through the view, opening documents and so forth. The document creation ability of Server.Load is similar to creating a simple mail message. Documents can be created, but the tool enters text into the Subject field and text plus an attachment into the Body field. This is not appropriate for most applications, since they generally have a number of fields and require different types of information to be entered into these fields.

With Server.Load for Domino 5.01 there is enhanced scripting language capability for the Notes Client Simulation. It is now possible to create documents in an application with many different types of fields. In addition, the HTTP protocol is useful in testing Domino Web sites, including the ability to enter information into a Web-based form and to activate buttons and other links on the page.
Server.Load can also be used to place additional user load onto the Domino Server. This can be used to simulate a more realistic environment since we can place user load onto the Domino server, which may not otherwise be possible when testing an application. It can also be used to place extra loads onto a “Live” server under test and perform a ‘What If’ scenario to see whether application performance is directly related to the number of users on the server.

**Client Performance**

Performance is generally a user perception issue. Even if the application, server, and network are not the cause of an apparent problem, it may still be reported that there is a problem with application performance.

Once all server-side issues have been eliminated as the cause, you should consider the different client hardware configurations that probably exist in your organization. Individual client circumstances, such as available memory and processor type, can significantly affect performance. If there are wide discrepancies in performance from one client to another, these client-focused issues should be further investigated. Testing the client performance can be done using a number of benchmark applications. These applications can be written in Lotus Notes and run on several clients.

**Usage Pattern Performance Issues**

To determine whether the performance issue is a user-related one, a set of scripts can be developed that test operations in a set sequence. It is important that the scripts are detailed enough to describe every issue that a user has. For example, the test might press a button only once for a short period of time, and if nothing happens in 10 seconds then the button may be pushed again. These tests can be run at several locations by different testers in order to set a benchmark figure for the total operation time.

Some third-party tools provide the ability to record actions such as user interaction, which can then be replayed at multiple locations.
Application Performance Troubleshooting

When an application is planned, prototyped, and developed, it is often very difficult to establish how it will perform when rolled out to the “live” user community. The real world is more complicated than any test scenario that you can invent during development. The primary reasons for this are as follows:

• Live users are very hard to simulate. For example, it is difficult to imitate the many different ways that users enter information, from someone familiar with the system who is very efficient at it, to someone who is new to using a computer and who may take a lot of time and make many errors while entering information.

• Budget constraints often mean that test environments are not similar to the live environment.

• External factors (such as a virus or a faulty network card) are hard to simulate or anticipate. Therefore, while it is not worth attempting to simulate an environment that has these factors, they should be taken into account.

• Development time scales often cause accelerated work flow. For example, the actual life cycle of an appraisal process may be a full year, but it may only be tested for one week.

• Use of an application often changes over time as it is adapted to changing circumstances.

• Test data is often not comparable to live data. This applies both to content and to volume.

These are the reasons an application might not perform up to expectations, even though it was tested before rollout.

We can now start to look at the individual components necessary to perform troubleshooting. The information above shows the limitations between testing and production; one of the purposes of troubleshooting is to try to make these parallel where possible.

The Environment

One of the first steps in application troubleshooting is to set up a test environment, which should be as similar as possible to the live environment in which the problems are occurring.
To achieve this you must gather data from the live environment, specifically the following items:

- The number of users of the application
- The number of documents of different types and the size of each document. This also includes the type of information stored in the document; for example, the application may behave differently with images stored in rich text fields instead of formatted text.
- View index sizes
- Application performance measurements
- Environment bandwidths
- Application usage

**Constructing Your Test Environment**

Generally it is not possible to conduct performance tests on an actual production system, as there is a requirement not to cause disruption and many tests will require changing forms and views, and modifying or adding test data. Using the information gathered from the live environment, set up a test environment as similar as possible to the situation in which performance problems were encountered.

Set up your test environment using the information collected in the production environment. To minimize the differences between the two systems, you should also consider implementing the following conditions for test purposes:

- Simulating poor network performance
- Simulating poor server performance
- Requiring the user to perform tasks in the system as if they were in a production environment

There are a number of factors which will generally interfere with setting up an environment that is identical to the production environment. However, by identifying what you need to test and how you can best use the resources available to simulate a production scenario, you will be able to gauge performance problems and the benefits of the modifications that you have made.
There are a number of different utilities and methods available to help you create the desired environment. However, a number of basic tasks should be done before any tests and measurements are taken:

- **Clean client environment** — Many applications, including Notes clients and Web browsers, cache information to improve performance. The Notes client will cache information in the CACHE.DSK file. Delete this file before each test. Web browsers cache information in different ways; generally you will need to delete Temporary Internet Files (which may include ‘cookie’ files), plus any images that are also stored on the local computer.

- **Clean network** — A clean network is also a desirable feature, as it enables you to place your own load on the network and not disrupt other people or have other network traffic disrupt your testing.

- **Clean server** — Domino Servers also cache a lot of information. In addition, once an application has been accessed once, view indexes will have been built. Any caching or indexing should be taken into account in the different tests that you perform. Typically you should ensure that there are no cached images (for Domino HTTP access). By default, these are stored in the following directory relative to the Domino Data Directory:

  ```
  domino\cache
  ```

  You also need to ensure that view indexes that will be accessed are either already built (by accessing them prior to the test) or are not built.

  To remove view indexes from a database, run the following command:

  ```
  load compact sales.nsf -d
  ```

  This example removes all view indexes from the sales.nsf database.

**Dial-up Networking**

The use of a modem and a dial-up networking application gives you the ability to test poor bandwidth access and also to record network throughput. When you configure the modem connection speed you may also decide to slow the connection speed down.

When you have a slow connection, you may notice things that are not obvious when you are accessing the application using a LAN connection. For example, you may notice a long delay when loading an item on a form or opening a view. This method of using a very slow connection is good for revealing performance bottlenecks.
From the following figure you can see that the number of bytes sent and received are recorded. It is important that you keep other network activity to a minimum if you intend to use the response times and network traffic size for measurement purposes.

### Spray Daemon

The “spray” command is a UNIX command which can be used to send network packets of a specific size between a client and a host. This is a very useful command if you are working on a “clean” network. Using a spray type command, we can simulate network load in a controlled manner. The syntax of the spray command is:

```
/usr/sbin/spray Host [-c count] [-d delay] [-i] [-l length]
```

The flags in the square brackets are all optional. An example of this would be:

```
/usr/sbin/spray odin -c 1200 -d 2 -l 1000
```

In this example, 1200 packets of 1000 bytes each will be sent, with a 2 microsecond delay between packets.

### IPTrace

The “iptrace” daemon is a tool used to provide packet tracing for Internet protocols. It is available for both UNIX and OS/2 platforms. It is useful for collecting network information at a very low level about your environment. The iptrace tool listens to a particular TCP/IP port on the server. By default Domino uses port 1352 for sessions with Notes clients and port 80 for HTTP sessions. Below is an example of the command to collect Notes client traffic to the server.

```
iptrace -i en0 -p 1352 /tmp/notes.trace
```
The -i flag tells the command to listen on the ‘en0’ interface and the -p flag tells the command to collect only information from port 1352. The information is then put into the notes.trace file in the /tmp directory.

The Lotus Enterprise Center of Competency uses this utility extensively when comparing production environments with test environments.

**Server Clock**

The server_clock NOTES.INI setting can be placed on the server. As mentioned previously, this setting enables us to record the Notes RPC on a server. This enables us to see what the traffic activity is like on our Domino server.

On a heavily used Domino server, however, the amount of information that will be collected is very high and it is unlikely to be of much use. The following is an example of the RPCs that are displayed on a server when a single user accesses it for the first time and opens a single database:

```
67118440 START_SERVER 16230 ms (16210 ms NETIO) TCPIP 00050825 Rcvd 424 Sent 388
67134680 OPEN_DB 0 ms (0 ms NETIO) TCPIP 00050825 Rcvd 0 Sent 290
67134690 SEARCH 0 ms (0 ms NETIO) TCPIP 00050825 Rcvd 0 Sent 3978
67134730 CLOSE_DB 0 ms (0 ms NETIO) TCPIP 00050825 Rcvd 0 Sent 0
67147620 OPEN_DB 10 ms (0 ms NETIO) TCPIP 00050825 Rcvd 0 Sent 290
67148400 DB_MODIFIED_TIME 0 ms (0 ms NETIO) TCPIP 00050825 Rcvd 0 Sent 44
```

182 Performance Considerations for Domino Applications
This information can be read into another application and used to track what is happening at an RPC level, as well as the amount (quantity in bytes) of information sent and received by the server.

**Note**  The server_clock setting must be used carefully, as it relates to the information that is being generated and transmitted by the server. If there are a number of clients connecting to the server, all client information will be reported. For information specific to an individual client, use the “Client_Clock” setting described later in this chapter.

 execution control list
The Execution Control List (ECL) is a security feature built into the Notes client to prevent unauthorized access to resources. It may also be used to enable the troubleshooter to monitor some actions, such as @Db commands.

By refining the ECL you will be notified when commands are executed that fall into a specific category, for example accessing external information, where you may not have noticed that the application has an @DbLookup command in a Validation field that is very slow.

**Design Synopsis**
The synopsis that the Notes client produces is a very useful tool to aid you in finding particular design elements and references. It allows you to select one, several, or all design elements, and displays the design details for the selected items. You can then either scroll through the information that is collected for you or use the find feature. Synopsis output can also be saved as a database document.

The R4.x Design Synopsis was very useful for collecting standard information about applications designed for Notes clients. Unfortunately, it did not display any LotusScript formulas or events, and various Web elements such as HTML attributes were not reported.

With the R5.x design client, the Design Synopsis is much more powerful. In addition to reporting on various new design elements (such as pages and outlines), it will now display LotusScript, HTML, Javascript and Java code. The synopsis can also be stored in a database, which can enable you to compare the design synopsis from different points in time to see design changes.
Tip  For a more detailed and complete set of information about your 4.x applications, use the R5 Designer client to get the design synopsis.

Note   There are also a number of very good third-party tools that can be used to analyze the contents of a Domino application in a similar way. Some of these tools have special functions that allow further analysis. For example, some tools can be used to compare two databases to identify the differences in design.

NotesPeek
The Notes Design Synopsis displays a very flat view of the Domino database. While this is useful for finding particular information, it is often difficult to navigate. NotesPeek is a tool that looks at the elements (or objects) that comprise a Domino application in a graphical way. It is also useful for displaying some extra information that may be hard to see when only using the Notes client, such as the number of deletion stubs and Agent Notes. NotesPeek also displays the information in the actual containment hierarchy that is native to Notes.

The following figure shows how you could use NotesPeek to view documents that are otherwise hidden. It is useful for determining what profile documents are being used in an application, and also for looking at the deletion stubs to see what they were previously.

The NotesPeek tool is available for download from the Notes Web site at:
http://www.notes.net

Select the downloads area and search for the version of NotesPeek that you want.
Note The NotesPeek tool is provided on a “Use at Your Own Risk” basis. Support and warranty for it is not provided. For the full terms and conditions of the product, refer to the “README.TXT” file provided.

Server.Load
The Server.Load tool provided with Domino can be used for two purposes when constructing your test environment. The main use for the tool is to simulate server load. Server.Load allows you to simulate up to 255 (512 in R5.0.1) users from a single Windows NT client. The R5.0.1 version of Server.Load allows you to provide more detailed testing of a specific application. For example, you can conduct a test to simulate Notes users performing the following tasks:

1. Open a view in a database.
2. Create a document.
3. Fill in the document in a specified manner.
4. Save the document.
5. Create a random number of additional documents at a random period.
6. Close the database and open another.

This type of script could be created and run on the Domino server while you are testing the new response times for the application. If the application has Web-based functionality, this can be tested with just as much, if not more, power as other Notes applications, since the HTTP syntax can be used to run Domino Agents or perform other actions. With the HTTP tool that comes with Server.Load you can also test the difference in the performance of standard HTTP and HTTP using Secure Sockets Layer (SSL). However, you cannot use Server.Load on a site that is running a mixture of the two environments.

Tip You can use the R5.0.1 Server.Load tool to perform actions against an R4.x Domino Server. To do this you will need to have the R5.0.1 Notes client loaded on the Load machine. This will allow you to test HTTP applications and to provide more detailed application testing using NRPC. It can also be used in the production environment and test environments in order to simulate a single user session and collect the response times and other metrics. This is very useful in comparing how the two environments perform under similar conditions.

Testing an Application
The methods for testing an application are slightly different for a Notes client and a Web browser; however, the underlying techniques to perform the testing are similar. Examine the performance of the application function by function. This section describes the factors to consider for each function.
**Database open**

If a database takes a long time to open, check the following items:

- Database size information, including the number of documents and the percent used. If the database has a large amount of free space (the percent used is low), this will affect access times to the application. This can be fixed by compacting the database.

- The number of unread documents in the database. If the database contains a large number of unread documents, this will slow down the loading of the database. The client builds the list of unread documents when the database is being loaded.

  **Tip** In Domino R5.0 it is possible to set a database property ‘Don’t maintain unread marks’, which will speed up the loading of a database with a large number of documents.

- What does the database do on opening? This can be determined by looking at the database properties and pressing the *launch* tab. These options will point you in the direction of where to look next. For example, if the database opens a specific view, you can investigate that view; if it is a navigator then the navigator can be investigated, and so on.

- If the database is one with high activity, or if it is used as a portal to other databases, the access method that is used to get into the application should be checked, and the method by which people access your application may need to be modified to distribute users.
Tip In Domino R5.0 the server console command ‘show dbs’ enables you to see which databases are currently open. Below is a sample of the output from ‘show dbs’

<table>
<thead>
<tr>
<th>Database Name</th>
<th>Refs</th>
<th>Mod</th>
<th>FDs</th>
<th>LockWaits/AvgWait</th>
<th>#Waiters</th>
<th>MaxWaiters</th>
</tr>
</thead>
<tbody>
<tr>
<td>d:\domino\data\domlog.nsf</td>
<td>1</td>
<td>N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d:\domino\data\statmail.nsf</td>
<td>1</td>
<td>N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d:\domino\data\statrep.nsf</td>
<td>1</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d:\domino\data\events4.nsf</td>
<td>8</td>
<td>N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d:\domino\data\mail.box</td>
<td>1</td>
<td>N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d:\domino\data\names.nsf</td>
<td>50</td>
<td>Y</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d:\domino\data\log.nsf</td>
<td>1</td>
<td>Y</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Views
Views are a common source of problems in application performance. They can cause a problem when you attempt to open them, when you are navigating through them, when you are attempting to refresh them, and when you are attempting to access information in them, either directly or indirectly.

Often, specific features in a view involve a trade-off between what performs well (speed of opening) and what works the best (ease of access to correct information).

The size of a view index is a good gauge of how complex the view is. This can be collected in three ways:

1. Using a Server Console Command.
2. From the Design tab on the view properties.
3. From the Notes Log.

The following section describes each of these methods individually:

Using a Server Console Command
If you have administration rights to the Domino server, use the remote console to issue the following Server Console command:

```
show database walt/redsampx.nsf v
```
This example displays the size of the view indexes for the ‘redsampx.nsf’ database in the ‘walt’ subdirectory. The following figure shows the output from this command:

![Remote console for server 'admin/Flock']

Here we can see the sizes of all of the views in the ‘Redbook Samples’ database. The size of the view index will be dependent on the number of documents and the design of the view.

**Using the Notes Log**

If you do not have access to either the Design of the Views or the Remote Server Console, you can use the Notes log to show you the size of views for the database. A task called StatLog runs daily on Domino servers. It collects database information and writes it to the log file. If you go into the ‘LOG.NSF’ database and open the ‘Database\Sizes’ view, you will see a list of databases in descending order of total database size. If you go down this list, find your database and open the document, you are presented with a list of all of the views in the database and the size of each.

**Note** The log also displays the size of private views stored in the database. These views are shown with square brackets around them, such as [My Documents]. If you see a large number of these views in a database, this is a cause for concern with respect to database indexing.

**View Opening**

The speed with which a view is opened will depend on the index options for the database, the amount of information sent to the client, and whether the view index must be rebuilt prior to the view being opened. In situations where there is a large amount of information being continuously entered into the application, the view will take a long time to open.
To stop the indexer from updating the view each time it is opened, set the Refresh Index view option to a more appropriate setting, such as “At most every ‘x’ hours.”

**Note** If reader names are used to hide some documents, it is important that you test the view with a Notes ID that can only see a limited number of documents. If you test with a designer’s ID, you will most likely be able to see all documents and your testing will not be valid, since no documents will be filtered.

**View Navigation**
You navigate through a view by either scrolling up or down the view, expanding and collapsing sections, and changing the sorts of columns in the view. In general, everything that the view does with navigation must be tested.

In particular, for views displayed on the Web, navigation can affect performance since each time you scroll, a new page is sent from server to client. Using view applets can help in this area.

**Accessing view information**
Accessing the information held in a view can be done either directly or indirectly. Direct access means that the user selects the document in the view and opens it. Performance problems with this are discussed in detail when Document access is addressed.

Indirect access is when the information held in a view is accessed either through @functions or with one of the various programming languages that are accessible to Domino. When indirect access is used, it is important that views are simple, and that the indexes are up to date or not refreshed automatically. Otherwise a delay will occur while the view is built.

**Tip** You can also use hidden views which are tuned for high performance data access.

**Tip** Hidden views can be seen by holding the Ctrl and Shift keys down when you access a database. By opening views that you will be using to access data indirectly, you will invoke the indexing process on these views.

**View Actions**
When a view is opened, information is downloaded regarding what actions are available. If these are “Action buttons,” the required icons are also sent to the client to be displayed on the Web. However, the code behind the icons is not sent at this time. Therefore, if there is a large amount of code behind an action button, it may take some time to receive this from the server. However, it is unusual for the information held behind these buttons to be of such a size that this is noticeable.
Forms
In most applications, the form is where most of the work is done. It is also an area that can be tuned considerably to improve user perception of the application performance.

What is Taking Time to Load?
It is often very difficult to determine what is causing slow performance. Fortunately, with forms, things can be simplified somewhat.

When a slow connection is used to open a form, you can see the loading pause at various points throughout the loading of the document. This can be helpful when trying to troubleshoot a poorly performing form.

The following are some of the reasons that a form might open slowly:

- Network problems. This method of testing can highlight the network as the problem.
- Graphics. As mentioned previously, large graphics will take time to download. If they have been scaled in Notes, they are still stored in the original size and scaled when the form is loaded.
- Formulas and events. The presence of many formulas and events will have an impact. These may be in fields (visible or hidden), hide when formulas, or form events (such as QueryOpen).
- Design Elements not stored in the form, such as shared fields, subforms and script libraries.
- Computed text.
- Complex tables.

In addition to looking through the design of the form, you can see which LotusScript is executed when the form loads, and during subsequent events that are triggered throughout the form, by turning on the LotusScript debugger when you enter a form.

Tip Try to remove fields or design elements, either individually or in groups, to see if there is a performance improvement when the form is loaded. This is useful when trying to find the specific elements that are causing delays.

Tip Try breaking up what is loaded into blocks by placing a pause in the form, such as a default formula in an editable field that displays an @Prompt, but does not populate the field. This will enable you to see whether certain portions of the form are taking longer than others to load. An example of the formula is as follows:

```
@Prompt([OK]; “Loading Paused”;; “The loading of this form has been paused for performance analysis. Press OK to continue”);@Return(“”)
```

190 Performance Considerations for Domino Applications
Enabling the Client_Clock makes it possible to see at a low level (RPC level) how much communication is being transmitted between the client and the server for each event. This can be hard to relate to what is happening at the client, but it is a good indication if there is a lot of “chat” taking place.

The ECL can be used to trap @Db functions and access to external information such as Profile documents, external files, or documents in other databases.

When a Web client is used, the Domino log can provide useful information about the interaction between the browser and the server.

**Delays While Navigating Through a Form**

You may experience delays while moving through a document. These delays can have a number of causes if you are using a Notes client. Check whether any of the following items are set in your form. They may be the cause for the delay:

- The form property ‘Automatically Refresh Fields’ is set.
- The ‘Refresh Fields on Keyword change’ is set for a keyword field.
- There are LotusScript or JavaScript events behind some of the fields.
- The LotusScript Class ‘NotesTimer’ has been used to periodically perform a task.

**Saving and Closing a Document**

A number of things will slow down the saving of documents. Events are often triggered that will cause the application to perform other actions, such as a WebQuerySave event calling a Domino agent to process the document that has just been created by the Web browser.

For the Web browser, check to see if there is an agent run by the WebQuerySave event in the Programmers Pane. For older versions of Notes, check whether there is a ‘$WebQuerySave’ field.

For Notes clients, the Translation and Validation formulas on Editable fields will be run, as will the formulas in Computed and Computed for Display Fields. Finally, if there is LotusScript in any of the form events, these may also be run. Ideally all validation should be done in one place.

**Agents**

Agents can be categorized into three types:

- Interactive Agents
- Background or Server Agents
- Web Agents
Interactive agents are agents that run on a client and either require input from the client when they are running or make the Notes client 'busy' while running, so that the user cannot use it for anything else.

Server agents run on the server, using the server ID. These are often scheduled agents.

Web agents are similar to server agents. Even though it is the Web Client that initiates the agent, it is run on the server. One difference is that the Agent Manager initiates Server Agents and the HTTP task initiates Web Agents. Another difference is the IDs used.

When writing agents to maximize performance, it is important that they run in as short a time frame as possible, and that the impact on the environment when they are running is kept to a minimum. Troubleshooting agents, therefore, is a combination of monitoring the entire environment (such as monitoring the CPU and memory usage with Operation System tools) and monitoring the amount of time that it takes to run the application. In some environments it is very important to ensure that the agent runs quickly, due to restrictions in the execution time limits for server-based agents.

The use of the Billing Class for Agents can be extremely helpful in troubleshooting agent run times and how frequently they are being run.

If agents are consuming too much of the environmental resources, you must narrow down the area of the agent that is causing this. This can be challenging unless you record times that parts of the agent are running. Code Profiling, described later in this chapter, is a method for doing this.

---

**Top Performance Items**

As nearly every application and the environment in which it runs is different, it is not possible to give a definitive list of how to develop an application. If an application is small and the development speed of the application is important, you may decide to use a number of design features which could slow down a larger application. This is not poor development, but it is important that the limitations of the application development technique are considered and documented.
Below is a list of design elements that should be considered when troubleshooting performance. They often will highlight specific performance-related areas:

# Views

<table>
<thead>
<tr>
<th>Element</th>
<th>Comment</th>
<th>Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Today, @Now, and so forth</td>
<td>Will cause the index to be marked as not up-to-date when used in a view</td>
<td>Use techniques in Appendix B-1, “Time/Date Views”</td>
</tr>
<tr>
<td>Reader names</td>
<td>May cause very slow performance when the user only has access to a small percentage of documents</td>
<td>Use private views or navigator</td>
</tr>
<tr>
<td>Complex categorization</td>
<td>Will cause a large view index</td>
<td>Refine view</td>
</tr>
<tr>
<td>Sorting ‘on the fly’</td>
<td>Will cause a large view index</td>
<td>Only sort columns that require sorting, don’t duplicate what another view shows</td>
</tr>
<tr>
<td>Unread marks displayed at all levels</td>
<td>Dramatically slows down a view, as unread marks are calculated for each user when they open the view</td>
<td>Use the ViewShowOnlyUnread’ @Command</td>
</tr>
<tr>
<td>Many expandable documents</td>
<td>All information is not sent up the network to the client when collapsed</td>
<td>Use the view property ‘Collapse all when database is first open’</td>
</tr>
<tr>
<td>Use of response hierarchy</td>
<td>Maintaining a response hierarchy can slow down view access times</td>
<td>Use response hierarchy to link forms together via form design and not view</td>
</tr>
<tr>
<td>Large number of documents in view</td>
<td>A complex selection formula will slow down a view, but too simple a selection formula, such as SELECT @All, can also slow a view down if there are a large number of documents</td>
<td>Use a carefully constructed selection formula</td>
</tr>
</tbody>
</table>

continued
Many different views
An index has to be built and maintained for every view
In R5 you can use the ‘Show Single Category’ view option to give you only specific documents

View applet
Can be slow to load over limited network bandwidth
Restrict the use of the View Applet for when you require the functionality

Elements | Comment | Alternative
--- | --- | ---
Many different views | An index has to be built and maintained for every view | In R5 you can use the ‘Show Single Category’ view option to give you only specific documents
View applet | Can be slow to load over limited network bandwidth | Restrict the use of the View Applet for when you require the functionality

### Forms

<table>
<thead>
<tr>
<th>Elements</th>
<th>Comment</th>
<th>Alternative</th>
</tr>
</thead>
</table>
| Large number of fields on document | This will slow down the load and use of the document | Create multiple forms or use a more interactive form with prompts and dialog boxes to improve perception
| Many @Db. commands | Often the information used for keywords is used multiple times throughout the document | Ensure that the fields are not refreshed unless necessary and try to load the keyword information only once at load time
| Numerous ‘Hide When’ formulas | These are generally used to control a user’s access throughout the document | Use dialog boxes to prompt the user for information, or use multiple forms or sub-forms. Try to use the ‘Hide When’ check boxes instead of formulas
| Large number of complex tables | The larger the number of tables, and the more complex these tables are, the slower the load and refresh time of the form will be | Use tabs in the form design when possible; minimize the complexity of tables
| Auto Refresh fields | The form option to auto refresh fields is often only used to update fields when based on a particular event | Use field events to change the required fields or use Hotspots
| Validation formulas in many fields | Validating every field on the form when saving or refreshing takes time; for the Web it is done on the server | Use field events to validate as the form is computed, or a single event as the form is posted (LotusScript or JavaScript depending on your client)

194 Performance Considerations for Domino Applications
Performance and the Network

You may not realize that certain coding styles put a much heavier load on the network than others. Here we will show you how to get an insight into network load from the comfort of your very own PC.

Your Deployment Environment

Some applications will be deployed in environments where the network plays a more influential role than it does in other environments. When an application is on a wide area network (WAN), network considerations become much more important than if the application is used on a local area network (LAN). A simple calculation of bandwidth alone is not sufficient to answer the question, “Is my network okay?” You need to consider not only the location of your servers, but also the paths between servers.

If your deployment environment is a local area network (LAN), you can skip the rest of this chapter.

Notes Talks

Notes can be very talkative in the network: “chatty” is a term that is often used. By that we mean that Notes sends many messages between the client and server to do its tasks. Not only does Notes send many messages, most of them are small. Remember, the database often resides on the server (and this includes both the data and design), but when the application runs, the actual code is run on the client. That means that the code must be sent to the client for execution. The data must also be sent to the client whenever necessary. Any results, of course, must be sent back to the server.

A very useful piece of information, then, is to be able to determine what is happening in the space between the client and server when your application is running.

Notes Remote Procedure Calls

When it comes to checking the performance of your application on the network, having information about the Notes Remote Procedure Calls (NRPCs) being generated by your application is a good starting point to getting a sense of how your application will perform.

As implied by the name, remote procedure calls involve the client requesting to run a procedure at the server. The procedure can be simple or complex and the server may do a lot of work or a little, but in any case the one thing that is necessary is communication between the client and server.

For example, if the code on the client says: “Open Database ‘A’,” that request must be sent to the server, since the client does not have database ‘A’ locally.
The Designer Is in the Dark
One of the problems with designing for performance, especially in a wide area network environment, is that the designer of the code is not easily aware of what client-server communication is being generated when the code is being written. Notes is client-server by nature, but this characteristic is built into Notes at a low level. The designer has little influence over when the client will “talk” to the server. This is good since the designer can worry about more important things and doesn’t have to operate at a system programmer level. However, the drawback is that it can easily lead to applications that put a heavy load on your network.

How Can We See Remote Procedure Calls?
In Notes, there are often several ways to code a particular function. If we can see the RPCs that an application generates for a given design, we then have some information to choose one design over another. We can choose the design that bests suit the application requirements and at the same time minimizes network load.

Notes has a built-in RPC tracing facility which can be enabled. This is easily done by turning on the Client_Clock variable in your NOTES.INI file (by default, the NOTES.INI file does not contain a reference to this variable). We can enable the RPC tracing facility by including a line in the INI file that sets the Client_Clock variable to ‘1,’ as follows:

1. Shut down Notes.
2. Edit the NOTES.INI file:
   - For Notes Release 4.x and earlier, the file will be in your WINDOWS directory. Normally the file will be C:\WINDOWS\NOTES.INI
   - For Notes Release 5 and above, the file will be in your NOTES directory. This will normally be C:\NOTES\NOTES.INI
3. Add the following lines to the Notes.INI file:
   ```ini
   Client_Clock=1
   Debug_Console=1
   Debug_Outfile=C:\RPCOUT.TXT
   ```
   Only Client_Clock is mandatory; the other two are optional.
   It does not matter where in the file you insert these lines. It is a good idea to put them near the top so you can find them easily.
   The Debug_Console line is used to display the RPCs in a debug pop-up window as they occur. If you’re happy to just put the output into a file, leave out the Debug_Console line.
The file for which the name appears in the Debug_Outfile line will be created and will contain the results of the trace. You can pick any location for the results file, but make sure that the required sub-directories exist. If the path does not exist, Notes will not create it for you. A nonexistent path will not generate an error, but no output file will be created.

4. Save the modified INI file.
5. Restart Notes.

**Note** When running Notes multiple times, the new file will always have the name specified by the Debug_Outfile INI variable. If a file by that name already exists, it renames the old file by appending a number to the end of the old file name. The numbering starts at 1 and increments each time a rename is required. In this way, a history of all RPC output files is automatically maintained for you.

**Tip** After you have traced the RPCs, reset your INI file by re-editing NOTES.INI and placing a semicolon in front of the three lines added above. This comments out the INI file information, thus disabling the trace. However, the next time you want to enable the RPC trace, you don’t need to re-key the information.

**Examining NRPC Output**
When you look at the output from an RPC trace, you may be surprised at the volume of data being generated. At the same time, you will wonder “What does it all mean?” The answer is not trivial, but with trial and error you will get a feel for what is happening in the RPC output.

Appendix B-8, “Using NRPC Output,” has more details about RPCs.

---

**Code Profiling**

When complex script is used in an application, it is often hard to understand what is happening, how long it takes and how often it happens for individual routines or subroutines. Code profiling is a method of logging these actions. We can use code profiling to get a better understanding of what the code is doing and how long it is taking. It is helpful to use code profiling in the development and troubleshooting of LotusScript and Java code.
In the Notes client you can enable the LotusScript Debugger, which will enable you to see when LotusScript is being executed; however, this feature is not available to the Web client. Code profiling will give you considerable power for logging Web client-initiated LotusScript in order for the troubleshooter to monitor which agents are being run by a client, when they are being run and how long they are taking. This gives you an alternative to printing lines of code to the screen in order to see what is happening when agents are running.

The Domino Object Model includes a special class for logging. This can create entries in either a text file or a Domino database. You use this class to record what your code is doing, that is, for creating a “Code Profile.”

Typically you will not want to profile small amounts of LotusScript or Java. As the agents and classes become more complicated, the requirement to collect information about them becomes more important.

We are interested in recording the total time taken to run the agent, class or application, how long functions and subroutines take to run, and the number of times each subroutine is invoked throughout the entire execution. By using the NotesLog class you also have the advantage of logging Errors that your code generates as well as other messages generated.

**Simple Plug Ins**

One of the purposes of code profiling is to ensure that you have reusable modules that are simple to plug in. The Script Library is the common place for this in a Domino application. Common subprocedures or functions can then be created and called every time an agent is run or a subroutine is invoked.

The following is an example of a subprocedure to be called when we start an agent:

```vbscript
Sub ProfilerInit(strProgramName As String)
    If (gLog Is Nothing) Then
        Set gLog = New NotesLog(strProgramName)
        Call gLog.OpenNotesLog(“”,”i:\notesdb\los\debug.nsf”)
        nTabs = 0
    End If
End Sub
```

Prior to any specific operation being called, we may wish to call the following subprocedure from our script library:

```vbscript
Sub ProfilerRoutineBegin(strRoutineName As String)
```

198 Performance Considerations for Domino Applications
If Not(gLog Is Nothing) Then
    Call gLog.LogAction(Space$(nTabs) & strRoutineName_ & " BEGIN")
    nTabs = nTabs + 5
End If

End Sub

There are also similar subprocedures that are called when you end your operation:

Sub ProfilerRoutineEnd(strRoutineName As String)
    If Not(gLog Is Nothing) Then
        nTabs = nTabs - 5
        Call gLog.LogAction(Space$(nTabs) & strRoutineName_ & " END")
    End If
End Sub

Finally, just before you finish the agent, you can call:

Sub ProfilerTerminate()
    If Not(gLog Is Nothing) Then
        Call gLog.Close()
    End If
End Sub

A small extract of your code could be as follows:

Option Public
Option Explicit
Use "Profiler"

Sub Initialize
    ' Profiler_BEGIN
    Call ProfilerInit("Sample Agent")
    Call ProfilerRoutineBegin("Initialize")
    ' Profiler_END

Chapter 5: Troubleshooting an Existing Application  199
Dim session As New NotesSession
Dim db As NotesDatabase
Dim note As NotesDocument

Set db = session.CurrentDatabase

Set note = CreateNewDocument(db)
Call AppendItemsToDocument(note)

'PROFILER_BEGIN
Call ProfilerRoutineEnd(“Initialize”)
Call ProfilerTerminate()
'PROFILER_END

End Sub

Function CreateNewDocument(db As NotesDatabase) As NotesDocument
    Dim note As NotesDocument

    'Profiler_BEGIN
    Call ProfilerRoutineBegin(“CreateNewDocument”)
    'Profiler_END

    Set note = New notesdocument(db)
    Call note.replaceitemvalue(“Form”, “AppFormName”)
    Call note.replaceitemvalue(“Subject”, “New Document”)

    Set CreateNewDocument = note

    'Profiler_BEGIN
    Call ProfilerRoutineEnd(“CreateNewDocument”)

200 Performance Considerations for Domino Applications
Sub AppendItemsToDocument(note As NotesDocument)
    Dim session As New notessession
    Call note.replaceitemvalue("From", session.username)
    Call note.replaceitemvalue("DocumentType", "0")
End Sub

This is expressed graphically in the following diagram:

The diagram illustrates the following:

1. ‘Main’ calls ‘Agent Start’ in the profile script, which creates an entry in the Agent log.
2. Before we call ‘Func 1’ there is a log entry created in the Agent log by the ‘Function Start’ routine.
3. ‘Func 1’ returns back to ‘Main’, which will then log that the ‘Func 1’ has completed by calling ‘Function End’, which in turn creates a log entry.

4. This is repeated each time a function is called by the main routine and could be repeated if any of the subroutines call different subroutines.

5. Before ‘Main’ ends, a log entry is created by the ‘Agent End’ function, which logs the time that the agent finishes.

The following figure shows some sample output from the agent log when we run it for three different agents.

---

**Summary**

In this chapter, we described what elements and features cause an application to show poor performance. Once we identified them, we described some of the tools that are available to help us to find performance hot spots. We finished by looking at two methods that could help us to review performance problems. The first method, reviewing RPCs, enabled us to look at how much communication there was between the client and the server. The second method, code profiling, allowed us to monitor the execution of our code.

---

202 Performance Considerations for Domino Applications
Appendices

The following appendices are grouped as follows:

- Group A: Application examples and tool descriptions.
- Group B: Programming Techniques (time views, seamless switching between databases, cascaded deletes, and so on).
- Group C: How some of the features of Domino work and some limitations.
Appendix A-1
How Do You Put 8GB of Data in a 4GB Bag?

This appendix presents a case study of how to design and implement a large-scale Domino application with focus on capacity and scalability. It describes our experiences with an IBM internal program with a tiered, distributed data architecture, a large data volume, and a large user population. Using this application as an example, we discuss:

• Design alternatives
• Design details of the chosen solution
• Performance tests run before deployment

This appendix is a modified version of a paper that was presented at an internal IBM Best Practices conference in May 1999. It was written and presented by Ralph Hueske, Pramod Singh, and James Willard, from IBM Global Services.

Introduction to the Application

IBM is tracking all of its business opportunities using a single application. Here an opportunity is tracked from birth to closure (win or lose). Using a single application enterprise-wide provides many benefits, including the following:

• Overview and prioritization of all business opportunities in all customer industries and segments
• Allocation of the right people across organizational barriers to handle the opportunity
• A consistent process for communicating with the customer

The IBM opportunity management application was originally developed to run under the VM host operating system and to use its own proprietary database. Due to the downswing of IBM’s VM host environment this application is being moved to the Lotus Notes platform.
Problem Statement

We begin with the basic application requirements. In particular, these are:

- Huge database requirements, that exceed 15GB worldwide. Partitioning this database by geographical region reduces but does not eliminate the problem. The Americas region (currently 7GB) and the Europe, Middle East, and Africa regions (currently 6GB) each significantly exceed the 4GB limit within Lotus Notes Release 4.x.

- A user population of over 50,000 worldwide, with approximately 20,000 registered users within the Americas and nearly that number in Europe, Middle East, and Africa.

- Key user access profiles:
  - Users within a given geographical deployment will typically need to access fewer than 100 documents.
  - Any given document will typically need to be accessed by fewer than 12 users within the geography.
  - Any given user may be assigned to work with any document; therefore, it is not possible to determine (a priori) a partitioning scheme for the user population.
  - No user is permitted access to a document unless specifically assigned as a participant in that document.

- Double Byte Character Set support is required for the Northern Asia-Pacific region (Japan) as well as key countries within the Southern Asia-Pacific region (Korea, China, and Taiwan).

- Seamless, “real enough” time (on the order of less than hourly) coexistence with the legacy application during the gradual migration of users to Lotus Notes. This migration is anticipated to span two to four months for each deployment.

The primary reason for the gradual migration requirement was to ensure that the infrastructure could support the large and active user population. By allowing as few as one user at a time migration, the infrastructure could be continually monitored and adjustments made before the network and servers became overloaded and the application unusable. Alternatively, a “big bang” approach, where a large user population is migrated at the same time, would not only make it difficult to manage the infrastructure, but would make it extremely difficult, if problems occurred, to determine which infrastructure component or components were constrained.
Coexistence with the DB2/MVS database was complicated by a complex data model. It is not a one-record to one-document mapping and NotesPump was inadequate to handle the nontrivial data transformations required in the transfer of data between DB2/MVS and Lotus Notes without additional programming. The logical choice was to write a custom interface between DB2/MVS and Lotus Notes that utilized the IBM MQ Series messaging interface.

- Disconnected operation is required by end users for the purpose of updating information in existing documents as well as creating new documents.
- Document activity can reach 75,000 updates (or creates) per ten hour day worldwide, with peak activity in the Americas region exceeding 30,000 per day (or nearly one/second).
- All reporting and analysis will be done through a DB2/MVS relational database providing a worldwide Information Warehouse.

**Alternative Solutions**

In order to meet the requirements and also comply with the internal standards and guidelines, several architectural solutions were explored. Lotus Notes 4.x has a size limitation of 4GB, with a performance implication if the database size grows to more than 3GB. Every architecture solution was measured against the following five important questions which had to be addressed.

1. How do we put all of the data (or documents) in a single Lotus Notes database with a 4GB size limit?
2. How do we manage the large user population or distribute the users to get optimal performance?
3. How do we achieve participant-level security and keep the database size small on the users’ workstations?
4. How do we make the application coexist seamlessly in “real enough” time with the legacy VM-based application?
5. How do we comply with the applicable internal standards and guidelines?

**Three-Tiered Distributed Data Architecture — MQ on Server**

One of the very first designs that we considered was to distribute the documents on application data servers. In this design, each server would contain a replica of the primary database and documents for only those users who were registered on that server. There would not be any replication between these servers. Each document would contain an
Authors and Readers field. All of the participants (their Notes IDs) would be listed in the Authors/Readers field, and when a user replicated, he or she would only get those documents in which their own ID is contained in the Authors/Readers field. This means that when users replicated their application database with the Application Data Server, they would only receive those documents in which they were listed as a participant. Application database agents on these servers would run every 30 minutes to synchronize the data with DB2/MVS using MQ Series APIs.

This solution met the first four requirements but failed on the fifth architecture requirement. The main drawback of this design was that it violated internal guidelines. Application data servers cannot have agents running every 30 minutes due to the severe performance impacts on the user population when attempting to access those servers.

Since one of the main customer requirements was for the Lotus Notes environment to coexist with the legacy VM application, we needed the data on Lotus Notes to be synchronized frequently. This would enable the legacy VM application users to access the updated documents within a reasonable time (say a few hours). Failure to accomplish this violates our fourth architecture requirement.

**Three-Tiered Distributed Data Architecture — MQ on client**

The thought that went into the second design was how could we take the first design and modify it so that we didn’t have to run the agents on the application data servers every 30 minutes, but still synchronize the application data with DB2/MVS.
One way this could have been done was to use MQ Series APIs on the client machine to synchronize the data with DB2/MVS (refer to the next figure). In this design, whenever a user updated a document on the client, they had a choice to update the document in DB2 directly. That would have allowed legacy users to access the updated documents as they happened. Application data servers would have to have a nightly agent which would synchronize the Opportunity databases on Notes with DB2, thereby closing the loop.

This design eliminated the problem of running the agents every 30 minutes on the application data servers, but had some other drawbacks. With this approach there was more chance of users losing their changes if other users were going to update the same document. Furthermore, installation of the application would have required an MQ Series dynamic link library on every client workstation, making the installation difficult and also requiring additional software on the client. This was not in compliance with stated internal workstation standards. Also, users would have to work in connected mode more often, thereby taking away the Lotus Notes feature of working in a disconnected mode.

Four-Tiered Distributed Data Architecture — MQ on Staging Server

Finally, we decided to put another layer of servers (called staging servers) between DB2/MVS and the application data servers. The agents could then run every 30 minutes on these staging servers to synchronize the application data with DB2/MVS. Since end users are never allowed to access these staging servers, it is not a problem to run frequently scheduled agents, as the agents will not directly impact any end user access. Replicas of the application database could reside on the application data servers, with each replica containing documents for only those users who are registered on that particular application data server.
Since a single Lotus Notes Release 4.x database could not handle the size requirements for the Americas or Europe, Middle East and Africa, we decided that no single document will exist on more than one staging server. Each staging server will contain at most 20,000 base documents (plus all of their associated response documents), which will keep the database size optimal without any performance degradation. This allows the total number of base documents to be distributed among multiple staging servers and also makes the architecture scalable. If the number of base documents grows too large, the deploying geographic centers can add another Staging Server. In order to make sure the documents only end up on one Staging Server, a pseudo-random algorithm will be used.

In this design (shown in the figure above), Staging Servers will have Lotus Notes agents running every 30 minutes, sending and receiving new or modified data to and from DB2/MVS using MQ Series. Each Staging Server will replicate with all Data Servers using a PULL-PULL replication model. Each staging server will pull all its documents and the data servers will pull their documents. Then, when the client replicates its application database, it will pull down all of the documents in which it is a participant (that is, the client’s Notes ID is in the Reader Names field) and which contain changes from the legacy system. The implementation details of this solution are described in the next section.
The Tiered, Distributed Data Architecture

Supporting the stated requirements within a Notes environment places several demands upon the design and development process. As previously described, there were several alternatives that were available, but each had its own drawbacks and some had severe limitations that would have adversely impacted the customer’s ability to access the data in a timely manner. To satisfy the greatest number of customer requirements, as well as minimize the amount of application code that would have to be developed that did not directly support end user functionality, it became clear that there were four key customer requirements that had to be considered as pivotal. Specifically, these are:

- The application data had to be accessible by the users of the legacy VM application in real time using fully supported IBM program products. Failure to exploit program products would require that extensive application code be written to implement infrastructure support that is inherent in the program products. This drove the decision making towards the use of DB2/MVS as the natural host environment for the legacy data.

- Access to the application data by the users of the Notes application had to be responsive and timely. For the re-engineering of the application to be successful, the end user must want to work in the Notes environment rather than be forced into it. To do this the application had to exploit the replication of required data onto the client while screening out data that was not needed. This implies that there will be a server copy of the database (.nsf) as well as individual replicas of the database on each client workstation.

- The application data that was viewed or modified within one environment had to be kept synchronized with the data in the other environment within a reasonable time frame (or real-enough time). To achieve this there had to be frequent synchronization of the data during the normal business day. Typically this was on the order of every half hour. This high-frequency synchronization activity could not be performed on normal application data servers within our worldwide Notes architecture standards and guidelines, and therefore required the introduction of a staging server copy of the database so that timely synchronization with DB2/MVS could take place. To achieve the level of individual data element control required by the application, this synchronization was performed using MQSeries in conjunction with the MVS Transaction Management System (CICS).
Data had to be partitioned across multiple replicas of the database since no single instance of the database could contain all of the data within a geographical deployment center. This required the introduction of a scheme to isolate parts of the data onto separate replicas without permitting all of the data to ever reside on a single replica.

What is the Tiered Architecture?

The following figure illustrates the basic structure of the tiered architecture. As you can see, there is a relational data repository, DB2, residing on MVS; a staging server that maintains data synchronization between the Lotus Notes documents and the DB2/MVS database; an application data server (or working set server) that is accessible by the client for replication, and finally the client workstation where the end user operates. At this point, we are only considering one staging server and one application data server so that we can focus on the tiered nature of the architecture. We will explore the data distribution to multiple staging servers later. Let’s examine each of these tiers in more detail.

The DB2/MVS Tier
An interim data model was constructed based upon the logical data requirements of the original legacy VM application (which has its own proprietary database). This model was then transformed into a DB2 physical database design. Migration utilities were developed to move the data from the internal data structure used by the VM application to the DB2/MVS physical database. Simultaneously, the legacy VM application data access layer was modified to use the SQL/DS interface to interact with the DB2/MVS data tier. Data stored in this tier is in pseudo 3rd normal
form (loosely this means that data depends upon the key, the whole key, and nothing but the key) with prudent denormalization done to enhance end user performance and facilitate the interface with Lotus Notes.

The legacy application also has interfaces to both upstream and downstream applications. These interfaces use DB2/MVS batch programming under static SQL to perform their data access. These updates are immediately available to the VM end user interface while there is a slight delay in delivering this information to the Lotus Notes end user interface.

This tier of the architecture is the only place where all of the application data resides in a single database. All of the other tiers are designed to contain the data in a distributed fashion. This is a typical back end data store for legacy applications running on either VM-based mainframe systems or MVS-based mainframe systems. A similar back end data store could exist in other environments where the data is maintained by a legacy application in a relational database management system (RDMS).

The use of such a relational database system forms a solid foundation for the data requirements of the operational system. Data can be stored in normalized form with prudent denormalization to focus on critical operational performance characteristics. It also facilitates the interface between the operational environment and an information warehouse where data archiving, reporting, and mining is possible. This dramatically relieves the operational environment from supporting query and reporting functions and enables the information warehouse to implement dramatic denormalization of the data to enhance the performance of predefined structured reports. It also supports reasonably efficient ad-hoc query capabilities without undue performance impacts on the operational environment.

The Staging Server Tier

The relational data model used within DB2/MVS was translated into a document-based data model with appropriate duplication of data between documents to satisfy performance and meet end user functional requirements. These documents are stored on one or more staging servers within the Lotus Notes environment. The contents of these documents are kept synchronized with the DB2/MVS tier through the use of an MQ Series messaging interface. An agent is used on the staging server to handle the transaction messaging interface. The other half of the messaging interface is a CICS transaction that handles the document-to-relational-data transformation as well as ensuring that referential integrity is maintained within DB2/MVS.

Synchronization occurs periodically (typically every half hour) on the staging server. All communications are initiated by a Lotus Notes agent running on the staging server. Its first task is to perform a protocol exchange with CICS. The next step is to transmit the data from any modified documents to DB2/MVS. MQ Series handles the conversion from
ASCII to EBCDIC character sets (and vice versa). Once all new and/or modified information in the Lotus Notes environment is sent, the CICS side of the interface transmits all new and modified data within DB2/MVS to Lotus Notes. The messaging interface is a document-oriented data transmission in both directions, with all fields represented in a generalized data stream. All transformations required to move the data from the document model to or from the relational data model is handled by the CICS transaction. This includes, but is not limited to, the support for the data denormalization required within the Notes documents (for example, the same field occurring in several different documents) as well as maintaining the referential integrity required within DB2/MVS. All base documents and their related response documents reside on this staging server.

The Application Data Server Tier
Since end users are not permitted direct access to staging servers, it is essential for the application data to also reside on one or more application data servers (working set servers). End users are assigned to a single application data server where the documents they are allowed to access will reside. These documents are synchronized with the staging server through normal Lotus Notes replication on a high priority basis, and with the client’s workstation on a frequency determined by the end user’s replication settings, typically every 20 minutes.

The use of such a delayed synchronization interface between the relational database and the Lotus Notes document structure, and subsequent replication with the application data server, would not be suitable for some time-critical business functions since it will introduce noticeable delays in data currency. However, it is also arguably inappropriate for Lotus Notes to be the principle end user interface when time-critical functionality or transactional integrity is required. Neither of these restrictions apply to this application since we are dealing with a “real enough” time business requirement. “Reasonably” current data is appropriate for the supported business process; therefore, this small delay in providing data currency is not a problem.

The Client Workstation Tier
As with most Lotus Notes-based applications, the best end user-perceived performance is available when using a local replica. To this end, all of the Lotus Notes users of this application (with the exception of key administration personnel) are required to use a local replica of the database. Access to the application data server copy of the database is prohibited for the normal user. A local replica also allows each user to have access to their data while operating in a disconnected mode. The full functionality of the application is available while disconnected; however, as is typical with the use of a local replica, the data is not available to others unless and until the client has replicated with their assigned application data server. Data replicated to the
client is controlled through the use of a Reader Names field on all base documents and related response documents that are required by an individual. Based upon the original requirements, the number of documents that a single individual will require is typically less than 100. This keeps the size of the client database small and ensures very reasonable performance characteristics.

Data Distribution and Scalability

Now that we have seen each of the tiers of the architecture, we can examine the scalability and data distribution strategy that is employed to support the very large number of base documents as well as the large user population. These two aspects of the data distribution and scalability are easily separated into the two server tiers. The large user population can be best addressed at the application data server tier, and the very large data requirements are best addressed at the staging server tier. Let’s examine the scalability regarding the user population first.

End User Data Distribution

The key to user scalability lies with distributing the end users across the application data servers as shown in the figure below. In order to keep to a minimum the amount of data that needs to be stored (and handled) by a single application data server, the end user population is divided among multiple servers.

Each end user is assigned to a specific application data server based upon their likelihood of sharing access to any given opportunity. The name of the assigned data server is stored in an Authorized User Profile maintained by the application. Whenever a user is granted access to a given base document,
that document and all of the related response documents that the user requires will have the end user’s Notes ID and the ID of their assigned application data server added to the Reader Names field.

This process ensures that all of the documents required by the end users assigned to an application data server will be replicated to that server. Furthermore, if there are no end users registered to a particular data server that require access to a given base document, the name of that application data server will not appear in the Reader Names field; thus, that document will not be replicated to that application data server. Hence, we have distributed the user population across a finite set of application data servers with only those documents required by the users registered to that application data server being replicated to that server.

The application data servers can be viewed as holding a distributed subset of documents based upon the user population assigned to each server. There is minimal duplication of documents across servers and maximum sharing of those documents on a single server. Since there is a reasonable amount of teaming within the application business process, this provides a suitable way to distribute the data while not precluding cross-teaming activity where necessary.

**Data Distribution Across Staging Servers**

To facilitate the distribution of data across multiple staging servers and to ensure that a base document and all of its related response documents reside on the same staging server, the concept of an “owning” staging server is introduced. Each base document is assigned an “owning” server at the time it is created (see the following figure).

---

216 Performance Considerations for Domino Applications
Currently this is done using a random distribution process. A base document and all of its related child documents are therefore “owned” by one and only one staging server. This is accomplished by storing the staging server assignment in the base document, and results in the Reader Names field within the base document containing the Notes ID of the assigned staging server. This information is copied to all related response documents as well.

The random assignment of one, and only one, staging server to each base document and its related response documents causes the distribution of those documents across all of the defined staging servers.

To ensure that these documents are replicated to the appropriate application data servers, each of the staging servers must replicate with all of the application data servers. This type of mesh interconnection, which on the surface seems complex, is relatively straightforward in practice, considering the number of staging servers required to support the data. Even the largest geographical deployment center only requires four to five staging servers.

Since only those documents that belong on a given application data server will have the Notes ID of that data server in the Reader Names field, only a subset of opportunities will be replicated between a given staging server and a given data server. As you can see, the staging server tier has the data distributed in a random manner to maximize the use of each staging server while equalizing the activity on any given staging server. This also enables the workload peaks to be spread across all of the servers when multiple time zones are being supported.

Each staging server has a pair of MQ Series messaging queues defined between the server and the DB2/MVS host environment. This enables all of the DB2/MVS data synchronization to take place in parallel.

**Data Flow Across the Tiers**

There are a wide variety of data flows that can occur in any reasonable application. Many of these flows are not particularly interesting and simply involve the replication or synchronization of data between the various tiers of the architecture. However, to help understand the characteristics of this data structure it will be helpful to address a typical scenario from the fundamental business process. The basic steps in this process are:

1. Creation of a new base document by an individual.
2. Addition of other individuals who will be working on the document to the list of participants.
3. Recognition by an individual that they have been assigned to work on a document.
The process as handled within the application does not vary significantly whether the individual is working through the legacy VM interface or using the new Lotus Notes interface. The most significant difference is the flexibility that is provided for enhancing the data accuracy within Notes through the use of prompts, pick lists, dialog boxes, and on-the-fly data validation that cannot be easily done within the legacy environment. Therefore, we will examine the process from the perspective of an end user operating within Lotus Notes. We will not dwell on the business data that is entered, but rather we will focus upon what is done to “move” the documents through the multiple tiers.

Creating a Document
When an end user wants to create a new base document, that activity can be performed while totally disconnected from the network. All pertinent reference information is available on the client workstation in a separate reference database. The creation process guides the user through the data entry process, and when complete, it creates a base document along with a participant (response) document that contains information about the person creating the base document. At the same time, a randomly selected staging server is chosen to become the “owning” staging server for these new documents. There may also be other response documents created, depending upon the business data that is entered during the creation of the base document.

Before the creation of the base document is complete, the following Lotus Notes IDs are added to the Reader Names field in both the base document and all of the associated response documents:

- ID of the owning staging server — ensures that the owning staging server will get a copy of the base document and all of its related response documents.
- Administrator support role — ensures that there will always be someone who will have access to these documents on the servers.
- ID of the individual creating the base document — ensures that the creator will have access to the documents that they have just created.
- ID of the application data server where this end user is assigned — ensures that the intervening application data server will get a copy of the base document and all of its related response documents.

At this time the only individual (aside from the administrators) who has access to these documents is the person who created them. However, the base document and all of its supporting response documents can readily move throughout the tiered distributed data architecture as illustrated in the next figure.
For example, this document and all of its supporting response documents (or more simply the document set) will be replicated as follows:

1. At the next scheduled (or manually triggered) replication with the application data server, the document set will be replicated to the server.

2. At the next scheduled replication between the application data server and the owning staging server, the document set will be replicated to the staging server.

3. At the next scheduled DB2/MVS data synchronization with the owning staging server, the document set will be synchronized with the DB2/MVS relational database.

Of course it would not contribute to the business process of managing this document set if there were no other people who had visibility to the set. Hence, we must now add other people who will participate in the business process and work with this document set.

### Adding Participants to a Document Set

Since the only individual with access to our new document set is the person who created it, only that individual can add other participants. This is done within the application by allowing any existing participant to add one or more other individuals as participants in the document set. When each new participant is added to the set, a new Participant response document is created and its Reader Names field is copied from the base document.
The Participant document contains the information necessary to update the Reader Names fields within the entire document set; however, the updating of the Reader Names fields is not permitted by the client workstation. The only exception is when it is originally being created and the Reader Names fields must be initialized. All subsequent updating of the Reader Names fields within the document set are handled by the “owning” staging server when a new Participant document arrives. This ensures that there are no replication conflicts created as a result of two existing participants adding new participants at essentially the same time.

If two or more Participant documents arrive at the owning staging server and they reference the same individual, all but one of them is deleted. In a similar vein, participants are removed from a document set by marking their Participant document “logically” deleted. The unique Participant documents that are not “logically” deleted are then used to refresh the contents of the Reader Names fields of the document set.

Participant information is also synchronized with DB2/MVS, thereby enabling legacy VM users to become participants in the opportunity. In a similar manner, participants added by the legacy VM user will create Participant documents when they arrive at the owning staging server.

**Propagating Documents to Other Participants**

Now that we can add and remove participants to a document set and we know how the Reader Names fields are updated, it is relatively straightforward to see how the document set arrives on the client workstation of a newly-added participant. Once the owning staging server has updated the Reader Names fields of the set, the documents flow to the appropriate application data servers during the next replication cycle as illustrated in the figure below. The set is then ready to be replicated to the client workstation when the end user’s next replication occurs (either scheduled or manual).
Minimizing Application-Induced Replication Conflicts

There are two types of replication conflicts: those created by users and those created by the application.

Replication conflicts created by users are minimized by using field-level merge replication. This enables different individuals to update the same documents at essentially the same time as long as they don’t modify the same field within the document. This is unlikely, since the workflow of the business process minimizes the possibility of multiple participants updating the same field within a document set.

Replication conflicts created by the application should also be eliminated. To accomplish this, all field updates to a document set must be done at a common point; for example, the staging server. Some examples of updates that need to be performed at the staging server are:

- Base document number assignment — performed when a new document set is detected at the staging server
- Reader names field updates — performed when a new, updated or deleted participant is detected
- Event history consolidation — performed when a new event document is detected
Performance Characteristics

The IBM Worldwide Integration Test Center (WWITC) performed a series of performance tests on the application. The following is the summary of their environment and test results.

The testing model used consisted of four application data servers and two staging servers with MQ Series communications channels to a single DB2/MVS host system. They did not include the legacy VM application as part of their performance model since it would be phased out as the end users were migrated onto the Lotus Notes environment.

The modeled workload, in terms of transaction and data volume, was based on the estimates for North America. The performance test had five test cycles, the first four of which examined individual components of the application while the fifth tested the complete application system. While the first four test cycles are interesting from a micro point of view, they do not reflect the overall performance of the application since they focus principally on the client.

Overall Performance Test — Part 1

The performance test of the complete application system was run in two parts. In the first part of this cycle, document sets were created and replicated to application data servers, then to staging servers, uploaded to the DB2/MVS host and replicated back to the application data servers. The average throughput rate for the various parts of the replication/synchronization are shown in the following table. The next two figures show the CPU utilization on the various application data servers during the two hour test period and include the client as well as the staging server load.

<table>
<thead>
<tr>
<th>Solution part</th>
<th>Average throughput rate in seconds per document set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client to application data server</td>
<td>0.5</td>
</tr>
<tr>
<td>Application data server to staging server</td>
<td>0.26</td>
</tr>
<tr>
<td>Agent processing on staging server</td>
<td>0.72</td>
</tr>
<tr>
<td>MQ Series agent on staging server</td>
<td>0.83</td>
</tr>
<tr>
<td>MQ Series and MVS processing time (end-to-end)</td>
<td>3.1</td>
</tr>
</tbody>
</table>
Appendix A-1: How Do You Put 8GB of Data in a 4GB Bag? 223
Overall Performance Test — Part 2

The second part of this test cycle was run while some replication was still occurring between the staging server and the application data servers as a result of part one. The workload mixture was 90% updates and 10% creates, which is typical of a production activity mix. The average throughput rate for the various parts of the replication/synchronization are shown in the following table. The next two figures show the CPU utilization on the various application data servers during the two hour test period and include the client as well as the staging server load.

<table>
<thead>
<tr>
<th>Solution part</th>
<th>Average throughput rate in seconds per document set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client to application data server</td>
<td>0.75</td>
</tr>
<tr>
<td>Application data server to staging server</td>
<td>0.51</td>
</tr>
<tr>
<td>Agent processing on staging server</td>
<td>1.07</td>
</tr>
<tr>
<td>MQ Series agent on staging server</td>
<td>1.4</td>
</tr>
<tr>
<td>MQ Series and MVS processing time (end-to-end)</td>
<td>1.67</td>
</tr>
</tbody>
</table>

**CPU Utilization**

Working Set Servers 1 & 2
Benchmark Run 5B

Time Interval in 2 Minutes (9:18 to 10:45)

224 Performance Considerations for Domino Applications
Based upon the complete test results WWITC has concluded that the application has passed the WWITC performance test certification and is ready for deployment.

**Future Applicability**

In Lotus Notes Release 5, database size limitation is bound only by the operating system-imposed limits, and is certified for 64GB. Even though Lotus Notes Release 5 supports a 64GB database size, it is unknown how the database performs when there are more than 1 million documents.

Tiered data architecture will still be applicable in a Lotus Notes Release 5 environment as it distributes the workload, document, and user population across multiple servers.
Appendix A-2
High Performance Reference Databases

The technique described in this appendix is useful for reference databases with low change frequency and a large number of documents. It was first documented by Jasper Schroder from IBM Netherlands in 1998. It describes an implementation using the database structure in Notes R4.x.

Note   The issues originally addressed with this technique have been solved in Domino R5, but it is still a very interesting technique for creating compact, high-performance reference databases.

I have created a lightweight directory in Notes R4.x. The technique used to build this database is useful for other reference databases with low change frequency and a large number of records. Example: in the Netherlands a list of valid ZIP codes could be stored this way. I have chosen a directory because it easily proves that the concept works.

Traditionally very big directories in Notes R4.x have two major challenges:

• The first challenge for the user is that a very big directory opens very slowly, especially when the user opens the database for the first time. The administration of the unread marks causes this problem. For each database Notes keeps track of the (un)read documents even if the unread marks are not shown in any of the views.

   Note   For databases in the Domino R5 structure, use of unread marks can be turned off

• The second challenge is the replication of the database. The replication from an enterprise server to a workstation can take a very long time — up to several hours.
Based on presentations of how Domino R5.0 would include a mobile light-weight directory, I wondered if I could create such a version for Notes release 4. The answer is yes. Here is an example of how my implementation compares to a traditional implementation of a directory for all IBM employees:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Lightweight directory</th>
<th>Traditional directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database size</td>
<td>200MB</td>
<td>&gt; 1GB</td>
</tr>
<tr>
<td>Number of documents</td>
<td>3,000</td>
<td>400,000</td>
</tr>
<tr>
<td>Number of fields</td>
<td>400,000</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Number of views</td>
<td>One, lookup on name only</td>
<td>Many</td>
</tr>
<tr>
<td>First open</td>
<td>Within 30 seconds (including loading the graphical navigator)</td>
<td>User aborts before completion</td>
</tr>
<tr>
<td>Full replication</td>
<td>&lt; 1 hour</td>
<td>&gt; 10 hours</td>
</tr>
</tbody>
</table>
| Data                    | Application field data only | All employee data |}

The information for multiple employees is stored in one document as follows:

- A multi-value field containing the employee names concatenated with the employee number to make each field value unique
- A field containing the links between the employee number and the field name in which the employee information is stored (the index)
- A field for each employee containing the employee information; the “fields” (name, phone, etc.) within the field are separated by one separator character
- A field for the country code (only employees from the same country are stored in a document, this makes selective replication to workstations possible)
The following table gives an example of the contents of a document:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>Schröder, J (Jasper)(788076109&lt;br&gt;Schröder, T (Taro)(788076112&lt;br&gt;Schröer, M (Marcel)(788076108&lt;br&gt;empindex</td>
</tr>
</tbody>
</table>

With the view setting “Show multiple values as separate entries” turned on for the column showing the “key” field (name and number) you create a view with all the employees in it. As Notes does not tell you in the line which entry of the multiple value field is shown, you are restricted to a one column view.

The user cannot open a document from this view, as the user can in other Notes applications, because the opened document contains the data for multiple employees and in the application you cannot determine which employee is selected in the view. So I created two forms, one to do a lookup for employee information, and one for selecting names to be inserted in the addressee fields in a memo.

Here is the formula for looking up employee information:

```plaintext
FIELD AllText := AllText;
key:=@PickList([Custom]:[Single];“”;“People”;“People”;“Select a person”;1);
index:=@DbLookup(“”;“Cache”;“”;“People”;key;“Index”);
item :=
@Replace(@Middle(key;“(“;”)”);@Word(index;“%”;1);@Word(index;“%”;2));
```

Appendix A-2: High Performance Reference Databases 229
content := @DbLookup("":"Cache";"";"People";key;item);
@SetField("AllText";@If(@IsError(content);"Error";content));
@Command([ViewRefreshFields])

Content

In the form, the displayed fields are computed from the field “AllText” which contains all the person information.

Note In the above situation, you have to go to the lightweight directory to do a lookup. You can also perform the lookup from another database. An example of this is included, together with a sample lightweight directory database, in the Web material that supports this redbook. Look in the appendix “Additional Web material” for instruction on how to get the sample databases.

The lesson learned here is that it is possible to create a big directory in a Notes database which:

- Can be accessed reasonably fast (works over the phone for me)
- Replicates easily over the enterprise servers
- Can be replicated to a local workstation (a directory with all employees of IBM Netherlands is 3.5MB and replicates within 5 minutes)

While a directory has been used in this example, the construction used is useful for other large reference databases as well.

The maintenance (insertions/deletions) of the data is more complicated than in other applications. You can no longer delete a record by deleting a document in the database. Instead you have to process the index or rebuild the document with all the records within. For an example of how to do this, check out the Lightweight Name and Address Book template described in the appendix “Additional Web material.”
Appendix A-3
Third-Party Tools for Performance Measurements

There are a lot of different requirements for testing and reporting on Domino applications. A number of tools come with Domino that can be used to test servers and applications, in addition to monitoring a server’s general health.

A large number of third-party tools have also been developed that can either do things similar to the standard Lotus tools, but in a different manner (for example, link the server monitoring tools into a single operations console that can be used to monitor other services, like a mainframe) or can provide a different use (for example, test an application in a different way).

In this appendix, we look at a number of these third party tools and discuss what use they could provide to application performance.

This appendix is by no means a comprehensive list of tools that are available, nor does it in any way endorse a particular tool or set of tools.

Monitoring Tools

The first set of tools that we discuss are tools that are used to monitor the health of either an application or a server. These tools are often best used to assist you in troubleshooting your applications.

Candle Pinnacle Performance Manager

The Pinnacle suite of tools allows you to monitor a large number of statistics about your Domino environment, including statistics that are not accessible using the standard Domino Administration tools. This includes items such as time taken to open a view, or what resources are being used on your server. For more information, see the Candle Web site at:

www.candle.com
Candle ETEWatch

End-To-End Watch (ETEWatch) is a relatively new product by Candle. It provides warnings and statistics for application performance. The difference between this product and the previously described one is that ETEWatch monitors “round-trip” performance. In other words, it works from a client and is triggered when the user begins a process and completes when the server has sent its response back. This is a very useful tool for finding applications that are not meeting their expected performance. Information on ETEWatch can also be found on the Candle Web site.

Tivoli Manager for Domino

Tivoli has developed a large number of tools for managing enterprise information systems. One of the many modules that is available from Tivoli is Tivoli Manager for Domino. This module monitors Domino servers and reports on their status as one of its tasks. It has two modes of monitoring. The first is capturing statistics and events that Domino reports, which are then processed locally and may be forwarded on; this is called “In-Band” monitoring. The second type of monitoring, called “Out-of-band,” does not rely on the availability of Domino in order to collect its statistics. More information on the Tivoli tools can be found at:

www.tivoli.com

Coast WebMaster

There are a number of excellent tools on the market to monitor Web site activity. WebMaster from Coast Software Inc. is one. A key feature of this tool is that it can reassemble the contents of the Domino log of HTTP events into much more meaningful information, such as the most requested images or pages. This information can then be used to analyze the usage pattern and reconstruct the site if required. The Coast Web site can be found at:

www.coast.com
Stress Testing and Load Tools

Another area for which several tools have been developed is stress testing and loading. These tools often operate in a similar fashion to Server.Load, but many have other features that make them useful.

**Websizr**

Prior to Server.Load for Domino R5, it was not possible to place load on the HTTP task of Domino using Server.Load. The Websizr tool allows you to send HTTP requests to a Web server. What this means for Domino is that the HTTP task can have HTTP requests sent to it. These can be in any URL formats. Websizr is provided by Technovations. More information can be found on their Web site at:

[www.technovations.com](http://www.technovations.com)

**Groupsizr**

The Groupsizr tool, also from Technovations, works in a similar fashion to Server.Load for Domino R4.6. It simulates Notes Remote Procedure Calls (NRPC) to a Domino database and server. The tool can simulate a large number of user and record response times. Information on this tool can be found at the Technovations Web site.

**Proactive Assistant**

The Proactive Assistant tool works in a different manner than most tools for monitoring application performance and placing load on the server. Most tools send RPCs to the server and do not interact with the Notes client. Proactive Assistant, on the other hand, simulates a client’s session by navigating through the fields on a form and filling them in. The time taken to do this is logged and a report can be produced.

This is very useful to see what effect the actual design of a database has on performance. The tools that simulate NRPCs all work “behind the scenes” in Domino, sending the commands that the Notes client would send and not requiring the client to send these commands. The Proactive Assistant tool can be used to provide proof that by changing an application’s design we are improving the performance of the application. In addition, it can also be used to test the difference in Notes client performance between two different environments. This can be done by generating a script and using it in both environments, then checking the difference in the response times.

More information on Proactive Assistant can be found at the G2 Associates Web site at:

[www.g2sys.com](http://www.g2sys.com)
Proactive Load

Also from G2 Associates, Proactive Load is another tool that can be used to place load on a Domino server, similar to Server.Load. It uses the same scripting language as Proactive Assistant, so a Proactive Assistant client can be set up to perform an operation and Proactive Load can be used to simulate the load on a server of another 500 people (the tool will allow for even more) performing the same tasks. Again this tool generates Notes RPCs, which in effect are what the Notes client will be sending to the server.

Information on this tool can also be found on the G2 Associates Web site.
This appendix describes the various options available when creating views which contain time/date @functions in the selection formula or in a column formula.

Much of this material is from the Lotus Technical Paper 147114, available in the Lotus Knowledge Base at http://support.lotus.com/

Background

Time/Date views in Notes can cause performance problems, yet they are a very important way to display information to the user.

To appreciate the value of Time/Date views, think about the following examples:

- A view that displays only Severity One problems that have been entered today.
- A view that displays only orders placed within the last seven days.
- A view that displays only technotes updated within the last 30 days.
- A column in a mail file that displays either the time or the date a message was sent. If the message was sent today, it displays just the time; otherwise, it displays just the date.

Why do these views cause performance problems? The reason is that the selection formula or column formula contains @functions such as @Today or @Now and as such, the formula result is never static. Because of the changing nature of the @function results, the index needs to be updated every time the view is accessed. Unless you have a small set of data, this is going to make the view slow to display.

Several techniques have been developed to allow time/date views to have good performance, even if they contain a large set of data.
Creating Time/Date Views Using Selection Formulas

Here we describe three ways to create time/date views using selection formulas, and we identify the pros and cons of each method.

Use @Today in the Selection Formula

This is the easiest solution and is always up to date, but it can have the worst performance of all the methods. Try to avoid using this type of selection for large views.

In this example we want to select all documents that have the field DueDate within seven days of today's date.

Selection formula
Select all documents with a DueDate within seven days of today.

```
SELECT @Adjust(DueDate; 0;0;7;0;0;0) > @Today
```

Advantages
- This kind of view will always be up to date.
- There are no other tasks to perform.

Disadvantages
- When you open the view the index must be rebuilt. If the number of documents is more than a handful, there will be a noticeable delay, perhaps many seconds for large views.
- The view refresh symbol will always display in the top left, indicating (incorrectly) that the view is out of date. This will encourage the user to click it (or press F9). Doing so will cause a similar delay to the one experienced when the view was opened.
- Whenever UPDATE or UPDALL runs against the database, this view will need to be completely rebuilt.

Use @Environment in the Selection Formula

This solution is also simple and has good performance, but requires an environment variable to be kept up to date.

We use the same example as before: that is, we want to select all documents that have the field DueDate within seven days of today's date.

Selection formula
Select all documents with a DueDate within seven days of an environment value.

```
SELECT @Adjust(DueDate; 0;0;7;0;0;0) > @Environment("Today")
```
Other tasks

- You must create a scheduled agent that runs every night, after midnight to update the value of the environment variable, $Today. This agent does not have to reside within the database that has the time/date view, but the agent must run on every server that contains a replica copy of the database. The formula for the agent can be as simple as:

  ```
  @SetEnvironment("Today"; @Text(@Today));
  ```

- The view index must be rebuilt once per day, after the environment variable has been set. This can be accomplished using:

  ```
  UPDALL dbName -r -t viewName
  ```

  You can create a program record to perform this view update.

Advantages

- There is no delay when opening the view.
- The view refresh symbol behaves normally. That is, the symbol appears only if updates occur while the view is open.
- When UPDATE or UPDALL runs against the database, the view will update very quickly.

Disadvantages

- If the agent fails to run, the view will be out of date.
- If you make a replica copy on a new server and forget to create an agent on this server, the view will not function properly (in most cases, the selection formula will return an error and the view will display nothing). In fact, prior to the first time the agent runs, the view will not function properly.
- If the program record fails to execute, the view will be out-of-date (until either the server performs the view rebuild, or a user presses SHIFT+F9 while in the view).
- If you make a replica copy on a new server, and forget to create a program record for that server, the view will correctly display the first time it is used, but will thereafter be out-of-date (until either the server performs the view rebuild, or a user presses SHIFT+F9 while in the view).

Use @Text in the Selection Formula

This third solution is simple, has good performance, and needs no environment variable.

Again, the same example is used here: we want to select all documents that have the field DueDate within seven days of today’s date.
Selection formula
Select all documents that contain “Today” in the list of dates within seven days of DueDate. Use date-to-text formatting that translates today into “Today”.

\[
\begin{align*}
ddt & := \@Date(DueDate); \\
dwk & := ddt : \@Adjust(ddt; 0;0;1;0;0;0) : \@Adjust(ddt; 0;0;2;0;0;0) : \\
& \quad \@Adjust(ddt; 0;0;3;0;0;0) : \@Adjust(ddt; 0;0;4;0;0;0) : \\
& \quad \@Adjust(ddt; 0;0;5;0;0;0) : \@Adjust(ddt; 0;0;6;0;0;0); \\
dwk\_fmt & := \@Text(dwk; "T1S3"); \\
\end{align*}
\]

SELECT @If (@Contains(dwk\_fmt; "Today"); @True; @False);

Other tasks
• The view must be updated once per day, after midnight. This can be accomplished using:

\[\text{UPDALL dbName -r -t viewName}\]

You can create a program record to perform this view update.

Advantages
• There is no delay when opening the view.
• The view refresh symbol behaves normally. That is, the symbol appears only if updates occur while the view is open.
• When UPDATE or UPDALL runs against the database, the view will update very quickly.
• You do not have to run an agent every night to update an environment variable.
• The view will never simply “not function properly.” The worst case scenario is that the view will not be up to date. When it is first created, it will be up to date automatically.

Disadvantages
• If the program record fails to execute, or if you make a replica copy on a new server and forget to create the program record for that server, then the view will be out-of-date (until either the server performs the view rebuild, or they press SHIFT+F9 while in the view).

238 Performance Considerations for Domino Applications
Creating Time/Date Views Using Agents

This alternative to using a time/date selection formula relies on agents making changes to the documents themselves, or moving the document into or out of a folder.

Save a DisplayInView Indicator in the Document

This method relies on the fact that every document has a computed field which will contain a specific value (mark) whenever the document is saved. All documents with this value will appear in a view of recently modified documents. A scheduled agent is then run (daily) to unmark documents which are older than seven days, so that they no longer appear in the view.

Selection formula:
Select all documents with the “DisplayInView” mark set.

SELECT DisplayInView = "1"

Other tasks

- Code is required in the QuerySave event to initialize the DisplayInView field to “1” during document save. In our example, the setting would be dependent on the value of DueDate. If DueDate is within seven days of today, we set the indicator; otherwise we clear it. This QuerySave code must be in every form that creates documents which will appear in the view.

- An agent must be run daily to update the DisplayInView field:

```vba
'S** Update all ‘DisplayInView’ indicators
Sub Initialize
  Dim s As New NotesSession
  Dim db As NotesDatabase
  Set db = s.CurrentDatabase
  Dim dc As NotesDocumentCollection
  Dim datetime As New NotesDateTime( "" )
  Dim doc As NotesDocument
  Set doc = dc.GetFirstDocument
  While Not doc Is Nothing
    Appendix B-1: Time/Date Views  239
```
doc.DisplayInView = ""

Call doc.save( True, True )

Set doc = dc.GetNextDocument( doc )

Wend

End Sub

Advantages

- There is no delay when opening the view.
- The view refresh symbol behaves normally. That is, the symbol appears only if updates occur while the view is open.
- When UPDATE or UPDALL runs against the database, the view will update very quickly.
- There is no need for a program record in the Domino Directory.

Disadvantages

- If the agent is not run, the view will be out-of-date.
- If the database replicates, changes must propagate to other replicas before the first users enter the database in the morning.
- The database search may be resource intensive for large databases. This is particularly important if many views require time/date document updates, either in this database or others on this server.

Put Documents into a Folder

Put documents into a Most Recent folder. Run a scheduled agent (daily) to remove documents which are older than seven days.

Selection formula

Not applicable. Folders do not have selection formulas. Documents are moved in and out of folders under program (or user) control.

Other tasks

- Code is required in the QuerySave event to put the document into the Most Recent folder. In our example, the action would be dependent on the value of DueDate. If DueDate is within seven days of today, we put the document into the folder. If not, we remove it from the folder, if required. This QuerySave code must be in every form that creates documents which will appear in the folder.
• An agent must be run daily to remove old documents from the folder:

Sub Initialize
    Dim s As New NotesSession
    Dim db As NotesDatabase
    Set db = s.CurrentDatabase
    Dim view As NotesView
    Set view = db.GetView( "Most Recent" )
    Dim doc As NotesDocument
    Set doc = view.GetFirstDocument

    '*** Clean out the folder
    Do While Not doc Is Nothing
        doc.RemoveFromFolder( "Most Recent" )
        Set doc = view.GetNextDocument( doc )
    Loop

    '*** Re-populate the folder
    Dim dc As NotesDocumentCollection
    Dim datetime As New NotesDateTime( "" )
    Set dc = db.Search( {@adjust(DueDate; 0;0;7;0;0;0) > _
        @Today}, datetime, 0 )
    Call dc.PutAllInFolder( "Most Recent" )
End Sub

Advantages
• There is no delay when opening the folder.
• There is no need for a program record in the Domino Directory.
• There are no modifications made to documents (placing a document into a folder updates the folder, not to the document).

Disadvantages
• If the nightly agent is not run, the folder will be out of date.
• If the database replicates, changes must propagate to other replicas before the first users enter the database in the morning.
• The database search may consume significant CPU resources for large databases. This is particularly important if many views require time/date document updates, either in this database or others for this server.

• Users may move documents into and out of the folder. There is no ACL setting to prevent this activity.
Appendix B-2
Dynamic Script Library Loading

This technique is taken from a presentation by Dean Garyet, IBM Software Delivery and Fulfillment.

Background

Normally, script libraries are statically bound to design elements, such as forms or agents. Syntax checking and name resolution occur at compile time.

When a design element is loaded, all the script libraries named in its Use statements are also loaded.

For applications that are highly object-oriented, it is common for script libraries to be used to define the object classes. Usually there is one class per script library. This makes it easier to manage in a team development environment and avoids any 64K limitations.

Problem

We run into problems, however, when our class library grows. Large class libraries have many interdependencies among classes.

Many dependencies exist at compile time that aren’t relevant at run time. For example, only one or two methods of a class might be called during a given execution. The classes or methods actually used at run time may depend on the type of data being processed.

Example

Use "Schedule"

Class X
  ...
  Sub y
    Dim sched As Schedule '< Only reason Schedule needed
      ...
    End Sub
  ...
End Class
In this example, class X, which might have many methods, uses the Schedule class in only one of its methods. But any user of the class must deal with the loading and linking time for the Schedule script library, whether they call the y method or not.

Example
Use "Tape"
Use "Diskette"
Use "CDROM"

Sub processDoc(doc As NotesDocument)
    Dim part As Variant
    Select Case doc.ClassName(0)
        Case "Tape"
            Set part = New Tape
        Case "Diskette"
            Set part = New Diskette
        Case "CDROM"
            Set part = New CDROM
    End Select
    Call part.setDoc( doc )
    Call part.doSomething
    ...
    Delete part
End Sub

This example shows a situation where we have to include three script libraries, even though we might only need one of them at runtime.

We have a subroutine, processDoc, that will process a NotesDocument. Since our database can contain many different types of documents, we’re not sure what kind we’ll be asked to process, so we use a Select Case to check the type of document, then create an object of the corresponding type.

Once we have the right type of object, we associate it with the document and call a method. The method might be overloaded in each class, so the processing is tailored to the type of document being processed.

Including all the script libraries necessary for compilation can result in unnecessarily loading most of them at runtime, even though they won’t necessarily be used during that particular execution of the design element.

244 Performance Considerations for Domino Applications
When a script library is loaded the first time after a database is opened, it must be downloaded from the server to the client, which can take some time, depending on communication speeds. Even after the script library has been cached at the client, it must be linked at runtime each time a Use statement executes, and linking is a time-consuming activity (although this has been improved in Domino R5.0).

Finally, when a script library uses many other script libraries, compile times can be long for developers.

**Circular references**

A more serious problem is a common occurrence when dealing with many interdependent classes: circular references crop up. In this example, class X has a method, xx, that needs to construct an object of type Y, so it must use the Y script library. But class Y has a method, yy, that needs to construct an object of type X, so it must use the X script library. This causes a compilation error. Of course, this is an overly simplified example; often there are many more than just two classes involved in the circular reference loop.

```plaintext
Use "Y"          Use "X"
Class X          Class Y
  ...            ... 
  Sub xx         Sub yy 
      Dim y As Y    Dim x As X
  ...            ...
End Sub          End Sub 
  ...            ...
End Class        End Class
```

The above situation is avoided in languages such as C++ by allowing the declaration of the class to be in a separate file from the implementation of the methods. In this example, you can see that the declaration for class X (its interface) does not require the use of class Y, so the compiler would be able to compile the declarations without encountering a circular reference. Unfortunately, LotusScript requires the class declaration and implementation to be in the same compilation module.

**Solution**

We need a way to dynamically load script libraries on demand for only those classes that are needed at runtime for the data being processed. We also must avoid compilation problems.

The solution is to use the LotusScript Execute statement to execute a Use statement at runtime for the classes needed at that time. The Execute
statement can be used to compile and execute the LotusScript supplied to it. Since compilation and execution of dynamic LotusScript will not be performance friendly, we want to make sure that only a small amount of code is compiled at that time. The technique described here is optimized so that the amount we compile is small and the compile is only done once per class.

The NewObj Function

We wrote a small function, NewObj, to encapsulate the technique. NewObj is meant to be used in the same way as a class constructor (the New method). You pass in the name of the class of object you want to create, and the function returns an object of that class as a Variant.

Function NewObj( className As String ) As Variant

Note that classes to be constructed via NewObj must conform to the following conventions:

1. The class must reside in a script library of the same name.
2. The class constructor cannot require any parameters since no parameters are passed into NewObj (other than the class name).
   \textbf{Note} We did not need to pass parameters to the class constructor in our application. However, we could have allowed parameters to be passed into NewObj, which in turn would be passed into the constructor if necessary. This would impose the restriction that the constructors of all classes to be used with NewObj would need to accept the same parameter.
3. Each class to be used with NewObj must provide a companion “Factory” class that can make objects of the class.

The Factory class

The Factory class must be defined in the same script library as the class itself. This class is used to minimize the amount of compilation that has to be done at runtime.

In this example, the class MyClass has a companion class named MyClassFactory. The MyClassFactory class has a “produce” method that returns a MyClass object. Again, the “produce” method does not accept any parameters, so the MyClass.New method cannot require any parameters either.

\textbf{Class MyClass}

\ldots

\textbf{End Class}

\textbf{Class MyClassFactory}
Function produce As Variant
    Set produce = New MyClass
End Function

End Class

Now let’s look at how the NewObj function is implemented to take advantage of these conventions.

Maintain a Cache of Reusable Factory Objects
First, the NewObj script library maintains a private cache of Factory objects. The cache is implemented as a list of Variants that can hold different types of Factory objects. The list tag names the type of Factory stored at that location. For example, factories(“Schedule”) will contain a ScheduleFactory object. The list is kept at file scope, so once the NewObj script library is loaded, it stays in memory until the script library is unloaded.

Whenever NewObj is called, the cache is checked to see whether a Factory object has already been created for the requested class. If we haven’t previously created a Factory object for the requested class, then we’ll use the Execute statement to compile and run a script that loads the script library for the class and constructs a Factory object. Notice how we build the script source dynamically by combining fixed text with the className parameter value.

Once we load the script library, it remains loaded and available until the module into which it is linked unloads. For example, if NewObj is called from an agent, the script library loaded by NewObj remains available until the agent finishes running.

Once the Execute script loads the script library for the class, it creates a Factory object for the class and stores it in the factory cache. Notice how we have to use a global variable to communicate between the Execute script and the NewObj function.

Once we have a Factory object for the requested class, we can use its “produce” method to construct an object of the class. Since we keep the Factory object in the cache, we can reuse it for future requests for objects of the same class.

Public newObj_factory As Variant
Private factories List As Variant

Function NewObj( className As String ) As Variant
    If Not IsElement( factories( className ) ) Then
        Dim script As String
        script = |
        Use "| & className & |
    End If
End Function

Appendix B-2: Dynamic Script Library Loading  247
Sub Initialize
    Set newObj_factory = New | & className | Factory
End Sub

| Execute (script)
    Set factories( className ) = newObj_factory
End If
    Set NewObj = factories( className ).produce
End Function

You may wonder why we use the Factory classes at all. Script libraries are cached in the cache.dsk on the client. Is there a difference in caching between standard linking and dynamic linking? The answer is that Factory classes are used to minimize the number of times that the Execute script needs to be compiled when NewObj is repeatedly called for the same class. For example, look at this code:

Dim doc as NotesDocument
Dim obj as Variant
Set doc = collection.getFirstDocument
While Not doc is Nothing
    Set obj = NewObj( "MyClass" )
    Call obj.setDocument( doc )
...
    Delete obj
    Set doc = collection.getNextDocument
Wend

In this example, NewObj is called for every document in a collection. Without the Factory class, the Execute statement within NewObj would be encountered in every call to NewObj. The Use statement within the Execute script runs faster after the first time, since the script library being used is cached (indeed, it is already loaded into memory and linked). But still, a compile is required. The loop runs much faster if we avoid the repetitive compiles through the use of the Factory class, and then use the Factory object to generate the object of the desired class.

Usage Example

Here we show you how to rewrite the earlier processDoc example to use NewObj.

248 Performance Considerations for Domino Applications
First, we have to load the NewObj script library with a Use statement. Then, in the subroutine we can call NewObj to create and return an object corresponding to the type of document we have. As before, we then associate the document with the object and call methods of the object class to do the desired processing.

Notice that we didn’t have to use the Tape, Diskette, and CDROM script libraries here as in the previous example. This results in shorter compile times.

Use "NewObj"

Sub processDoc( doc As NotesDocument )
    Dim part As Variant
    Set part = NewObj( doc.ClassName(0) )
    Call part.setDoc( doc )
    Call part.doSomething
    ...
    Delete part
End Sub

Evaluation

Benefits
The key benefit of this approach is that script libraries are only loaded when needed. Once the library is loaded, there is hardly any further penalty for using NewObj.

Another major benefit is that compile times are shortened, since all the classes required for objects constructed at runtime do not have to be loaded at compile time. Returning to our previous example, here the X script library doesn’t have to use the Y script library at compile time. Instead, when a Y object is needed at runtime, the Y script library is loaded dynamically. Of course, this comes at the price of sacrificing syntax type checking, since Variants are used.

Also, as in this example, the circular-use problem is solved. Here, X and Y no longer need to use each other, since they are loaded at runtime as needed.

Use "NewObj"  Use "NewObj"

Class X          Class Y
...             ...
Sub xx           Sub yy

Appendix B-2: Dynamic Script Library Loading  249
Dim y As Variant        Dim x As Variant
Set y = NewObj("Y")        Set x = NewObj("X")
              
...                              ...
End Sub             End Sub
            ...
End Class            End Class

Limitations
The limitations imposed by NewObj are minor. As noted before, classes to be
used with NewObj must conform to some conventions. Also, this technique
only works for classes, not script libraries containing only standalone subs or
functions (although standalone subs or functions defined in the same script
library as a class can be called once the script library is loaded via NewObj).

Performance tradeoffs
As with most solutions, this one has its tradeoffs. On the positive side,
NewObj minimizes the amount of code that needs to be compiled during
development and loaded and linked at runtime. Also, script libraries are not
loaded until just before they are needed, so users don’t have to deal with the
performance penalties of waiting for code to be loaded that they may or may
not need.

On the negative side, there is a small performance cost in compiling the
Execute script. There is also a small penalty for using Variants instead of
typed objects, but in many cases where polymorphism is required this
penalty is unavoidable, even without using NewObj. For example, in the two
examples shown earlier, a Variant was required whether using Select Case or
NewObj. Although not a performance consideration, it is also true that using
Variants reduces type safety.

Conclusion
In conclusion, the technique shown here yields big savings in the time
needed to load and link script libraries by loading only those libraries
needed at runtime; it allows script libraries to be structured better for team
development by allowing one class per script library; it results in shorter
compile times during development; and it eliminates circular use errors
during compilation.
Appendix B-3
Displaying Processing Status

When unavoidable processing time occurs, the user will benefit if visual feedback is provided about the status of the processing.

This appendix contains samples of methods of displaying progress information to a user during extended processing.

Status Line Messages
Displaying a brief message in the status line at the bottom of the Notes window is the most basic technique used to display progress to the user. Status line messages are displayed by using the LotusScript “Print” function.

Messages such as:

100 documents processed...

Followed by:

200 documents processed...

Can be an effective and simple way to show the current status to a user.

If the message contains some indication of how much processing is left, then the user has an even better indication of the current status:

300 documents processed, 1202 left to go...

One problem with this last technique is that sometimes processing is via a view, and it is not easy to determine how many documents are in a view. The function “ViewCount” described at the end of this appendix might be helpful.

Status Line Progress Bar
This technique is similar to the prior one, except that progress is displayed via a graphical progress bar in the status line area. Although only text is actually sent to the status line, the characters used simulate a graphical progress bar.

Sample code:

Sub Click(Source As Button)
    Dim s As New NotesSession
End Sub
Dim db As NotesDatabase  
Set db = s.CurrentDatabase  
Dim dc As NotesDocumentCollection  
Set dc = db.AllDocuments  
Dim doc As NotesDocument  
Dim i As Long  
Dim ii As Long  
Set doc = dc.GetFirstDocument  
i& = 0  
While Not doc Is Nothing  
 ' *** Process the document  
 ' <whatever processing required>  
 ' *** Display progress  
 i& = i& + 1  
 Call PercentComplete(50 , i& , Clng(dc.Count))  
 Set doc = dc.GetNextDocument(doc)  
 Wend  
 End Sub  
 Function PercentComplete(barLen As Integer, _   
 curCnt As Long, _      
 totCnt As Long) As String  
 ' *** Purpose: Displays a percent complete progress indicator  
 ' *** in the print area at the bottom of users screen.  
 ' *** barLen = The length of the progress bar (# of chars)  
 ' *** curCnt = The current count  
 ' *** totCnt = The total count (this count makes 100%)  
 Dim curLen As Integer  
 Dim perComp As Integer  
 Static svPC As Integer  
 Static svLen As Integer  
 ' *** Calculate length and percent complete  
 curLen = (curCnt * barLen) \ totCnt  
 perComp = (curCnt * 100) \ totCnt  
 252 Performance Considerations for Domino Applications
'*** Only print if values have changed from last print
If (curLen <> svLen Or perComp <> svPC) Then
  If (curLen < barLen) Then
    Print Ustring(curLen+1, Chr(127)) & _
    Ustring(barLen-curLen, "-") & _
    Chr(127) & " " & _
    Cstr(perComp) & " % complete"
  Else
    Print Ustring(barLen+2, Chr(127)) & _
    " " & Cstr(perComp) & " % complete"
  End If
'*** Save new values
  svPC = perComp
  svLen = curLen
End If
End Function

Instead of a text message, the status bar shows an (almost) graphical display of progress.

How Many Documents Are in That View?
Anyone who has dealt with progress bars has had to solve this problem. To effectively display a progress bar, you need to know two vital pieces of information:

1. The total processing required.
   For example, if you are looping through documents in a document collection, the total processing is the number of documents in the collection (dc.Count).

2. The amount of processing already done.
   You need to keep track of how much has been processed. In the above example, a count of how many documents have been processed is all that is required.

With these two pieces of data, we can effectively display processing status to the user. Graphically, this display is usually represented by a line (or bar) of reasonable length, with a portion of it highlighted in a contrasting color. The highlighted length should be the same percent of the total length, as the current processing count is to the total processing count.
The key bit of data, which is often difficult to determine in a Notes application, is the amount of data that is to be processed. In particular, the number of documents in a view is not obvious.

Looping around the view to count the documents and then displaying the progress bar as we loop around a second time is a poor use of resources (both computer resources and the user’s time). We need a better way to determine the number of documents in a view.

ViewCount
The following routine returns the number of documents in any non-categorized view. It uses a binary chop technique to get to the end of the view very quickly. For example, in a database of over 43,000 documents, it took only 23 calls to GetNthDocument to determine the exact number of documents in the view (which was 43,487). That means that the overhead of getting the total count for progress bar use was much less than 1/10 of a percent.

Function VwCount(vw As NotesView) As Long

    ’*** Pass in a NotesView object
    ’*** Returns the number of docs in the view
    ’*** Note: If the view is categorized - this function
    ’***     counts the top-most categories only
    Dim docLo As NotesDocument
    Dim docHi As NotesDocument
    Dim done As Integer
    Dim fellOff As Integer
    Dim rCtr As Integer
    Dim inc As Long
    Dim posLo As Long
    Dim posHi As Long
    ’*** Initialise
    VwCount& = 0
    done% = False
    fellOff% = False
    rCtr% = 0
    If (vw Is Nothing) Then
        ’*** No view
        Exit Function

254 Performance Considerations for Domino Applications
End If
Set docLo = vw.GetFirstDocument()
If (docLo Is Nothing) Then
    '*** Empty view
    Exit Function
End If
rCtr% = rCtr% + 1
posLo& = 1
'*** Start somewhere (must be a power of 2)
inc& = 512
'*** Scan forward 'til we fall off the end
'*** (but double our increments each loop)
Do While (Not done%)
    Set docHi = vw.GetNthDocument(posLo& + inc&)
    rCtr% = rCtr% + 1
    While (docHi Is Nothing And Not done%)
        '*** We fell off the end - come on back
        fellOff% = True
        If (inc& = 1) Then
            '*** If we get this far we know we found
            '*** what we’re after
            VwCount& = posLo&
            done% = True
        Else
            '*** Try a smaller increment
            inc& = inc& \ 2
            Set docHi = vw.GetNthDocument(posLo& + inc&)
            rCtr% = rCtr% + 1
        End If
    Wend
    '*** Jump ahead
    If (Not done%) Then
        posLo& = posLo& + inc&
If (fellOff%) Then
    '*** If we’ve seen the end,
    '*** we halve our increments
If (inc& > 1) Then
    '*** but only if we can halve them
    inc& = inc& \ 2
Else
    '*** Hey - we’re done
    VwCount& = posLo&
    done% = True
End If
Else
    '*** If we haven’t a clue how big this is,
    '*** we double our increments
    inc& = 2 * inc&
End If
End If
Loop
End Function

Important ViewCount does not work on categorized views. The GetNthDocument method returns only the top level in a view. The count for categorized views will be the count of top-level categories.

Progress Bar Using Undocumented API Calls
If you search for “progress and nnotesws.dll” in the Notes/Domino Gold Release Forum on http://notes.net, you may find postings with code for accessing the built-in Notes progress dialog through undocumented calls. While this may be a cool feature, note that these calls are not part of the documented Notes C API and you therefore use them entirely at your own risk. Lotus and Iris strongly discourage their use, will not provide any support for them and may cease to provide them in any future version of Domino.
Appendix B-4
Sorting

There is generally not a great need for sorting when building applications with Domino; however, if you do need to sort, this appendix will help you to determine the method to use.


Sorting Scope and Limitations

The scope of this appendix is limited to sorting internal arrays of strings using LotusScript. The techniques used, however, can be applied to many other types of sorting, dealing with many other data types.

Since we will be dealing with internal arrays, we immediately have a size limitation of less than 16KB for our array. The exact amount less is about 300 bytes (some LotusScript overhead, maybe), but the routines described in this appendix have been tested with up to 16,300 entries in the arrays.

Sorting Algorithms

There are many sorting algorithms documented in many programming books. We’ve tested various algorithms and have picked out a few for inclusion here.

The ones we picked are interesting for a number of reasons. Our criteria for inclusion included the following factors:

- Is the algorithm fast?
- Is the algorithm a common one, but slow?
- Is the algorithm easy to understand?
- Does the algorithm scale well?
- Does the algorithm work well under all conditions?
Where Can You Find Sorting Routines?

We have seen a few sorting routines described in the various Domino forums and picked some of these for testing. We had some interesting results with the very first one we chose to test:

- Several comments in the forums mentioned that, although there were few comments in the code, it was not a problem. The routine was a “library” routine so maintenance would not be required. However, we found one bug and two performance inhibitors.

- The routine was obviously copied from either a book or another program (it had wrong syntax). The main looping variables were declared as:

  ```plaintext
  Dim i, j, k, upper, lower, temp As Integer
  ```

  As we can see, this means all looping variables are variants, except for “tmp”. As we also are aware, variants are much slower than integers.

- One part of the code tested against 0, which is fine when your array bounds start at 0. This code failed for arrays starting at 1 or greater.

- The routine had the option of sorting with or without case sensitivity. It worked, but the code used a simple “<” for the compare. The “StrCompare” internal LotusScript function is faster and takes into consideration the collation table. Actually, for case-sensitive sorting there was no appreciable difference in speed, but for non-case-sensitive sorting, the “<” code was twice as slow, since it had to “same-case” each string before the compare.

  **Tip**  Don’t pick an algorithm that doesn’t have credentials.

After our experience with our first sort routine, we decided to do a bit more investigation, so we used a book called *Algorithms* by Robert Sedgwick to give more credibility to this document.

**Bubble Sort**

We’ll start with the Bubble sort. We can hear some of you say “I knew that was a good way to sort,” while we can also hear cries of “Why is that one in here?”

The Bubble sort is described here because most of you have seen it during your education. Unfortunately, it performs very badly in most cases. There is one case, however, where the Bubble sort is the fastest type of sort — if the array is already in order.
Below is the code:

Public Sub bubbleSort( ar() As String )

' *** For i = 1 to N-1
' ***  // Bubble the largest item to the end
' ***  // of the array on each loop
' ***  For k = 1 to N-i
' ***    If item at k+1 is less than item at k Then
' ***     Swap them
' ***  Loop
' ***  If nothing was swapped - exit loop
' *** Loop

Dim Lower As Integer
Dim Upper As Integer
Dim i As Integer
Dim k As Integer
Dim swapped As Integer
Dim tmp As String

Lower% = Lbound( ar$( ) )
Upper% = Ubound( ar$( ) )

For i% = 1 To Upper% - Lower%
    swapped% = False
    For k% = Lower% To Upper% - i%
        If ar$( k%+1 ) < ar$( k% ) Then
            ' *** Swap them
            tmp$ = ar$( k%+1 )
            ar$( k%+1 ) = ar$( k% )
            ar$( k% ) = tmp$
            swapped% = True
        End If
    Next k%
Next i%
Selection Sort

This is a very easy sort to understand and is very useful when dealing with small amounts of data. However, as with Bubble sorting, a lot of data really slows it down. Selection sorting does have one advantage over other sort techniques. Although it does many compares, it does the least amount of data moving. Thus, if your data has small keys but large data area, then selection sorting may be the quickest.

Below is the code:

```vbscript
Public Sub selectionSort( ar( ) As String )

    '*** For i = 1 to N-1
    '***  // Find the smallest in the unsorted part of the
    '***  // array on each loop and remember its index
    '***  Set minimum index iMin to i
    '***  For k = i+1 to N
    '***    If item at k is less than item at iMin Then
    '***      Set iMin to i
    '***  Loop
    '***  Exchange the item at iMin with entry at i
    '***  Loop

    Dim Lower As Integer
    Dim Upper As Integer
    Dim i As Integer
    Dim k As Integer
    Dim min As Integer
    Dim tmp As String

    Lower% = Lbound( ar$( ) )
    Upper% = Ubound( ar$( ) )

End Sub
```

260 Performance Considerations for Domino Applications
For i% = Lower% To Upper% - 1
min% = i%
For k% = i% + 1 To Upper%
    If ar$( k% ) < ar$( min% ) Then
        min% = k%
    End If
Next
' *** Swap 'i' and 'min'
If (i% <> min%) Then
    tmp$ = ar$( min% )
    ar$( min% ) = ar$( i% )
    ar$( i% ) = tmp$
End If
Next

End Sub

Insertion Sort

This sort can be useful when adding one or more items to an existing array which is already in order. The new items are added to the array and the array is resorted. With small additions, this can be fast, but sorting an entire array can be very time consuming for large amounts of data.

Below is the code:

Public Sub insertionSort( ar( ) As String )
' *** For i = 1 to N-1
' *** // i is the upper bound of the sorted part
' *** // of the array
' *** // On each main loop, take item at i+1 and see
' *** // where it inserts in the sorted part of the
' *** // array
' *** // As you pass each item > i, move it up one
' *** // slot, making room for i when we get there
' *** newitem = item at i+1

Appendix B-4: Sorting 261
'*** For k = i to 1
'*** If newItem is less than item at k Then
'*** Move item at k up one
'*** Else
'*** Exit the loop
'*** Loop
'*** Put newItem in at k+1
'*** Loop

Dim Lower As Integer
Dim Upper As Integer
Dim i As Integer
Dim k As Integer
Dim v As String

Lower% = Lbound( ar$( ) )
Upper% = Ubound( ar$( ) )

For i% = Lower% To Upper% - 1
  v$ = ar$( i% + 1 )
  For k% = i% To Lower% Step -1
    If ( v$ < ar$( k% ) ) Then
      ar$( k% + 1) = ar$( k% )
    Else
      Exit For
    End If
  Next
  If ( k% <> i%) Then
    ar$( k% + 1 ) = v$
  End If
Next

End Sub
Quick Sort

This sorting routine is very complex but also very fast. The implementation shown here uses recursion and can easily blow the stack if data is already in reasonable sort order. For example, when data is already in order, the maximum number of items we could sort before we got the “Out of stack space” message was 191. When sorted in reverse order, this number rose to 202. When we had random data, however, we had no trouble sorting the full 16,300 items allowed by the limits on array size.

Another problem with QuickSort is that when data is in reasonable order, it takes a lot longer to sort, approaching the time taken for Selection sort.

The limitation regarding the stack can be removed by coding a push-pop list instead of using recursion. That implementation is a bit slower, but can sort any data configuration without blowing up. Unfortunately, this does not overcome the problem of data already in reasonable order. QuickSort is not quick in that case.

Below is the code:

```vbscript
Public Sub quickSortRecursive( ar( ) As String, L As Integer, R As Integer )
    '*** Standard quicksort implementation.
    '*** Unfortunately it is pretty easy to blow the stack!
    '*** Routine taken from Sedgewick

    Dim i As Integer
    Dim k As Integer
    Dim s As String
    Dim tmp As String

    If R% > L% Then
        '*** Partition the array into two 'halves'
        s$ = ar$( R% )
        i% = L% - 1
        k% = R%
        Do While True
            Do
                i% = i% + 1
```
Loop While ar$( i% ) < s$
Do
  k% = k% - 1
  If k% < L% Then Exit Do
Loop While ar$( k% ) > s$
If i% >= k% Then
  Exit Do
End If
'*** Swap them
  tmp$ = ar$( i% )
  ar$( i% ) = ar$( k% )
  ar$( k% ) = tmp$
Loop
  '*** Swap them
  tmp$ = ar$( i% )
  ar$( i% ) = ar$( R% )
  ar$( R% ) = tmp$

  '*** Now use QuickSort recursively for each ‘half’
  Call quickSortRecursive( ar$, L%, i% - 1 )
  Call quickSortRecursive( ar$, i% + 1, R% )
End If

End Sub

We leave it up to the reader to devise a non-recursive way to implement QuickSort.

Heap Sort

Heap sort is another complex sorting routine. It is based on a “complete binary tree” algorithm and is well understood; however, the code can be daunting. It is almost as fast as QuickSort but is also consistent for any initial data sequence.

Below is the code:

Sub heapSort( ar() As String )
  '*** 1. Reorder the array so that it is a “complete
binary tree”. This tree satisfies the “heap condition”: Each node has a key greater than (or equal to) the keys of its children.

1. For size = N to 1

Reduce the array size by one by removing the largest (which is always at index 1)

Save this largest in the vacated spot at the end of the array

Loop

The resultant array is in order (using the same space as the completely removed array)

Routine taken from Sedgewick

Dim i As Integer
Dim v As String
Dim midpoint As Integer
Dim Upper As Integer
Dim Lower As Integer

Upper% = Ubound(ar)
Lower% = Lbound(ar)
If (Lower% = Upper%) Then Exit Sub

Convert the data to “complete binary tree” order
midpoint% = lower% + (upper% - lower% + 1) \ 2 - 1
For i% = midpoint% To Lower% Step -1
   Call downHeap(ar$, i%, Upper%)
Next

On each loop, remove the largest (at ‘Lower’) and move it to the top
i% = Upper%
Do
   v$ = ar$(Lower%)
   ar$(Lower%) = ar$(i%)

Appendix B-4: Sorting 265
ar$(i%) = v$

i% = i% - 1

Call downHeap(ar$, Lower%, i%)

Loop Until i% <= Lower%

End Sub

Sub downHeap(ar() As String, st As Integer, upper As Integer)
'*** This routine places the item at index 'st' into
'*** it's correct spot
'*** The tree 'below' the 'st' node must already be
'*** in correct heap order
Dim k As Integer
Dim jay As Integer
Dim lower As Integer
Dim midpoint As Integer
Dim v As String

lower% = Lbound(ar$)

midpoint% = lower% + (upper%- lower% + 1) \ 2 - 1

k% = st%

v$ = ar$(k%)

While k% <= midpoint%
    jay% = k% + k% - lower% + 1
    If (jay% < upper%) Then
        If (ar$(jay%) < ar$(jay%+1)) Then jay%=jay%+1
    End If
    If (v$ >= ar$(jay%)) Then Goto wExit
    ar$(k%) = ar$(jay%)
    k% = jay%
End While

wExit:

If (k% <> st%) Then ar$(k%) = v$

End Sub

266 Performance Considerations for Domino Applications
You can get even more performance improvements by putting the “downHeap” subroutine directly into the main line (duplicating the code). This does make it harder to understand, however.

Shell Sort

Here it is — the final routine in a long list of sort methods. Does that mean it’s the best? Well, in our opinion, yes, the shell sort seems to be the best general purpose sort routine to use.

Shell sort is almost as fast as QuickSort, has similar speed regardless of the starting sequence of the data, and is very simple. Its only negative aspect is that no one can explain why Shell sort works this well.

Below is the code:

```vba
Public Sub shellSort( ar( ) As String )

' *** Do a shell sort
' *** Notice the lack of other comments
' *** - I’m not sure how it works!
' *** Routine taken from Sedgewick

Dim Lower As Integer
Dim Upper As Integer
Dim botMax As Integer
Dim i As Integer
Dim k As Integer
Dim h As Integer
Dim v As String

Lower% = Lbound( ar$( ) )
Upper% = Ubound( ar$( ) )

h% = 1
Do
    ' *** Determine starting ’h’
    h% = (3*h%) + 1
Loop Until h% > Upper% - Lower% + 1
```

Appendix B-4: Sorting 267
Do
    h% = h% \ 3
    botMax% = Lower% + h% - 1
    For i% = botMax% + 1 To Upper%
        v$ = ar$( i% )
        k% = i%
        While ar$( k% - h% ) > v$
            ar$( k% ) = ar$( k% - h% )
            k% = k% - h%
        '*** A 'goto' - sorry 'bout that
        If (k% <= botMax%) Then Goto wOut
    Wend
wOut:
    If (k% <> i%) Then ar$(k%) = v$
Next
Loop Until h% = 1
End Sub

---

**Pros and Cons**

This summary of the strengths and weaknesses of each sorting method will help you decide the best method for you.

**Bubble Sort**

**Pros:**
- Easy to understand and code.

**Cons:**
- Only works at a reasonable speed with small data.

**Selection Sort**

**Pros:**
- Easy to understand and code.
• Has the fewest “moves” of any sort algorithm, regardless of the data arrangement. This means that if you’re dealing with data that is sorted on small keys, but with a large data component, then the Selection sort may prove to be quicker, since it moves the least amount of data.

Cons:
• Only works at a reasonable speed with small data.

**Insertion Sort**

Pros:
• Can be used to add to existing sorted arrays.

Cons:
• When sorting an entire array, it is slow with large data.

**Quick Sort**

Pros:
• Fastest sort when data is in random order.

Cons:
• Complex to understand.
• For data that is in order or reverse order, it can take ten times as long as with random data.
• Needs a lot of stack space. The recursive version blows the stack when data is in a reasonable order at the start.

**Heap Sort**

Pros:
• Is well understood. Since we can easily determine the path through the binary tree, we can guarantee the maximum compares and moves.
• The data order has no major affect on the performance.
• Scales very well.

Cons:
• Very complex code.

**Shell Sort**

Pros:
• It is very fast. The only faster sort was the QuickSort, and then only when the data was in random order.
• It is easy to code. The only complexity is dealing with “h” (see code).
• It scales well. Time for 800 records was 0.33 sec, for 1600 it was 0.87 sec and for 3,200 it was 2.10 sec.

• It works well with most (maybe all) arrangements of the data. When the array was already sorted or reverse sorted, the routine was only marginally faster than with random data.

• It has the least number of “compares” of all the sorts for any data arrangement.

Cons:

• The technique is not well understood. No conclusion has been reached as to why it works.

• A poor selection of “h” can lead to slow sorting.

• There may be an arrangement of data that causes slow sorting.

Other Things to Consider

When sorting, you will have to decide the following:

• Is the sorting case-sensitive (including accent-sensitive)?

• What sequence do we need to sort, ascending or descending?

Case Sensitivity

A very simple rule should be applied here: Use the LotusScript “StrCompare” built-in function to do the compares. It has the same speed as the “<” operator but works equally fast for any case or accent situation. To simulate case-insensitive sorting with the “<” operator, you will need to “same-case” both sides of the comparison. This takes time and will slow your sort down by as much as 100%.

Sort Order

During our tests, we found that sort order had no effect on the test results.

Results

Here we have some graphical results of various data sorted by the different sort routines.

Sorting Times

The following graph shows the times for sorting various numbers of strings. The times are for the sorting operation only — data setup is not included.
As you can see, the bubble sort, selection sort and insertion sort techniques have times that grow in proportion to the square of the number of items to sort. The quick sort, heap sort and shell sort times grow in proportion to the log of the number of items.

![Number of Seconds to Sort Strings](image)

**Compares and Data Moves**

Another factor to consider in sorting is how many compares are done and how many data moves need to be done. These factors relate to your data type — large data may mean that you want a sort routine that moves data the least. Complex comparisons may mean that you want a sort routine that compares the least amount of times.

Shell sort was best for the number of compares.

![Number of Compares](image)
Selection sort was best for the least number of moves:

![Number of Data Moves Chart]

### Conclusion

Sorting can be a complex issue to decide. The amount of time taken to get your data into arrays to allow you to use a sort routine may be longer than the actual sorting. You then will need to get the sorted data out of those arrays. You may be able to sort the data as you are preparing it or use the resulting array as is.

There are many factors that influence the type of sort to use. Maybe creating an extra view will solve all your problems.

In any case, you’ll need to think about your sorting needs. The examples here use internal arrays of strings, so the problem is nicely encapsulated. For these examples, the shell sort routine seems to be the easiest to code and provides excellent performance.

If you are in a position to require a sophisticated sorting routine, spend some time understanding these routines, and do some reading. The differences in performance can be amazing!
Appendix B-5
Switching Seamlessly Between Databases

In multiple database applications, one of the important thing is to navigate between the different databases seamlessly. This appendix describes some of the techniques by which this can be achieved.

Switching Between Databases

In a multiple database application, the data is distributed among the different databases that constitute the application. Navigators will help switching between the databases. In a typical application consisting of more than 10 Domino databases, using navigators alone will not solve the problem. As the user switches between databases, a new window is opened for each opened database. Since Lotus Notes R4 limits the maximum number of windows opened, the application should be coded to take care of such situations.

**Note** The limitation on the number of opened windows no longer exists in Lotus Notes R5.

The other important thing in switching between databases is to make the physical file name of the database and the server name on which the database is hosted available to the application. This can be done by hard-coding the file name of the database in the code, by using script libraries, by using profile documents, and so on.

Sample Application

The following sample application illustrates some of the techniques of seamlessly navigating between databases.

This sample application is a simple banking application which maintains the following data:

- Customer data
- Accounts data
- Transactions
The end users of this application will be the clerks of the bank. The customers approach the clerk to open a new account and to do money transactions. The users are authorized to access all the above-mentioned types of data.

The three different types of data are stored in different Domino databases.

The user should be able to switch between these databases without having to close the application. The navigation between the databases should be transparent to the users. When the user highlights a customer data document in the Customer database, and switches to the Transactions database, the transaction documents for that customer should be highlighted.
For example, the user selects the customer document of John, and switches to the accounts database, by clicking the Accounts hotspot button.

When the user switches to the Accounts database, the related documents in the Accounts database for John are highlighted.
In a similar way, when the user switches to the Transactions database, the related documents will be highlighted. Also, the application does not open multiple windows when the user navigates through the different databases.

The following steps are involved in the process.

1. When the user clicks the navigator hotspot button, the following information get stored in the environment variables:
   - Logical name of the destination database
   - View name in the destination database
   - Name of the navigator in the destination database

2. Using the logical name of the destination database, the application obtains the physical file name and the server name of the database, and the name of the server on which the database is hosted.

3. In a similar way, the application gets the logical name and the physical name of the current database and the name of the current server.

4. Also, the key of the currently selected document is stored in the environment variable. This is used to highlight the related documents in the destination database.

5. All this information is stored in the environment variables.

6. Using these environment variables, the destination database is opened and the related documents are highlighted.

7. The current database is closed.

In this sample application, script libraries and agents are used to achieve the required functionality.

All three databases use the following script libraries, namely:

- LibViewNavigator
- LibSystemSetup

These script libraries are used in the agents of the databases. The databases contain the following agents:

- SetIniVars
- ViewSwitchAgent
The navigator hotspot buttons run these agents. These agents call the methods of the classes defined in the script libraries.

LibSystemSetup
This library contains the definition for the class SystemSetup. This class has the following methods:

- New
- GetDBType
- GetDatabaseName
- GetAllDBNames
Note  In order to make this example as simple as possible, we have
minimized the code in this library and hard-coded our values. In a normal
production application the values would be stored in a setup database and
read from there. The system setup library would also satisfy other purposes
in addition to serving database types and names to facilitate switching
between databases.

Class Variables
The class contains an instance of the NotesDatabase class and the
NotesSession class.

    session As NotesSession
    curDB As NotesDatabase

New
The constructor of this class actually creates an instance of the NotesSession
class and the NotesDatabase class.

    Sub New
        Set session = New NotesSession
        Set curDB = session.currentDatabase
    End Sub

GetDBType
Following is the LotusScript code of the method GetDBType.

    Function getDBType As String
        If (curDB.FileName = “redcust.nsf”) Then
            getDBType = “CUSTOMER”
        ElseIf (curDB.FileName = “redacct.nsf”) Then
            getDBType = “ACCOUNT”
        ElseIf (curDB.FileName = “redtran.nsf”) Then
            getDBType = “TRANSACTIONS”
        ElseIf (curDB.FileName = “redmain.nsf”) Then
            getDBType = “MAIN”
        End If
    End Function

This method returns the logical name of the current database.

278  Performance Considerations for Domino Applications
GetDatabaseName
Following is the LotusScript code of the method GetDatabaseName. The logical name of the database is passed as the parameter to this method.

Function getDatabaseName(dBType As String, DBFName List As String) As Integer
    DBFName(0) = “copenhagen.lotus.com/Asgard”
    If (Ucase(dBType) = “CUSTOMER”) Then
        DBFName(1) = “redcust.nsf”
    Elseif (Ucase(dBType) = “ACCOUNT”) Then
        DBFName(1) = “redacct.nsf”
    Elseif (Ucase(dBType) = “TRANSACTIONS”) Then
        DBFName(1) = “redtran.nsf”
    Elseif (Ucase(dBType) = “MAIN”) Then
        DBFName(1) = “redmain.nsf”
    End If

    getDatabaseName = 0

    If (DBFName(0) <> “”) And (DBFName(1) <> “”) Then
        getDatabaseName = 1
    End If
End Function

The first parameter to this method is the logical name of the database. The second parameter is a string list. The method returns this list with two strings. The first string contains the name of the server. The second string in the list contains the physical file name of the database.
**GetAllDBNames**

Following is the LotusScript code of the method `GetAllDBNames`. This method takes a text list as parameter. The logical names of all the databases of the application are returned through this list to the calling program.

```livescript
Function GetAllDBNames(DBList List As String)
    DBList(0) = "CUSTOMER"
    DBList(1) = "ACCOUNT"
    DBList(2) = "TRANSACTIONS"
End Function
```

**LibViewNavigator**

This script library contains the definition for the class `ViewNavigator`. This class has the following methods:

- SwitchToDb
- SetIniVars

The `LibViewNavigator` script library uses the `LibSystemSetup` library.

**SetIniVars**

Following is the LotusScript code for this method.

```livescript
Sub SetIniVars
    Dim curSession As New NotesSession
    Dim dbType As String
    
    'Get the logical name for the database to switch to.
    
    dbType = curSession.getEnvironmentString("RED_SWITCHTO_DB")
    Call Me.switchToDb(dbType)
End Sub
```

This method gets the logical name of the database from the environment variable and calls the other method of the class.
SwitchToDb
This method takes the logical name of the database as the parameter. Following is the script code for this method.

Sub switchToDb(dbType As String)
    Dim curSession As New NotesSession
    Dim getInfo As New SystemSetup
    Dim destDBFileName List As String
    Dim curDBFileName List As String
    Dim curDB As NotesDatabase

    Set curDB = curSession.currentDatabase
    ' Check, if we are on a customer related form. If so, set the search key'
    Dim curDoc As NotesDocument
    Set curDoc = curDB.unprocessedDocuments.getFirstDocument()

    If Not (curDoc Is Nothing) Then
        If (curDoc.KeyNo(0) <> '') Then
            Call curSession.SetEnvironmentVar("RED_SWITCHTO_KEY", curDoc.KeyNo(0))
        End If
    End If

    ' Convert the logical name to physical server and filename'
    If (getInfo.getDatabaseName(dbType, destDBFileName) = 0) Then
        MessageBox("Cannot find the location for" + _
        "databasetype " + dbType)
        Exit Sub
    End If

    ' Get database name for current database

Appendix B-5: Switching Seamlessly Between Databases 281
dBType = getInfo.getDBType

If (getInfo.getDatabaseName(dbType, curDBFileName) = 0)
Then
    MessageBox("Can not find the location for databasetype " + dbType)
    Exit Sub
End If

' Set the environment strings

Call curSession.SetEnvironmentVar("RED_SWITCHFROM_SERVER", _
curDBFileName(0))

Call curSession.SetEnvironmentVar("RED_SWITCHFROM_DB", _
curDBFileName(1))

Call curSession.SetEnvironmentVar("RED_SWITCHTO_SERVER", _
destDBFileName(0))

Call curSession.SetEnvironmentVar("RED_SWITCHTO_DB", _
destDBFileName(1))

End Sub

This method performs the following steps:

1. It obtains the Key information from the selected document in the view, and sets the corresponding environment variable.
2. Using the logical name of the destination database, it obtains the physical file name and the server name.
3. It obtains the logical name of the current database.
4. Using this logical name, it obtains the physical file name and the server name.
5. Finally, it sets the environment variables.

**Navigator Hotspot Button**

Following is the code behind the navigator hotspot button *Accounts*.

@SetEnvironment("RED_SWITCHTO_DB"; “ACCOUNT”);
@SetEnvironment("RED_SWITCHTO_VIEW”; “vwAcDetails”);
@SetEnvironment("RED_SWITCHTO_NAVIGATOR”; "Nav1”);
@Command([ToolsRunMacro]; "(setIniVars)“);
@Command([ToolsRunMacro]; "(viewSwitchAgent)“)

282 Performance Considerations for Domino Applications
The logical name the database which has the account details is *Account*. This is the logical name of the database to which the user wants to switch. This value is stored in the environment variable. The second and third lines of the code store the name of the view to be switched and the name of the navigator respectively. The fourth statement calls the agent *setIniVars*.

This agent uses the script library *libViewNavigator*. Following is the code for the agent *setIniVars*.

```
Sub Initialize
    Dim nav As New ViewNavigator
    Call nav.setIniVars
End Sub
```

This agent creates an instance of the class *ViewNavigator* and calls the method *setIniVars*. (Don’t confuse this method with the agent.)

The *setIniVars* method obtains the logical name of database to switch. In this particular case, it will be *Account*.

The *setIniVars* method in turns call the other method *SwitchToDB* and passes the logical name of the database to this method.

Now, the method *SwitchToDB* obtains the Key value for the selected document. In this particular case, it will be 903412. This value is also stored in the environment variable.

The *SwitchToDB* method creates an instance of the class *SystemSetup* from *libSystemSetup*.

The *switchToDB* method calls *getDatabaseName* method of *SystemSetup* class. The logical name of the database to be switched is passed to this method. This method returns the physical file name and the server name.

The *switchToDB* method calls *getDbType* method of *SystemSetup* class. This method returns the logical name of the current database.

The *switchToDB* method calls *getDatabaseName* method of *SystemSetup* class. The logical name of the current database is passed to this method. This method returns the physical file name, and the server name.

The physical file name and the server name of both the current database and the database to be switched are stored in the environment variables.
Now the second agent, viewSwitchAgent, is called. Following is the code for this agent.

```lisp
REM "Get the server/file of the database which launched us";
  f_server := @Environment("RED_SWITCHFROM_SERVER");
  f_db := @Environment("RED_SWITCHFROM_DB");

REM "Otherwise get where we are supposed to go";
  s_view := @Environment("RED_SWITCHTO_VIEW");
  s_key := @Environment("RED_SWITCHTO_KEY");
  s_server := @Environment("RED_SWITCHTO_SERVER");
  s_db := @Environment("RED_SWITCHTO_DB");
  s_navigator := @Environment("RED_SWITCHTO_NAVIGATOR");
REM "Go there ...";
  @PostedCommand([FileOpenDatabase];s_server:s_db;s_view;
                  s_key;"";"1");
  @PostedCommand([OpenNavigator]; s_navigator);

REM "Switch back to previous database and close it.";
  @PostedCommand([FileOpenDatabase];
                  f_server:f_db;"";"";"";"1");
  @PostedCommand([FileCloseWindow]);

REM "Expand at current position in the view we\’ve just opened.";
  @PostedCommand([ViewExpand]);
  @All
```

This agent obtains the physical file name of the database to switch and the server name from the environment variables and opens it.

The current database window is closed.
Appendix B-6
Supporting Roles Locally Without Enforcing Consistent ACL

This appendix introduces a technique for local access to roles without enforcing a consistent ACL. Before diving into the actual implementation please note that

- Local roles cannot be used as a security feature with this technique.
- Nothing comes for free — supporting roles locally without Enforcing Consistent ACL requires more application maintenance.
- The performance improvement in use of roles only helps people using the Notes client in the following circumstances (in order of least speed to highest speed):
  - Forms used to create or edit documents on a server-based replica that contain @UserRoles references (in fields, events, hide-whens, and controlled-access sections formulas and so on), which are evaluated on load or refresh. The effect is greater the more references to @UserRoles you replace.
  - Forms like those described above also automatically refresh the document when the user tabs between fields because of one of the following:
    - The form setting “Automatically Refresh Fields”
    - Certain fields having the setting “Refresh Fields on Keyword Change”
    - Other events that reference @UserRoles or cause refreshes
  - Forms like those described above that are being accessed via a high-latency (such as dial-up) Notes client connection.
- This was all tested using Notes Client 4.5.2 and Notes Server 4.5.5 but should apply to higher versions as well.

This appendix was written by Brendan Hoar.
Workarounds for Two Related Issues: The Performance Hit from @UserRoles and Emulating Local Roles Without Using Enforce a Consistent ACL

At my company, we generally cannot use “Enforce a consistent Access Control List across all replicas of this database” since we have scores of domains that have different internal ACL requirements. Using the feature is equivalent to disabling cross-domain replication.

However, we often want to use Roles in those databases for which people create local replicas. While roles are never to be used for security purposes, they make offering different features to different types of users easy to maintain and to code.

I began developing a workaround that allows including the role functionality in local replicas without entailing a great deal of maintenance by Notes development staff, or requiring hard-coded user names in forms or keyword lists that would inevitably get out of sync with the groups used in the ACL. While optimizing my @DBLookups, I happened to have the Notes Debug Window open (see Appendix B-8, “Using NRPC Output”) to examine server transactions, to understand the performance hits of various things while I was experimenting.

I correlated a large number of server transactions (24 in one particular form) to the large number of @UserRoles function calls I was making on a form and its sub-forms. The @UserRoles functions were being called both when the document was loaded as well as when it was refreshed, due to the “Refresh fields on keyword change” setting, a feature that was required for the application. Most of the calls were in Hide-When formulas, but there were a few scattered about elsewhere.

After some experimenting, I found that every time an @UserRoles function is executed from a document stored on a server replica, you get a pair of client-to-server transactions such as:

(819-10223) GET_SPECIAL_NOTE_ID: 153 ms. [16+16=32]
(820-10223) OPEN_NOTE: 381 ms. [28+5910=5938]

Yes, the values are small, but they can (and did) add up. The timing values above are real-world fast ethernet values. Imagine a dial up connection with 12 pairs of those every time you refreshed the form or tabbed out of certain fields!

286 Performance Considerations for Domino Applications
An interesting point is that the value of @UserRoles is not cached as part of the open database object. Every time the @UserRoles function is called, the Notes client checks with the server. If you have many slightly different hide-when formulas, even disabled ones, you generate double that number of server transactions on every document refresh.

Did I also mention I had a solution for Local Roles too? Let’s combine the two solutions into one, since that’s what I did.

For the Local Roles feature, create a form called LookupRoles with two fields:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoleName</td>
<td>Editable Keyword, Single-Value</td>
<td>Default keyword list = @UserRoles</td>
</tr>
<tr>
<td>Readers</td>
<td>Reader Names, Computed, Multi-Value</td>
<td>Formula = RoleName</td>
</tr>
</tbody>
</table>

Also, create a simple view called LookupRoles with the following selection and column formulas:

<table>
<thead>
<tr>
<th>View alias</th>
<th>Selection formula</th>
<th>First column value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LookupRoles</td>
<td>‘Select Form = “LookupRoles”</td>
<td>Readers</td>
</tr>
</tbody>
</table>

Create and save one document for each of the roles in the database. Remember to do this any time you add a role to the application or write a script agent that keeps them up to date.

Now we have role lookup documents that will replicate to local replicas only when the end user has that role to begin with. If I only have [Role1] on the ACL and not [Role2] and [Role3], my local replica will only contain the document for [Role1] in the view when I replicate it down.

Back to @UserRoles performance. Let’s make it so that @UserRoles function is only ever called once, and that is only on document load (or create). Let us also include the lookup to the LookupRoles view during that process.

On all forms or subforms that use @UserRoles, replace all references to “@UserRoles” with just a reference to the field “UserRoles”. Remember to also do this for hide-when formulas in: all table cells, blank lines, action buttons, access controlled section formulas, you name it.
Then add the following two hidden fields at the top of the form (in this order):

<table>
<thead>
<tr>
<th>Field name</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserRolesOnLoad</td>
<td>Text, Multi-Value, Computed For Display</td>
<td>Formula that looks up local roles when document loads and caches them in the document for future reference — See the actual code below</td>
</tr>
<tr>
<td>UserRoles</td>
<td>Text, Multi-Value, Computed</td>
<td>Formula that removes the cached roles before the document is saved — See the actual code below</td>
</tr>
</tbody>
</table>

Here is the actual code for the field UserRolesOnLoad:

REM “Only initialize on document load”;  
@If(@IsDocBeingLoaded;@True;@Return(UserRolesOnLoad));  

templookuproles :=  
@If(@DbName *= "";@DbColumn("";"";"";"";"LookupRoles";1);"");  

lookuproles :=  
@If(@IsError(temploookuproles);"";temploookuproles);  

REM “The field also makes sure that @UserRoles is only ever called once per document. @UserRoles makes a performance hit on all server-based documents. Forms with this code will only”;  

REM “call @UserRoles once on load. Granted, this might sometimes be redundant with the code above, but we include it as a safety feature.”;  

realroles := @UserRoles;  
roles := @Unique(@Trim(lookuproles:realroles));  
roles

And here is the code for the field UserRoles:

REM “Remove cached roles when document is being saved”;  
@If(@IsDocBeingSaved;@Unavailable;UserRolesOnLoad)

Caution  If the person’s roles change, but they already have a local replica of their existing role documents (with the ‘old’ roles) the roles may still exist in the local replica. One way to address this is to have a server-based agent that touches all role documents once a day to make sure they are rechecked during replication. Another more cumbersome way is to have users with changed roles clear the replication history and re-replicate before the “local roles” can be guaranteed to be correct.
Due to issues with use of Computed For Display fields in hide-when formulas in Notes 4.x, UserRoles has to be a computed field, not a Computed For Display field. Lastly, since we do not want one user’s roles saved on a document and passed on to the next reader/editor of that document, the UserRoles field must be set by the code above to @Unavailable during the save (this is the only time UserRoles acts differently than the real @UserRoles). You may need to check for mailing as well in your application.

Therefore, make sure that you do not reference the UserRoles field from the formula language during a save (or possibly during mailing).

The end result in my application: Incredibly fast form navigation and refresh, plus the benefit of Roles on local replicas when “Enforce a consistent ACL...” will not do the job.
Appendix B-7
Cascaded Deletions Example

In multiple database applications the related documents are distributed among multiple databases. This appendix describes some techniques to delete related documents.

Cascading the Documents

Cascaded deletion is a process by which, when a document is deleted, all the related documents are also deleted.

Relational Databases

In relational databases, the data resides on multiple tables. Rows of different tables are related through the foreign key. The relational database applications are based on the Entity-Relationship model, where both the entities and relationship between the entities of the system are represented by tables.

For example, consider an inventory management application developed using a relational database. The Supplier table contains the details of different suppliers. The Parts table contains information about the different items in the inventory. The Supplier-Parts table has information about who supplies which part.

<table>
<thead>
<tr>
<th>Supplier Table</th>
<th>Supplier ID</th>
<th>Supplier Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123,567</td>
<td>James Wood</td>
</tr>
<tr>
<td></td>
<td>867,458</td>
<td>John Joe</td>
</tr>
<tr>
<td></td>
<td>512,890</td>
<td>Siva Sunder</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parts Table</th>
<th>Part ID</th>
<th>Part Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1A</td>
<td>Scale</td>
</tr>
<tr>
<td></td>
<td>2D</td>
<td>Steel bar</td>
</tr>
</tbody>
</table>


The Supplier ID is the primary key for the Supplier table and the Part ID is the primary key for the Part table. These primary keys are used as foreign keys in the Supplier-Part table. Relational databases will not allow the row for the part steel bar in the Parts table to be deleted because it is referred to by some of the rows in the Supplier-Part table.

In other words, relational databases will not allow the user to delete a row, if the primary key of the row is used as foreign key in any other table.

But the relational database application can be programmed to delete the rows in different tables which are related. To delete the row for steel bar in the Parts table, use the following steps:

1. Delete all the rows in the Supplier-Part table that refer to steel bar (Rows 2 and 3).
2. Now delete the row for steel bar in the Parts table.

These deletions can be done using the Structured Query Language (SQL).

**Domino**

Since Domino is not a relational database, Domino by itself does not enforce Primary key or Foreign key in the documents. To cascade the related documents in Domino applications:

You have to identify a Primary key for each document.

Use this Primary key as Foreign key in related documents.

The Primary key should be information that uniquely identifies the document in the database.

There are several ways to generate unique keys. See the section “Using Keys to Relate Data in Different Databases” in Chapter 2 for a discussion of this.

The user can designate one of the fields in the form as the Primary key for the document. For example, in a banking application, the customer’s social security number can be used as the Primary key (as permitted by law). In such cases, the application should be coded in such a way that the user does not create multiple documents with the same key value.

After identifying a suitable method to generate the unique keys, you should establish a mechanism to store the primary key of the document in the related documents as foreign key.
Consider a banking application. The primary key of the customer document should be stored as foreign key in the documents that hold the account details of that customer. Similarly, the transaction documents will store the primary key of the account document.

This can be achieved by allowing the users to create the child documents only from the parent document and inheriting the primary key value from the parent document.

In the banking application example, the users should be allowed to create the account documents only from the customer document, so that the account document inherits the primary key of the customer document. In the same way the transaction documents should be created only from the account document.

If the application is a multiple database application, the name of the server and the database on which the child document is created should be stored in the parent document. For example, if the account documents are stored in a different database, the database name where the account documents are created and its server name should be stored in the customer document.

If all the documents are stored in the same database, set a flag in the parent document to indicate that it has one or more child documents.

This way, when the user deletes a parent document, the child documents can also be deleted recursively.

To maintain data consistency, the users should not be allowed to paste documents in the database. This is accomplished by programming the QueryPaste event of views.
Deleting the Related Documents

Once the relationship between documents is established, the application
should be coded to handle cascaded deletions.

To make the cascaded deletions easier, create views for each document type
by the foreign key.

What follows are the steps to delete all the related documents, when a
document is deleted from the database.

1. Check to see whether the document has one or more child documents.
2. If so,
   • Get all the child documents from the child document view (views by
     the foreign key) using the GetallDocumentsByKey method.
   • For each document returned, repeat step 1.
3. Else,
   • Delete the document.

If there are a large number of related documents, then the time taken to
delete all the documents will also be great. Second, if the users have
selectively replicated the documents and work locally, then deleting all the
related documents will not be possible.

The related documents may reside on different servers. When the user
deletes a document, the server hosting the child documents may be shut
down temporarily.

In such situations, the users can mark the documents as deleted. The actual
deletions can be performed using a scheduled agent running on the server.

Example
The following example shows a simple method of deleting related
documents using the FTSearch method. The example considered here is the
same banking application which was introduced in Appendix B-5,
“Seamlessly switching between databases.”
The Customer Details view of the Customers database has an action button Cascaded Delete 1. The LotusScript code behind this action button is as follows:

```livescript
Sub Click(Source As Button)
    Dim session As NotesSession
    Dim db As NotesDatabase
    Dim docCollection As NotesDocumentCollection
    Dim setup As New SystemSetup
    Dim DBList List As String
    Dim DBName List As String
    Dim keystr As String
    Dim searchstr As String
    
    Set session = New NotesSession
    keystr = session.Currentdatabase.UnprocessedDocuments.GetFirstDocument.KeyNo(0)
    searchstr = "FIELD KeyNo contains " + keystr
    Call setup.GetAllDBNames(DBList)
    Forall dbitem In DBList
        If (setup.getDatabaseName(dbitem, DBName) <> 0) Then
            Set db = session.GetDatabase( DBName(0), DBName(1) )
            
            Set docCollection = db.FTsearch(searchstr,0)
            Call docCollection.StampAll( "DeleteDoc" , "1" )
        End If
    End Forall
End Sub
```

When a user selects a document in the view and clicks the action button Cascaded Delete 1, the selected document as well as all the related documents in the Accounts and Transactions database will be marked as deleted.
As you can see, all the related documents share the same key. The program obtains the list of all databases in the application and searches each database using the key to get the document collection. Then all documents in the collection are marked as deleted.

*FTSearch* method is used to search the documents. The documents are marked as deleted using *StampAll*. This method is faster than updating and saving the document.

The documents that are marked as deleted appear in the view *Deleted Customer Details*. These documents can be deleted either by an agent or manually.
Appendix B-8
Using NRPC Output

The Notes Remote Procedure Calls (NRPCs) is the method that the client and server use to communicate with one another. This is very much a low level, under the cover, network traffic. The user of an application will only see the lightning bolt in the bottom corner of the Notes screen.

As developers, we do not have control over our network design and size. However, we can build our applications to minimize the traffic that is passed down the network, or try to spread the amount of network traffic that we send at once. By monitoring the RPCs that we send from a client we can make faster design decisions.

This appendix provides information on the RPCs and how to use the information that is passed from them.

Note The use of the NOTES.INI variable Client_Clock is not supported or documented by Lotus, and may be removed at a later date without warning or acknowledgement.

Enabling the Output

As mentioned earlier in this book, the output of RPC is enabled by entering the following line into the NOTES.INI file:

Client_Clock=1

Accompanying this setting we also need to tell the Notes Client where to direct this output. It can be either to a window, with the following NOTES.INI setting:

Debug_Console=1

Or to a file with the setting:

Debug_Outfile=c:\temp\debug.txt

In the NOTES.INI file, where “c:\temp\debug.txt” is the name for the text file. You may wish to enable both.
Using the RPC Information

The information that is generated by enabling the Client_Clock is as follows:

\[(\text{Counter-Time [Thread]}) \text{ RPC: Time Taken [Bytes Sent + Bytes Received = Total]}\]

So, for the following line:

\[(2-42 \ [2]) \text{ GET_UNREAD_NOTE_TABLE: 9 ms. } [290+44=334]\]

what we are interested in is that when the RPC ‘GET_UNREAD_NOTE_TABLE’ was called, it took 9 ms to complete, 290 bytes were sent to the server and 44 bytes were received. We can also use the Time value to help separate different events that we initiate.

**Tip** When using the Client_Clock output, pausing between actions will make it a lot easier to break the output down into specific chunks.

**Tip** To be able to map what you are doing to the RPCs that are output, the ‘Debug_Console’ setting and ‘Debug_Outfile’ can be used together. After you have finished your action, check the counter on the Console (the DOS window that is brought up in the background when you enable Debug_Console) screen, and record this number next to the action that you performed. Later when you look through the file you created with the ‘Debug_Outfile’ entry, you can match the numbers that you recorded with the entries in the file.

The RPC relates to the C procedure that is called. Some of these C procedures have been exposed and documented through a CAPI toolkit, though most are internal to the Notes and Domino communications. However, by looking at the name of the procedure called and matching that to what you were doing at the time, it is often relatively obvious what that particular RPC is doing.

The following table shows some of the most common RPC calls.

<table>
<thead>
<tr>
<th>RPC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open_Session</td>
<td>Authenticate with the server and establish a session</td>
</tr>
<tr>
<td>Open_Database</td>
<td>Find and open a database</td>
</tr>
<tr>
<td>Open_Note</td>
<td>Send the contents of a note (data document or design element)</td>
</tr>
<tr>
<td>Open_Collection</td>
<td>Open a view</td>
</tr>
<tr>
<td>Read_Entries</td>
<td>Send a list of information from a view or search, usually follows Open_Collection</td>
</tr>
<tr>
<td>Find_By_Key</td>
<td>A view lookup via DBLookup</td>
</tr>
<tr>
<td>Get_Special_Note_ID</td>
<td>Send info from the ACL</td>
</tr>
<tr>
<td>Close_db</td>
<td>Close DB Session</td>
</tr>
</tbody>
</table>

298 Performance Considerations for Domino Applications
Examples - Open Database
Here we walk through two different examples of a very basic routine to manually open a database on our desktop.

1. Simple Database
This first database is a blank database.

   \[(56-126 \ [66])\] OPEN_DB: 9 ms. \[126+290=416\]  
   \[(57-126 \ [67])\] GET_UNREAD_NOTE_TABLE: 14 ms. \[290+44=334\]  
   \[(58-126 \ [68])\] OPEN_COLLECTION: 30 ms. \[120+8612=8732\]  
   \[(59-126 \ [69])\] OPEN_NOTE: 19 ms. \[48+8458=8506\]  
   \[(60-126 \ [70])\] READ_ENTRIES: 9 ms. \[46+112=158\]  
   \[(61-147 \ [71])\] CLOSE_COLLECTION: 11 ms. \[12+0=12\]

We can use this as a baseline measurement for opening a database. We know what amount of data is transmitted and the number of RPCs that are required to perform this operation.

2. Database with Script
This database is a blank database. Its only difference from the database above is that it has a database script with a ‘PostOpen’ event to get a profile document for the current user.

   \[(47-99 \ [57])\] OPEN_DB: 15 ms. \[126+290=416\]  
   \[(48-99 \ [58])\] GET_UNREAD_NOTE_TABLE: 6 ms. \[290+44=334\]  
   \[(49-100 \ [59])\] OPEN_COLLECTION: 16 ms. \[120+484=604\]  
   \[(50-100 \ [60])\] OPEN_NOTE: 5 ms. \[48+310=358\]  
   \[(51-100 \ [61])\] READ_ENTRIES: 9 ms. \[46+52=98\]  
   \[(52-100 \ [62])\] GET_NAMED_OBJECT_ID: 4 ms. \[82+24=106\]  
   \[(53-100 \ [63])\] OPEN_NOTE: 5 ms. \[48+248=296\]  
   \[(54-116 \ [64])\] CLOSE_COLLECTION: 5 ms. \[12+0=12\]

From this output we can now compare the two simple operations to see what effect the implementation of the ‘PostOpen’ event to load a Profile Document has. In this example we can see that two extra RPCs were run, how long they took, and the amount of network traffic that was added.

Example Creating - Using a Discussion Database
What follows is an example of the output from using a discussion database (Created from ‘discsw50.ntf’ template). The following operations were performed:

1. Open Database
2. Create Main Document
3. Save Document
4. Read a different Main Document
5. Create Response document
6. Close Database

The output was annotated after completion with notes that we took while running through the steps above (as described in the ‘Tips’ earlier in this appendix).

***** My Annotations start with *****

***** Click on database Icon

(1-228 [1]) GET_SERVER_NAMES: (Connect to RED_MAILSERV/RED/M/Lotus: 6785 ms) (OPEN_SESSION: 111 ms)
134 ms. [42+16=58]

(2-238 [2]) OPEN_DB: (Connect to freja/Red5: 0 ms)
(Exch names: 0 ms) (Authenticate: 45 ms.)
(OPEN_SESSION: 9 ms)
14 ms. [126+290=416]

(3-238 [3]) GET_UNREAD_NOTE_TABLE: 8 ms. [290+44=334]

(4-238 [4]) OPEN_NOTE: 28 ms. [48+1408=1456]

(5-238 [5]) OPEN_COLLECTION: 12 ms. [32+652=684]

(6-238 [6]) READ_ENTRIES: 78 ms. [46+4064=40692]

(7-238 [7]) CLOSE_COLLECTION: 0 ms. [12+0=12]

(8-238 [8]) DB_MODIFIED_TIME: 95 ms. [14+44=58]

(9-238 [9]) GET_MULT_NOTE_INFO_BY_UNID: 13 ms. [360+694=1054]

(10-239 [10]) DB_REPLINFO_GET: 9 ms. [14+32=46]

(11-239 [11]) DB_REPLINFO_GET: 9 ms. [14+32=46]

(12-239 [12]) DB_INFO_GET: 8 ms. [14+140=154]

(13-239 [13]) OPEN_NOTE: 11 ms. [48+1324=1372]

(14-239 [14]) OPEN_COLLECTION: 20 ms. [120+8808=8928]

(15-240 [15]) READ_ENTRIES: 7 ms. [46+236=282]

***** End of open Database

***** Press Action Button to Create Document

(16-261 [16]) OPEN_NOTE: 84 ms. [48+55568=55616]

(17-263 [17]) DB_MODIFIED_TIME: 9 ms. [14+44=58]

(18-263 [18]) OPEN_COLLECTION: 7 ms. [32+34=66]

(19-263 [19]) FIND_BY_KEY: 8 ms. [540+26=566]
(Entry not found in index)

300 Performance Considerations for Domino Applications
***** Document open

***** Pressed Categories Keyword button
(20-297 [20]) DB_MODIFIED_TIME: 8 ms. [14+44=58]
(21-297 [21]) OPEN_COLLECTION: 20 ms. [32+34=66]
(22-298 [22]) READ_ENTRIES: 6 ms. [46+624=670]
***** Keyword dialog box displayed with no entries

***** Pressed Save & Close action button
(23-330 [23]) UPDATE_NOTE: 11 ms. [2016+100=2116]
(24-330 [24]) LOCATE_NOTE: 6 ms. [34+30=64]
(25-331 [25]) READ_ENTRIES: 7 ms. [46+398=444]
(26-331 [26]) DB_REPLINFO_GET: 8 ms. [14+32=46]
(27-331 [27]) GET_MODIFIED_NOTES: 6 ms. [26+42=68]
***** Returned to view

***** Opened document in Read Mode
(28-350 [28]) NIF_OPEN_NOTE: 85 ms. [42+55638=55680]
(29-351 [29]) OPEN_NOTE: 9 ms. [48+1668=1716]
(30-351 [30]) READ_ENTRIES: 8 ms. [46+624=670]
(31-351 [31]) UPDATE_COLLECTION: 17 ms. [24+12=36]
(32-351 [32]) READ_ENTRIES: 4 ms. [46+828=874]
(33-351 [33]) READ_ENTRIES: 5 ms. [46+828=874]
(34-351 [34]) FIND_BY_KEY: 9 ms. [540+26=566]
(Entry not found in index)
***** Document displayed

***** Selected attachment & launch
(35-377 [35]) GET_OBJECT_SIZE: 9 ms. [28+24=52]
(36-377 [36]) READ_OBJECT: 32 ms. [26+16424=16450]
(37-379 [37]) READ_OBJECT: 29 ms. [26+16424=16450]
(38-379 [38]) READ_OBJECT: 25 ms. [26+16424=16450]
***** Removed Repetative lines
(106-387 [106]) READ_OBJECT: 23 ms. [26+14184=14210]
***** Attachment Launched

Appendix B-8: Using NRPC Output
***** Pressed the New Response Button
(107-434 [107]) OPEN_NOTE: 36 ms. [48+18706=18754]
(108-435 [108]) OPEN_NOTE: 90 ms. [48+55568=55616]
(109-435 [109]) OPEN_NOTE: 25 ms. [48+9956=10004]
(110-435 [110]) OPEN_NOTE: 15 ms. [48+7310=7358]
(111-436 [111]) FIND_BY_KEY: 8 ms. [540+26=566] (Entry not found in index)
(112-436 [112]) FIND_BY_KEY: 10 ms. [540+26=566] (Entry not found in index)
***** Response document Displayed

***** Pressed Save & Close action button
(113-486 [113]) FIND_BY_KEY: 11 ms. [540+26=566] (Entry not found in index)
(114-487 [114]) GET_NOTE_INFO: 5 ms. [18+102=120]
(115-487 [115]) UPDATE_NOTE: 17 ms. [2366+100=2466]
(116-487 [116]) LOCATE_NOTE: 9 ms. [34+34=68]
***** Returned to Main Document

***** Closed Main document
(117-514 [117]) READ_ENTRIES: 8 ms. [46+560=606]
(118-515 [118]) DB_REPLINFO_GET: 5 ms. [14+32=46]
(119-515 [119]) GET_MODIFIED_NOTES: 5 ms. [26+42=68]
***** Returned to view

***** Closed database
(120-530 [120]) CLOSE_COLLECTION: 10 ms. [12+0=12]
(121-530 [121]) CLOSE_COLLECTION: 3 ms. [12+0=12]
(122-530 [122]) CLOSE_COLLECTION: 8 ms. [12+0=12]
(123-530 [123]) CLOSE_DB: 3 ms. [14+0=14]
***** Returned to desktop

The above is a demonstration of how you can use the RPC output to review what communication is happening between your client and the server at a given time.
Appendix B-9
Method to Populate a Database with Simulation Data

One of the problems we often encounter is predicting how a Domino database will perform with a large number of documents. This appendix describes a method to populate a database with a large number of documents. It was written by Boris Minnaert from IBM Netherlands.

Overview
Basically, the idea is to create a large number of documents with agents, with the help of the third-party tool TeamStudio. This way you should be able to get some idea about database size (before and after building the view indexes), view index sizes, time needed to replicate, time needed to open a view, time needed to open a document, time needed to save a document (times both measured when LAN connected and when dialing in with a 28K modem) and so forth.

Now, you may think that creating the code for the agent is a challenge. Trust me, it’s not. The real challenge lies in thinking of a way to generate a good set of values. You have to think of the typical data contained in a field, which in some cases can be hard to predict. You’ll probably have to modify the value-generating routine a few times, because you’ll probably notice side effects of the routine. Remember to consider the impact of your value-generating routine on performance inhibitors, like view categorization.

Assumptions
You’re nearly done with development of a large scale database.

Limitations
Any simulation is only as good as the subject knowledge you put into it.
This method is easiest when a large portion of the fields are numeric.
This simulation doesn’t include the effect of a large number of users, like multiple users accessing the same database at the same time, or the effect of unread marks.

Details
The goal is to create a large number of documents with an agent. In practice, most people don’t like creating such an agent, because it can involve a lot of tedious coding — especially when there are many fields.
Now here’s the trick: TeamStudio can help you.
With TeamStudio you run a design analysis of your database. This will include a comma-delimited listing per form of all the fields used. Check out the TeamStudio Analyzer or TeamStudio Designer Help if you don’t know how to do this. More information about these products is available on the Web at:

www.teamstudio.com

What follows is a list I got using this technique (yes, quite a number of fields). Note that in this particular case I was lucky, because most of my fields are supposed to contain numeric values.

SaveOptions, CMNotes_Pull_TMP, CMNotes_Pull_Flag, IW_Pull_TMP, IW_Pull_Flag, Delete_Flag, Form, Country_Code, Company_Code, CTS_Comment, HdrTitle, HdrSecClass, Contract_Number, Legal_Number, Contract_Desc, Customer_Number, Customer_Name, Local_Curr_Ind, ITD_Revenue, Revenue_Total, ITD_Invoiced, Curr_Mult_Factor, ITD_Price_Rev, ITD_Plan_Rev, ITD_Actuals_Rev, YTD_Pricing_Rev_1, YTD_Actuals_Rev_1, ITD_Plan_P_Rev, ITD_Price_Var_Rev, ITD_Plan_Var_Rev, ITD_Price_Cost, ITD_Plan_Cost, ITD_Actuals_Cost, ITD_Pricing_Cost, ITD_PlanP_Cost, YTD_Pricing_Var_Cost, YTD_PlanP_Var_Cost, YTD_Pricing_GP, YTD_Plan_GP, YTD_Actuals_GP, YTD_Pricing_GPP, YTD_Plan_P_GPP, File_Complete, Status_Exposure, Contract_Sign_Date, Contract_Sign_Date_Alt, Start_Date, Start_Date_Alt, Contract_Create_Dt, Contract_Create_Dt_Alt, End_Date, End_Date_Alt, Work_Number,
Project_Desc, CFTS_LastLoad_TMP, FCS_Data, PACT_LastLoad_TMP, CMR_LastLoad_TMP, Claim_Data, Ledger_Data

Just use a word processor and a spreadsheet to transform this in only a couple of minutes to code like the code below. I’m not including all the lines, but you’ll get the idea, and I’m sure you’ll appreciate the time saving factor. Make sure to take care of special items like Form and SaveOptions, and flags. You might want to use the DelayUpdates property of the NotesDatabase class to improve the performance of this agent.

DB.DelayUpdates = True
For i = 1 To 10000
Set Doc = New NotesDocument(DB)
Doc.Form = "MyForm"
Doc.ITD_Revenue = 10000 + Round(i/10,0)
Doc.Revenue_Total = 10000 + Round(i/10,0)
Doc.ITD_Invoiced = 10000 + Round(i/10,0)
Doc.Curr_Mult_Factor = 10000 + Round(i/10,0)
Doc.ITD_Price_Rev = 10000 + Round(i/10,0)
Doc.ITD_Plan_Rev = 10000 + Round(i/10,0)
Doc.ITD_Actuals_Rev = 10000 + Round(i/10,0)
REM ...etc...
Doc.Claim_Data = 10000 + Round(i/10,0)
Doc.Ledger_Data = 10000 + Round(i/10,0)
Call Doc.save(True, True)
If Fraction(i/25) = 0 Then
Print "Contract " & i
End If
Next i

The next step is to try to simulate the effect of users updating the documents. In my case I used a simple sub called MyReplaceValue(Doc as NotesDocument, ItemName as String) to add a random number to the numeric values and then used the word processor and spreadsheet again to create code like:

Randomize
Do While Not Doc Is Nothing
Call MyReplaceValue(Doc, "ITD_RevAD_IL")

Appendix B-9: Method to Populate a Database with Simulation Data  305
Call MyReplaceValue(Doc, “ITD_RevAD_SL”)
Call MyReplaceValue(Doc, “ITD_RevAD_OEMHW”)
Call MyReplaceValue(Doc, “ITD_RevAD_OEMSW”)
Call MyReplaceValue(Doc, “ITD_RevAD_TL”)
Call MyReplaceValue(Doc, “ITD_CostAD_IL”)
REM ...etc...
Call Doc.save(True, True)
If Fraction(i/25) = 0 Then
    Print “Updated “ & i
End If
Loop
Sub MyReplaceValue(Doc As NotesDocument, ItemName As String)
    On Error Goto ErrorLabel
    Dim Item As NotesItem
    Dim ItemValue As Integer

    Set Item = Doc.GetFirstItem(ItemName)
    If Not Item Is Nothing Then
        ItemValue = Item.Values(0)
        ItemValue = ItemValue + Cint(10 * Rnd)
        Call Doc.ReplaceItemValue(ItemName, ItemValue)
    End If
    Exit Sub

ErrorLabel:
    Print “Error; resuming”
    Exit Sub
End Sub

Important  You will probably need a couple of runs of the agent with different user IDs to accumulate enough data in the $UpdatedBy and $Revisions items..

306  Performance Considerations for Domino Applications
Appendix C-1
Limits for Application Development Features

This list of limits in Domino R5.0 and R4.6 has been compiled from Notes Help and the Lotus Knowledge Base.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Limit in Domino R5.0</th>
<th>Limit in Domino R4.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum size of database</td>
<td>Up to 64GB (32GB for OS/2) or defined by OS file size limit</td>
<td>Up to 4GB or defined by OS file size limit</td>
</tr>
<tr>
<td>Maximum size of view</td>
<td>128GB or database size limit</td>
<td>512MB or database size limit</td>
</tr>
<tr>
<td>Maximum number of view in a database</td>
<td>No limit; however, as the number of views increases, the length of time to display other views also increases</td>
<td>same</td>
</tr>
<tr>
<td>Maximum number of columns in a view</td>
<td>289 ten-character columns; dependent upon number of characters per column</td>
<td>same</td>
</tr>
<tr>
<td>Maximum number of fields in a database</td>
<td>~ 3000 (limited to ~ 64K total length for all field names). You can enable the database property “Allow more fields in database” to get up to 64K uniquely-named fields in the database, but certain features that use the list of all field names will not work properly such as full-text querying, the Designer’s list of field names, and agents that modify by form.</td>
<td>~ 3000 (limited to ~ 64K total length for all field names).</td>
</tr>
<tr>
<td>Maximum number of forms in a database</td>
<td>Limited only by database size</td>
<td>same</td>
</tr>
<tr>
<td>Maximum size of a text field</td>
<td>15K</td>
<td>same</td>
</tr>
<tr>
<td>Maximum size of a rich-text field</td>
<td>Limited only by available disk space up to 1GB</td>
<td>same</td>
</tr>
</tbody>
</table>

*continued*
<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit in Domino R5.0</td>
</tr>
<tr>
<td>Maximum size of a single paragraph in a rich-text field</td>
</tr>
<tr>
<td>Maximum number of entries in an ACL</td>
</tr>
<tr>
<td>Maximum number of Roles in an ACL</td>
</tr>
</tbody>
</table>
| Limits for titles and names | Database title: 96 bytes  
Field name: 32 bytes  
View name: 64 bytes  
Form name: 64 bytes  
Agent name: 127 bytes including all cascading names. Each name cannot exceed 64 bytes | Database title: 96 bytes  
Field name: 32 bytes  
View name: 64 bytes  
Form name: 32 bytes  
Agent name: 127 bytes including all cascading names. Each name cannot exceed 64 bytes |
| Maximum number of columns in one table | 64 | same |
| Maximum number of rows in one table | 255 | 255 |
| Maximum number of cascading views in a database | 200 | 200 |
| Maximum number of outline entries | ~21,000 entries | Not applicable |
| Maximum number of outlines in a database | There is no limit | Not applicable |
| Maximum number of framesets in a database | There is no limit | Not applicable |
| Maximum length of search query string | In Notes: no limit  
From a Web browser: 254 bytes | |
| Maximum length of environment variable in NOTES.INI | 62 characters (You can add longer names, but this can cause applications to fail) | same |
| Maximum length of Key string parameter for GetDocumentByKey and GetAllDocumentsByKey of the NotesView class | 32,750 characters | 200 characters |

308 Performance Considerations for Domino Applications
Appendix C-2
View Indexing Comparisons

In this appendix we analyze a number of different view characteristics and the effects they have on index size and view update speed.

The views tested and the data used is only one of many possible combinations. The results cannot always be applied to every application nor every situation.

View Characteristics

There are many criteria we can use to decide which kinds of views to test. In this analysis we investigate the major view types and most common column properties and, although the study does not cover every situation, the majority are included. Conclusions, therefore, are not hard and fast rules, but can be thought of as good generalizations.

Most of the performance literature available, including this redbook, mentions a few common things about maximizing view performance. Those comments led to testing the following view characteristics:

- Number of columns
  Compare views that differ only by the number of columns being displayed.

- Categorization
  Compare effects of categorization versus not categorizing.
  Compare categorization using separate columns for categories versus using a single column that contains the combined categories (by using the backslash (\) between categories in the column formula).

- Sorting
  Compare sorting a column vs. not sorting it.
  Compare dynamic sorting of columns.

- Column calculations
  Compare using form fields to store calculations versus doing the calculation in the column formula.
  Compare how the size of data displayed in a column affects things.
- Number of fields in a document
  Compare effects of adding fields to the documents but not changing the view in any way.

Test Data

Having decided what view characteristics we are analyzing, we now need some test data. This is the hard part. How do you get data that simulates a real application?

We could get data from a real application, but then we are simulating a single variation from an infinite number of possible application data sets. Also, real data may be sensitive. Using real data may require the test database to have a set of matching forms for the data, and so on.

What we chose to do was to generate our own data. There are several reasons for this choice:

- It’s relatively easy (we don’t have to find an application).
- As we change database size, the distribution of the data can be controlled.
- We don’t have to worry about “real” data (from a security standpoint).
- We can tailor the data attributes to suit the tests we want to do.

We need a relatively complex agent to generate data which has the characteristics we want. We don’t just want all random data. We need some random data but we also need data that is chosen from a limited set (so we can categorize). We want some numbers, dates, text and rich text.

Test Cases

We built a database that has not only the view test cases of this appendix, but many of the other test cases used to validate performance information in this redbook. The database is available on the redbook Web site.

We found that when testing with database sizes of 500 or 1000 documents, the performance numbers are not very meaningful. Domino does a lot of internal partitioning of the sections inside the NSF file. The sizes of these partitions are such that when the number of documents are small, doubling things sometimes has no noticeable effect on space or time. You need to have a reasonable volume of documents for the numbers to have some validity.
We ended up with four database sizes:

- 2000 document
- 4000 documents
- 8000 documents
- 16000 documents

We tested on both Notes Release 4.5 and 5.0.

When doing the “number of fields” test, we just duplicated all the fields in the documents (except for system fields and rich text).

We developed a few agents to help generate data. The document creation agent used the following basic logic:

- The form was selected from five document types (A, B, C, D and E). The selection was random (using the Rnd() function). When a type D document was being created, we also sometimes added one or more response documents (type F).

- Three string fields were created and used for categorization. Each of these fields was populated from a fixed set of data. Field one contained “Country” information, field two was “Sport” and field three, “Year.” The data for each document was created by randomly selecting from a list of ten countries, eight sports and four years.

- 18 numeric fields were created. These were populated by random numbers. The six “Count” fields were integers in the range of 10 to 20. The six “Totals” were integers in the 0 to 10,000 range. The six “Average” fields were the average of the total divided by the count (thus the “Average” fields were fractional numbers).

- Two date fields were included. These were set to random dates in the range of today plus or minus 30 days.

- 20 text fields were created. These were set to various text of random lengths. The lengths ranged from 0 to 166 characters.

- Two rich-text fields were created. These were populated from a sample document that had three pre-created rich-text fields. The first was typical word processing text: paragraphs, bullets, fonts and color. The second was a table (4 rows and 4 columns). The third was a button with some @functions behind it. Two of these rich-text fields were randomly selected and copied from the sample when creating documents.

The rich-text fields had no embedded objects (no attachments or active objects).
The above was the maximum configuration for a single document. Many documents included only a portion of this data, but always from the same set:

- Form types A and B contained all the fields.
- Form type C contained half of the fields.
- Form type D was like C but contained no rich text. We added response documents (type F) to some of these (see type F).
- Form type E contained only three numeric fields and three text fields.
- Form type F contained one text field and one rich-text field. Type F documents were always responses to either a type D or to another type F. We created a random number of type Fs (0 to 3) for every type D document. When this number was not zero, half the time we would create an extra response layer (another type F) between the parent and the responses being created.

The creation agent generated 2000 main documents plus response documents. To get the other database sizes, the same data was duplicated (via another agent).

When adding fields to the documents, all existing fields were duplicated (except system fields, like the Form field, and rich text fields).

---

**View Testing**

A number of views were set up to analyze the test cases. In general, the first view in each series is a “base” view from which we can compare the effects of the test.

The actual tests were run as follows:

1. A new copy of the database was created on the server.
2. An agent was run to create 2000 documents. In fact, since some of the 2000 documents had responses, 2714 “notes” were created in the database.
3. Another agent was run to create the calculated column fields using the same formula that the “column calculation” views use.
4. All view indexes were built (Ctrl-Shift F9).
5. The “Show Database <db name> V” command was issued on the server remote console to get the view index sizes.
6. An agent was run to double the documents to 4000 (which meant the database had 5428 “notes”).
7. Steps 4 and 5 were run again.
8. The agent was run again to double the documents to 8000 (which meant the database had 10856 “notes”).

9. Steps 4 and 5 were run again.

10. The agent was run one last time to double the documents to 16000 (which meant the database had 21712 “notes”).

11. Steps 4 and 5 were run again.

12. An agent was run to double all the fields in each “note” in the database.

13. Steps 4 and 5 were run again.

14. The results were entered into a Lotus 1-2-3 spreadsheet and the graphs were created.

During the course of testing, there were several times when we realized that some “things” were missing. For instance, we needed an extra view type, or a different data type, or another agent to do things. So, although the above steps were the final result, much testing was done in different sequences. Some ad hoc results listed at the end of this appendix describe the findings that we discovered during the process for which individual test cases were not created.

**Number of Columns**

There were four views in this test.

**NC: 7 Columns**

This was the base view. It had seven columns of various types and had no sorting nor categorization.

**NC: 14 Columns**

Similar to the base view except that every column was duplicated. The column formulas in the duplicate columns were identical to the original.

**NC: 14 Different columns**

Similar to the base view except every column was duplicated but the column formula was then altered so that the duplicate column resulted in different data from the original column.

**NC: 28 Columns**

Similar to the base view except that every column was quadrupled. The column formulas in the duplicate columns were identical to the original.
Test Results
The following graph depicts view index size versus number of columns:

![View Index vs. Number of Columns](image)

Note The middle line on the graph is actually two lines on top of one another: NC: 14 Columns and NC: 14 Different cols.

As was expected, the number of columns directly relates to the space requirements. However, in the NC: 14 Columns view there were seven duplicate columns (they contained the same formula and properties as seven other columns). In the NC: 14 Different cols view, each of the 14 columns had a slightly different formula. As can be seen by their identical space requirements, view indexing is not smart enough to optimize when two (or more) columns are the same, but that is unlikely to happen in the first place.

Categorization
There were four views in this test.

CAT: No Sorting
This was the base view for categorization. The view had six columns. The first three columns were text data (which will be categorized in other views in this section). The remaining three columns contained unsorted text and numeric data. In this base view, no column was sorted or categorized.

CAT: No Categories
Similar to the base view except that the first three columns were flagged as sorted, ascending. In this view we still have no categories.
**CAT: Separate Columns**
Similar to the base view except that the three first columns were flagged as categorized, ascending. The last three columns were left unsorted.

**CAT: Single Column**
Similar to the CAT: Separate Columns view except that the three category columns were combined into one single category column using “\” to concatenate the three fields together. This view, therefore, had only four columns, not six, and the first one only was categorized, ascending.

**Test Results**
The following graph depicts view index size versus categorization:

![View Index vs. Categorization Graph]

This one surprised us. It seems that the sorting required for categorization takes up some space, but the categorization itself takes up almost no space at all. Likewise, if you examine the timing section later in this appendix, you see that categorization does not consume huge amounts of time either.

Of all the things you can do with a view that hinders performance, categorization seems to have the least effect.

**Sorting**
These were seven views in this test.

**SRT: No Sorting**
This was the base view. It had seven text and number columns, none of them sorted.

**SRT: 1 Column**
Similar to the base view except that the first column was sorted.
SRT: All Columns
Similar to the base view except that all seven columns were sorted.

SRT: All, 1 Dynamic
Similar to the base view except that all columns were sorted and the second column had a dynamic resort flagged (descending).

SRT: All, 3 Dynamic
Similar to the base view except that all columns were sorted and the second, third and fourth columns had dynamic resorts flagged (descending).

SRT: None, 1 Dynamic
Similar to the base view except that the second column had a dynamic resort flagged (descending).

SRT: None, 3 Dynamic
Similar to the base view except that the second, third and fourth columns had a dynamic resort flagged (descending).

Others
During the course of testing, there were also some sorting views created with five dynamically sorted columns and others where the dynamic sorting was done ascending vs. descending. A final set was also create where the dynamic sorting was set to “Both” ascending a descending. Test results from these views are included elsewhere in this appendix.

Test Results
The following graph depicts view index size versus sorting:

---

316 Performance Considerations for Domino Applications
**Note**  The bottom line on the graph is actually two lines on top of one another: SRT: No sorting and SRT: 1 Column.

As expected, the more we sort, the more space it takes. Some points to note, however:

- Sorting a single column takes the same amount of space as not sorting at all. In fact, the time to rebuild the index, as we see later in this appendix, is also the same for one sorted column versus no sorting.

- Dynamic column sorting takes up much more space than “normal” sorting. And as we discuss in the timing section later in this appendix, the time to rebuild the index also takes much longer when dynamic column sorting is involved.

**Column Calculations**

There were five views in this test.

**CC: Column Calcs**
This was the base view. All three columns had simple calculations coded in them.

**CC: Complex Calcs 1**
Similar to the base view except that the calculations for columns two and three are much more complex (a lot more Formula language involved). The resulting data displayed in the columns, however, was the same simple calculation used in the base view. In other words, we are doing a whole lot more calculating but ending up with the same result.

**CC: Complex Calcs 2**
Similar to the CC: Complex Calcs 1 view except that, instead of displaying simple results, the complex results of the calculations were displayed in the columns (the size of the data displayed was much larger).

**CC: No Calcs**
Similar to the base view except that the view used three pre-calculated fields that existed in the documents. The formulas for these pre-calculated fields were identical to the formulas used in the base view. These pre-calculated fields were displayed in the view instead of doing the calculations.
**CC: No Calcs 2**
Similar to the CC: Complex Calcs 2 view except that the view used three pre-calculated fields that existed in the documents. The formulas for these pre-calculated fields were identical to the formulas used in the CC: Complex Calcs 2 view. These pre-calculated fields were displayed in the view instead of doing the calculations.

**Test Results**
The following graph depicts view index size versus column calculations:

![View Index vs. Column Calculations](image)

**Note**  The bottom line on the graph is actually three lines on top of one another: CC: Column calcs, CC: Complex calcs 1 and CC: No calcs 1. As we can see, space requirements have almost no relation to the complexity of the calculations happening in the view columns. The major factor affecting view index size seems to be the size of the data displayed in the column.

Don’t be fooled, however. Take a look at the timing section later in this appendix for some other interesting information about column calculations.

**Number of Fields**
There were no special views in this test. The results of prior tests were compared against results after the number of fields in the database was increased.

An agent was run on the 16,000 document database to duplicate the document fields. System fields (like the Form field) and rich-text fields were not duplicated.
Test Results
The following graph depicts view index size versus number of fields:

![Graph](image.png)

After the agent was run on the 16,000 document database, the size of the database increased from 192MB to 219 MB, but the size of the indexes remained constant. The increase in database size was due to each document requiring more storage to hold the larger document. The views did not require extra storage since the new fields were not part of any view.

In other words, view index size is related to the information displayed in the columns, and not related to the number of fields in the documents.

View Index Rebuild Times
In all the prior examples, we have been examining view index size. View index size represents a large percentage of the database size in most cases, and as we have mentioned before, the larger the database, the slower things run.
The following graph is a measure of the time to rebuild the indexes for the test views. The times were measured with a stopwatch on the 4000 document database. Each view was rebuilt 3 times and the average was used. When a rebuild took a longer time than normal (noticeably larger than the other two values), that value was thrown out and the index was rebuilt again. We did this because the server was obviously busy doing other tasks at the time (we could not stop others from using the server). This happened very infrequently during the tests.

![View Rebuild Times](image)

As we can see, doing many column calculations takes a lot of time. Likewise, dynamic column sorting takes time.

One other point to note: CC: No calcs 2 takes twice as long as CC: No calcs 1 simply because there is more information that has to be stored in the index. There are no calculations being done in either of these two views (the calculations are done on the form and stored in fields on the form), but because the second one displays a large amount of data, its view index is much larger and thus, rebuilding takes much longer.

**Note**  The time to *display* views with complex calculations or large data did not appear to be different from displaying simple views with small data. Likewise, we did not notice any difference in the time for a view refresh for these types of views. The calculation time is apparent only when the entire index is rebuilt. Of course, if your data changes a lot, then a view refresh might affect many documents and thus slow down the refresh.
Summary

View testing is an interesting area and was fun to do. We found that the results were surprising in several ways:

- Categorization is not too bad from a space or time perspective.
- Dynamic column sorting is a killer when it comes to both space usage and rebuild time.
- The amount of information displayed in each column can really cause your space requirements to shoot up.
- Complex calculations severely affect the view index rebuild time.

Ad Hoc Results

A number of ad hoc tests were done during the other tests. Some points to note:

- Sorting ascending versus sorting descending had no appreciable difference in either time taken or view index size.
- Defining a column with dynamic sorting in both directions (ascending and descending) doubles the time and space required for that dynamic sort. In other words, if sorting the column one way adds 1MB of space to the index, sorting both ways adds 2MB. Likewise, if sorting one way adds 10 seconds to the rebuild time, sorting both ways adds 20 seconds.
- Results using Domino R5.0 seemed similar to Domino R4.x, but exhaustive tests were not done.

Conclusions

View indexing is a complex subject and your application will have special needs, and therefore, different results. But we think, given the above test results, and other prior experience, it is safe to say the column sorting and size of data displayed in a column are the two most significant factors in determining view index size and rebuild time.
Performance Considerations for Domino Applications
Appendix C-3
Fast Access to Reader Names Protected Documents

This appendix describes why reader names can affect view performance and then goes on to describe a technique of gaining access to documents with reader names fields without getting hit with bad-performing views.

Much of the information is an extract from the Lessons Learned section on the IBM Notes/Web Application Center of Competence Web site — an IBM intranet Web site.

Reader Names and View Performance

A Possible Scenario

The following is something that might be overheard at the coffee machine in a Domino development shop:

Developer #1: “I’ve got a problem. My database has only 10,000 documents in it but my views take over 10 seconds to open. The documents all have reader names fields and maybe that’s the problem.”

Developer #2: “Well, my database has over 100,000 documents in it with reader names fields and my views come up consistently in one second. I doubt your problem has to do with reader names fields.”

Well, developer #2 is wrong, the problem developer #1 is having is because of the reader names fields. But why, then, does developer #2 not have a performance problem when that database is 10 times the size?

Reader Names

Reader names fields and their effect on performance have been discussed in many places, but we had not found “the answer” until we started an in-depth study into this feature of Domino.

First, let us say that the reader names feature of Notes is extremely useful. It basically provides the only way to guarantee that a document can only be seen by persons referred to in the reader names field (either directly listed, part of a listed group or who have a role which is listed in the reader names field).
Also, the actual access to a document with a reader names field is very fast. The extra processing required to check if the user is referred to in the reader names field is minimal. Why, then, do some applications that use reader names fields suffer from slow response time?

The answer lies in the fact that the vast majority of Domino applications access documents via a view. And the only area that causes noticeable degradation of performance when using reader names fields is the area of view index preparation for a user.

Fine, but that still doesn’t help us understand the problem discussed by the two developers in the scenario above! To explain it, we have to go into a bit more detail about how Domino handles views.

**Domino and Views**

As your know, a view is a way of presenting a subset of all the documents in the database to the user. The “SELECT” formula for the view determines how many documents are in this subset. Some presentation aspects of a view (the color of the heading, the fonts, and so on) are stored as part of the view design and are independent of the volume of documents in the view. Other aspects of the presentation (sorting and categorization) are affected by view document volume. This is because things like sorting affect the index of the view. Of course we all realize that each view has an index and many views have several alternate versions of the index. The “main” index is a collection of summary information for every document that satisfies the selection criteria. An additional version of the index is maintained for each column that has dynamic ascending or descending sorting. If a column has both types of sorting, then two additional versions are maintained for that column, not one.

Now, let’s see where the reader names enter into it. A view index is stored on the server and contains entries for all the documents that satisfy the selection criteria for that view. The index is built (and maintained) without reference to reader names, since the same index is used by all users of the database. When the view is presented to a user, however, any reader names field must be taken into consideration by the server before the view is passed on to the user. This is where we can run into performance issues.

A few other things to consider about views:

- Each document in a view might be of a different type, and even if they are of the same type, one document may have a reader names field and another may not.
• Each entry in a view index not only contains a reference to the actual document, but it also includes summary information for each column in the view. The view is displayed without actual reference to any of the documents because the summary data is used.

• The client is the actual driver for displaying views. The user does some action and as a result, the client requests view information from the server.

• The client is smart enough to request only enough data from the server to fill the window. On the initial display of a view, the client might request 30 or 40 records from the server. The client displays the first window for the user. Depending on window size and font versus hardware considerations, the user may actually only see 20 or so records. As the user scrolls, the client presents more records that it has cached. If there are not enough records in the client’s cache to fulfill the user’s request (he may be scrolling a page or just a record), the client will ask the server for another batch of records.

An Anatomy of a Client View Request

Now we are in a position to describe the events that lead to the two developer’s comments. But first, there are two more things about the developer’s applications that relate to the situation.

1. Developer #1’s application deals with employee/manager information. Managers can generally see things about their employees but employees can only see things about themselves. Thus the average user of the application can see only a handful of documents. An employee can see only five, ten or maybe only a single record; managers can see more records. perhaps 50 to 100.

2. Developer #2’s application is like a sensitive pricing database. Users see information on a need-to-know basis depending on what area they work in. The average user can see about 10 percent of the records in the database.

Now that we know a bit more, let’s see what happens when users use developer #1’s application:

• The user requests to open a view.
• The client asks the server for 30 records from the view, starting at record #1.
• The server locates the index of the view.
• The server processes the index:
  1. Set index position to the requested record number.
  2. Get the next record.
3. Does this entry have reader names? (use summary data to determine this)
   - No — feed the record to the client buffer.
   - Yes — check the user ID to see if he/she is in the reader names list
     (this may involve going to the Domino directory to resolve groups
     or to the database ACL to resolve roles). If the reader is allowed to
     access the record, feed it to the client buffer. If not, skip this record.

4. Have I got enough records in the buffer yet?
   - No — go back to 2.
   - Yes — continue on to 5.

5. Send the data buffer back to the client:
   - The client receives the returned data and puts it into some internal
     cache.
   - The client displays records from the cache to fill the window.

When the user scrolls, the client might be able to handle it from the cache
(scroll a few records) or might have to request more data from the server
(steps as above, but with an updated starting point).

So, why does developer #1 have a response time issue? It’s because the client
asked for 30 records and the user is only allowed to see ten records in the
database. In attempting to fill the buffer with 30 records for the client, the
server must scan all 10,000 records in the index before it can determine that
there are no more records for the user. The server ends the search and
returns the ten records in the buffer to the client. However, scanning an
index of 10,000 records where reader names are involved will take 10
seconds (and that’s optimistic). The user is not at all happy with the response
(especially if a few hops in a wide area network are involved as well).

Then how about developer #2? Well, re-examine the steps taken by the
server to satisfy the client request. In developer #2’s case, the user can see
10% of the records. That means (on average) 1 in 10 records that are
processed by the server pass the test and are sent to the client buffer. So after
only 300 records or so, the buffer is full (containing 30 records) and is sent
back to the client.

Processing 300 records, even with reader names fields, is quite fast. As the
user scrolls, the client gets more data from the server, and again only
300-odd records need to be processed to satisfy the client. The user of this
application is happy with the response times (unless the wide area network
really gets in the way).
Some Interesting Observations

We had an insight to the above situation while examining an application with reader names that was having problems. Before writing this article, we verified our many assumptions by building two databases as above, one with 10,000 records and another with 100,000 records. The designs of both the databases were in fact the same. The only difference was that in the 10,000 document database, only a single record had our test user’s ID in the reader names field, while for the 100,000 document database, 10 percent of the records had the test user ID in the reader names field. During our various investigations, a number of other interesting facts about reader names came to light, as described below.

Sequence May Affect Performance

If the view is sorted, then the user may have access to the data in “chunks.” In the above example, our 10 percent of the records were randomly spaced throughout the database. The server never needed to process more than about 300 records between user “response” times. However, if the 10 percent were all near the end of the view, then the server would need to scan a great deal more data before finding 30 records to satisfy the client. Further requests by the client would be fast, but the initial one would be slow. So, depending on the location of the user’s data in the view, response times might not be consistent.

Categorization Can Help

If you categorize the view and set it to be “collapsed on open,” then access to the view is very fast for the initial entry regardless of how many records you are allowed to see. Since only categories are displayed when collapsed, the data is found quickly and passed to the user (there are no reader names fields on categories). To open a category, however, may be slow — it depends on how many records are in the category. The same reader names principle applies — the server needs to scan all records in a category to determine either that the buffer is full or the category has ended.

Note  This categorization technique can be an effective way to improve response times. You in effect break up the single view into many smaller sections. The server only processes one section at a time. One drawback, of course, is that the user might see many empty categories (unless you are using Domino R5.0). With Domino R5.0 you can specify that empty categories are to be hidden. We do not know if scanning by many empty categories will affect performance.
Empty Views Are More Costly
If the user has no records in the view, the view takes three times as long to open! We think this is a Notes “limitation.” From network traffic data (using IPTRACE), it can be seen that when a user has one or more entries in the view, the client makes a single request to the server for the view information. But if there are no entries in the view for the user, the client makes three requests to the server for the same view information. Our guess is that there is internal processing that accesses some internal cache three times. Under normal processing, the first access fills the cache (from the server) and the other two just use the data in the cache (no server access). But if the cache is not filled on the first try, then each subsequent access to the cache again tries to fill it (by accessing the server).

Index Size Is Not Affected
Reader names fields don’t affect the size of the index any more than the same field without the READERS flag. Index sizes are important factors to consider, not so much when dealing with reader names fields, but in determining database size. An unindexed database can easily become five times bigger after indexing — depending on view design.

The size of an index is determined by six things:

- The number of documents that satisfy the SELECT criteria
- The number of columns in the view
- The number of fields in the documents
- The number of columns that are categorized
- The number of columns that are sorted
- The number of dynamic sorts there are on the columns

The size of the index does not appear to affect response times if other things are constant. If we have a view with 1,000 documents where we can see 10 percent versus a view of 100,000 documents where we can see 10 percent, the response times are similar.

Other Things to Watch For
Other activities in your application may inadvertently cause an access to the view which may cause a long delay:

Example 1: Dynamic Sorting
If you have a dynamic sort on a column, pressing on that column heading for the sorted view is in fact re-accessing the view through another version of the index. If it took 5 seconds to open the view initially, it will take 5 seconds again to display the alternate sort version (and another 5 seconds to go back to the original view). Be careful: if you have used a categorized view to make response times okay, any dynamic sorting might blow you away.
The sorted view uses a linear version of the index (no categories on dynamic sorts), thus the full index may again need to be scanned for a result (not just one category).

**Example 2: Saving a Document**
When you save a document that you opened from a view with reader names, the server may automatically refresh the index, causing a delay equivalent to opening the view in the first place. If the script in any part of the form ever does a ViewRefresh you get the delay as well. There may be other situations that cause the server to refresh the index and these will cause delays. Setting the view attribute Refresh index to “Auto, at most every ‘x’ hours” does not seem to make a difference in this situation.

**Summary**
If you use reader names fields in your documents and have a database with many documents (as a ballpark figure, this number is around 2,000) and the number of documents “visible” to the user is 5 percent or less, then special attention will need to be given to the design of views of those documents. The designer needs to know a lot more about the proposed data in the application to be able to judge if the response times for views will be a problem. If it looks like there might be an issue, various techniques can be employed to speed up access to views containing reader names documents.

The following techniques can be used to reduce the delays in views that deal with reader names fields:

1. Use categorization to segment the volume of documents in the view so each category has a small enough number to make response times reasonable. Remember — you must flag the view as “collapsed when opened” in this case.
2. Spread your documents across multiple databases. (This sounds easy but it has more pitfalls than you might imagine.)
3. Access your documents without going through a view. This technique is further explained later in this appendix.
4. If the only purpose of a view is to navigate to a particular document, and if the application is one where the user needs to only update this single document during this application session, then you might consider closing the view behind the scenes at the same time that the document is presented to the user. In this way, any saves done by the user will not incur the view refresh delay. Also, when the user exits the document he’s already one step closer to exiting the application.

These techniques are not the only ones that might solve a reader names problem. Among the other possible techniques are using “personal on first use” views and folders. These methods were not tested.

---

**Getting Fast Access to Reader Names Documents**

The use of this technique is applicable to databases where the user can see only a few percent of the documents. As outlined earlier in this appendix, if the number of documents viewable by the user is relatively high (more than around 8 percent), then the effect of reader names fields on performance is diminished and normal Domino coding practices can be used. It’s when the number of documents accessible to the user is less than 5 percent and the database size is several thousand documents or more that the following techniques are most fruitful.

**Getting at the Documents**

So, how do we get at the documents if we have no view selection to find them? Well, we cheated a bit when we implied that a view is not involved. In fact, we do use a view to access the documents, but the view is of documents that do not have reader names fields. This is the key: if a view contains only documents that do not have reader names fields, then regardless of the number of documents in the database, access to the view is very fast. When a view index is up to date, access to the view, even with GetDocumentByKey, is very quick and from a user perspective, is not related to the number of documents in the view. The answer, then, is to maintain a parallel set of skeleton documents which are used to access the real documents. The skeleton documents have no reader names fields and views of these are fast to access.
The technique is as follows:

1. Create an additional form in your database design. This form should have the bare minimum of fields from the “real” document — just enough to satisfy selection criteria for the original views. The form also needs one more field: the universal ID of the related document.

2. Modify the QuerySave event in the “real” document to detect the first save of the document (document.IsNewDoc property). On this first save only, create a skeleton document as a partner to the real document being saved. This skeleton document contains the universal ID of the real document and is used to access the real document.

3. Change all navigation to the real document to access it via the skeleton document. Once the skeleton is found, bring up the real document without using a view of that document (for example, using the workspace.EditDocument method).

4. Trap the “paste” event to ensure pasting documents is either not allowed or that it maintains any skeleton document.

5. Trap the “delete document” event to ensure deletion is either not allowed or that related skeleton documents are deleted too.

Although we have removed the long delays of accessing views with reader names fields, we have also introduced a number of negatives with this solution:

- There is extra code to write and maintain. Although it’s relatively straightforward code, it’s still code.
- The integrity of the links must be periodically checked via an agent (even though in theory this can’t go wrong).
- The extra documents take up space in the database, as does any view of those documents. In actual fact, the views with reader names might no longer be required (unless an administrator needs one). The removal of these large views can equal the addition of both the documents and the skeleton document views. We ran a test where we removed two reader names views, added the complete set of skeleton documents and two skeleton document views. The resulting database was less than 5 percent larger that the original.
• But, worst of all, we lose the nice UI to access multiple documents with reader names (the view). This last aspect is the single one that may deter potential users of the technique. All other things can be hidden from the user (they do not need to know the internals) but this last one affects what the user sees. It affects usability and aesthetics. A nice UI must be built to let users get at their set of documents and manipulate them. This new UI will use form design rather than view design. Although using a form to display a set of documents is quite easy, techniques of opening a document from a form are very limited compared to double-clicking a line in a view.

However, in a database of 10,000 documents, the reader names views takes over 10 seconds to open while the alternate technique opens the document in under a second (neglecting the network). When the number of documents gets up to 100,000, the time to open the view is over 2 minutes while the alternate remains at under a second.
Appendix C-4
LotusScript versus Formulas

This appendix is an extract of Lotus Technical Paper 142346 which does some comparisons between LotusScript and the Formula language. The information presented here is the result of tests to process data in a database using LotusScript compared to using @formulas.

Performance Testing

LotusScript and the Formula language have different strengths. This is great for the Notes community, because it means that the breadth of tasks that can be performed has been increased since the release of Notes R4.0. There are, however, some overlapping areas, and it is in these areas that we focused our testing. The question we try to answer is, “If I can easily use LotusScript or the Formula language, which should I use?”

Background

We considered the different cases where one might use LotusScript or formulas.

Updating Fields Within a Single Document
When processing a single document, the following operations were considered:

- Setting document default values
- Performing document field validations
- Looking up data from other Notes databases

We found that the difference in speed is so slight as to not be meaningful (fractions of a second) with the medium-sized documents and forms we tested. With regard to ease of development, LotusScript provides one centralized place in the form to put your code for validating all the fields, which is helpful. Formulas, on the other hand, tend to be considerably shorter and faster to code if there aren’t very many fields to validate. As a general rule of thumb, formulas are a good choice when you are trying to perform a task that formulas provide, as they are close to the core of Notes and have no additional overhead associated with them. Our testing with medium-sized documents did not show meaningful speed differences, however. One LotusScript consideration is that the back-end extended class syntax is slightly faster than front-end methods.
Working With Many Documents (Agents)
Many times we need to process lots of documents at a time. Agents are often used for this. The following types of updates were examined:

- Updating field values in all documents in the database
- Updating field values in some of the documents in the database
- Looking up data from other Notes databases and updating documents based on this data

Though the differences for one lookup might be small, repeating the process many times amounts to significant time differences. We focused our testing in this area.

Points To Consider
Some points to consider when looking at our results:

- Is there a substantial difference in the time to complete a task, or is the difference negligible?
- How easy is it to code this task in LotusScript versus the Formula language?

Test Methodology
This section details the hardware, software, and methodology used to perform the tests described in this appendix.

Equipment
The following equipment was used:

- P75 IBM ThinkPad (755CX) with 24MB of RAM and approximately 100MB of available disk space.
- Windows 95, with never more than Lotus Notes and the Explorer loaded into memory.
- Notes R4.
- The ThinkPad was always used as a stand-alone, but it was always plugged into a wall socket. No modem connections.
- The location document was always specified as non-LAN.
Test Setup Within Notes
This was the setup in Notes:

- We took 2700 Person records from the Lotus Name and Address Book, and copied them into a local database. This database had a standard People view, and we ended up creating several hidden views, as well (see below).

- Of these 2700 names, we copied and pasted 500 into another local database. We called this database the 500 document database. It contained the agents which ran our time trials.

- We also copied and pasted 2000 of the Person records into a 5000 document database, with the same agents, as another set of data points (the other 4000 records had no corresponding documents in the Name and Address Book).

- And finally, we copied and pasted those same 2000 Person records into a 48,000 document database (same setup as smaller databases).

Additional Comments
The methodology used was to perform three trials in a row, using the average of the 2nd and 3rd trials. Between each trial, we reset the documents by running an agent to clear one field. Between each trial, the database was closed and reopened.

We did notice that the first trial was usually longer than the 2nd and 3rd trials, often by 10 percent or more. Possible explanations are:

- View caching by the Notes client software.

- When a database is first opened, you will often see a message on the status bar, “Releasing unused space” or “Performing consistency check.” If these actions occur on the lookup database, we wouldn’t notice anything but an increased time for the agent to complete its work. Since we performed three consecutive trials, we felt that the 2nd and 3rd were reliable enough and close enough to average.

In the time trials using the 48,000 document database, if the first and second times recorded were within 3 percent of each other, we used those two times, and did not record a third time. This was because each trial itself was so time consuming.
Test Results

We tested many different LotusScript and @formula methods to determine if there were optimal methods for updating documents, or for looking up data from another Notes database. We did find some rules for optimizing code (see Analysis section), but perhaps more interesting, we found that some methods excelled for small numbers of documents, but were bested by other methods as the number of documents to update increased.

In particular, we found that the LotusScript Walk the View method was the fastest method if the number of documents being updated was very small compared to the total number of documents in the database (less than 1 percent of the documents in the database). From 1 to 15 percent, our tests found the best method was LotusScript NotesDatabase Search. And, from 15 percent up, @formulas were the fastest.

Simple Actions performed just as @formulas did, but because they are more limited in scope, we were not able to perform all of the tests (for instance, we could not look up data using Simple Actions). For these reasons, we have not posted the Simple Actions results.

Full Text Search methods for LotusScript and @formulas were very fast, but there are some caveats. For the reasons which follow, and because of time constraints, we have not listed time trials with Full Text Search methods:

- If the index was not up to date, the agent would spend so long updating the index that the end result was a poor time trial.
- Full text index sizes affect the performance of this kind of agent.
- Full text searches work slightly differently from database Search or @formula SELECT statements. In those searches you can use @Begins. Full text searches don’t allow for that. So, in our testing, the full text query A* would find Amery (as desired), but it would also find D’Amour (not desired). This problem can be coded for, but it adds to the complexity of the agent.
- The Full Text Search method never even came in first place in our time trials.

What follows are the various tests we performed, with brief introductions, the numeric results, and the code used to perform these tests. In the accompanying Freelance presentation you can see the charts which interpret this data. In the Analysis section which follows we have summarized the findings.

**Note** Please be aware that these numbers and findings are meant to indicate “trends” that we found for our tests, and these tests are not necessarily benchmarks. Remember that all our testing was done on a stand-alone machine so that possible network inconsistencies and issues would not affect...
the tests. These findings are published here to give some information regarding performance trends that LotusScript and @formulas will display.

**Update All Documents**

We started out by updating all 500 documents in the “500” database. Our first test was running a LotusScript agent. It was surprisingly slow (about 200 seconds), so we ran an @formula agent and found that it did the same work in about 30 seconds! The LotusScript method we first used was one that gets a handle to a view that contains all the documents and then “Walks the View” to access each document and update it. When we tried another method, getting a collection of documents via the UnProcessedDocuments property of the database, the time for the LotusScript agent to run improved significantly, to 90 seconds. Using the AllDocuments property for the database gave the same results, 90 seconds.

**Inserting a 40-Byte Text String Into a Field**

We found that the amount of data being written to the field did effect the time to update. For consistency, we used a 40-byte text string for our update value (approximately the same length as the string which is looked up in the “lookup” tests that follow).

The results for inserting a 40-byte text string into a field:

- @Formulas: 30 seconds
- LOTUSSCRIPT Unprocessed: 89 seconds
- LotusScript Walk the View: 201 seconds
- LotusScript database Search: 89
- LotusScript AllDocuments: 89

**Inserting the Result of a Lookup Into a Field**

Our next test changed only one variable. Instead of inserting a known 40 byte text string, we inserted a value that we had to look up from another database. In this test, we noticed that the @formula speed differed greatly when optimized (see Analysis for optimization notes). Although each method was slower in this test than in the previous test, they were all slower by approximately the same percentage.
The results for inserting the result of a lookup (approximately 40 bytes) into a field:

- @Formulas (not optimized): 44 seconds
- @Formulas (optimized): 37 seconds
- LotusScript Unprocessed: 100 seconds
- LotusScript Walk the View: 240 seconds
- LotusScript database Search: 100
- LotusScript AllDocuments: 103

**Update All Documents (30 Fields)**

We next modified the formulas and script to look up 30 fields (we added 24 additional fields to the Person records in our copy of the Name and Address Book). This slowed the LotusScript code (Walk the View) down to 300 seconds, but it slowed our initial @formula to 600 seconds! Upon re-coding the @formula (to look up against 30 columns in a hidden view, as opposed to looking up 30 fields in a document), we brought that time down to 400 seconds. And upon further re-coding the @formula to look up against one big concatenated column (with all 30 fields) in a hidden view, we brought the time down to a mere 50 seconds. See the Analysis section for further details on this optimization.

**Update a Subset of All Documents**

Our next test was to update only certain documents in our database. Immediately, we found that @formulas perform much better than LotusScript when running against many documents, but much worse when running against few documents. In an attempt to quantify this difference, and the differences between each LotusScript method, we ran three tests (different numbers of documents) in each of three databases (different database sizes).

It may not be apparent, but when charted, these different methods all come out showing a linear relationship between number of documents to update and the time it took to run the agent. It is easier to examine that relationship in the graphs included in the Freelance Presentation which accompany this paper online (see [http://support.lotus.com/sims2/sims_or2.nsf/85256151005bf9528525614100588964/cb3cea9d70864fbd8525638f0061c6f5?OpenDocument](http://support.lotus.com/sims2/sims_or2.nsf/85256151005bf9528525614100588964/cb3cea9d70864fbd8525638f0061c6f5?OpenDocument) or search the Notes/Domino Knowledgebase at [http://support.lotus.com](http://support.lotus.com) for document number 2478.)
Using the 500 Document Database
In the 500 document database, we ran tests inserting a 40-byte text string into the FullName field, and we also ran tests where that text string was the result of a lookup into a copy of the Name and Address Book. In these tests, the agents updated 10 documents, 100 documents, and 200 documents. In all cases, there were 500 documents in the view, and only those 500 documents in the database (all of these @formulas were optimized).

Insert a 40-byte text string into a field in 10 documents out of 500
Results were:
- @Formulas: 13 seconds
- LotusScript Unprocessed: 11 seconds
- LotusScript Walk the View: 7 seconds
- LotusScript database Search: 3 seconds
- LotusScript AllDocuments: 11 seconds

Insert a 40-byte text string into a field in 100 documents out of 500
Results were:
- @Formulas: 18 seconds
- LotusScript Unprocessed: 26 seconds
- LotusScript Walk the View: 54 seconds
- LotusScript database Search: 20 seconds
- LotusScript AllDocuments: 27 seconds

Insert a 40-byte text string into a field in 200 documents out of 500
Results were:
- @Formulas: 23 seconds
- LotusScript Unprocessed: 45 seconds
- LotusScript Walk the View: 108 seconds
- LotusScript database Search: 41 seconds
- LotusScript AllDocuments: 46 seconds

Inserting a text string that comes from a lookup
Following are the results when the text string inserted into the FullName field comes from a lookup into the copy of the Name and Address Book.
Insert result of a lookup into a field in 10 documents out of 500
Results were:

- @Formulas: 12 seconds
- LotusScript Unprocessed: 11 seconds
- LotusScript Walk the View: 7 seconds
- LotusScript database Search: 4 seconds
- LotusScript AllDocuments: 11 seconds

Insert result of a lookup into a field in 100 documents out of 500
Results were:

- @Formulas: 18 seconds
- LotusScript Unprocessed: 31 seconds
- LotusScript Walk the View: 57 seconds
- LotusScript database Search: 25 seconds
- LotusScript AllDocuments: 31 seconds

Insert result of a lookup into a field in 200 documents out of 500
Results were:

- @Formulas: 23 seconds
- LotusScript Unprocessed: 53 seconds
- LotusScript Walk the View: 115 seconds
- LotusScript database Search: 49 seconds
- LotusScript AllDocuments: 52 seconds

Using the 6000 Document Database
We then performed the same tests in the 6000 document database. Because both types of test had such similar results in the 500 document database, we only tested the lookup for the 6000 document database:

Insert result of a lookup into a field in 10 documents out of 6000
Results were:

- @Formulas: 176 seconds
- LotusScript Unprocessed: 125 seconds
- LotusScript Walk the View: 7 seconds
- LotusScript database Search: 23 seconds
- LotusScript AllDocuments: 138 seconds
Insert result of a lookup into a field in 100 documents out of 6000
Results were:
- @Formulas: 184 seconds
- LotusScript Unprocessed: 148 seconds
- LotusScript Walk the View: 58 seconds
- LotusScript database Search: 47 seconds
- LotusScript AllDocuments: 164 seconds

Insert result of a lookup into a field in 1000 documents out of 6000
Results were:
- @Formulas: 238 seconds
- LotusScript Unprocessed: 356 seconds
- LotusScript Walk the View: 616 seconds
- LotusScript database Search: 293 seconds
- LotusScript AllDocuments: 382 seconds

Using the 48,000 Document Database
Finally, we performed the same tests in the 48,000 document database. This
time, we only tested the case where we insert a 40-byte text string.
Additionally, if the first two time trials were within 3 percent of each other,
we did not perform a third time trial (this was both because the time
required to conduct these trials was becoming prohibitive, and also because
the times were coming in so close to each other, in contrast to the wide
discrepancies we had found in the smaller databases).

Insert a 40-byte text string into a field in 10 documents out of 48,000
Results were:
- @Formulas: 1282 seconds
- LotusScript Unprocessed: 990 seconds
- LotusScript Walk the View: 8 seconds
- LotusScript database Search: 161 seconds
- LotusScript AllDocuments: 1042 seconds
Insert a 40-byte text string into a field in 1000 documents out of 48,000
Results were:

- @Formulas: 1395 seconds
- LotusScript Unprocessed: 1107 seconds
- LotusScript Walk the View: 667 seconds
- LotusScript database Search: 408 seconds
- LotusScript AllDocuments: 1232 seconds

Insert a 40-byte text string into a field in 1900 documents out of 48,000
Results were:

- @Formulas: 1349 seconds
- LotusScript Unprocessed: 1329 seconds
- LotusScript Walk the View: 1275 seconds
- LotusScript database Search: 611 seconds
- LotusScript AllDocuments: 1487 seconds

Analysis

Every case is unique, but we do want to give some sort of summary from all these tests.

Deciding What to Choose

Here we summarize our findings, to help us decide when we should use LotusScript and when we should use @formulas.

Agents Processing Documents

Regarding our agent tests for performance, LotusScript vs. @Formulas:

- LotusScript Walk the View method excels at running against just a few documents within a large database. Specifically, less than 1 percent of the total documents in the database.
- LotusScript database Search was the fastest LotusScript method when the percentage of documents was between 1 and 15 percent of the total number of documents in the database.
- Once the percentage of documents being updated exceeds 15 percent, @formulas were the fastest method.
- We found that the LotusScript Full Text Search method was very fast, but not as fast as LotusScript database Search.

342 Performance Considerations for Domino Applications
• @Formulas using Full Text Search were also very fast. For small subsets of documents, they outperformed @formulas which did not utilize Full Text Search; however, for those small subsets they were still not as fast as LotusScript Walk the View or LotusScript database Search. For larger subsets, @formulas not using Full Text Search are faster than those which use it.

• Almost all of our tests involved either filling a single field with a hard-coded value, or filling a single field with a value looked up from another database. When we looked up multiple fields, instead of just one, we found that LotusScript methods were affected less than @formulas. This difference, however, is small compared to the performance differences noted above.

• Simple Actions perform as do @formulas.

Tip We found that for both LotusScript and @formulas, coding techniques can make a huge difference in performance.

@Formulas (@DbLookup and @DbColumn Formulas)
When using @Db lookupformulas:

• Set a temp variable to the returned value, and then refer to that temp variable. This will avoid performing the same lookup multiple times within one formula.

• Use “Caching” whenever possible.

• Lookup against hidden views with a minimum amount of data displayed.

• Use column number rather than field name.

• If the formula will need to return many fields, avoid performing multiple lookups where each lookup returns one item. Instead, concatenate all of the fields you will need to look up into one column, and perform your lookup just against that column. Then parse the returned data into the appropriate fields in the destination document.

Note This method cannot always be used. You may not know in advance which fields you will have to look up, or you may not have designer control over the lookup database.

LotusScript
Some things to think about with LotusScript:

• LotusScript often gives several methods for getting at the same piece of data. In the case of our tests, we found that using the ColumnValues property saved about 10 percent over simply getting a handle to the document and then using extended class syntax (such as, x = doc.fieldname).
• The Walk the View method is an excellent choice when trying to find a small subset of the documents in the database to process. With this method, there is virtually no overhead associated with the number of documents in the database or in the view.

• Using the database Search, UnProcessedDocs, or AllDocuments method is faster than the Walk the View method when you need to update many documents, because the collection used in this method provides faster document access than does a Notes view (which is what the Walk the View method utilizes). With these methods, once you have a collection of documents you then use an If statement to see if the current document meets a criteria to be updated.

Rules of Thumb
Some rules of thumb:

• The more bytes written to a field, the longer it takes to perform the update.
• Writing data to a field which you have looked up takes longer than writing that same data if it is hard-coded.
• The more fields you need to update, the longer it takes.
• The more documents you need to update, the longer it takes.
• Overwriting with the same data is faster than overwriting with different data.
• Running against documents which you don’t need to update takes time.

Other Factors to Consider
Some other things to note:

• For formulas which perform lookups, @formulas are much easier to code, and to maintain, than are LotusScript scripts. On the other hand, they usually require special (often hidden) views to lookup against. The maintenance of these views takes some time and effort and can also create an issue for overall database size. LotusScript doesn’t require any such overhead. Two LotusScript search methods, Search and FullTextSearch, can be used to find a collection of documents that match criteria without using an indexed (sorted column) view to find documents.

• LotusScript gives the ability to get data from rich-text fields. @Formulas do not.

• LotusScript can easily update documents in other Notes databases. @formulas can accomplish this (with a great deal more work) only if run in the foreground.
Summary

And finally, an overall summary:

- Use LotusScript if you want to perform actions which require iteration. For example, processing all databases on a server or all fields on a document or working with a looping agent.

- Use LotusScript if you need to do any of the following: ACL manipulation, OLE-Automation, calling external C functions, using the LotusScript Data Object.

- Use LotusScript if you are updating a small percentage of the documents in a database.

- Use @formulas in the following places: Selection formulas for views, Column formulas, Field Formulas.

- Use @formulas for simple tasks, to keep your development time minimized. In many cases, you can accomplish the same task with many fewer lines of code with @formulas.

- Use @formulas if you are updating a large percentage of the documents in a database.
Appendix C-5
Execution Order of Notes Form Events and Formulas

The following list was generated by embedding messagebox commands and @Prompt statements into all the possible events and formulas on a test form that contained a mixture of field types. Although the form did not include all possible types or evaluation combinations, the results can be used to gain an understanding of the execution order.

The test form contained five fields, listed from top to bottom in the following order:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field type</th>
<th>Field details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>Editable/Text Field</td>
<td>With Default Value, Input Translation and Input Validation Formulas</td>
</tr>
<tr>
<td>From</td>
<td>Computed When Composed/Authors Name Field</td>
<td>With Value Formula</td>
</tr>
<tr>
<td>Counter</td>
<td>Computed/Number Field</td>
<td>With Value Formula</td>
</tr>
<tr>
<td>DisplayNum</td>
<td>Computed For Display/Number Field</td>
<td>With Value Formula</td>
</tr>
<tr>
<td>Body</td>
<td>Editable/RTF Field</td>
<td>With Default Value Formula</td>
</tr>
</tbody>
</table>

Execution Order Results

In the examples below, scripts (events) are in italics, and formulas in regular text.

Composing a Document

*Form - Initialize Event*
Form - Window Title Formula
*Form - QueryOpen Event*
Subject Field - Default Value Formula
*Subject Field - Initialize Event*
From Field - Value Formula
*From Field - Initialize Event*
Counter Field - Value Formula
*Counter Field - Initialize Event*
DisplayNum Field - Value Formula
DisplayNum Field - Initialize Event
Body Field - Default Value Formula
Body Field - Initialize Event
Subject Field - Entering Event
Form - PostOpen Event

**Saving Document with an @Command([FileSave]) or FileSave from Menu**

Form - QuerySave Event
Subject Field - Input Translation Formula
Counter Field - Value Formula
DisplayNum Field - Value Formula
Subject Field - Input Validation Formula

**Closing the window with an @Command([FileCloseWindow]) or Closing Window**

Form - QueryClose Event
Form - Terminate Event
Subject Field - Terminate Event
Form Field - Terminate Event
Counter Field - Terminate Event
DisplayNum Field - Terminate Event
Body Field - Terminate Event

**Reopening an Existing Document in Read Mode**

Form - Initialize Event
Form - Window Title Formula
Form - QueryOpen Event
Subject Field - Initialize Event
From Field - Initialize Event
Counter Field - Initialize Event
DisplayNum Field - Value Formula
DisplayNum Field - Initialize Event
Body Field - Initialize Event
Form - PostOpen Event

**Toggling from Read Mode to Edit Mode with Document Open**

Form - QueryModeChange Event
Subject Field - Entering Event (Note: Depends on Cursor Position)
Form - PostModeChange Event

348 Performance Considerations for Domino Applications
Toggling from Edit Mode to Read Mode with Document Open (and with No Changes)

*Form - QueryModeChange Event*
*Form - PostModeChange Event*

Toggling from Edit Mode to Read Mode with Document Open (with Changes Made and Saving)

*Form - QueryModeChange Event*

**Do You Want to Save Your Changes? - Answer YES**

*Form - QuerySave Event*
*Subject Field - Input Translation Formula*
*Counter Field - Value Formula*
*DisplayNum Field - Value Formula*
*Subject Field - Input Validation Formula*
*Form - PostModeChange Event*
*Form - QueryClose Event*
*Form - Terminate Event*

**Subject Field - Terminate Event**
**Counter Field - Terminate Event**
**DisplayNum Field - Terminate Event**
**Body Field - Terminate Event**
*Form - Initialize Event*
*Form - Window Title Formula*
*Form - QueryOpen Event*
*Subject Field - Initialize Event*
*From Field - Initialize Event*
*Counter Field - Initialize Event*
*DisplayNum Field - Value Formula*
*DisplayNum Field - Initialize Event*
*Body Field - Initialize Event*
*Form - PostOpen Event*

Toggling from Edit Mode to Read Mode with Document Open (with Changes Made but without Save)

*Form - QueryModeChange Event*

**Do You Want to Save Your Changes? - Answer NO**

*Form - PostModeChange Event*
*Form - QueryClose Event*
Moving the Cursor from One Editable Field to Another

First Field - Exiting Event  
Second Field - Entering Event

Pressing F9 to Refresh Fields While in Edit Mode

Subject Field - Input Translation Formula  
Counter Field - Value Formula  
DisplayNum Field - Value Formula  
Subject Field - Input Validation Formula

Form - PostRecalc Event

Note  Closing the document after changing it results in the prompt: “Do you wish to save this document?” Selecting Yes performs the same as a Save followed by a Window Close. Selecting No performs the same as a Window Close.

Note  In Notes 4.0, it appears that the Initialize and Terminate events of Action Hotspots and Buttons either in the form or embedded in an RTF field do not get activated.

Note  In order to enter an Initialize or Terminate script into a Computed or Computed for display field, it may be necessary to first set the field to Editable, add the scripts and then change the field definition back to Computed or Computed for Display.
Appendix C-6
How Domino Handles Document URL Requests

This appendix describes what happens in Domino under the cover when a URL is submitted from a Web browser. It is an excerpt from the article “The Architecture of the Domino Web Server - Part 2” by Richard Schwartz. You can view the entire article in Iris Today, the technical Webzine located at www.notes.net. This Web site is produced by Iris Associates, the developers of Domino and Notes. Visit Notes.net to obtain the latest version of the article as well as other related articles.

Domino.Commands

Users interact with Domino applications through commands issued as URLs. Each command specifies a top-level object and 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.


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

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

?OpenForm
The ?OpenForm command executes a similar sequence of steps:

1. Create a new document in memory.
2. Load the Notes form referenced in the URL.
3. Scan the Notes form and execute all default value formulas and computed field formulas, adding items to the in-memory document.
4. Run the appropriate agent if an item named $$QueryOpenAgent exists.
5. Render the Notes 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 the a Web server. If and when the browser sends edited information back, the Domino Web Server will re-establish all the data structures necessary to handle the updates.

?CreateDocument and ?SaveDocument
?CreateDocument and ?SaveDocument are the two commands that receive HTTP POST data generated by browsers when a user clicks the Submit button on an HTML form and saves the data in Notes documents. Submit buttons on HTML forms generated by the ?OpenForm command send the ?CreateDocument command, and Submit buttons on HTML forms generated by the ?EditDocument command send the ?SaveDocument command.
command. The two commands follow similar sequences of steps. The steps for \texttt{CreateDocument} begin with:

2. Open the Notes form referenced in the URL.
3. Scan the Notes form and execute all default field formulas and computed value formulas, adding items to the in-memory document.

\texttt{SaveDocument}

The steps for \texttt{SaveDocument} begin with:

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

Then both commands continue on essentially the same path:

4. Create items for all data sent in the POST data.
5. Scan the Notes form and execute all translation, validation and computed value formulas, updating items in the in-memory document and returning validation errors as HTML.
6. Scan the Notes form and eliminate any computed-for-display items from the in-memory document.
7. Run the appropriate agent if an item named \texttt{$\$QuerySaveAgent} exists.
8. Save the in-memory document to the appropriate database file.
9. If the \texttt{$\$Return} item exists, run the appropriate formula from the Notes form, and 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.

One thing worth noting is the fact that formulas that a user might have thought executed only once at the time that an HTML form was sent by either the \texttt{EditDocument} or \texttt{OpenForm} command will actually execute again when \texttt{CreateDocument} or \texttt{SaveDocument} is processed. This can result in some confusion if these formulas have time-dependent values, or are based on lookups of constantly changing information.
Appendix C-7
Ways to Control Caching of Pages Served by Domino

This appendix was written by Andrew Wharton, and was first published as an article in Iris Today, the technical Webzine located at www.notes.net. This Web site is produced by Iris Associates, the developers of Domino and Notes. Visit Notes.net to obtain the latest version of the article as well as other related articles.

Note Since this article was written, more support for client-side caching has been added to Domino. On every page being served by Domino there is now a Last-Modified header. This means that a Web browser that already has a copy of the page in its local cache does not have to reload the page unless it has been changed since the last time it was downloaded.

Note Until and including the current versions of Domino (R4.6.6 and R5.0.2) use of the feature named “Use JavaScript when generating pages” will always turn off the command cache. Iris is looking into how this restriction can be removed in the future.

Expanded Command Caching Introduced in Domino 4.61

Since its initial release, the Domino HTTP server has had the ability to keep a volatile, in-memory cache of requested HTML pages called the Command Cache. The very nature of a dynamic Web applications server would preclude the caching of all Web pages. Some Web pages, however, are “static enough” that they can be reused when the same URL or command is issued from a browser. This section describes how the Command Cache works so that you can design applications that take advantage of its capabilities — and the performance gains it provides.

Command Cache Fundamentals

The Command Cache saves Web pages in memory in the event that the same Web page URL will be requested in the future, and that the cached page will still be valid to serve for that URL. To do this correctly, the cache must understand the nature of the page being saved, and it must know how to determine if that page is still valid for serving. While the Command Cache has been part of Domino since its inception, the role of the Command Cache is continually expanding as we make it more and more capable of understanding pages and determining their validity.
Caching URL Commands

The following types of Domino URL commands (and the HTML that they generate) are currently the only candidates for the Command Cache:

- ?OpenDatabase
- ?OpenView
- ?OpenDocument
- ?OpenForm
- ?ReadForm

Other commands (for example, ?EditDocument) are not considered for caching. Domino only serves pages from the Command Cache if the URL request exactly matches the URL of the cached pages, even though Domino URL syntax allows multiple ways to specify the same URL command (by using replica IDs instead of database names, or view names instead of view UNIDs).

Caching @Functions

Determining the algorithm to make an accurate and timely decision as to which Web pages are cacheable is one stumbling block to making a very useful caching system. In 4.6, we considered the very presence of @functions (among others) too volatile, and thus, did not consider any pages with @functions as candidates for caching. Unfortunately, this meant that many Web pages could not take advantage of caching and the performance gains that it provides.

In R4.61, much has changed. As part of the ongoing effort to provide performance improvements in the Domino server, we now provide the ability to cache Web pages that contain @functions. This is a very simple statement with very powerful implications for the Web site administrator, the Web application designer and the Web application user.

Domino 4.61 now has the ability to analyze all macro language (@function) formulas for their dependencies through the Formula Analyzer. Rather than exclude all pages that contain any @function formulas from the Command Cache, the Formula Analyzer intelligently examines the formula on the page and decides its level of volatility. By doing this first, Domino can decide immediately whether the command is too volatile to be cached (for example, the Web page contains @Now) and, perhaps more importantly, under what conditions a cached command becomes invalid.
The logic for these caching schemes is based on very conservative thinking. In all cases where the slightest bit of ambiguity exists, Domino will not cache the page. We considered the performance loss of re-rendering the resultant HTML to be a small tradeoff to the possibility of sending out-of-date, incorrect, or inappropriate information to the requesting user.

The Formula Analyzer is turned OFF by default in 4.61, meaning that any @function will result in the page not being cached. To enable the Formula Analyzer, place the following line in your NOTES.INI file:

```
DominoAnalyzeFormulas=1
```

It is important to note that without this setting, the Command Cache behaves exactly the way that it did in 4.6; you will still get the benefit of command caching, just without the expanded usefulness of caching Web pages where @formulas were used in the process.

### Cache Flags

When the Formula Analyzer examines a command invoking @formulas, flags are assigned to that cache candidate based on the evaluation of the @formula(s), among other things. The following flags are used by the Domino HTTP server to determine the cache strategy, explained in the next section. (A cache candidate can and often does have multiple flags.)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OffDb</td>
<td>The page uses data outside the current database. This includes the use of CGI variables.</td>
</tr>
<tr>
<td>TimeVariant</td>
<td>The page uses time-variant data (such as @Now).</td>
</tr>
<tr>
<td>HadEffect</td>
<td>The page has an important side effect (such as @DialogBox).</td>
</tr>
<tr>
<td>UsedEnv</td>
<td>The page uses the environment (NOTES.INI). This does not include CGI variables.</td>
</tr>
<tr>
<td>UserVariant</td>
<td>The page is dependent on the user’s identity. This includes using any data or design note that includes Read ACLs, Readers fields, Authors fields or controlled access sections.</td>
</tr>
<tr>
<td>DesignUserVariant</td>
<td>The page is from a database that has protected design elements.</td>
</tr>
<tr>
<td>DbData</td>
<td>The page uses data in the database other than the referenced document. This includes all views, embedded views in forms, and so on.</td>
</tr>
<tr>
<td>UsedDocId</td>
<td>The page uses the document’s ID.</td>
</tr>
<tr>
<td>UsedNewDoc</td>
<td>The page uses a newly-created in-memory note.</td>
</tr>
<tr>
<td>Unknown</td>
<td>The page does something that couldn’t be analyzed (such as executed LotusScript).</td>
</tr>
<tr>
<td>Error</td>
<td>The page generated an error of some sort.</td>
</tr>
</tbody>
</table>
Cache Strategy

Domino uses the cache flags to determine the cache strategy for a given command. A cached command can have multiple cache strategies. The most restrictive strategy wins. The strategy is stored with the command if it is cached, and is used to help determine whether the command is still valid each time it is retrieved from the cache. The cache strategies are:

<table>
<thead>
<tr>
<th>Cache strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DontCache</td>
<td>Don’t cache the response at all.</td>
</tr>
<tr>
<td>Document</td>
<td>Invalidate the cached response when the document changes.</td>
</tr>
<tr>
<td>HadEffect</td>
<td>The page has an important side effect (such as @DialogBox).</td>
</tr>
<tr>
<td>DbDesign</td>
<td>Invalidate the cached response when the database design changes.</td>
</tr>
<tr>
<td>DbData</td>
<td>Invalidate the cached response when any of the data in the database changes.</td>
</tr>
<tr>
<td>OnlyAnonymous</td>
<td>Cache the response, but only serve it when the user is anonymous.</td>
</tr>
</tbody>
</table>

Any commands with the following flags are not cached (the DontCache strategy):

- OffDb
- TimeVariant
- HadEffect
- UsedEnv
- Error
- Unknown
- UserVariant (if authenticated)
- DesignUserVariant (if authenticated)
Here is an example of how command flags set the cache strategy. We will annotate the cache flags Flag(flag), where flag refers to the flags above. The cache strategy will be annotated as Strategy(strategy), where strategy refers to the strategies above. Using this notation, cached commands are assigned a cache strategy that defines how the cache is invalidated. This is illustrated by the following pseudocode:

Sub CACHESTRATEGY
CASE OPENDOCUMENT
  CACHESTRATEGY = Document
  If Flag(OffDb) or Flag(TimeVariant) or Flag(HadEffect) or Flag(UsedEnv) then
    CACHESTRATEGY = DontCache //don't cache
    return CACHESTRATEGY //we're done
  End If
  If Flag(UsedDocId) and Flag(UsedNewDoc) then
    CACHESTRATEGY = DontCache //don't cache
    return CACHESTRATEGY //we're done
  End If
  If Flag(UserVariant) or Flag(DesignUserVariant) then
    If USER_AUTHENTICATED then
      CACHESTRATEGY = DontCache //don't cache
      return CACHESTRATEGY //we're done
    End If //not authenticated
    CACHESTRATEGY = CACHESTRATEGY + OnlyAnonymous //continue
  End If
  CACHESTRATEGY = CACHESTRATEGY + DbDesign //continue
  If Flag(DbData) then
    //remove Document strategy and add DbData strategy
    CACHESTRATEGY = CACHESTRATEGY - Document + DbData //continue
  End If
  return CACHESTRATEGY
CASE OPENFORM
  // etc.
End Sub
You Are Smarter Than Your Server

The Command Cache does a good job understanding the page creation process and automatically invalidating pages, but it is not perfect. Because of its conservative nature, it errs on the side of not caching pages in order to guarantee the correctness of the page returned. You may decide that certain pages can be cached where the server determines that they cannot be. We’ve provided controls so that you can override the cache behavior where appropriate. The following fields can control the use of the cache to some extent:

$CacheOptions — If the value of this field is the text string “0,” then the response is not cached.

$CacheValid — The value of the numeric text string N will be evaluated and will protect the response from validity checks for N seconds. This setting can be globally set by using the NOTES.INI setting

DominoDefaultCacheValid=N

The default for the HTTP server is N=0.

The $CacheValid field lets you tell the cache that this page should be considered valid for a certain number of seconds regardless of what Domino determines the cache strategy to be. Consider a simple home page that is being continually edited. As this page would be given the “Document” strategy, the cache entry would become invalid each time the page is edited. Let’s say you consider it acceptable that the home page is not continually updated as a tradeoff for performance. You can communicate this to Domino by creating a $CacheValid field on the document with a value of “180.” This means that the results of the page should be considered valid for 180 seconds. After that time, the normal validity checks will take place.

Viewing the Cache Strategy and Flags for a Page

If the NOTES.INI variable “DominoTraceCmdCache=1” is set, the cache strategy and the cache flags will be included in the HTTP header information that is sent to the browser when a page is served. To view this information, you can use a tool that displays this information (such as a wire-sniffer) or download the following Java application (9 Kb) written by Bob Congdon at Iris. This tool is not a supported product; it’s just intended to be a useful utility.

Make sure that CLASSPATH has “.” on it, and run the application by entering “java SpyFrame” at the command line. While Bob developed the application using the JDK 1.1.1, it should run fine with JDK1.0x, since it doesn’t use any JDK 1.1.1-specific classes.
The following header information is sent:

X-DominoCmdCache-EvalInfo — Following this is a comma-separated list of flag words (information pieces) that the Formula Analyzer set in the process of evaluating the page (see above).

X-Domino-CmdCache-Strategy — Following this is a comma-separated list of flag words that denotes the cache strategy that was assigned to the command based on the inspection of the EvalInfo (see above).

X-Domino-CmdCache-ValidSeconds — This is the number of seconds that the response will be assumed to be valid without the benefit of a validity check. This is the time set in either the $CacheValid field of the DominoDefaultCacheValid NOTES.INI variable.

X-Domino-CmdCache-CheckValid — This is the clock time that states when the validity check will again be allowed.

X-Domino-CmdCache-DataMod — This is the recorded data modification date that is used for the validity checking.

X-Domino-CmdCache-DesignMod — This is the recorded design modification date used for validity checking.

Cache Statistics

You can check the efficacy of your Command Cache by monitoring the following server statistics:

Domino.Cache.Command.Count — This is the actual number of commands that are contained in the Command Cache.

Domino.Cache.Command.MaxSize — This is the maximum number of commands that can be cached. This is the number configured from the Server document in the Public Address Book using the “Maximum Cached Commands” field.

Domino.Cache.Command.HitRate — This is the percentage ratio of the number of times a valid cache entry is found in the cache to the number of times the cache was investigated for a cache entry.

Domino.Cache.Command.DisplaceRate — This is the percentage ratio of the number of times that a new cached command displaces an aged command to the number of times the cache was investigated for a cache entry.
List of @Functions

The following is a list of @functions and the Eval flags that are set at compute time. If a function is not listed here, safely assume that there is no associated Eval flag set. The Eval flag Depends means that the evaluation of the entire formula will determine the Eval flag set. If the @function says “Fallback,” that means that there is an evaluation that is Web server-specific and this is the non-Web version. Its converse is “Web.”

@Accessed — OffDatabase,UsedDocId
@Certificate — OffDatabase
@Command - Web — Depends
@Command([Compose]) — Depends,DbDesign,OffDatabase
@Command([FileSave]) — HadEffect
@Created — UsedDocId
@DbColumn - Fallback — UserVariant,DbDesign,DbData,Unknown,Depends,OffDatabase
@DbCommand - Fallback — Unknown
@DbCommand - Web — Depends
@DbLookup - Fallback — Depends,Unknown,DbData,DbDesign,UserVariant,OffDatabase
@DbManager — DbDesign
@DbTitle — DbDesign
@DocumentUniqueID — UsedDocId
@Environment — HadEffect,UsedEnvironment
@GetDocField — DbData,UserVariant
@GetPortsList — UsedEnvironment
@GetProfileField — DbData,UserVariant
@InheritedDocumentUniqueID — UsedDocId
@MailEncryptSavedPreference - Fallback — UsedEnvironment
@MailEncryptSentPreference - Fallback — UsedEnvironment
@MailSavePreference - Fallback — UsedEnvironment
@MailSend - Fallback — HadEffect
@MailSignPreference - Fallback — UsedEnvironment
This template is designed to facilitate pagination.

Appendix C-7: Ways to Control Caching of Pages Served by Domino  363
Special Notices

This publication is intended to help you develop applications using Lotus Domino Release 5.0 and Lotus Domino Release 4.6.

The information in this publication is not intended as the specification of any programming interfaces that are provided by Lotus Domino. See the publications section of the announcement for Lotus Domino and related products for more information about what publications are considered to be product documentation.

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

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one); and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available subject to appropriate terms and conditions, including, in some cases, payment of a fee.

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

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

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

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

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

- AIX
- AS/400
- DB2
- IBM®
- MQSeries
- OS/2
- OS/Warp
- ThinkPad®

The following are trademarks of Lotus Development Corporation in the United States and/or other countries:

- Domino
- Lotus 1-2-3®
- LotusScript®
- Lotus SmartSuite®
- Notes
- Notes Mail®

366 Performance Considerations for Domino Applications
NotesPump
NotesSQL
Lotus®
Lotus Notes®
RealTime Notes
SmartIcons®

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power to Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

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

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries (for a complete list of Intel trademarks see www.intel.com/tradmarx.htm).

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

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product or service names may be the trademarks or service marks of others.
Additional Web Material

Additional Web material is referenced in this book and can be found on the IBM Redbooks Web site. The material is described in the following table.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 5602a2.zip      | Contains two databases that illustrate the techniques described in Appendix A-2, “High Performance Reference Databases”:  
|                 | - LWDir.nsf contains the compressed people records and shows different ways to search and see details for them.  
|                 | - mail46lw.nsf is a mail database where the memo form has an extra action button named GetPerson that looks up and gets people information from LWDir.nsf |
| 5602a2x.zip     | Contains template (lwab.ntf) for databases where an agent creates a compressed version of the people information in Domino Directory.  
|                 | The template also contains scheduled agents to keep the database in synch with Domino Directory. Note that the data format used in this template is slightly different from the one used in the databases in 5602a2.zip, but the concepts it illustrates are the same. |
| 5602b4c2.zip    | Contains database with test data used for writing appendices B-4, “Sorting” and C-2, “View Indexing Comparisons.” |
| 5602b5b7.zip    | Contains three databases (redcust.nsf, redtran.nsf and redacct.nsf) that illustrate the techniques described in Appendices B-5, “Switching Seamlessly Between Databases” and B-7, “Cascaded Deletions Example.” |
| rc2034.pdf      | Softcopy version of A Planning Model for Lotus Notes Applications by Bucky Pope, IBM Watson Research. |
| reorgprocess.pdf| Softcopy version of Analysis and Process for Balancing Domino Mail Servers at the IBM Global Services North Data Centers by John Capurso, Jon Champlin, Bill Pointer, Bucky Pope, Lorrie Renz. |
| rc94265.pdf     | Softcopy version of Characterizing Lotus Notes Email Clients by Bucky Pope, IBM Research Report. |
How to Get the Web Material

To get the Web material, point your Web browser to:
ftp://www.redbooks.ibm.com/redbooks/sg245602

Alternatively, you can go to the Redbooks Web site at:
http://www.redbooks.ibm.com

Select Additional Materials and open the file that corresponds with the redbook form number.
Related Publications

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

International Technical Support Organization Publications

For information on ordering these ITSO publications see, “How To Get ITSO Redbooks.”

- **Lotus Domino for AS/400: Performance, Tuning, and Capacity Planning**, IBM form number SG24-5162
- **Lotus Domino R5 on OS/2 Platform Exploring New Features and Interoperability**, IBM form number SG24-5497
- **Lotus Domino for S/390 Performance Tuning and Capacity Planning**, IBM form number SG24-5149
- **Netfinity and Domino R5.0 Integration Guide**, IBM form number SG24-5313, Lotus part number CT7BKNA
- **Lotus Domino R5 for IBM RS/6000**, IBM form number SG24-5138, Lotus part number CT7BHNA
- **High Availability and Scalability with Domino Clustering and Partitioning on Windows NT**, IBM form number SG24-5141, Lotus part number CT6XMIE
- **HTTP Server Performance and Capacity Planning**, IBM form number SG24-5645
- **Building a Portal with Lotus Domino R5**, IBM Redpaper REDP0019 (only available in softcopy from the IBM Redbooks Web site)
- **Lotus Domino 5.0: A Developers Handbook**, IBM form number SG24-5331, Lotus part number CC7EDNA
- **Creating Customized Solutions with Domino.Doc**, IBM form number SG24-5658, Lotus part number CC6X3NA
- **Developing Web Applications Using Lotus Notes Designer for Domino 4.6**, IBM form number SG24-2183, Lotus part number 12974
- **Lotus Notes 4.5: A Developers Handbook**, IBM form number SG24-4876, Lotus part number AA0425
- **Using VisualAge for Java to Develop Domino Applications**, IBM form number SG24-5424, Lotus part number CT6ENNA
Developing e-business Application Using Lotus Enterprise Solution Builder R3.0, IBM form number SG24-5405, Lotus part number CT69TNA

Connecting Domino to the Enterprise Using Java, IBM form number SG24-5425, Lotus part number CT6EMNA

Lotus Domino for AS/400: Integration with Enterprise Applications, IBM form number SG24-5345, Lotus part number CT7BMNA

LotusScript for Visual Basic Programmers, IBM form number SG24-4856, Lotus part number 12498

Lotus Solutions for the Enterprise, Volume 1. Lotus Notes: An Enterprise Application Platform, IBM form number SG24-4837, Lotus part number 12968 (available in soft copy only)

Lotus Solutions for the Enterprise, Volume 2. Using DB2 in a Domino Environment, IBM form number SG24-4918, Lotus part number CT69BNA

Lotus Solutions for the Enterprise, Volume 3. Using the IBM CICS Gateway for Lotus Notes, IBM form number SG24-4512 (available in soft copy only)

Lotus Solutions for the Enterprise, Volume 4. Lotus Notes and the MQSeries Enterprise Integrator, IBM form number SG24-2217, Lotus part number 12992

Lotus Solutions for the Enterprise, Volume 5. NotesPump, the Enterprise Data Mover, IBM form number SG24-5255, Lotus part number CT69DNA

Enterprise Integration with Domino for S/390, IBM form number SG24-5150

Other Lotus-Related ITSO Publications

The publications listed in this section may also be of interest:

Moving from Novell GroupWise to Lotus Domino R5, IBM form number SG24-5650, Lotus part number CT6QENA

Moving from cc:Mail to Lotus Domino R5, IBM form number SG24-5649, Lotus part number CT6Q9NA

A Roadmap for Deploying Domino in the Organization, IBM form number SG24-5617, Lotus part number CT6P8NA

The Three Steps to Super.Human.Software: Compare, Coexist, Migrate; From Microsoft Exchange to Lotus Domino, Part One: Comparison, IBM form number SG24-5614, Lotus part number CT7QJNA

The Three Steps to Super.Human.Software: Compare, Coexist, Migrate; From Microsoft Exchange to Lotus Domino, Part Two: Coexistence and Migration, IBM form number SG24-5615, Lotus part number CT7QWNH

Lotus Notes and Domino R5.0 Security Infrastructure Revealed, IBM form number SG24-5341, Lotus part number CT6TPNA

372 Performance Considerations for Domino Applications
- The Next Generation in Messaging. Moving from Microsoft Mail to Lotus Notes and Domino, IBM form number SG24-5152, Lotus part number CT7SBNA
- Eight Steps to a Successful Messaging Migration: A Planning Guide for Migrating to Lotus Notes and Domino, IBM form number SG24-5335, Lotus part number CT6HINA
- Deploying Domino in an S/390 Environment, IBM form number SG24-2182, Lotus part number 12957
- The Next Step in Messaging: Upgrade Case Studies on Lotus cc:Mail to Lotus Domino and Lotus Notes, IBM form number SG24-5100, Lotus part number 12992
- The Next Generation in Messaging. Moving from Novell GroupWise to Lotus Notes and Domino, IBM form number SG24-5321, Lotus part number CT7NNNA
- From Client/Server to Network Computing, A Migration to Domino, IBM form number SG24-5087, Lotus part number CT699NA (available in soft copy only)
- Lotus Domino Release 4.6 on IBM RS/6000: Installation, Customization and Administration, IBM form number SG24-4694, Lotus part number 12969
- High Availability and Scalability with Domino Clustering and Partitioning on AIX, IBM form number SG24-5163, Lotus part number CT7J0NA
- Lotus Domino for AS/400: Installation, Customization and Administration, IBM form number SG24-5181, Lotus part number AA0964
- Lotus Domino for S/390 Release 4.6: Installation, Customization & Administration, IBM form number SG24-2083
- Porting C Applications to Lotus Domino on S/390, IBM form number SG24-2092, Lotus part number AB1720 (available in soft copy only)
- Managing Domino/Notes with Tivoli Manager for Domino, Enterprise Edition, Version 1.5, IBM form number SG24-2104 (available in soft copy only)
Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Collection kit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lotus Redbooks Collection</td>
<td>SK2T-8039</td>
</tr>
<tr>
<td>Tivoli Redbooks Collection</td>
<td>SK2T-8044</td>
</tr>
<tr>
<td>Application Development Redbooks Collection</td>
<td>SK2T-8037</td>
</tr>
<tr>
<td>Netfinity Hardware and Software Redbooks Collection</td>
<td>SK2T-8046</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (BkMgr Format)</td>
<td>SK2T-8040</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (PDF Format)</td>
<td>SK2T-8043</td>
</tr>
<tr>
<td>AS/400 Redbooks Collection</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>Transaction Processing and Data Management Redbooks Collection</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>Networking and Systems Management Redbooks Collection</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>System/390 Redbooks Collection</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>IBM Enterprise Storage and Systems Management Solutions</td>
<td>SK3T-3694</td>
</tr>
</tbody>
</table>

Other Resources

These publications are also relevant as further information sources:


• *Analysis and Process for Balancing Domino Mail Servers at the IBM Global Services North Data Centers* by John Capurso, Jon Champlin, Bill Pointer, Bucky Pope, Lorrie Renz. Available online at: ftp://www.redbooks.ibm.com/redbooks/sg245602


• *Lotus Knowledge Base* - contains Tech Notes and Papers. Available online at: http://support.lotus.com/
Web Sites

These Web sites are also relevant as further information sources:

• **http://www.lotus.com/itcentral**
  Lotus IT Central has a performance zone with news and technical literature about performance in Domino.

• **http://notes.net/**
  Notes.net from Iris — the developers of Notes and Domino — is a technical Web site with discussion forums, documentation and the Webzine *Iris Today* with many good articles about performance in Domino.

• **http://support.lotus.com/**
  Lotus Support Web site — Search using keywords or browse the Lotus Knowledge Base and locate helpful and informative tech notes and technical papers for the entire Lotus Product Family. This source of information contains the latest technical information updated hourly.

• **http://www.lotus.com/developer/**
  Lotus Developer Network — Lotus’ primary destination for the latest developer information and resources. Contains articles about new and current technologies along with relevant tips and techniques to help you build dynamic collaborative e-business applications.

• **http://www.ibm.com/developer/**
  The IBM developerWorks Web site is designed for software developers, and features links to a host of developer tools, resources, and programs.
How to Get IBM Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

  Search for, view, download or order hardcopy/CD-ROM redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.
  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**
  Send orders by e-mail including information from the redbooks fax order form to:

  In United States: usib6fpl@ibmmail.com

- **Telephone Orders**
  United States (toll free) 1-800-879-2755
  Canada (toll free) 1-800-IBM-4YOU

- **Fax Orders**
  United States (toll free) 1-800-445-9269
  Canada (toll free) 1-800-267-4455

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

---

IBM Intranet for Employees

IBM Redbook Fax Order Form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First name ___________________________ Last name ___________________________

Company ________________________________

Address ________________________________

City ________________________________ Postal code _______ Country _______

Telephone number __________ Telefax number __________ VAT number __________

☐ Invoice to customer number ________________________________

☐ Credit card number ________________________________

Credit card expiration date __________ Card issued to ___________________________ Signature ___________________________

We accept American Express, Diners, Eurocard, MasterCard, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.
Index

Symbols
%INCLUDE Files, 83
?CreateDocument, 352
?EditDocument, 352
?OpenDatabase, 356
?OpenDocument, 351, 356
?OpenForm, 352, 356
?OpenView, 356
?ReadForm, 356
?SaveDocument, 352

A
Access Control List (ACL) limits, 308
Access control requirements, 10, 19
Actor in Use Case Model, 41
Add-In Task, 25
Agents, 82
agent log, 201
billing class, 192
caching, 82
Java agent, 144, 145
LotusScript agent, 144
scheduled agent, 145
server agents, 192
troubleshooting, 191
AIX, 171
monitoring tools, 171, 172
AppendToTextList, 77
Applets, 137, 146
view applet, 130
Application design asynchronous processing, 85
case study, 205
Application design considerations access control requirements, 19
archive database, 28
background processing, 33
bandwidth requirements, 29
business process, 40
caching, 31
dynamic lookups, 37
enterprise integration, 23
field-level replication, 30
Local Area Network, 27
local replicas, 19
logging, 34
mirroring data, 23
mobile users, 22
one server in one location, 18
proof-of-concept prototype, 45
read-only data, 19
real-time access, 24
redundant data, 37
replication over WAN, 30
security, 21, 22, 23
setup, 28
several locations, 17, 28
several servers, 17, 27
several servers in several locations, 30
signaling required updates, 39
staging server, 23
switching seamlessly between databases, 38
up-to-date data requirements, 27
use of roles, 20
user feedback, 33
using scenarios, 40
Wide Area Network, 28
Application development features limits, 307
Application Performance Issues, 164
Application Programming Interface (API)
C API, 147
Domino Web Server Application Programming Interface (DSAPI), 147
Application setup, 28
Archiving, 28, 148, 158
keeping database size down, 13
maintaining links to archived documents, 149
Arrays, 61, 64, 66
AS/400, 171
monitoring tools, 171
Performance Explorer, 171
Asynchronous processing, 85
Attachments, 101. See Also Forms
Auto Refresh fields, 194
AutoReload, 74

B
Back-end Connectivity. See
Enterprise integration
Bandwidth, 29
dial-up networking, 180
simulating limited bandwidth, 46
Billing Task, 175
Business process, 40
Buttons, 96

C
CACHE.DSK, 81, 137, 138, 180
InitialCacheQuota, 139
location, 139
Caching
$CacheOptions, 360
$CacheValid, 360
@Db lookups, 56
groups and script libraries, 82
CACHE.DSK, 81, 138
cleaning server to test, 180
client-side caching (Web), 355
database cache on server, 31
design considerations, 31
design summary not cached at client, 76
DominoTraceCmdCache, 360
effect of Evaluate statement, 56
field and hotspot formulas, 93
formula analyzer, 357
hidden frame, 109
images, 137
inspecting Notes Remote Procedure Calls (NRPCs), 140
pages served by the Domino Web server, 355
script libraries, 81
subforms, 90
view actions, 122
Web browser, 108
Web command cache statistics, 361
Capacity bottlenecks, 40 extending, 9
Case study compact reference databases, 227
designing for large data volume and large user population, 205
Centralization, 8
Client. See Notes client; Web browser Client_Clock, 191, 196
Clustering, 11, 154
ICM, 11
Code profiling, 197
coding example, 198
Code Reuse via %INCLUDE, 83 via agents, 80 via LotusScript, 81 via subforms, 80
Command Cache, 355
COMPACT task, 126
Computed fields, 100
Computed for display fields, 57
Computed when composed fields, 60
Concurrent updates, 12 check in/check out, 12 controlling with Lotus Sametime, 12
Console command show dbs, 164, 175, 187
Constructing Your Test Environment cleaning the environment, 180
Cookies, 108, 180
CORBA, 146

Data. See Subsets of data subsets in applications, 15
Data age check, 13
Database access control requirements, 10 activity, 173 application examples, 14 application with several databases, 8 compact size, 228 compaction, 151 extending capacity, 9 improving response time, 9 internals (Domino R5.0), 151 limits for titles and names, 308 local replica, 19 overcoming limits, 9 sharing of data and/or functionality, 10 single database application, 8 size, archiving, 13 splitting up applications, 11 supporting mobile users, 10 supporting more users, 10 Database open, 73, 186 showing db open, 187 Database properties, 154 accessed (In this file), 158 allow more fields in database, 160 display images after loading, 156 document table bitmap optimization, 157 headline monitoring, 159 prevent overwriting deleted data, 158 prevent the use of stored forms, 155 size of $Revisions fields, 160 size of $UpdatedBy fields, 159 specialized response hierarchy, 158 unread marks, 156
Databases special, 16
DECS. See Domino Enterprise Connection Services Deferred processing, 33
Design elements. See Domino design elements
Design Synopsis, 183
third party tools, 184 DESKTOP.DSK, 115 Dial Up Networking, 168, 180 Do Loops, 60 DocumentUniqueID, 36 Domino design elements, 87 agents, 143 buttons, 96 CACHE.DSK, 138 forms, 88 framesets, 131, 132 outlines, 135 pages, 130 profile documents, 106 resources, 136 shared actions, 138
subforms, 89
tables, 91
viewing all elements using NotesPeek, 184 views, 88, 109 Domino Enterprise Connection (DECS), 153 Domino Enterprise Connection Services (DECS), 24 Domino extension manager, 147 Domino Global Workbench, 96 Domino Log, 174, 191
Domino Object Model, 69, 70, 198.
See Also Domino design elements; Domino Objects Domino Objects, 73 append item option, 79 AutoReload option, 74 back-end classes, 73 disabling autoreload, 74 extended syntax, 76 GetAllDocumentsByKey, 77 GetDocumentByKey, 77 GetDocumentByUNID, 76 lazy initialization, 71 NotesDocument, 74 object reuse, 72 opening a database, 73 reuse, 73 searching, 75 UnprocessedDocuments, 79 using AppendToTextList, 77 using columnvalues, 74 Domino OffLine Services (DOLS), 19 Domino Server Performance, 172 monitoring tools, 173, 174, 175, 176 DominoAnalyzeFormulas, 357 Double Byte Character Set support, 206 Dynamic lookups, 37

E
Encryption, 22, 150
Enforce Consistent ACL, 22 roles, 20
MQ Series messaging, 207
real-time access, 24
runOnServer method, 24
servlets, 25, 146
staging server, 23
Enterprise Solution Builder (ESB), 25
Environment Variables, 68
ESB. See Lotus Enterprise Builder
Events
- calculations per field type, 58
- combine field formulas, 104
PostModeChange, 59
Execution Control List (ECL), 183, 191
Extended syntax
- Domino objects, 76
External Programs, 84
Feedback, 33
Fields, 93, 99
- $Return field, 105
- CacheOptions, 360
- CacheValid, 360
- Revisions, 47
- Revisions field, 159
- UpdatedBy, 47
- UpdatedBy field, 159
- WebQuerySave, 191
Authors field, 12, 102, 208
basics, 99
caching, 93
calculations per event, 58
- combine field formulas into events, 104
computed field, 99
computed for display, 57
computed for display field, 103
computed when composed, 60, 99
DisplayInView indicator, 239
effects of a computed field, 100
encrypted fields, 120
HTML field, 102
limits, 307
number on form, 99
Reader Names field, 216
readers field, 19, 79, 102, 113, 128, 208
refresh, 104
rich text field, 101
shared fields, 100
summary attribute, 99, 128
validation with JavaScript, 69
- with HTTP prefix, 103
For loops, 60
Forall, 61
Forms, 88. See also Fields
- automatic field refresh, 104
- caching field and hotspot formulas, 93
- controlled sections, 95
defragmenting, 106
delays while navigating, 191
dialog boxes, 93
fields, 98
form formula, 127
graphics, 88
Hide When conditions, 94
JavaScript, 94
layout regions, 92
limits, 307
lookups, 95
number of fields, 99
pop-ups, 105
progressive disclosure, 93
rich text, 101
sections, 92
stored forms, 155
subforms, 89
tables, 91
time-date and numerical calculations, 94
top performance items, 194
troubleshooting, 190
Formula Analyzer, 93, 356
Formula language, 52, 53, 94
- AllChildren, 159
- AllDescendants, 159
- ClientType, 55
- DbColumn, 56, 95, 97
- DbLookup, 56, 95
- Environment, 236
- IsDocBeingEdited, 59, 96
- IsDocBeingSaved, 59
- IsDocumentBeingSaved, 96
- Now, 120, 128, 235
- Prompt, 190
- SetField, 98
- Subset, 97
- Text, 238
- Today, 120, 128, 193, 235
- Unique, 36
- UserName, 112, 120
- UserRoles, 55
caching @functions for Web browsers, 356
hide-when formula, 55, 95
IsDocBeingLoaded, 57
using hidden views, 57
versus LotusScript, 333
Framesets, 131, 132
caching information, 108
switching between databases, 132
FTSearch, 57, 75
G
GetAllDocumentsByKey limits, 308
GetDocumentByKey limits, 308
GetDocumentByUNID, 76
GetView, 75, 76
Graphics, 88, 92, 136, 142, 143
aspect ratio, 89
scaling, 89
H
Headlines, 141
- monitoring, 159
Hidden views, 57
Hide When, 55, 94, 194
HPUX, 172
- monitoring tools, 172
HTML, 54
HTTP task, 174
HTTP transactions
- troubleshooting, 174
I
If, 61
Images, 136. See also Graphics
Indexing comparisons for views, 309
Integer, 62, 67
Internet Cluster Manager (ICM ), 11
iostat, 172
iostat-x, 172
IPTrace, 181
J
Java, 53, 68
- applets, 146
- servlets, 25, 146
- using Java classes, 68
- using Recycle function, 69
JavaScript, 54, 69
- avoiding round-trips, 69
- caching information, 109
Index 383
onBlur event, 104
using, 94

K
Keys
@Unique, 36
DocumentUniqueID, 36
relating data, 35

L
LANs, 162, 195
Layout regions, 92
Lazy Initialization, 70
LEI. See Lotus Enterprise Integrator
Lightweight databases
case study, 227
Limiting bandwidth, 46
Limits
Application development features, 307
database size, 153
overcoming, 9
UNK table, 9
Load distribution
clustering, 11
replication, 12
splitting application into several databases, 13
Load tools
third party, 233
Local Area Network (LAN), 27
Local replicas, 19
Domino OffLine Services, 19
Local replication, 13
Locations, multiple
design considerations, 17
Log, 173, 174
Logging
of background processing, 34
Lookups
@DbColumn, 56
@DbCommand, 56
@DbLookup, 56
@IsDocBeingEdited, 96
@IsDocBeingLoaded, 57
@IsDocumentBeingSaved, 96
alternate search methods, 57
caching, 56
design considerations, 37
finetuning, 95
hidden views, 57
LotusScript, 96
monitoring via ECL settings, 183
using profile documents, 107
using the Evaluate statement, 56
Lotus Connector classes, 24
Lotus Enterprise Integrator (LEI), 24
LotusScript, 52, 53, 60, 94
%INCLUDE, 83
automatic conversion, 67
comparing data, 67
creating loops, 60
events, 104
FTSearch method, 75, 96
If statements, 61
integer division, 62
LotusScript Extensions (LSX), 52
QuerySave event, 104
reading files, 63
redim, 66
script library, 81, 122
search method, 57, 75, 96
variant types, 62
versus Formulas, 333
WebQuerySave event, 105
LotusScript Extensions (LSX), 52
lsol, 172

M
Measuring test results, 48
Memory leak, 170
Methodology, 39
Software Performance Engineering, 44
Use Case Model, 41
Mobile users, 10, 14, 21, 38, 207
compact mobile versions of big databases, 16
Domino OffLine Services, 19
security, 22
Monitoring tools
third party, 231
Multidatabase applications techniques, 35

N
Netstat, 166, 172
Network performance, 162, 164
dial-up networking, 168
netstat, 166
ping example, 165
ping utility, 164
tracert, 167
Notes client
@ClientType, 55
bandwidth requirements, 29
saving and closing a document, 191
testing an application, 185
Notes Global Designer, 96
Notes Log, 173, 188
Notes Remote Procedure Calls (NRPCs), 28, 48, 92, 182, 191, 195, 196
enabling, 196
examining output, 197
NOTES.INI, 106, 126, 152
CACHE, 139
Client_Clock, 196
Debug_Console, 196
Debug_Outfile, 196, 197
DominoAnalyzeFormulas, 357
DominoDefaultCacheValid, 360
DominoTraceCmdCache, 360
environment variables, 68
InitialCacheQuota, 139
limits, 308
NSF_Buffer_Pool_Size, 31
NSF_DbCache_MAXentries, 31
Server_Clock, 182
VIEW_REBUILD_DIR, 152
NotesDocument class, 74
NotesPeek, 141, 184
NotesUIDocument class, 74

O
Object oriented design, 84
Object reuse, 72, 73, 131
locating documents, 77
OLE, 52
objects, 101
On Disk Structure (ODS), 141, 151, 152
OS/2, 171
monitoring tools, 171
using IPTrace, 181
Outlines, 135
performance considerations, 135

P
Pages, 130
Performance
application factors, 4
application usage, 162
as responsiveness, 2
as throughput, 2
factors affecting performance, 161
guidelines for views, 126

384 Performance Considerations for Domino Applications
Subsets of data
    non-replica copies of databases, 15
    readers fields, 15
    selective replication, 15
SVMon, 171
Switching between databases
    framesets, 132
    Switching seamlessly between databases, 38
System Management Facility, 172

Tables, 194
    considerations, 91
    Domino R5.0 enhancements, 92
    limits, 308
    size, 91
    tabbed tables, 92
    timed tables, 92
Task Manager, 170
Test Environment, 179
    cleaning the environment, 180
    tools & techniques, 180
Third Party Tools, 176, 177
    monitoring tools, 231
    stress testing and load tools, 233
Time/date views, 235
    using agents, 239
Tprof, 171
Tracert, 167
Transaction Logging, 152
Troubleshooting, 161, 164
    agents, 191
    application performance, 178
    client performance, 177
    Domino server performance, 172
    forms, 190
    memory leaks, 170
    network performance, 164
    opening database, 186
    server operating system, 168
    Server.Load, 176
    usage patterns, 177
    views, 187

Unique Key table. See UNK table
UNIX, 181
    using IPTrace, 181

UNK table, 9
    more fields in database, 160
UnprocessedDocuments, 79
Unread marks, 127, 156, 193
UPDALL task, 124, 125, 126, 152
UPDATE task, 123, 124, 125, 126
URL requests
    ?CreateDocument, 352
    ?EditDocument, 351
    ?OpenDocument, 351
    ?OpenForm, 352
    ?SaveDocument, 352
    candidates for command cache, 356
Usage patterns, 177
Use Case Model, 41

Views, 109
    accessing information, 189
    actions, 122
    categorization, 120, 127, 129
    column design, 120
    column formula, 128
    columns, 127
    ColumnValues, 74
    data displayed, 122
    discarding index, 124
    DisplayInView indicator, 239
    dynamic sorting, 121
    formula, 122, 128
    getNextDocument, 75
    GetNthDocument, 75
    GetView, 76
    index size in notes log, 173
    indexing comparisons, 309
    internal information, 109
    limits, 307
    opening, 188
    performance guidelines, 126
    personal on first use views, 110, 112, 116
    private views, 110, 117
    refresh, 123
    removing indexes, 180
    response hierarchy, 158
    SELECT statement, 128
    selection formula, 120, 128, 157
    single category view, 114
    size, 9
    size in LOG.NSF, 110
    sorting, 121
    time/date views, 235

WANs, 162
Web agents, 192
Web browser, 89, 92, 93, 94, 102, 105, 143
    @ClientType, 55
    bandwidth requirements, 29
    caching, 108
    client-side caching, 355
    cookies, 108, 180
    Domino images, 142
    laying out pages, 141
    linking identity, 108
    number of lines in a view, 130
    profile information, 107
    saving and closing a document, 191
    servlets, 25
    temporary internet files, 180
    testing an application, 185
Web browsers
    how Domino handles document URL requests, 351
WebQuerySave, 191
While loops, 60
Windows NT, 169, 185
    monitoring tools, 169, 170

386 Performance Considerations for Domino Applications
IBM Redbook Evaluation

Performance Considerations for Domino Applications SG24-5602-00

Your feedback is very important to help us maintain the quality of ITSO redbooks.

Please complete this questionnaire and return it using one of the following methods:

- Use the online evaluation form at http://www.redbooks.ibm.com/
- Fax it to: USA International Access Code +1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?

___Customer  ___Business Partner  ___Solution Developer  ___IBM employee
___None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction  ___________________

Please answer the following questions:

Was this redbook published in time for your needs?  Yes _________ No _________

If no, please explain:

____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________

What other redbooks would you like to see published?

____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________

© Copyright IBM Corp. 2000

387
Performance Considerations for Domino Applications

Søren Peter Nielsen, Alessandro Toscano Mariscal, Umasuthan Ramakrishnan, Walt Simons, Chris Thornton