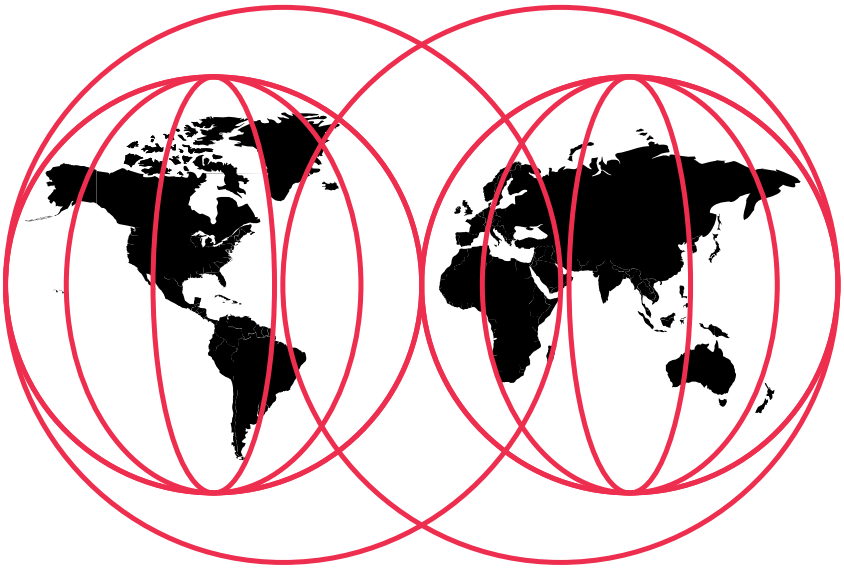
Lotus®

IBM

Lotus Domino Release 5.0: A Developer's Handbook

Fiona Collins, David Morrison, Søren Peter Nielsen, Sami Serpola, Reinhold Strobl



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5331-01

SG24-5331-01

International Technical Support Organization

Lotus Domino Release 5.0: A Developer's Handbook

September 1999

Take Note!
Before using this information and the product it supports, be sure to read the general information in the Special Notices section at the back of this book.

Second Edition (September 1999)

This edition applies to Lotus Domino Release 5.0.1.

Comments may be addressed to: IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© International Business Machines Corporation 1999. All rights reserved.

Note to U.S. Government Users: Documentation related to restricted rights. Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Preface xiii
The Team That Wrote This Redbook xiii
Comments Welcome xv
1 What is Lotus Domino? 1
Domino R5.0 Server 1
Domino Mail Server 1
Domino Application Server 1
Domino Enterprise Server 2
Services Offered by Domino Servers 2
Clients for Domino R5.0 4
Notes R5.0 4
Domino Administrator R5.0 6
Summary 8
2 Lotus Domino Designer: An Overview 9
Managing Your Workspace 10
The Design Pane 11
The Window Tabs 12
The Launch Buttons 12
The Programmer's Pane 13
The Objects View 13
The Reference View 14
The Script Area 14
Domino Design Elements 15
The Domino Database 15
Forms 15
Subforms 15
Views 16
Fields 16
Shared Fields 16
Tables 16
Action Buttons 16
New Features of Domino Designer 17
Outline Designer 20
Frameset Designer 21
Page Designer 22
Domino User Interface Applets 22
New Rapid Development Capabilities in IDE 23
Industry Standards Support 23
Multi-Client Applications Support 24
Multilingual Applications Support 24
Easy Access to Enterprise Data and Applications 25
3 Domino Design Elements: Basics 27
Domino Databases 27
Creating a Database 27
Changing the Database Properties 33
Using Design Synopsis 40
Summary 42
4 Forms 43
Forms 43
Specifying Form Properties 43
Giving the Form a Title 52
Objects View 53
Creating a Field 56
Performing a Test Run 57
Sharing and Reusing a Field 60

Field Properties 63
Rich Text Field (RTF) Applet 68
Using the $$Return Field 70
Field Events 74
Sharing Design Elements With Subforms 75
Subform Properties 77
Removing Subforms 77
Computed Subforms 78
Displaying a Different Form to Web Users and Notes Users 79
Working With Layout Regions 80
Creating a Layout Region 80
Working With Collapsible Sections 82
Creating a Collapsible Section 82
Using Tables 82
Create Tables Within Tables 84
Merge and Split Cells 84
Table Properties 86
Embedded Elements 92
Embedded Navigators 93
Embedded Date Picker 93
Embedded Outline Control 93
Embedded View 93
Embedded Group Scheduling Control 94
Embedded Folder Pane 94
Embedded File Upload Control 96
Other Features of Forms 97
Horizontal Rules 97
Computed Text 97
Buttons, Action Bar Buttons, and Hotspots 99
Images Within Forms 102
Copying Images 102
Importing Pictures 102
Using Image Resource 103
Alternate Text 103
Using CGI Variables 103
Table of CGI Variables Supported by Domino 103
Using a Field to Capture CGI Variables . . . 105
Using a LotusScript Agent to Capture CGI Variables 106
Summary 106

5 Views, Folders, and Navigators . 107

What is a View? 107
What is a Folder? 107
What is a Navigator? 107
Creating Views 107
Working With View Properties 111
Editing View Columns 114
Creating Calendar Views 117
Summary 118
Shared and Private Views 120
Shared Views 120
Shared, Personal-on-First-Use Views 120
Personal Views 121
Creating a Button on the Action Bar 121
Working With Views as a Developer 123
Views and the Web 125
Using the Default Display 125
Using Java Applets for Display 127
Using HTML Formatting for Views 128
Hints and Tips on Designing Views 132
Naming Views 132
Overview of Styles 133
Identifying Unread Documents 136
Using Categories in Views 136
Creating an All by Category View 137
Presenting Views to Users 138

Embedding Views 138
Hiding Views 140
Formatting Date and Time Columns 141
Formatting Numbers in Columns 142
Indenting Response Documents 142
Sorting Documents in Views 143
Exporting and Importing Views 145
Designing a Folder 148
Managing Access to Views and Folders 149
Creating a Read Access List 149
Creating a Write Access List 150
Using Navigators 150
Navigator Objects 151
Navigator Actions 151
Creating a Navigator 152
Adding an Action to a Navigator Object . . 154
Adding an Action Using @Functions or LotusScript 154
Testing a Navigator 154
Including a Navigator in the View Menu . . 155
Summary 156
6 New R5.0 Design Elements 157
Pages 157
Creating a New Page 158
Specifying Page Properties 159
Using Pages 161
Launching Pages 162
Framesets 163
Creating a Frameset 163
Changing the Layout of a Frameset 166
Outlines 167
Creating a New Outline 169
Embedded Outline 170
Resources 173
Images 173
Applets 175
Shared Fields 175
Script Libraries 176
Other 176
Summary 176
7 Agents 177
About Agents 177
Access Control 177
Restricted and Unrestricted Agents 178
Creating an Agent 179
Naming the Agent 179
Scheduling the Agent 180
Selecting Documents to be Processed 182
Specifying What the Agent Should Do 183
Displaying the Agent Pop-up Menu 186
Testing an Agent 187
Testing an Agent During Development . . . 187
Testing an Agent Before Copying it to a Live Database 187
Checking the Agent Log 188
Debugging Agents 188
Disabling Scheduled Agents 189
To Disable Individual Agents 189
To Disable All Automated Agents in a Database 189
Troubleshooting Agents 189
Agents and the Web 192
Running Multiple Instances of an Agent . . 193
WebQueryOpen and WebQuerySave Agents 193
Using the @URLOpen Command to Call Agents 194
Using a LotusScript Agent to Capture CGI Variables 194

Creating a Web Page Counter 195
Using Agents — Advanced Topics 197
Summary 199
8 Securing Your Application 201
Controlling Access to Domino Data 201
Overview of Domino Security Architecture 202
Using the Access Control List to Control Access to an Application 204
Setting Up and Refining the ACL 206
Roles in the ACL 210
Enforce Consistent ACL 211
Maximum Internet Name and Password Access 212
Changing the ACL Programmatically 212
Using Outline Control to Hide Parts of an Application 214
Using Directory Link Files to Control Access to an Application 214
Controlling Access to Views and Forms 215
Controlling Access to Views 215
Controlling Access to Forms 216
Preventing Printing, Forwarding, and Copying of Documents 219
Controlling Access to Documents 219
Read Access 220
Editor Access 222
Combining Readers and Authors Fields . . 223
Controlled Access Sections 224
Use of Hide-When Formulas 225
Using Encryption for Field Security 225
Authentication on the Web 227
HTTP Basic Authentication 227
Secure Sockets Layer (SSL) 228
Domino and SSL 230
When to Use Internet Security 230
Defining Web Users 231
Programming Considerations 232
Using @UserRoles 232
Using @UserName 232
Using @ClientType 233
Using @UserNameList 233
Password Field 234
Controlling if Users Paste Documents into Database 234
Hiding the Design of a Database 234
Other Security Options and Considerations . . 234
Using Signatures for Security 234
Access Control for HTML and Other Files . 237
APIs for Customized Authentication, Encryption, and Signing 239
Backup and Restore 240
Developing a Plan for Securing Your Application 240
Key Design Issues 242
Distinguishing True Security Features 244
Summary 245
9 Searching 247
Adding Search Capabilities to Your Web Site . 247
Search-Related URLs 248
Search View URLs 248
Search Site URLs 249
Full Text Indexing 249
Customizing Search and Result Forms 251
Creating a TeamRoom Search 252
Customizing Search Result Forms 256
Search Site Databases 258
Creating a Search Site Database 258
Multi-Database Full Text Indexes 260
How Users Search Using a Search Site Database 261

Customizing Search Site Result Forms 261
Summary 262
10 Programming for Domino 263
Programming in Notes 263
Simple Actions 263
Formula Language 264
LotusScript 267
The Domino Object Model 269
Domino Front-End UI Objects 270
Domino Back-End Objects 270
Object Hierarchy 273
Using Domino Objects from LotusScript . . 274
Understanding Front-end and Back-end Classes 276
Using Domino Objects From Java 278
Programming With LotusScript 278
The Event Model 278
Event Type and Sequence 280
Action Object 284
Using LotusScript in Web Applications . . . 285
How Scripts and Formulas Are Executed . . 285
LotusScript Programming Tips and Considerations 290
General Suggestions 290
Use Consistent Variable Names 291
Reserved Fields 292
Using Script Libraries 293
Using a Template Database 295
Catching Errors at Compile Time 297
Improving Form Performance 297
When to Use Formulas and LotusScript . . . 298
Using the Evaluate Function to Combine LotusScript and Formulas 300
Making Field Value Changes Effective 301
Using Validation Formulas and QuerySave 302
Working With a Rich Text Item and Rich Text Style 302
New Domino Objects in Domino R5.0 306
Error Handling 312
Using the Debugger 313
How to Enable the Debugger 314
Tracing Programs Without a Debugger . . . 318
Using JavaScript 320
Using JavaScript in Domino Design Elements 321
Where JavaScript Gives You Access in Domino 323
Mapping Domino Objects to the Document Object Model 323
Examples of Adding JavaScript to Forms . . 328
Cookies and Domino 334
LiveConnect — JavaScript Access to the Domino Classes 337
Accessing an Applet From JavaScript 337
Accessing Java/CORBA Applets via LiveConnect 338
External Tools 341
The API for Domino and Notes 341
Summary 342
11 Advanced Domino Programming 343
Java 343
Agents, Applets, Applications, and Servlets 345
Adding CORBA to the Picture 346
Benefits of Using CORBA 347
How and When to Use CORBA 347
Compiling and Running a Java Program . . 348
Runtime Requirements 349

Remote Calls to lotus.domino Package ... 349
Applet Calls to lotus.domino Package ... 351
Creating a Java Agent ... 353
CORBA/IIOP ... 357
Benefits of Using CORBA ... 357
CORBA Architecture ... 358
Internet Inter-ORB Protocol (IIOP) ... 358
CORBA and Domino ... 359
Coding the CORBA Applet ... 361
Domino and OLE Automation ... 367
Accessing the Domino Object Model Using OLE Automation ... 367
Sending Information From Excel Using Domino ... 369
More Examples ... 372
Runtime Errors and Debugging ... 374
Accessing Other Applications From Notes Using OLE Automation ... 376
Classes of OLE Objects ... 377
LotusScript Functions and Methods to Use for OLE Applications ... 378
Embedding OLE Objects ... 380
Using OLE Automation Without Embedding ... 385
Troubleshooting ... 387
Considerations Before Using OLE Embedding/Automation ... 390
The LSX Toolkit ... 391
What Is an LSX? ... 391
Using an LSX ... 392
Using the LSX Toolkit ... 392
Overview ... 392
What the LSX Toolkit Contains ... 393
Creating an LSX ... 395
The LSX Development Process ... 396
Using the LSX Wizard ... 396
Compiling the LSX ... 406
Using LSX Data Types ... 406
Using Data Type Descriptions ... 408
Using LotusScript System Services ... 409
Testing an LSX ... 413
The LSXTEST Tool ... 413
The LSXRUN Tool ... 414
Deploying an LSX ... 414
The LSX Runtime Environment ... 414
LSX Installation ... 415
LSX Registration ... 415
Architecture and Advanced Design ... 416
Understanding the C++ LSX Class Framework ... 419
LSX Design Decisions ... 423
Accessing LSX Class Method Arguments ... 425
Accessing LSX Class Property Arguments ... 427
Summary ... 428
12 Development Dos and Don'ts ... 429
Before You Write a Single Line of Code ... 429
Lesson 1 - Getting a Business Sponsor ... 429
Lesson 2 - Communication ... 429
Lesson 3 - Ensure That There is a Real Business Need ... 429
Lesson 4 - Understanding the Deliverables ... 430
Lesson 5 - Planning Your Application ... 430
Lesson 6 - Even Domino Has Limitations ... 430
Lesson 7 - Project Scope Creep ... 430
Creating Your Application ... 431
Lesson 1 - Use Professional Graphics ... 431
Lesson 2 - Design the Outlook as Thoroughly as Possible ... 431
Lesson 3 - Try to Standardize on a Web Browser ... 431
Lesson 4 - Comment Your Code ... 431

Lesson 5 - Try to Avoid Hard Coding 431
Lesson 6 - Use the Appropriate Design Elements and Events 432
Lesson 7 - Provide Meaningful Error Messages 432
Lesson 8 - Document Your Application . . . 432
Lesson 9 - Be Aware of Performance Options 432
Handing Over Your Application to Production 434
Lesson 1 - Perform Quality Assurance of the Application 434
Lesson 2 - Supply an Installation Test Verification Case 434
Lesson 3 - Document the Application Requirements 435
When Your Application is Deployed in Production 435
Lesson 1 - Define a Maintenance Server . . . 435
Lesson 2 - Get Feedback From Your Users . 435
Summary 436

13 Introducing DECS and Database Connectivity 437
Installing and Running DECS 437
Supported Data Sources 438
Setting Up Connectivity to DB2 439
Testing Connections With LCTEST 440
Running LCTEST 440
Configuring DECS 441
Using the Connection Server Administrator . . 441
The DECS Administrator Navigator 442
Creating a Simple RealTime DB2 Connection 443
Creating the Lotus Notes Database 444
Creating the Activity Document 446
Populating the Lotus Notes Database With Key Data 448
Running the DECS Activity 448
Accessing More Than One Table 449
LotusScript Extension for Domino Connectors 452
Reading Data From an External Source into a Notes Form 452
Updating Data From Notes to an External Source 463
Multi-Value Data 467
Summary 470

14 Using Other Database Connectivity Tools 471
NotesSQL 471
What is ODBC? 472
SQL Grammar Conformance Level of NotesSQL 473
Technical Advantages 477
When to Use NotesSQL 478
Hardware and Software Requirements . . . 478
Installing NotesSQL 479
Adding a Lotus Notes Data Source 480
Connecting to a Data Source 484
Mapping Notes Names to SQL Names 485
The Universal Relation 485
Using SQL Tables from Derived Forms and Views 486
Column, Index, Table, and View Names . . 488
Using Notes Views as Indexes 489
View Column Definitions 490
Data Types 490
Summary of Supported ODBC SQL Grammar 493
Known Limitations or Problems With NotesSQL 497
Example: Accessing Notes From Visual Basic 498

Example: Using NotesSQL With Microsoft Active Server Pages (ASP) 502
Domino Driver for JDBC 506
What is JDBC? 506
Domino Driver for JDBC Data Types 507
Creating a Connection 509
Using IBM VisualAge for Java Version 1.0 . 516
Using IBM WebSphere Application Server . 516
To Configure and Run Domino Driver for JDBC Using IBM WebSphere 516
LotusScript Data Objects and ODBC 518
Data Resource Access 518
Database Access Facilities 518
What is ODBC? 519
LotusScript:DataObject (LS:DO) 521
What Is LS:DO? 521
Concepts 522
Architecture 523
When to Use LS:DO 524
Differences Between LS:DO and ODBC 524
Programming Environment 524
Functionality 525
Performance 525
Software Requirements 526
Registering ODBC Data Sources 527
USELSX Statement to Enable LS:DO 528
Mapping Data Types Between RDB and Notes DB 529
How to Trace and Debug LS:DO 529
LS:DO Class Library 533
Server Side Processing for Web Applications . 546
How Does This Work? 550
Digging Deeper 551
Running Multiple Instances of an Agent . . 554
Using @DB Functions to Access Other Databases Through ODBC 555
When to Use 555
How to Use @DB Functions 556
@DBColumn 556
@DBLookup 557
@DBCommand 557
Summary 558
15 Domino Global WorkBench 559
Concepts, Databases, and Tools in Domino Global WorkBench 562
Domino Global WorkBench Databases . . . 563
What is Tagging? 563
The WorkBench 564
The Standalone Tagger 564
The Synchronizer 565
Localizing an Application 565
Setting up the Project and Tagging the Database 566
Running Checks Using Pseudo-Translation 577
Preparing the Glossary for Translation . . . 584
Translation 586
Building Language Databases 589
Skipping Terms During Tagging Versus Marking Terms as “Do Not Translate” . 591
The Difference Between Updating and Rebuilding 592
Preparing Your Database — Tips for Developers 593
In General 593
Use Aliases 597
Temporary Variables 598
Use Formulas Instead of Simple Actions . . 599
Hide-When Formulas 599

Exclude Paragraphs From Translation Using the DO_NOT_TAG Style 599
Avoid Shared/Personal on First Use Folders or Views 600
Concatenated Sentences 600
Handling Translatable Lists 602
Keywords 602
Fonts 604
Pay Attention to Length Limitations 604
LotusScript 605
Keep Translators Informed 606
Preparing an Existing Database 607
Make a Backup 607
Create a Design Synopsis 607
Summary 610
16 Domino Ad Hoc Workflow 611
What is Workflow? 611
Workflow Design Considerations 612
Creating a Database Using the Approval Cycle Template 613
Using the Workflow Document 617
Working With the Request 618
Approval Cycle Database: Design 619
How Does a Form Flow? 619
How Is the Approval Cycle Database Organized? 619
A Closer Look at the ApprovalLogic Subform . 623
The Major Fields 624
Creating a New Request 624
Submitting a New Form for Approval 626
Approving a Request 631
Denying a Request 633
Summary 634
Appendix A Domino URLs 635
Domino URL Command Syntax 635
Opening Servers, Databases, and Views . . 637
Opening Framesets, Pages, Forms, Navigators, and Agents 639
Opening, Editing, and Deleting Documents 641
Opening an Anchor Link 642
Opening Documents by Key 642
Advantages of Using Keys Instead of Universal ID 643
Searching for Text with Domino Search URLs 645
Appendix B Shortcuts 647
Workspace Keys 647
Function Keys 648
Dialog Boxes 649
Keys for Editing Documents or Designing Domino Objects 649
Appendix C CORBA Internals 651
CORBA Objects 655
Special Notices 657
Additional Web Material 661
How to Get the Web Material 661
Related Publications 663
International Technical Support Organization Publications 663
Other Lotus-Related ITSO Publications 664
Redbooks on CD-ROMs 665
How to Get ITSO Redbooks 667
IBM Intranet for Employees 667
Index 671
ITSO Redbook Evaluation 681

Preface

This document describes how to develop applications using Lotus Domino Release 5.0.

The earlier chapters in the book introduce some of the basic design elements of a Domino database such as the database itself, forms, views, folders, and navigators. Those readers that are familiar with developing applications using earlier releases of Lotus Domino may want to move straight to the chapters that introduce the new features in Domino 5.0, such as framesets, pages, outlines, resources and headlines.

The book then describes how to use the programming languages available in Domino Designer; the formula language, LotusScript, JavaScript, Java, IIOP and CORBA, C++ API and the LSX toolkit. Some good practical advice on Domino development do's and don'ts is then outlined before discussing the Domino Enterprise Connectors (DECS) which are used to access data from external data sources, along with chapters describing how to use NotesSQL, ODBC, and the LotusScript Data Object (LSDO).

The book then explains how to create multilingual international applications for Domino and the Web using the Global Workbench tool.

Finally, the topic of creating workflow applications with the Approval Cycle template is covered with an in-depth look at the LotusScript in the ApprovalLogic subform.

This redbook was written for Domino technical specialists and programmers, customers, IBM Business Partners, and the IBM and Lotus community who need a good technical understanding of how to develop applications using Lotus Domino R5.0.

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.

Fiona Collins is an International Technical Support Specialist for Notes and Domino at the International Technical Support Organization Center at Lotus Development, Cambridge, Massachusetts. She manages projects whose objective it is to produce redbooks on all areas of Domino. Before joining the

ITSO in 1996, she provided technical support for Lotus Notes/Domino and the AS/400, for Lotus and IBM in the UK.

David Morrison is a senior Lotus Notes consultant working for IBM e-business services in the United Kingdom. David specializes in designing and developing large scale Internet and data integration applications based on Lotus Notes and Domino technology. David has over 10 years experience working in the IT industry, with five of these spent working with Lotus Notes and Domino.

Søren Peter Nielsen works as an IT Architect for IBM Global Services in Denmark with Domino development and Domino infrastructure. During his more than 15 years at IBM, Søren has worked with solutions for industries like Small and Medium Business, Banking, Insurance, News & Media, Manufacturing, Public Sector, Travel & Transportation, and Petroleum. Søren is a Certified Lotus Professional at the Principal level in Application Development and System Administration.

Sami Serpola works in IBM e-business Services in Finland as a Lotus Notes and Domino Application Development Consultant, designing and developing Lotus Notes and Domino solutions for IBM customers.

Reinhold Strobl works for the IBM Product Support Services Department in Vienna, Austria, where he provides specialist support to customers for Lotus Domino and support for other aspects of application development. Before joining the IBM support organization, Reinhold was a software designer and developer working out of an IBM software development laboratory.

A number of people have provided support and guidance. In particular, we would like to thank **Chris Reckling**, Product Manager for Domino Designer. In addition, we would like to thank the following people from Lotus Cambridge (unless otherwise noted):

- Stephen Auriemma
- Mike Bisacre
- Janet Bowers, Lotus Notes Application Center of Competence, IBM Boulder
- Martin Broekhuysen
- Paul Castiglione
- Shu Chen
- Bill de la Vega
- Gary Devendorf
- Carl Hero
- Bernadette Kelly

- Carl Kriger
- Thierry Mayeur
- Avshalom Neer
- Mary Peterson
- Moshe Rappoport, IBM Zurich
- Michael Roche
- Mark D Smith
- Barry Wand, IBM Global Services Notes Database team, IBM Austin
- David Watkins, e-business Services, IBM UK
- Lauren Wendel
- Stanley Wood, IBM Global Services Network Computing Office
- 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
What is Lotus Domino?

The Domino™ Server family is an integrated messaging and Web application software platform, for growing companies that need to improve customer responsiveness, and streamline their business processes.

Domino, the only solution built on an open, unified architecture, is trusted by the world's leading companies to deliver secure communication, collaboration and business applications. Domino R5.0 servers set a new standard for rich Internet messaging, ease of administration, and integration with back-end systems.

This chapter describes the Domino R5.0 Server Family, the services Domino R5.0 offers, and the clients for Domino R5.0.

Domino R5.0 Server

The Domino R5.0 Server is offered in different packages, to allow customers to pick the functionality that meets their current requirements and extends that functionality as their requirements change in the future. We will briefly describe the three Domino R5.0 servers below.

Domino Mail Server

Domino Mail Server is the newest member of the Lotus® Domino server family. Domino Mail combines support for the latest Internet mail standards with the advanced messaging capabilities and enterprise-scale reliability and performance of Lotus Domino. Its integrated, cross-platform services include Web access, group scheduling, collaborative workspaces, and newsgroups — all accessible from a Web browser or other standards-based client.

Domino Mail Server is used for messaging only. Customers that want to deploy their own applications on the Domino server should consider Domino Application Server or Domino Enterprise Server.

Domino Application Server

Domino Application Server is the leading integrated messaging and applications server. It delivers best-of-breed messaging as well as an open,

secure Web application platform. The server easily integrates back-end systems with front-end systems business processes.

This is the natural evolution of the Lotus Notes server from which Lotus Domino originates.

Domino Enterprise Server

Domino Enterprise Server is the server for customers requiring mission-critical, highly scalable deployments with uninterrupted access, and maximum performance under all conditions. It extends the functionality of Domino Mail and Domino Application Servers with high availability services such as partitioning, clustering, and billing.

This product was previously called Domino Advanced Services.

Services Offered by Domino Servers

Lotus Domino Servers offer a wide range of services. We will briefly describe the most important ones.

Object Store

Documents in a Domino database can contain any number of objects and data types, including text, rich text, numerical data, structured data, images, graphics, sound, video, file attachments, embedded objects, and Java™ and ActiveX applets. A built-in Full text search engine makes it easy to index and search documents. The object store also lets your Domino applications dynamically present information based on variables such as user identity, user preferences, user input, and time.

Directory

A single directory manages all resource directory information for server and network configuration, application management, and security. Domino includes user account synchronization between NT and Domino and is Light Weight Directory Access Protocol (LDAP) compliant. The directory is the foundation for easily managing and securing your Internet and intranet applications.

Security

The Domino security model provides user authentication, digital signatures, flexible access control, and encryption. Domino security enables you to extend your intranet applications to customers and business partners.

Replication

Bi-directional replication automatically distributes and synchronizes information and applications across geographically dispersed sites. Replication makes your business applications available to users around your company or around the world, regardless of time or location.

Messaging

An advanced client/server messaging system with built-in calendaring and scheduling enables individuals and groups to send and share information easily. Message transfer agents (MTAs) seamlessly extend the system to Simple Mail Transfer Protocol (SMTP)/Multipurpose Internet Mail Extension (MIME), x.400, and cc:Mail™ messaging environments. The Domino messaging service provides a single server supporting a variety of mail clients; Post Office Protocol V3 (POP3), Internet Message Access Protocol V4 (IMAP4), Message Application Programming Interface (MAPI), and Lotus Notes clients.

Workflow

A workflow engine distributes, routes, and tracks documents according to a process defined in your applications. Workflow enables you to coordinate and streamline critical business activities across an organization, and with customers, partners, and suppliers.

Agents

Agents enable you to automate frequently performed processes, eliminating tedious administration tasks, and speeding your business applications. Agents can be triggered by time or events in a business application. Agents can be run on Domino servers or Lotus Notes clients.

Development Environment

Domino Designer is general-purpose client software featuring an integrated development environment (IDE) that provides easy access to all features of the Domino server. We will focus on the features and functions of Domino Designer as well as the Domino Object Model in this redbook.

Domino Object Model

Domino offers a unified model for accessing its objects through back-end classes, whether you use LotusScript® or Java. This allows you to switch programming languages without having to learn new ways to program for Domino. Refer to Chapter 11: Advanced Domino Programming for more information on the Domino Object Model.

Live Integration with Enterprise Data

DECS (Domino Enterprise Connection Services) is part of the Domino Server. It is a Lotus developed technology, first shipped with NotesPump™ 2.5, that supplies an easy-to-use forms-based interface to achieve deep, integrated connectivity to external data from Domino applications. This allows developers to map fields in forms directly to fields in relational database tables, without storing any data within the Domino database.

Scalability and Reliability

Domino Enterprise Server enable you to cluster up to six Domino servers to provide both scalability and failover protection, to maximize the availability

of your groupware and messaging applications. Real-time replication technology keeps the clustered servers synchronized.

Note A Domino server is not the same as a file server. A file server provides access to shared resources such as printers and applications, and also manages network activity. Domino is an application-level server process that provides services necessary for the effective management of communications and applications.

Clients for Domino R5.0

Previous versions of Lotus Domino had one, all-purpose client that would be used by users, administrators, and application developers. With Lotus Domino Release 4.6, a special client for developers called Lotus Notes® Designer for Domino was introduced.

As a result of the strong focus on ease-of-use in the design of Lotus Domino R5.0, three individual clients are now available. They are:

- Notes™ R5.0: the users client
- Domino Administrator R5.0: the administrator's client
- Domino Designer R5.0: the developer's client

Most of the functionality in Lotus Domino can also be accessed from Web browsers. The Lotus Domino server includes a Web administration application. This redbook has its focus on Domino Designer R5.0. Chapter 2 is dedicated to an overview of Domino Designer R5.0. We will give a brief overview of the two other clients below.

Notes R5.0

Lotus Notes is the leading integrated e-mail and collaborative software for the Internet. In R5.0, Notes offers a more open, Web-like, customizable environment, so you can work the way you want, with all the power you expect from Notes.

The new Navigation Bar gives you instant forward, back, stop, and refresh actions, as well as access to search engines and the Web, from wherever you are in Notes.

Notes R5.0 has Bookmarks so that you can create links to Web pages, application views, documents, and forms for instant access. The new Window tabs allow you to keep track of multiple open windows, and navigate between them quickly. Notes R5.0 also has enhanced search capabilities, including search-by-form, fuzzy search, and the ability to perform a domain search — making information tracking quick and intuitive.

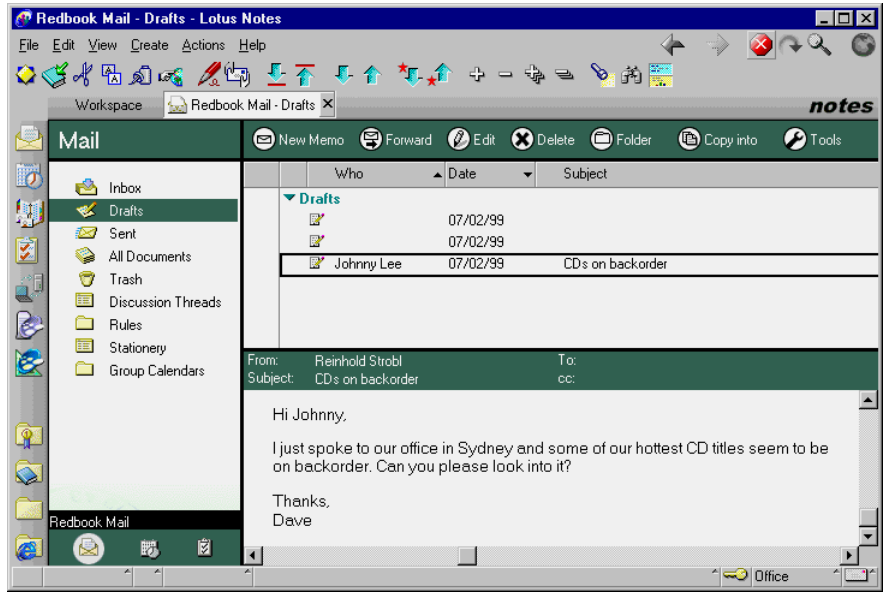
Headlines

With Notes R5.0, keeping on top of the latest and most important information is easy. The Notes R5.0 customizable Headlines page lets you select the information that you want to see first. You'll be alerted to important e-mail messages, tasks, or meetings for the day. You can even receive updates from intranet applications and view Web content dynamically — all from Headlines.

Each item on the Headlines page is a point of entry, so if you've received an urgent e-mail message, the full document is just a mouse click away. Plus, IT organizations can customize Headlines to feed corporate intranet information right onto the user's desktop.

Enhanced E-mail and Calendaring

The new mail and calendaring features in Notes R5.0 take the best of industry leading applications, such as cc:Mail and Lotus Organizer, and make them better. Notes R5.0 continues to build on its powerful integration by combining your mail and calendar preferences. You can preset preferences for every e-mail you send, including automatic spell check and sending all mail high priority with return receipt. Notes R5.0 mail also supports signature files, giving you a simple way to identify yourself and add pertinent information to every e-mail you send.



If you manage multiple calendars, Notes R5.0 now gives you the ability to view more than one calendar at a time. Choose to access multiple calendars for a "quick view" of who is available, or get more detail on another user's schedule if necessary. And when you need to take your calendar with you, Notes R5.0 gives you multiple print formats to choose from. Notes R5.0 streamlines the process of managing resources across domains, giving you greater access and control over conference rooms, AV equipment, and more.

Installation and Setup

Setting up Notes R5.0 is easy. Integration with dial-up networking entries means connections are created automatically for you as you install the software. Notes R5.0 also offers several preset configurations for even faster user setup. It's easier than ever for you to access your ISP mail account right from Notes. And if you're upgrading, you'll be ready to go as soon as you finish installation.

A Powerful Tool for Any Infrastructure

Current Notes users can continue to take advantage of all their Domino server-based applications like e-mail and calendaring. In addition, Notes R5.0 offers full standards support including POP, IMAP4, SMTP, LDAP v3, MIME, S/MIME, HTML, Java, Javascript, and X.509 certificates. So now, even users with non-Domino, standards-based back ends or that use ISP hosted mail at home will benefit from the power of Notes R5.0.

Domino Administrator R5.0

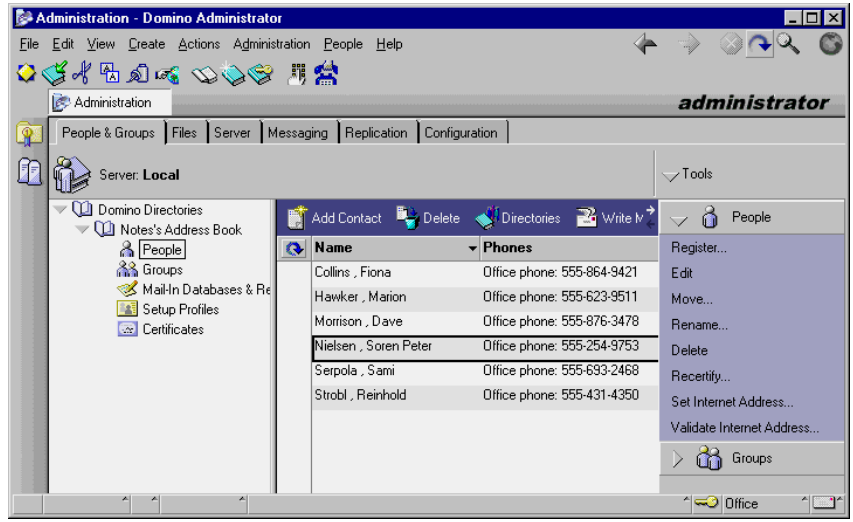
Domino Administrator R5.0 is a new, integrated administration control panel that provides simple, yet flexible administration. Administration benefits are universal whether you are a smaller company just getting started with Domino, or an enterprise managing a large-scale deployment, with thousands of people and applications.

The Domino Administrator R5.0 utilizes the Windows Explorer metaphor, providing an easy, intuitive interface and allowing drag-and-drop functionality for common administration tasks, such as moving a user. Important new server monitoring features now allow administrators to proactively monitor and manage an environment. Finally, administrators have the ability to centrally configure, manage, and enforce user desktop settings. All of these administration enhancements, and more, result in the most comprehensive server management tools and reduce the cost of ownership.

Domino Directories Administration Tab

The Domino Administrator provides logical groupings for administration functions and tasks via five specific interfaces reached via tabs across the top of the Administrator UI. These tabs are People & Groups, Configuration, Files, Server, Messaging, and Replication. On each tab, the UI is divided into three primary work areas or "panes."

- On the left, the Server Scope Pane gives administrators a complete hierarchical view of your Domino Server deployment.
- The Context Pane object on the top gives administrators a view of the specific database, directory, group, server, etc. that you are working on.
- The Results Pane on the bottom gives administrators immediate feedback and results of tasks you invoke.
- The Toolbar along the right side provides context-specific administration tools (also available via right-mouse click).



The People & Groups Tab in the Domino Administrator provides a central interface for all user and group management, such as user registration, certification, and group management.

From the Files Tab, Administrators can easily manage files and applications. Context-sensitive tools let administrators easily perform common database tasks such as check the disk status, move, compact, and more.

From the Server Tab, Administrators can get a graphical representation of the state of their servers, with details on the current status of specific tasks.

Installation

Domino Administrator R5.0 is not a stand-alone client, but is included as optionally installable with Domino Server R5.0 and Domino Designer R5.0.

As a developer you will need Domino Administrator R5.0 for tasks like signing databases.

Summary

In this chapter we have described the Domino Server Family:

- Domino Mail Server R5.0
- Domino Application Server R5.0
- Domino Enterprise Server R5.0

and the clients for Domino from Lotus:

- Domino Designer R5.0
- Notes R5.0
- Domino Administrator R5.0

Chapter 2
Lotus Domino Designer: An Overview

The Domino Web application server and Domino Designer provide a world-class Web development platform. They are optimized to enable you to build applications which facilitate the flow of information between your organization's enterprise systems and front-end business processes.

The Domino development environment offers you application services such as workflow, directory, messaging, and security which can be used to create high value business solutions.

The Domino Designer is an open application development environment that is intuitive and offers a high degree of developer productivity.

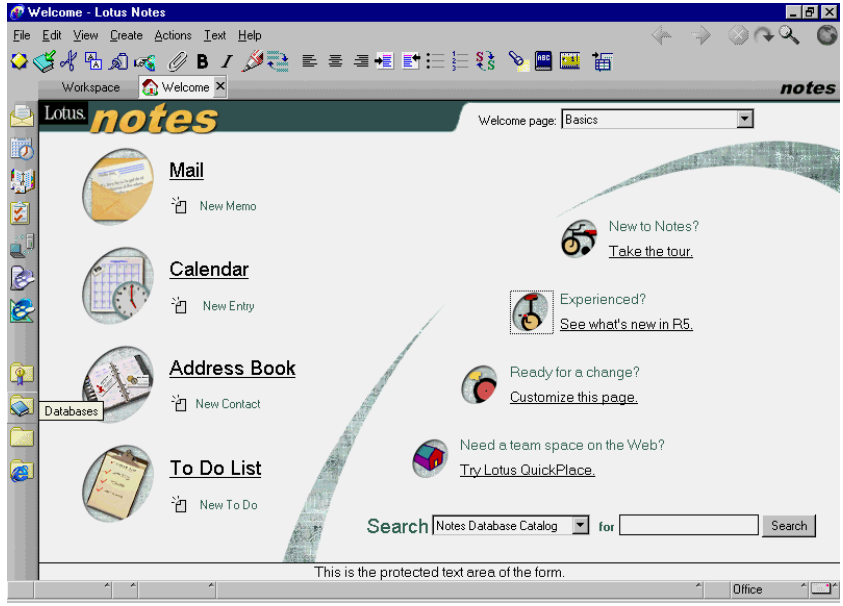
In Domino R5.0 Notes and Web development experiences have been integrated, bringing native Web technologies to the Notes environment and extending native Domino technologies to the Web environment. With Domino Designer you write your application once to run in both a Web browser and the Notes client. Several Domino design elements, previously offered only in Notes, are now available as Java applets to provide increased functionality to browsers. For example, a view served as an applet supports such "Notes-like" features as expandable/collapsible categories, resizable columns, and multiple document selection. A rich text editor applet allows users to style and format text in a Web document. In addition, you can now write applications in JavaScript which will support both the Notes client and the Web browser.

Lotus Domino Designer gives you the ability to build international applications with Domino Global WorkBench™ which contains a comprehensive set of tools to easily create, synchronize, and manage multilingual Domino applications.

Managing Your Workspace

This section gives you a short overview on the user interface of Lotus Domino Designer. As in Lotus Domino client, the workspace in Lotus Domino Designer is made up of several pages where the Domino databases are displayed as icons. One of the features of the Programmer's Pane is its sensitivity to context. You are very often just one mouse-click away from the action you want to perform.

When you start Lotus Notes your screen may look like this:



To access your workspace, click the Databases icon and then click Workspace.

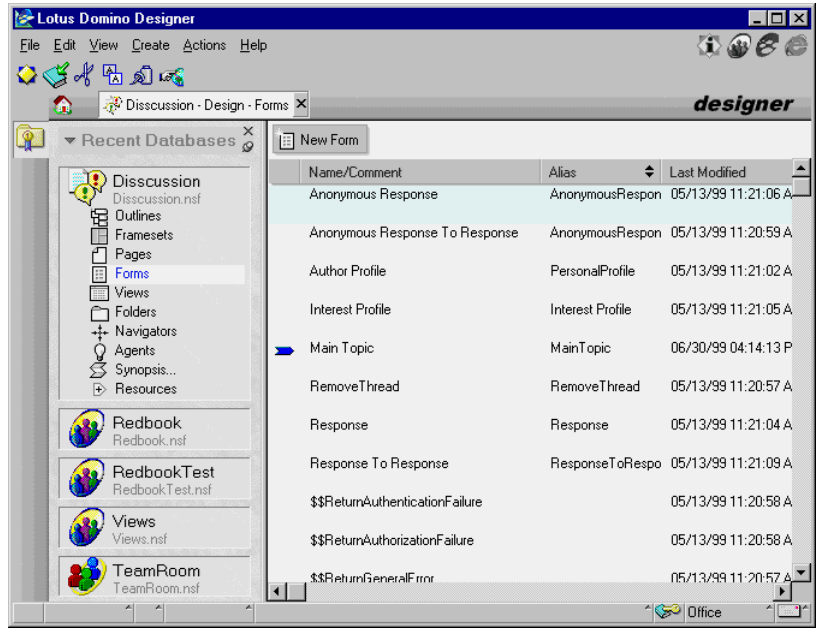
To open Domino Designer, click the Domino Designer icon or right click a database icon and select Database - Open in Designer from the pop-up menu.

You can of course also start Domino Designer by clicking its icon in the Start Programs list.

The Design Pane

The Design Pane gives you easy access to the design elements of the last 5 databases you have worked on.

Clicking the tab in the upper left corner of the designer bookmarks brings up the Design pane list.



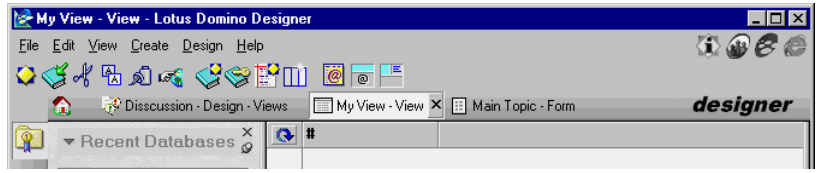
From here you can easily go to any of the design elements of an already listed database in the area called the work pane by simply clicking it.

Note As soon as you change the design of a Domino database this database is added to the site database list.

Clicking the push-pin in the upper right corner of the Recent Databases list fixes the list to the screen and stops it from automatically hiding. Clicking the push-pin a second time will cause the site database list to disappear when you click the programmer's pane.

The Window Tabs

Another enhancement to the user interface is the window tabs, which appear when a new design element is opened. The window tabs have replaced the Windows menu as a more intuitive way of tracking where you've been.



It's easy to see what you have previously opened and return to it quickly and easily by clicking the tab. Also, because the tabs have text titles you can easily close the windows that you don't want open without having them as the active window on the desktop. Just click the small x to the right of the tab to close that window.

Tip Using the window tabs is an easy way to cut and paste design elements or parts of them between applications.

Tip You can also use the keyboard to switch between the windows. CTRL+TAB allows you to move from window to window. To go to a specific window press ALT+W. Domino Designer then displays a number on each windows tab. Press the number displayed for the tab you want to select. To select a bookmark icon press ALT+B and select one of the displayed numbers.

The Launch Buttons

The Domino Designer R5.0 interface has a set of launch buttons in the upper right corner.

There are buttons which let you easily preview the results of your design changes. The following preview tools are available:

- Domino Client — Click the button showing the 3 people.
- Domino Web Browser — Click the button showing the tile.
- External Web Browser — Click the button showing the symbol of the installed browser.

Note The symbol shown for the external Web browser depends on the installed browser. For example, if you have two browsers installed you will see two buttons for the external browsers.

The button displayed to the left of the preview tools buttons (shown in the panel above) will launch the InfoBox for the selected design element.

The Programmer's Pane

The Programmer's Pane is made up of two parts:

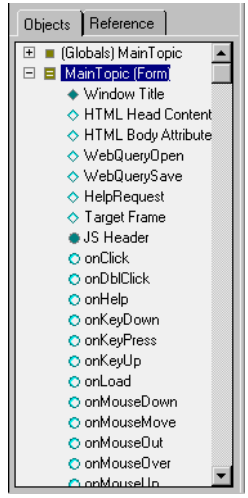
- Info list
- Script area

In the info list you can select one of two views:

- Objects view
- Reference view

The Objects View

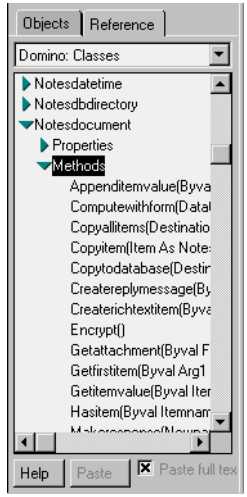
The Objects view gives you immediate access to any design element in your application and its associated events and attributes.



You can easily navigate through the list by clicking the plus (+) and minus (-) signs to expand or collapse the displayed list for a design element.

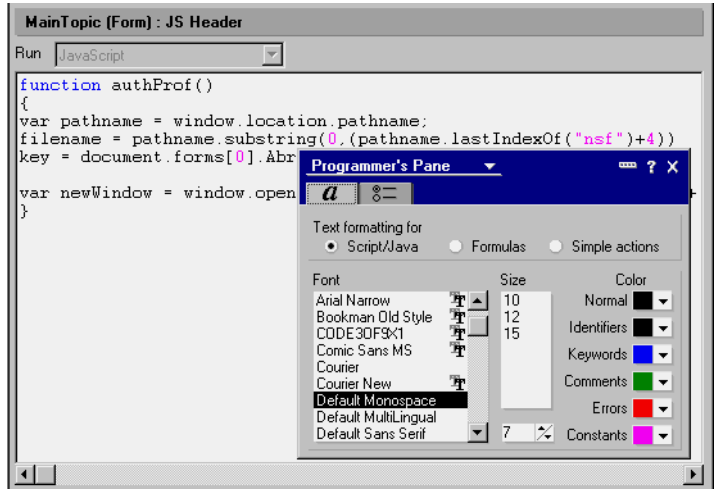
The Reference View

The Reference view is similar to the Objects view. In this view you get information about Domino objects.



The Script Area

Depending on the selection you make in the Objects view, the appropriate input window is presented in the script area. Using the Design Pane property box you can adjust the settings to your needs.



Domino Design Elements

As a developer of Domino applications you will work with the Domino design elements to build your application. The following section gives you a short overview of the design elements.

The Domino Database

A Domino database is a collection of related information stored in a single file. A Domino application uses at least one database. However, applications of a more complex nature may use several databases and may route information between databases on one or more servers. A database holds information about its design (see the description of the Domino design elements below) as well as data. Domino data is organized as documents. A document is defined as an object containing text, graphics, video, or audio objects, or any other kind of "rich text" data.

Forms

A form is a framework for entering and viewing information in a database. A Notes database contains documents created from one or more forms. A form can contain:

- Fields that store data
- Text that labels fields or gives instructions
- Subforms that store a collection of form elements that you want to use on more than one form
- Layout regions that combine graphics and fields in a way that affords greater design flexibility
- Graphics that make forms easier to understand
- Tables that summarize or organize information
- Objects (OLE, Subscriptions, Notes/FX™ fields), file attachments, URLs, and links that extend the reach of Notes documents
- Actions and buttons that perform functions automatically
- Background color and graphics that enhance the look of a document

Subforms

A subform is a collection of fields that you plan to use in more than one form. For example, you might create a corporate letterhead in a subform and then use the subform on a variety of business forms. Subforms can contain the same elements as a regular form.

Views

A view is a list of documents in a database. Depending on the selection criteria a subset or all documents of a database are displayed. The documents may be grouped or sorted based on their contents. Usually, the most important information contained in a document is shown in a view, too.

Fields

Fields are the individual elements on a form that store data. Fields determine what data a single document can contain. Each field in a document stores a particular kind of data, such as text, numbers, dates, or user names. Often users can enter and edit field values, but sometimes data is filled in or changed automatically.

The contents of a field can be displayed in documents and views or can be retrieved for use in formulas. A field can be defined for use on a single form or can be defined to be shared among multiple forms in a database.

Shared Fields

Shared fields behave like fields but may be used in different forms. If you change the properties of a shared field the changes are promoted to all occurrences of this field.

Tables

Tables help you to present data in a tabular way. You may use different order, styles, and colors to emphasize the data.

Action Buttons

Action buttons provide one-click shortcuts for routine tasks, and substitutes for menu choices. For example, they might allow users to compose, print, delete, or categorize documents, or give Web users, who don’t have access to the Notes menus, a way to click to edit, save, or close documents.

New Features of Domino Designer

From the beginning, Domino was designed to be a great environment for quickly creating secure, collaborative applications. Now, the Lotus Domino Designer has been enhanced to meet the needs of professional and corporate developers who create enterprise applications for the Web. As a result, the Lotus Domino Designer includes a set of new features and tools for rapid application development:

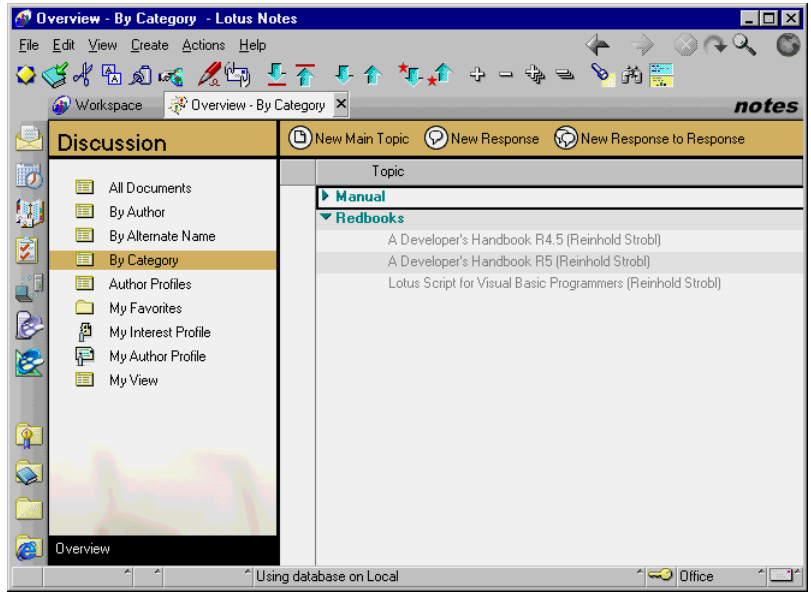
- **Outline Designer:** The outline designer is a visual toolset for simplifying Domino Web site design. It enables developers to design an entire site, link content to the site design, manage the links, and create a UI site navigation map component that can be used in site frames or on Web pages.
- **Frameset Designer:** The frameset designer provides visual tools and wizards to easily create multi-paned interfaces for Domino applications.
- **Page Designer:** Page designer, a WYSIWYG HTML authoring tool, supports a broad range of browser technologies. A number of improvements to tables and graphics file support provide complete control over page design and layout.
- **Domino UI Applets:** Three popular Notes user interface components are now available as Java applets. The Java applets provide the capability to quickly add these full-featured Notes design elements to browser applications.
- **New Rapid Development Capabilities in IDE:** Domino Designer now enables multiple work sessions to be open within tiled windows, provides a "movable" properties box for rapid manipulation of object properties, and offers a new Design Synopsis that provides access to all information about your application including application source code and administrative information.

To start with, we will now look at how these new Design elements can be used in Domino. The following figures show a database that has been developed to be used from the Notes client as well as from the Web. The following design elements have been used:

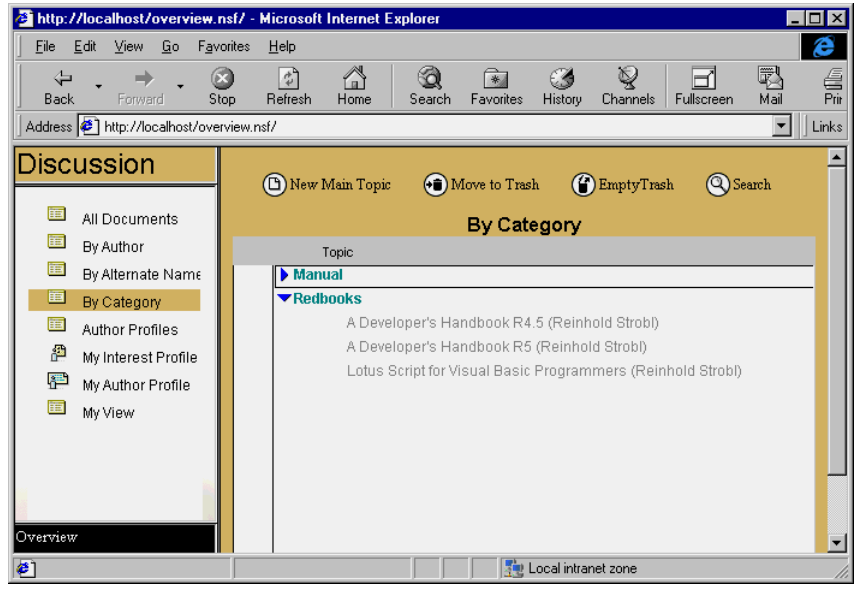
- Frameset
- Page
- Outline
- Domino User Interface Applet for views

One of the benefits of the new design elements is that you can build applications that have the same look and feel both in Notes and on the Web. Of course, there are some Domino functions which cannot be carried out on the Web, but using the Domino User Interface Applets help you to minimize this impact.

This is how the database looks when it is opened in the Notes client:



The following figure shows the same database opened in a Web browser:



As you can see they look very similar.

The design elements used are:

- *Page*

  The pages are shown in the three left frames of the screen. The middle left page contains an imbedded outline (the rectangle containing the selectable views).

- *Outline*

  The outline consists of seven entries. Each entry describes how the associated view is to be shown or provides a link to a profile document. In this instance, we have specified that each view should appear in the right frame of the screen.

- *Frameset*

  A frameset allows you to split the screen in several logical units. In this example there are four frames, three containing the pages described above, and a fourth one, containing the views.

- *Domino UI applet*

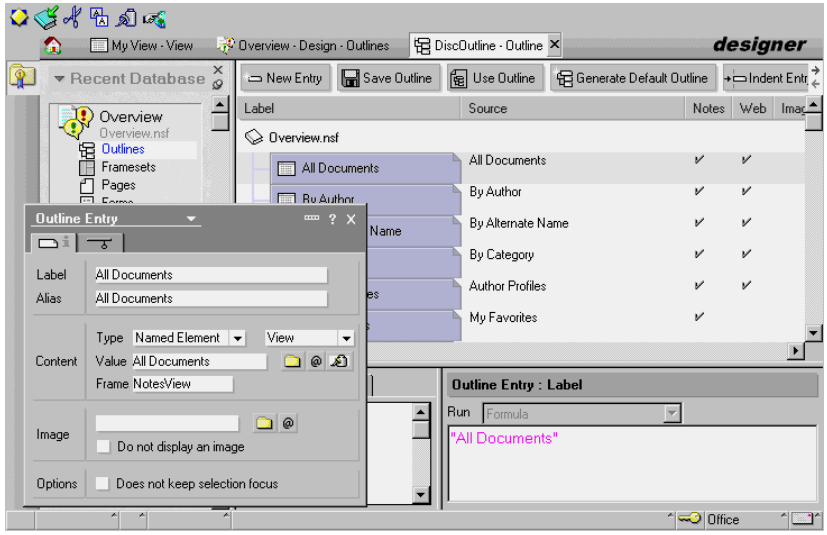
  The view applet brings some functionality to the browser. Using such an applet you can expand and collapse views without a server request, and you are also able to select documents.

The next section gives some more detailed information about each of the features and tools briefly described above.

Outline Designer

The outline feature lets you visually design the navigation of your Web site. In the same way that a table of contents serves to focus the reader's attention on the entire document and its organizational flow, the outline allows you to design the overall navigation of the application first before drilling down to the page or form level. A Web developer might think of this as site map creation. In addition to serving as an organizing structure for a site designer, an outline can be presented as a navigational tool to users, much as a table of contents guides a reader through a book.

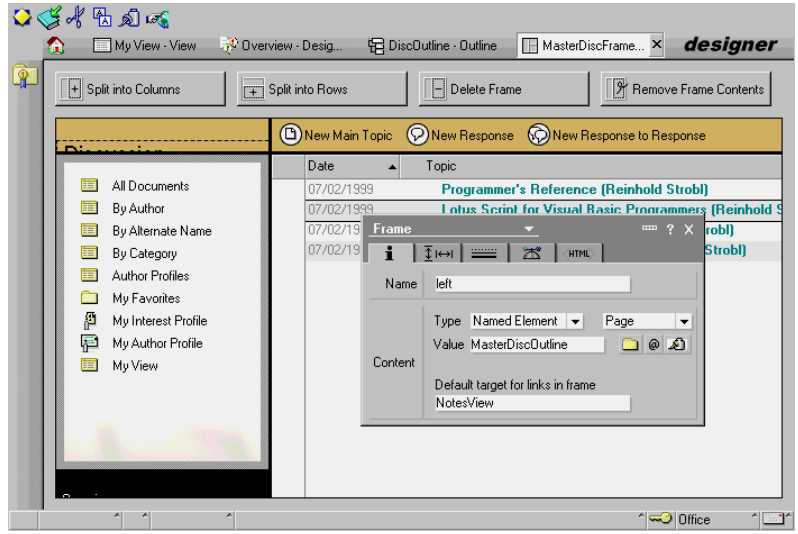
Using the outline feature, you can visually create dynamic Web site maps by dragging and dropping navigation tiles into a hierarchical structure and linking them to pages, forms, views, or URL addresses. You can set properties for the outline such as whether a design element is visible or invisible to users, and liven up the display with bitmap icons. You can then embed the resulting Outline control on a Web page or form and further enhance it with background colors and graphics. The completed site design is then available as a site map component, called an Outline control, that can be easily embedded on a Web page, form, or frame.



The Outline control automatically maintains links to the Web page or other elements no matter how often you rearrange the site. This reduces the administrative burden of managing large sites that are frequently updated. The Outline control is programmable, enabling you to add logic to your outline that dynamically updates the control. For example, an Outline control can display customized navigation options based on a Web user's identity or the location of the user within the site.

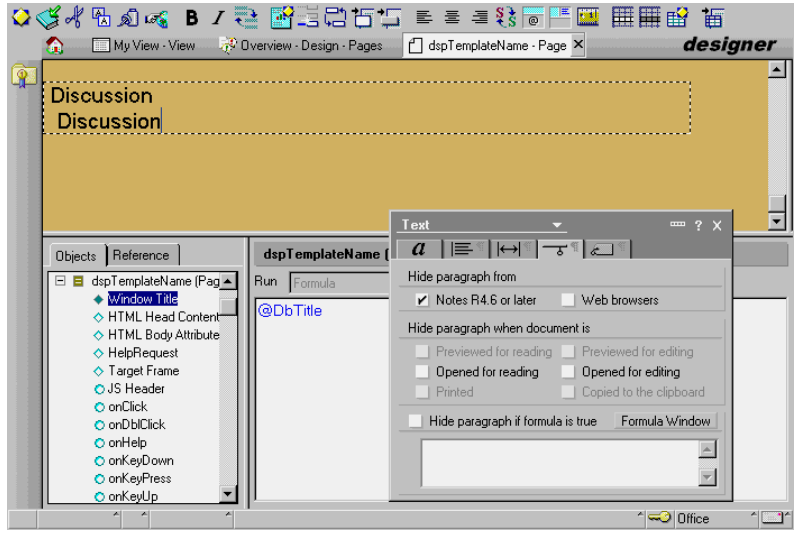
Frameset Designer

Framesets provide a standard way to set up a multi-pane interface for the user. The Frameset designer enables you to create framesets and then associate specific pages, views, forms, Java applets, ActiveX components, or any URL with each frame.



Page Designer

Using the Page designer you can create or import HTML Web pages. Page designer is a WYSIWYG HTML authoring tool that provides support for a broad range of browser technologies including HTML 4, image file formats, Java applets, ActiveX components, and multimedia objects. You can create or edit HTML in the Page designer using the WYSIWYG editor, by writing HTML source code, or by mixing both in a single page.



The Page designer provides you with a much improved level of control over the layout of your Web pages. You no longer have to work directly in HTML to create sophisticated page design and layout (although working directly in HTML is still an option).

Domino User Interface Applets

Several Domino design elements are now available as Java applets. Domino converts all Domino design elements, including embedded elements, to HTML when accessed by a browser. Optionally, however, you can specify that Domino sends an interactive version in the form of a corresponding downloadable Java applet. The applets enable more efficient operations by allowing work to be performed in the client-side applet.

There are four applets available — View, Text Editor, Action Buttons and Outline Control. The view is a unique Domino design element that provides a customizable interface for displaying and interacting with lists of documents or other objects. The text editor provides a fully functional rich text editor for creating and editing content. The Outline control offers customizable site navigation.

New Rapid Development Capabilities in IDE

Also several significant features have been added in Release R5.0 that can be taken advantage of throughout the environment for improved rapid application development. For example, you can open multiple work sessions at the same time either as tabbed work sessions within Designer or as tiled windows running within multiple instances of Designer. Manipulating visual objects within Designer is easy using a movable properties box. All properties associated with a Domino design element are available to you via point-and-click.

Furthermore, the design synopsis feature has been improved. Design synopsis provides a single location for you to browse or search all of the components and code logic within a database. The design synopsis provides you with an output of critical information about every component of the application for documentation or archival purposes. This output is customizable and can be displayed in a single document or in a database to be used at a later time. This feature is easily accessible from the workspace.

Industry Standards Support

Environments for developing e-Business applications must support the "standard" Web programming and scripting languages including Java, JavaScript, and HTML. Lotus Domino Designer provides full support of Web standards.

Within the Designer Programmer's Pane you have new choices of languages for writing and compiling code. Java is available for creating Domino agents, which are server-side applications that are initiated based upon events or schedules. Designer supports JavaScript in conjunction with a subset of the Document Object Model, a standard drafted by the World Wide Web Consortium (W3C). In addition, you can code in HTML directly in the Page designer and Forms designer.

Lotus Domino Designer also supports CORBA/IIOP for creating distributed applications. With Domino CORBA objects, you can write Java applications and Java applets that remotely access Domino services and data. Through the support of industry standards in the Domino Web application server and in Domino Designer, you are able to lower your cost of ownership and application maintenance by leveraging your existing developer skills.

Multi-Client Applications Support

One of the great benefits of Lotus Domino Designer is that you are able to develop a single application that runs in both the Notes client and Web browsers. The Designer now supports the latest Web standards including HTML 4, JavaScript, and Java. In addition, some Domino design elements are available as Java applets. This provides functionality, previously available only in Notes clients, to Web browsers. Furthermore, the CORBA/IIOP distributed object technology is supported in Domino providing an alternative to Notes Remote Procedure Call (RPC) for communicating between clients and the server.

Notes and Domino localization features allow Notes and Domino to be a premier platform for multinational organizations working across time zones, languages, and cultures.

Multilingual Applications Support

The Domino Global WorkBench is now completely integrated as a feature in Lotus Domino Designer, so you can create multilingual Web sites right out of the box. The strengths of Domino Global WorkBench reside in better enabling and serving multilingual multinational corporations and Web site developers who are implementing and rolling out Domino-based multilingual applications for use on the World Wide Web or on a Notes Network. Domino Global WorkBench turns Lotus Domino servers into an intelligent language server for the Web.

Domino Global WorkBench lets you localize the Web infrastructure, define the initial sets of languages supported in the Web site, and define the high level of synchronization between forms and pages across languages. By allowing you to localize all user-visible elements of an application, including field labels, buttons, bitmaps, and dialog boxes, as well as actual content, an application can be optimized for each individual user who can select their language of choice at run time.

You can create the design of a Web site in more than one language with Domino Global WorkBench by resourcing/localizing Notes design elements and objects stored in the Notes object store environment, and the development languages including HTML, LotusScript, or JavaScript. Domino Global WorkBench facilitates review and approval of localized documents through workflow process and ensures accurate linking and synchronizing of pages available in different languages, enabling content to appear simultaneously to all users worldwide.

Easy Access to Enterprise Data and Applications

Incorporating back-end data into everyday business processes maximizes the value of Domino applications. Domino applications provide core technologies for the security and control of business processes, forms routing, and approvals management. With new enterprise integration technologies, Domino applications are now able to incorporate traditionally difficult to reach data into those applications, becoming a key component of managed business processes.

Domino includes the ability to create Web applications that contain connectors to relational databases (for example, Oracle DB/2), Enterprise Resource Planning systems (for example, SAP), and transaction systems (for example, CICS, IBM MQSeries, and IMS). You can accomplish this either programatically (as available in previous versions) or with visual tools to native database drivers.

Domino Enterprise Connection Services (DECS) offers developers a visual tool and high performance server environment used to create Web applications that provide live, native access to enterprise data and applications. The visual tool presents an application wizard and online documentation to assist you to define external data source connections — DB2, Oracle, Sybase, text-based files, EDA/SQL, and ODBC — and fields within the Domino application that will be automatically updated with external connector data.

New Domino classes for enterprise data access will be available in LotusScript and Java. These classes enable you to customize applications to incorporate information from relational databases, transaction systems, and ERP applications from Domino according to your business needs. The Domino driver for JDBC, providing standard JDBC access to data in Domino databases, is also available. Using this driver, you can write Java applets and applications that use JDBC to access information in Domino databases.

Domino Connectors are modules which provide native connectivity to external sources such as relational database, ERP, or transaction systems. These connectors can be accessed through the forms-based development tool in DECS, or through the new Domino object classes using LotusScript or Java languages.

Lotus Enterprise Integrator (LEI), which is available separately, extends DECS functionality beyond real-time data sources to include support for high volume data transfer and synchronization. LEI provides visual tools to manage integration between data sources without programming, including the capability to initiate event-driven or scheduled high volume data transfers between Domino applications and relational databases and other enterprise applications. LEI also supports programmatic data transfers via LotusScript and Java Classes.

Chapter 3
Domino Design Elements: Basics

This chapter describes how to create and manage Domino databases. The chapter also includes a glossary of Domino design terms that an application developer would need to understand when creating a Domino application.

Domino Databases

The term "Domino database" refers to both Domino and Web databases. What makes it a "Web" database is the viewing mechanism — a Web browser instead of a Notes client, and the fact that it resides on a Domino server running the HTTP server task.

Traditional Web sites consist of different kinds of pages and the associated compound elements which are organized in hierarchical directory structures. When an HTTP request is issued to display a page, a new HTML coded file is opened.

With Domino, the Web site is structured through Notes databases designed in the Notes object store format. When an HTTP request is issued to display a page, a Notes element is opened through a Universal Resource Locator (URL) command and Domino translates it for viewing as a Web page.

Creating a Database

There are several different ways of creating a database. You can:

- Use an existing template.
- Use an existing database.
- Create a new database.

Once the database is created, you can modify most of the settings using the Database InfoBox. We will cover these options later in this chapter in the section "Changing the Database Properties."

Using an Existing Template

Domino Release 5.0 provides a series of written applications that can be used or customized for your own needs. Although there are many types of popular application templates, they are mainly designed to reveal the

underlying technology and development capabilities within Domino Release 5.0. Their main intent is not to be "out-of-the-box" applications.

If your application is identical, or similar to, an existing template provided with Domino, the most convenient way to create a new database is to use that template as your starting point. Most of the design work has already been done for you. The design elements of the individual templates can easily be copied and pasted into your custom applications.

Listing Available Templates
To see the list of available templates:

1. Choose File - Database - New. The list box on the New Database dialog box lists several templates.

   Tip The shortcut is CTRL+N.

2. Click the Show advanced templates check box. The list box at the bottom of the list displays additional templates. The templates listed are stored on your local workstation.
3. Select any template.
4. Click the About button to display the database Help document. It summarizes what the database can be used for.

To see additional templates stored on a server:

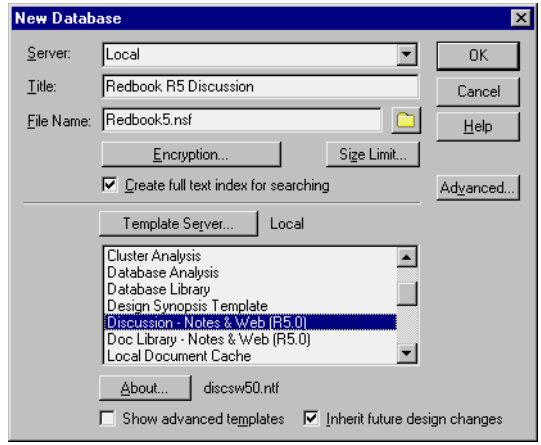
1. Click the Template Server button.
2. In the Server field, select the server you want to access. Additional templates are listed.

Creating the Database
Follow these steps to create the database:

1. Decide if the database will reside on your local workstation or on a server.
2. In the Title field, specify a meaningful title.
3. In the File Name field, specify a file name for the database. You can also take the file name that Domino provides automatically based on the database title.

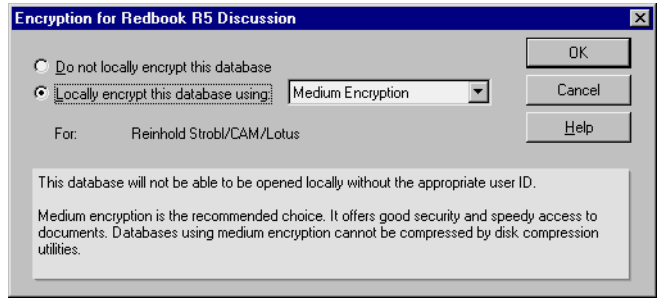
   Note The extension for a database file is .NSF. The extension for a database template file is .NTF.

   The following figure gives you an example of a completed New Database dialog box:



4. You can encrypt local databases to protect confidential data. This is useful if users have laptops that will be taken out of their business locations.

   To specify encryption, click Encryption. Specify the appropriate level of encryption. The following figure shows what you can do:

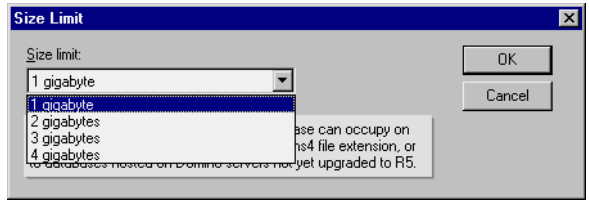


   Only the user shown in the Encryption window has access to the local database after it is encrypted.

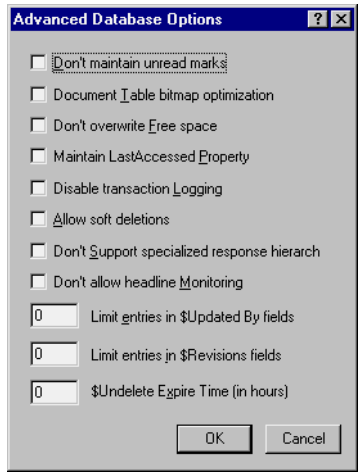
   Note Although Domino users can't access the encrypted database locally, they can still access the database if it is on a server. This also applies to Web users. If you would like to deny access to other users, use the Access Control List.

   Databases that you create or convert to the Release 5 format perform considerably better than databases created in previous releases. Because of these performance improvements, limiting the size to improve database performance is less important than in past releases. The size limit applies only to databases using the .ns4 file extension, or to databases hosted on Domino servers not yet upgraded to Release 5.

The Size Limit box looks like this:



5. If you want your new database design to stay synchronized with the design template, check Inherit Future Design Changes in the New Database dialog box.
6. For advanced options, click Advanced. The following dialog box will be displayed.



See the Domino R5.0 Help database for a description of these advanced database options.

Copying an Existing Database

Copying a database is similar to starting from a template, except that you will almost certainly want to change part of the design.

Listing Available Databases

To list the available databases:

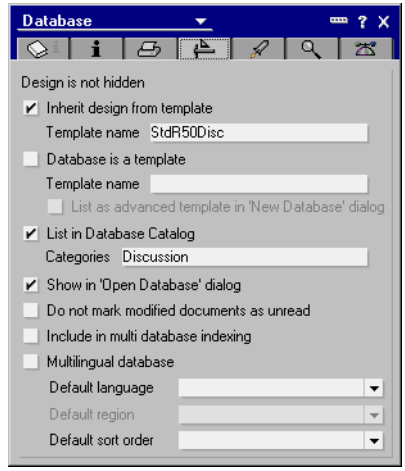
1. Choose File - Database - Open.

   Tip The shortcut is **CTRL+O**.
2. If required, specify the appropriate server name in the Server field to list additional databases.
3. Click About to browse the Help document of the database.

4. Click Bookmark to add the database to your bookmarks.

Creating the Database
To create the database:

1. In your Database folder, select the database icon that you want to copy.
2. Display the database pop-up menu by clicking the right mouse button.
3. Choose Database - Properties.
4. Click the Design tab. Make sure that the InfoBox shows that the design is not hidden. It should look like this:



5. Close the InfoBox.
6. Keep the database selected.
7. Choose File - Database - New Copy.
8. Select Local as the server name if you want to store the database on your local workstation. Select a server name if you want to store the database on a server.
9. Enter a title for the database.
10. Type a file name with the extension .NSF for the new database.
11. Select Database Design Only since you do not want to copy the documents that are stored in the database.
12. Deselect Access Control List because it could prevent you from modifying the database design in the future.

    Note Unless you deselect Access Control List, the access that you have to the copy of the database will be the same as the access you have to the original database.

The following figure shows an example of the Copy Database dialog box:

13. You can optionally select the following two features:

    - Encryption. This is a good idea if database will be stored locally, especially if the database contains confidential data, or if your users have laptops that they use in public environments.

      To encrypt the database, click Encryption and select the appropriate encryption level.

    - Predefine the maximum database size. Click Size Limit and specify the appropriate size.

    Note This applies only to databases using the .ns4 file extension, or to databases hosted on Domino servers not yet upgraded to Release 5.

14. Click OK to start the copy operation.

Creating a New Database

If the template or existing databases do not meet your requirements, you can create a completely new database. This means that you will have to create all the design elements, such as forms, pages, views, and fields. However, you can always copy existing elements from other databases and paste them into the new database.

1. Choose File - Database - New.
2. Type a title in the Title field.
3. From the list of available databases displayed at the bottom of the window, choose the -Blank- option.
4. Click OK. The new database will be added to your bookmarks.
5. Select the database that you just created, click the right mouse button, and select Open in Designer.
6. Domino opens the Lotus Domino Designer R5.0 workstation and you are ready to start the design of the database.

Changing the Database Properties

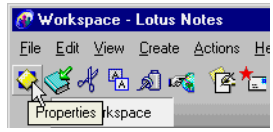
One of the strengths of Domino is the fact that it is sensitive to context. You are often just one mouse-click away from the properties of the object you are working on: outlines, fields, embedded elements, Java applets, buttons, forms, attachments, and databases.

Opening the Database InfoBox

To display the database InfoBox:

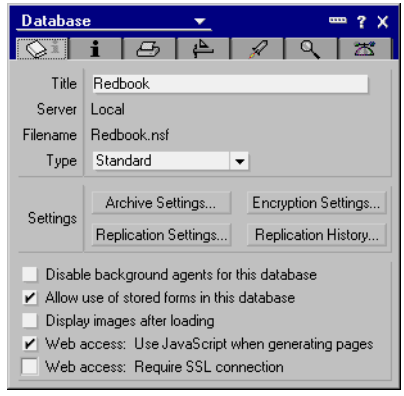
1. Display the database pop-up menu by clicking the right mouse button.
2. Select Database Properties.

   Tip You can also click the following InfoBox icon to display the database properties InfoBox. If the SmartIcons bar is not visible, select File - Preferences - SmartIcons from the menu and click Icon Bar under Show.



Specifying the Database Type, Replication, and Encryption

The Basics tab contains information about the database, such as its title, location, database name, the database type, the replication settings, and replication history. The Basics tab looks like this:

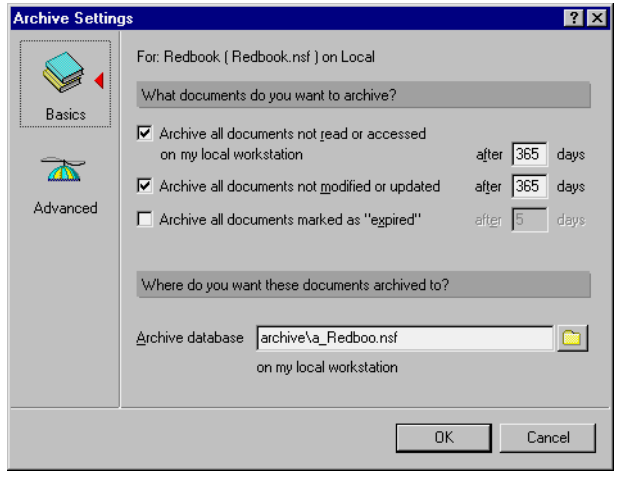


1. To set the Database type, you can select one of the following values:
   - **Standard.** The option used most of the time.
   - **Library.** A database with type Library is generally created from the Database library template. It is used to record and store information about the databases located on a Domino server or on a workstation.

This provides an easy way for users to browse the list of databases available to them

- **Personal Journal.** This database type allows you to store personal information. It has limited design elements and is meant for individual use.
- **Address Book.** Creates a database based on the Domino Directory format.
- **Light Address Book.** Creates a database based on the Domino Directory format, but with fewer features than the full Address Book.
- **Multi DB Search.** Used to specify a database type of Search Through Multiple Databases, which uses the SRCHSITE.NTF template. This type of database is used to configure searches among databases that have been designated to participate in Multi Database indexing by selecting the appropriate option in the Design tab of the database InfoBox.
- **Portfolio.** This type allows the user to keep a collection of databases that are used frequently or related.
- **IMAP Server Proxy.** Internet Message Access Protocol (IMAP) is used to send and receive electronic mail using the Internet.
- **News Server Proxy.** Allows the user to keep a collection of news group and conversations about news.
- **Subscriptions.** This type is used to keep track of user subscriptions to various databases and is used by the headlines databases.
- **Mailbox.** This type is used for mailing purposes.

2. Click Archive to display the Archive InfoBox. It looks like this:



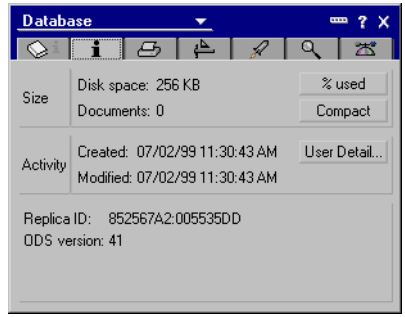
This option enables you to determine when documents are deleted and where archives of those documents are stored.

3. Click Encryption to display a window that enables you to specify encryption for the local version of the database.
4. Selecting Web Access. Use JavaScript when generating pages. This option allows the Domino server to use JavaScript to generate Web pages. By selecting this option you can use, for example, multiple buttons on the form.
5. Selecting Web Access. Require SSL connection. This option forces users to login to the database using Secure Sockets Layer.

Displaying General Database Information

1. Click the Information tab to display general information about the database, such as its size, and the number of documents stored.

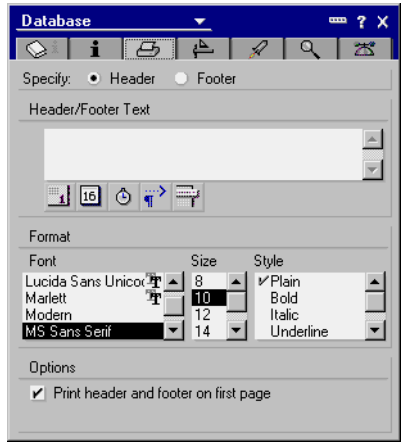
   The Information tab of the InfoBox looks like this:



2. Click User Detail to display information related to user activity.

Specifying Print Options

1. Click the Printer tab to specify options related to printing the database.
2. Use the icons under the Header and Footer options to define the date and time, tabs, and page numbering.



3. You can also select the font, size, and style.

   Note Printing properties don't take effect if you are printing from the Web.

Specifying Database Design Properties

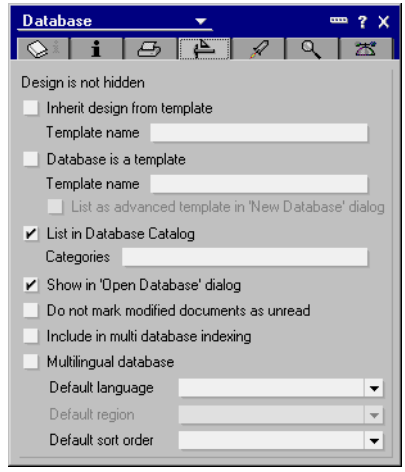
1. Click the Design tab to display or specify information concerning the design of the database.

   The example displayed in the figure below shows that the design of this database is not hidden. If you select Inherit design from template, the database automatically inherits all the changes made to the template if the template this database is based on is modified in the future.

2. If the database you are creating is a template, check the Database is a template check box.
3. Specify a name for the template.
4. If appropriate, select that the new template is listed as an advanced template. This indicates that the template should only be customized by Domino developers.
5. Deselect List in Database Catalog and Show in 'Open Database' dialog if the database is located on a server, contains sensitive data, and you do not want users to be able to see its name.
6. Select Include in multi database indexing if you want the index to be included in Multi Database Search Database site queries
7. Enable Multilingual database if your database will be used by multinational organizations across time zones, languages, and cultures.

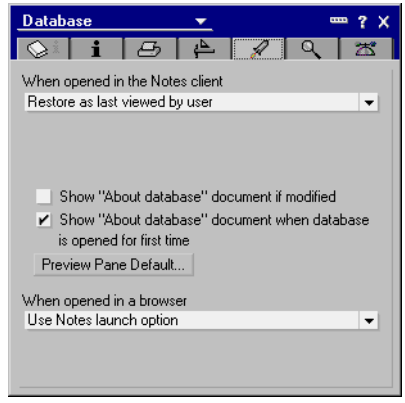
8. Select the Default language for this database.

   The following figure gives an example of the Design tab:

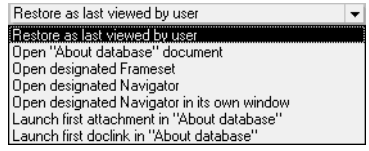


Specifying Launch Options

1. Click the Launch tab to define what users will see when they first open the database. The dialog box looks different depending on your choice of actions in the On Database Open drop-down list.

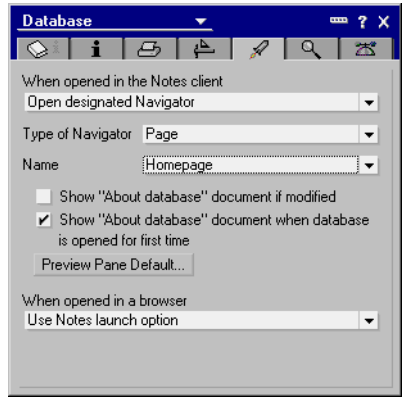


2. Select an option from the On Database Open drop-down list.



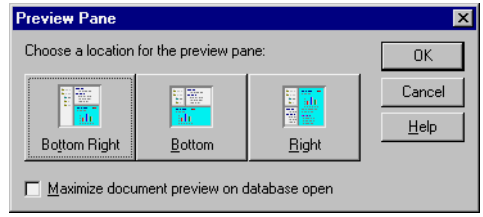
A wide variety of options is possible, for example:

- If you select open designated Frameset, then you need to select the frameset that you want to be opened. All available framesets are listed in the drop-down box.
- If you select Open designated Navigator options, another drop-down list is displayed, where you can select the kind of navigator that you want to open. You can choose folders, standard navigator or page. Choosing standard navigator or page will display a third drop-down list where you can select the actual navigators or pages to be used.



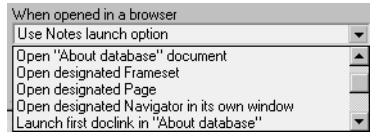
- Select Open Designated Navigator in its Own Window if you want the navigator to be displayed in a full screen. You would typically choose this option if the navigator or page consists of a large map or a workflow sketch.

3. You can specify the properties of the Preview Pane by clicking Preview Pane Default. You will be presented with a number of choices. Click the most appropriate property for the user.



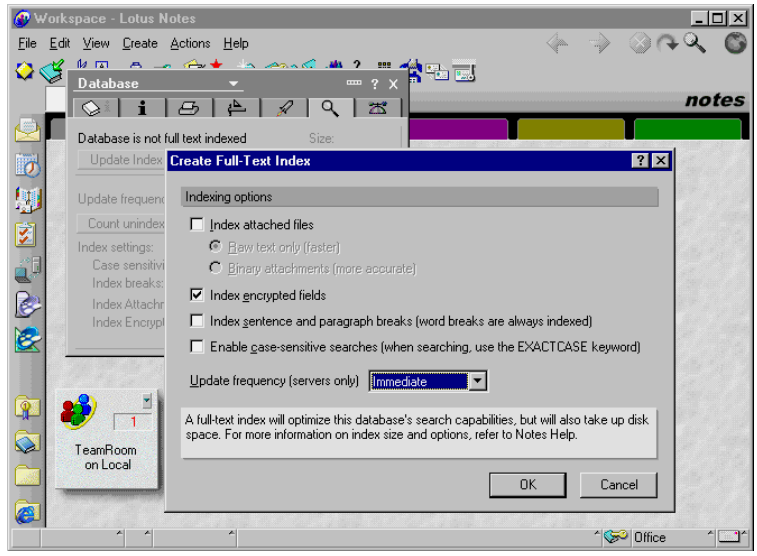
Note This option does not take effect on the Web, because Web users don't have the Preview Pane.

4. Select an option from the On Web Open drop-down list for Web users opening the database. A wide variety of options are possible.



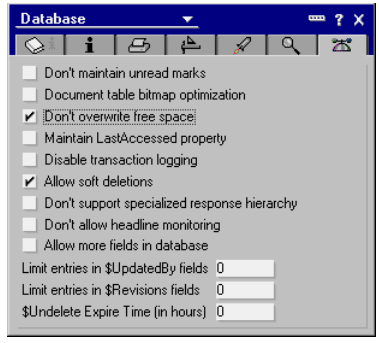
Specifying Full Text Indexing

1. Click the Full Text tab to create, update, or delete a full-text index, which allows for a fast retrieval of documents.
2. Select the update frequency as required.



Specifying Advanced Options

1. Click the Advanced tab if you would like to enable or disable fields in the database, or set the size limit for fields in the database.



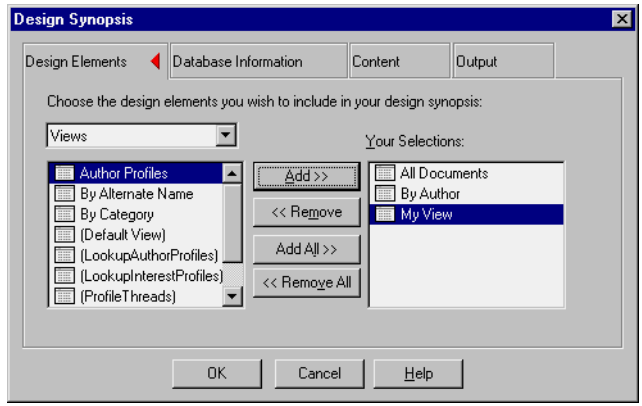
Using Design Synopsis

Design Synopsis gives you a single location where you can browse or search all of the components and code logic of a database. It provides you with an output of important information about every component of the application, for documentation or archival purposes. You may customize the output to your needs, and you can display it as a single document, or put it into a database to be used at a later time.

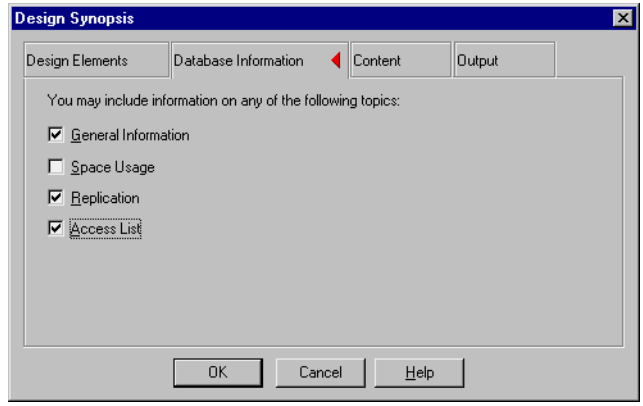
The Design Synopsis dialog box allows you to generate a detailed report on a particular database. In Release 5.0, the user interface of the dialog box has been enhanced to make it easier to use.

To create a design synopsis:

1. Select the database for which you want a report.
2. Choose File - Database - Design Synopsis.
3. Select the design elements (forms, views, shared fields, agents, and so on) which you want in your report. For each design element you select, select the individual elements that you want in your report. The Add button lets you select the elements of a design and add them one at a time to your report. The Add All button selects all the elements of a particular design element.



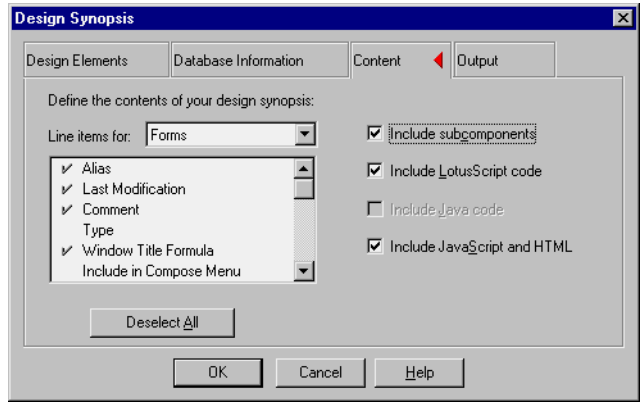
4. Click the Database Information tab to select the database information that you want in the report.



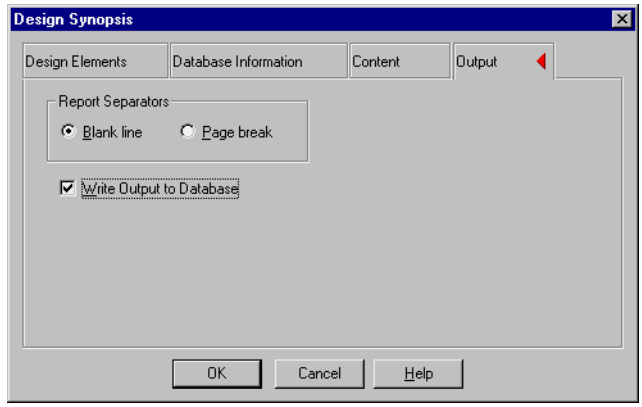
The following selections can be made:

- **General Information.** Gives information such as the database title, location, and categories.
- **Space Usage.** Calculates the file size, number of documents, space used by the database, and so on.
- **Replication.** Gives information on the replication settings for the database.
- **Access List.** Generates a list of users, groups, and servers in the ACL and specifies assigned access levels and access roles for each.

5. Click the Content tab to select the contents of the report for each design element.

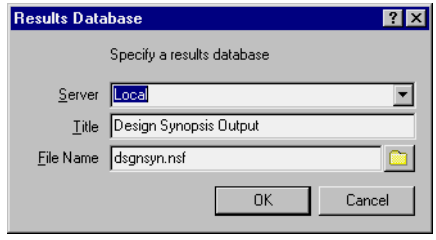


6. Click the Output tab to specify the output options. For example, if you want to write the report to a database, select Write Output to Database.



7. Click OK to generate the report.

   If you have selected Write Output to Database, a new dialog box is displayed where you can specify the database to receive the information.



Summary

This chapter explained some of the basics related to creating Domino databases and also provided a glossary for some of the most commonly used design elements in Domino. It also explained Design Synopsis, which enables you to easily see an overview of the design elements in your database.

Chapter 4
Forms

This chapter describes Domino forms; what they are as well as how to design and modify them. We also discuss the basic design elements used when creating a Domino database. In addition, this chapter explains how to show different information to Web users and Notes users.

When you have completed this chapter, you will also understand how to capture CGI variables into a form.

Forms

This section will guide you through Domino forms; what they are, the different types of forms, and how to create, design, and modify them. This section also discusses the design elements of the form, such as fields, tables, images, embedded elements, layout regions, and computed text. We will use the TeamRoom database template as a basis for demonstrating the Domino R5.0 elements. You will find the template for the TeamRoom database on your Domino server.

The form is the skeleton provided to users to enable them to enter data, either by typing or by using buttons. There is usually at least one form in a database, although a typical business application will have many forms, each targeted to the type of information that the user wants to save in the database.

The form contains all the design elements; fields to store the user's information, static text, buttons, sections, images, and subforms that help the user to enter the data into the database.

To create a new form, go to Form Designer and click New Form or choose Create - Design - Form. Alternatively, you can copy and paste a form from the Design Form pane and then customize the form to your needs.

Specifying Form Properties

The Form InfoBox contains all of the information related to forms.

To look at the form properties, do the following:

1. Go to the Form Designer.

2. To create a form, click New Form and the new form is displayed.
3. Click the Properties icon.

   Note If the Properties icon is not displayed on your workspace choose File - Preferences - Smarticon Settings. Select the option Show Icon Bar in order to display the icons. Instead of clicking the Properties icon you may click with the right mouse button on the design pane and select Form Properties from the popup window.
4. In the InfoBox, click the triangle in the middle of the InfoBox title and select Form. An InfoBox will be displayed which allows you to set the properties of the form. It consists of seven tabs:
   - Form Info
   - Defaults
   - Launch
   - Background
   - Header
   - Print
   - Security

Using the Form Info Tab

The Form Info tab stores general information about the form.

1. In the Name field, specify a name and an alias for the form.

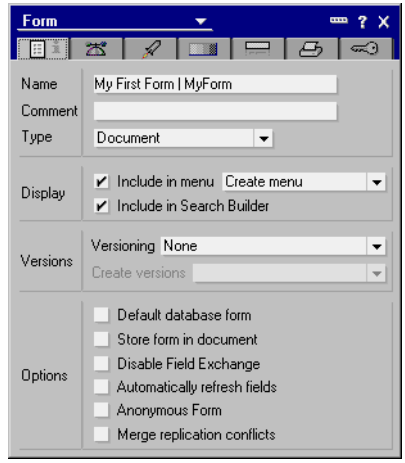
   By default, the form name appears as an item in the Create option on the menu bar. This is the name that the user sees. Therefore, make it as meaningful as possible.

   To specify an alias enter the vertical bar (|) and the alias after the form name in the Name field. It is recommended that you create an alias for each name. This is the name you will use in your code. Specifying an alias enables you to leave your code unmodified if, for example, the user requests to have the name of the form changed.
2. Try to keep the first character unique, because Domino will use the first unique character as a keyboard shortcut under Windows.

   Note You also can add the underline character (_) to define a keyboard shortcut.

   Example: _Document

   The D is now the keyboard shortcut character.



3. In the Comment field, you can enter some information about the form. This is useful for designers, and will help another designer who may need to modify the database design at a later stage.
4. Specify the form type. The form that we are creating in the example shown above is a Document type, which means that it is a main document. Response type means that it is linked to a main document of type Document. The response document cannot exist without a parent. A third type is the Response to Response type, which adds a third level to the document hierarchy.
5. If you want to include the form in the Create option on the menu bar, you have two options:
   - If there are only a small number of forms, they can be displayed directly in the Create Menu list. Up to nine forms can be listed in the Create option.
   - If there are many different forms, it is best to use the Create - Other dialog. You could add the most commonly used forms to the Create Menu list and then put the least used forms under the Create - Other dialog option.

   Note Create options are not applicable to Web users, because Web users do not have the Create menu.
6. Select the Search Builder check box to add the response form to the list of forms that users can search.

7. In the Versioning field, specify whether or not you want version control. The following options are available for Notes users but not for Web users:
   - None
   - New versions become responses
     Use this when the original version is the most important. The original document is listed first in a view.
   - Prior versions become responses
     Use this when the new version is the most important. The latest version is listed first in a view.
   - New versions become siblings
     Use this when all versions have equal importance. The original document is listed first in a view. All successive versions follow as additional main documents without introducing the risk of replication or save conflicts.
8. If you decided to work with versioning there are two create version options:
   - Manual-File, New Version creates a new version of the document when the user user chooses File - Save as New Version.
   - Automatic-File, Save creates a new version ot the document each time the user saves the document.
9. Domino uses a default form to open documents whenever their associated form has been dropped from the database design. You should select this option for the main form of the database.
10. Leave the Store Form in Document check box deselected.

    You must store the form in the document if, for example, a user that has no access to the database receives a document and has no access to the design of the form used to create the document. For Web users this option is supported in read mode, but do not use it for documents that need to be created or edited on the Web.

    Note Selecting the Store Form in Document option increases the amount of disk storage required to store each document based on that form.
11. The Disable Field Exchange check box, which is normally left deselected to enable data exchange with Notes/FX compliant applications, is not supported on the Web.
12. With the option Automatically Refresh Fields selected Domino recalculates fields automatically whenever a field value changes. It is not supported on the Web.

13. Select the Anonymous Form check box if you want authors or editors to anonymously enter documents into the database based on this form.

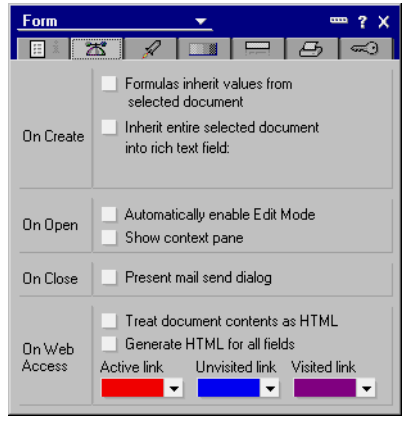
    Note Documents created with an anonymous form do not contain the $UpdatedBy field, but have an $Anonymous field with a value of "1". You will also need to make sure that the author's or editor's name does not appear in any other field of the form.

14. If desired, select the Merge replication conflicts option. Domino then merges conflicting edits into a single document whenever possible. If two users edit different fields in the same document, Domino saves the changes to each field in a single document.

    However, if two users edit the same field in the same document, Domino saves one document (which is saved first) as a main document and the other as a response.

Using the Defaults Tab

The Defaults tab lets you specify details regarding the usage of the form.



The Defaults Tab is divided in four sections:

1. On Create:
   - The Formulas Inherit Values from Selected Document option means that Domino copies the values of the fields in the parent document to the document or response document that is being created. The inherit does not work automatically, unless you specify the field name of the parent document in the field's default value event. Domino supports this feature for rich text fields as well as other field types.

   Note Inheritance from selected documents is supported on the Web, but only if the field is of type text, the document resides in the same database, and the database property Use Java Script when Generating Pages is selected.

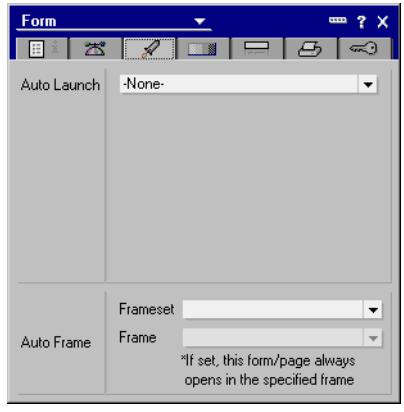
- The Inherit Entire Selected Document into Rich Text Field option defines how the fields of the parent document are displayed in the response document. To inherit the parent document a Web browser user must open an existing document before creating the new document. To enable Web users to create a new document from an open document you must provide a form action.

2. On Open:
   - The Automatically Enable Edit Mode option, when selected, opens the document in edit mode from either a Notes client or a Web browser.
   - The Show Context Pane option, when selected, displayes the parent document to end users in the preview pane of the Notes client.
3. On Close:
   - Select this option if you want the Mail Send dialog box to appear so that the users have the option of mailing, saving, signing, or encrypting a document.
4. On Web Access:
   - Treat Document Contents as HTML. When you select this option Domino passes all data on the form directly to the browser.
   - Generate HTML for All Fields allows the developer to use hidden fields for programming in Web applications.
   - Use the three color selections to determine how links will be displayed in a Web browser.

Using the Launch Tab

The Launch tab enables you to specify what happens when the document is opened.

Note The Auto Launch feature is not available for Web users.

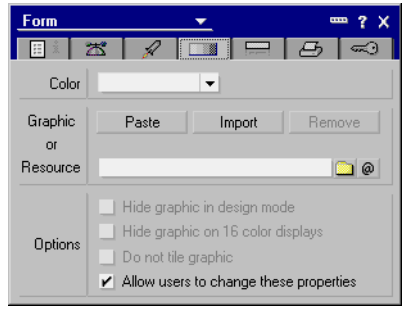


1. In the Auto Launch field, specify the type of action to take place when the document is opened. The following options are available:
   - None
   - First Attachment
   - First Document Link
   - First OLE Object
   - URL
2. If users are also going to use this form from the Web, make sure that *-None-* is selected in the Auto Launch field, because launch features are not supported on the Web.
3. The frameset drop-down list allows you to select a frameset which will open when the user opens the form.
4. When you have selected a frameset, you have to select a frame within that frameset where you would like to open the form.

   The frameset option works in the same way in a Notes client as in a Web browser.

Using the Background Tab

1. Click the Background tab to specify the options for the form background.



2. Specify the background color for the form using the Color drop-down box.
3. If desired, click the Paste Graphic button to paste a graphic image into the form. If the image is smaller than the form, Domino tiles the image to fit the size of the form.

   Note You need to copy an image to the clipboard before you click the button.

   Note Domino supports RGB colors, but if you want to insert very high quality graphics, select the option to import an image instead.

Tip Keep in mind that the cursor could be difficult to see on some displays if you choose a color like gray.

4. You can also import a graphic image into the form by clicking the Import Graphic button. Select the graphic image and click Import. The supported graphic formats are BMP, GIF, JPEG, PCX and TIFF 5.0.

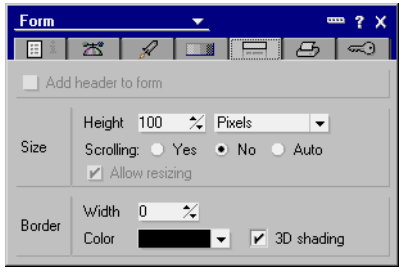
   Note If the image does not fill the entire page, it will be tiled automatically.

   Tip It is better to import rather than paste a graphic into the background because imported graphics are usually of better quality than pasted graphics. It is also easier to import a graphic than to copy the graphic to the clipboard and then paste it in.

5. By selecting Allow Users to Override Form Background Properties, users can change the background properties of a document that uses this form.

Using the Header and Footer Tab

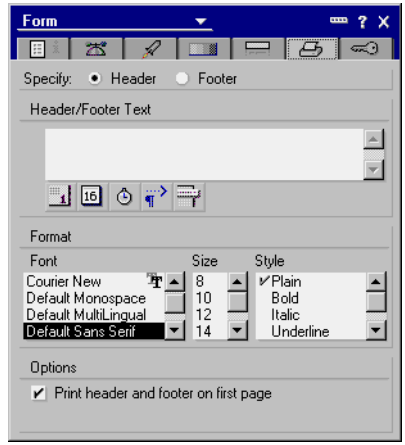
Header is a new feature in Domino R5.0 and works only in a Notes client. On the Web, Domino produces a table with the header information rather than a separate frame.



Headers can contain any element that a form can contain. The only caveat is that a table cannot be the first element in a header, it must be preceded by a text object, even if the text object is blank.

Using the Print Tab

1. Click the Print tab to specify the options that relate to printing a document based on the form.
2. Use the icons listed under the Header and Footer option buttons to define the date and time, tabs, and page numbering.

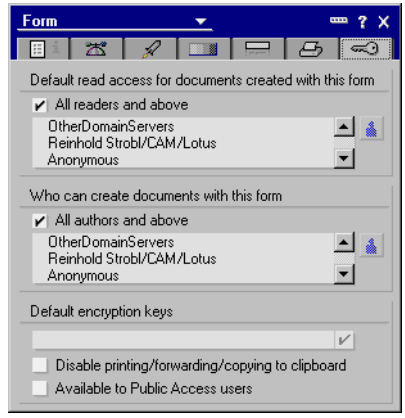


3. You can also select the font, size, and style.

Note The printing properties set here do not take effect if you are printing from the Web.

Using the Security Tab

Use the Security tab to define which users or user groups are authorized to use the form.



1. Deselect the All Readers and Above check box. This activates the blue person button to the right of the list.
2. Click the button. A window is displayed that allows you to select the users and groups from the different address books to which you have access.
3. Specify who can create documents with this form. The default is All Authors and Above. If required, deselect the check box, and click the

blue person button to the right of the list. A window is displayed that allows you to select individual users and groups.

4. If required, select Disable Printing/Forwarding/Copying to Clipboard. This makes it more difficult for users to distribute the documents created with this form to other users. It is recommended that you limit this option to confidential data.

   Note Selecting this option does not prevent the user from using other software to copy data to the clipboard.

5. Select Available to Public Access Users, if required. This allows users with No access or Depositor access to see specific documents and forms without giving these users Reader access to the entire database.

Giving the Form a Title

You can define actions to be performed when users trigger events as they compose, edit, or browse a document created with a form. The events can be defined in the Programmer's Pane which is displayed at the bottom of the screen. The Programmer's Pane is divided into two parts:

1. Objects view, which displays a hierarchical tree of Domino objects plus any you have added, with the associated events.
2. A second panel where you can enter different kinds of actions that define what events will do.

One event that you can define is the window title. When you design a form, make sure you give it a title. The title will then appear on the Notes title bar when a document is created or edited, based on the form in Notes and on the browser title bar when a document is created or edited in browser.

Note You should pay special attention to creating a window title when the application is to be used through the Web, because some of the functions used in the Notes client don't work on the Web.

Here is an example of a window title that can be used from either a Notes client or a Web browser:

```
@If(@IsNewDoc;"New Document";"Document: " + Subject + " created
by " + @Name([CN];@UserName))
```

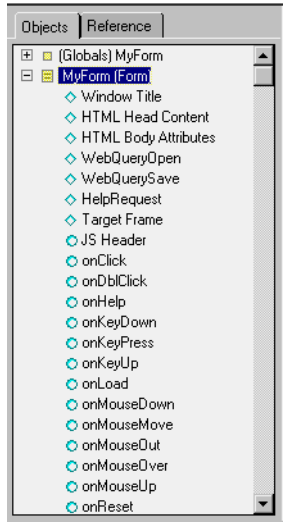
Subject is a field of the form. *@UserName* is a function that returns the whole name of the user. *@Name([CN];@UserName)* returns the common name part of the user name.

If the document is a new document (@IsNewDoc), the title is set to New Document. If the document already exists in the database, the title is set to Document: *Subject* created by *Username*.

Objects View

Objects view is one of the two parts of the Programmer's Pane. This pane displays a hierarchical tree of Domino's default objects, and the objects that you have created. Domino R5.0 provides a consistent programming environment across all of the Designer tools and supported languages. This new layout also helps you to see all of your code and available objects easily

The same pane is also displayed in the Reference View. It contains information about all Domino objects, formulas, and commands and it looks like this:



Form events are listed in the objects view and the following table lists all Domino form events:

Events	*Language*	*Description*	*Notes*	*Web*
Window Title	Formula	Gives the title for the form.	Yes	Yes
HTML Head Content	Formula	Contains all information, for example JavaScript, color and fonts, that occurs in the HTML Head section.	No	Yes
HTML Body Attributes	Formula	Contains all information that occurs in the HTML Body section.	No	Yes

continued

Events	*Language*	*Description*	*Notes*	*Web*
WebQueryOpen	Formula	Runs after a form is submitted from the Web. By default it runs an agent before a Web document displays. The formula must use the following syntax: @Command([ToolsRunMacro]; "agentname"). You can also enter different types of formulas here without calling an agent. **Tip** You can enter a LotusScript agent and then call that agent.	No	Yes
WebQuerySave	Formula	Must use the syntax: @Command([ToolsRunMacro]; "agentname"). You can also enter different types of formulas here without calling an agent. **Tip** You can enter a LotusScript agent and then call that agent.	No	Yes
HelpRequest	Formula	Allows you to display your own help for a form when the user presses the **F1** key.	Yes	Yes
Target Frame	Formula	The frame pages, views and forms will open if no default frameset has been specified in the hotspot action properties	Yes	Yes
JS Header	JavaScript	You can put JavaScript functions in here.	Yes	Yes
onClick	JavaScript	Runswhen an object is selected	Yes	Yes
OnDblClick	JavaScript	Runs when an object is selected with a double-click	Yes	No
onHelp	JavaScript	Runs when help is selected	No	IE only
onKeyDown	JavaScript	Runs when a key is pressed down	No	Yes
onKeyPress	JavaScript	Runs when a key is pressed	No	Yes
onKeyUp	JavaScript	Runs when a key is released	No	Yes
onLoad	JavaScript	Runs when document is being loaded.	Yes	Yes
onMouseDown	JavaScript	Runs when a mouse button is pressed down	No	Yes
onMouseMove	JavaScript	Runs when the mouse is moved	No	Yes
onMouseOut	JavaScript	Runs when the mouse is moved out of an object	No	Yes

continued

Events	*Language*	*Description*	*Notes*	*Web*
onMouseOver	JavaScript	Runs when the mouse is moved over an object	No	Yes
onMouseUp	JavaScript	Runs when a mouse button is released	No	Yes
onReset	JavaScript	Runs when document is reset by button. Triggers the form.reset.action.	No	Yes
onSubmit	JavaScript	Runs when document is submitted.	Yes	Yes
onUnload	JavaScript	Runs when a document is exited.	Yes	Yes
(Options)	LotusScript	Applies to all scriptable objects and provides an area for statements (Use, Option, Const, Def).	Yes	No
(Declarations)	LotusScript	Applies to all scriptable objects and provides an area where all global variables are declared.	Yes	No
Queryopen	LotusScript	Runs before a document is opened.	Yes	No
Postopen	LotusScript	Runs after a document is opened.	Yes	No
Querymodechange	LotusScript	Runs before a document is changed to Read or Edit mode.	Yes	No
Postmodechange	LotusScript	Runs after the user changes the document to Read or Edit mode.	Yes	No
Postrecalc	LotusScript	Runs after a document is refreshed (and values are recalculated).	Yes	No
QuerySave	LotusScript	Runs before a document is saved	Yes	No
PostSave	LotusScript	Runs after the document has been saved.	Yes	No
Queryclose	LotusScript	Runs before a document is closed.	Yes	No
Initialize	LotusScript	Runs when a document is being loaded.	Yes	No
Terminate	LotusScript	Runs after the document is closed.	Yes	No

Note Whether an event is supported on the Web or not depends on the browser used. Please see the Domino R5 Designer help: "Table of supported JavaScript objects for automated components" for more information on which browser supports which events.

Creating a Field

We will now take a look at how to create a field in a form. We will also see how to change the properties of the field.

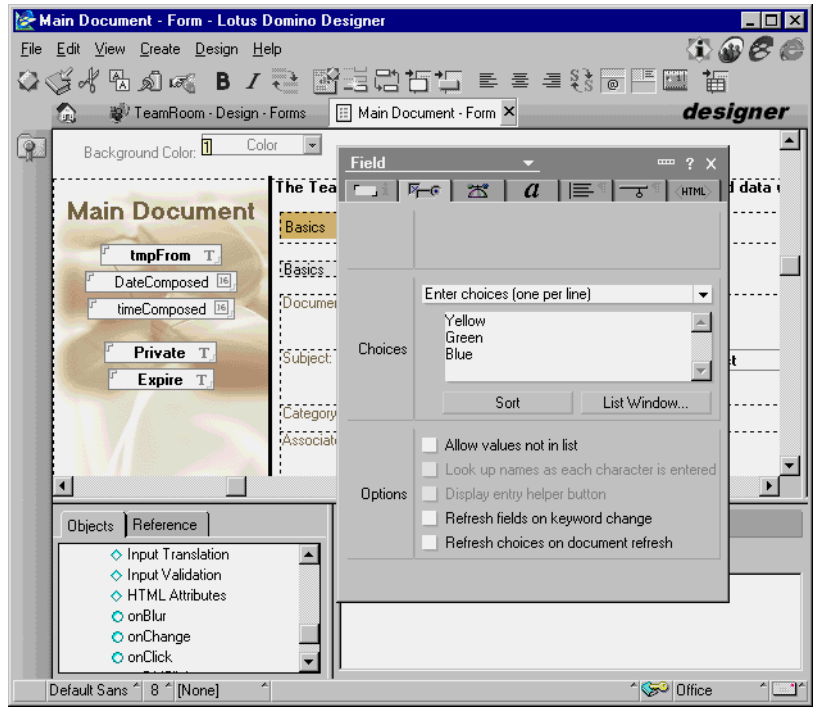
We will add a combobox field in the form. This field will demonstrate how to use keyword fields on the Web. The combobox field will allow you to choose one keyword. You will then change the color of the documents background on the Web.

1. From the standard navigator, choose Design, then Forms, and open the Main Document form listed in the view pane.
2. Type the static text, Background Color.
3. Next to the static text, create the field Color. To do so, choose Create - Field.

   Tip You can also display a pop-up menu by clicking the right mouse button and selecting Create Field.
4. On the InfoBox for Field, type a name for the field, for example, Color. The new name is now also shown in the Objects view.
5. In the Type field, choose Combobox.
6. In the field next to Type, leave Editable.
7. Enter 1 in the Tab field. This positions the cursor in this field when the document is opened.
8. Switch to the next tab.
9. In the Choices field, leave Enter choices (one per line).
10. In the Choices list box, type Yellow, Green and Blue. Separate the values by pressing **ENTER**.

    Note Remember to change the Frameset field in the forms InfoBox launch tab to -Blank-.

11. In the Programmer's Pane, leave the Default Value empty. The form and fields InfoBox should look like this:



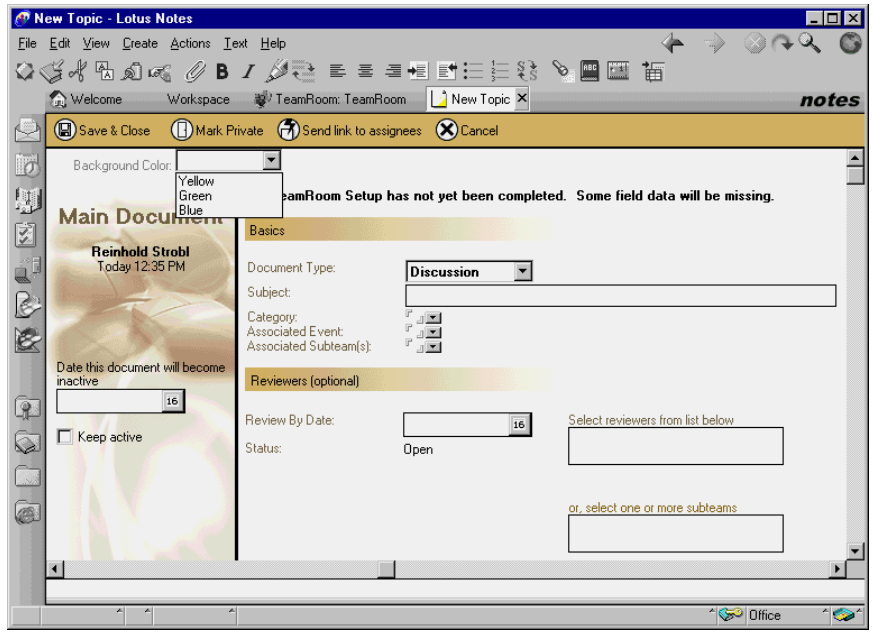
12. Check that on the ParagraphHide When tab no checkbox is selected.
13. Save the modified form by pressing the ESC key, and confirm that you want to save the form.
14. Close the information box that is displayed.

Performing a Test Run

To test your modification from a Notes client:

1. Open a Notes Client and select the TeamRoom™ database.
2. Double-click the TeamRoom database in your workspace. Once the database is open, choose Create - Main Document.

3. In the document, you will see that there is now an additional field. The document will look like this:



4. Select a color for the background and type a document title in the Subject box.
5. Click the Save & Close button on the Action bar.

   To see the document, select the By Date view and the document will appear in the list of documents.

   To test your modifications from a Web browser interface, you can use the Preview in Web browser option in Notes, which will enable you to preview your form without creating a document. To do this, choose Design - Preview in Web browser. If your database is on a Domino Server, Notes will then use the browser specified in your location document to preview the form. If your database is on a local machine, Notes will start the Local Web Preview Process.

   Note You can only preview local databases which are in the Notes data directory.

   Alternatively you can do the following:
6. Start your Web browser.

7. Enter the following command in the location field:

   ```
   http://"Server name"/Teamroom.nsf/Main+Document?OpenForm
   ```

   where *Server name* is the current server where the database is located, *Teamroom.nsf* is the database name and *Main+Document?OpenForm* opens the new document so that you can edit it.

8. You should see the new field in your browser.
9. Type in the Document title and choose the color for the background.
10. After you have completed the form, click Save & Close.

    Earlier we created a Color field to enable the user to change the background color of the form. We will now make some further updates to the form to add new functions:

11. From the Lotus Domino Designer R5.0, choose the Forms Design view.
12. Double-click the Main Document form in the View pane to open it.
13. Click the Color field.
14. Select the *onChange* event from the Objects view.
15. Type the following JavaScript code:

    ```
    document.bgColor=this.form.Color.options[this.form.Color.selectedIndex].text
    ```

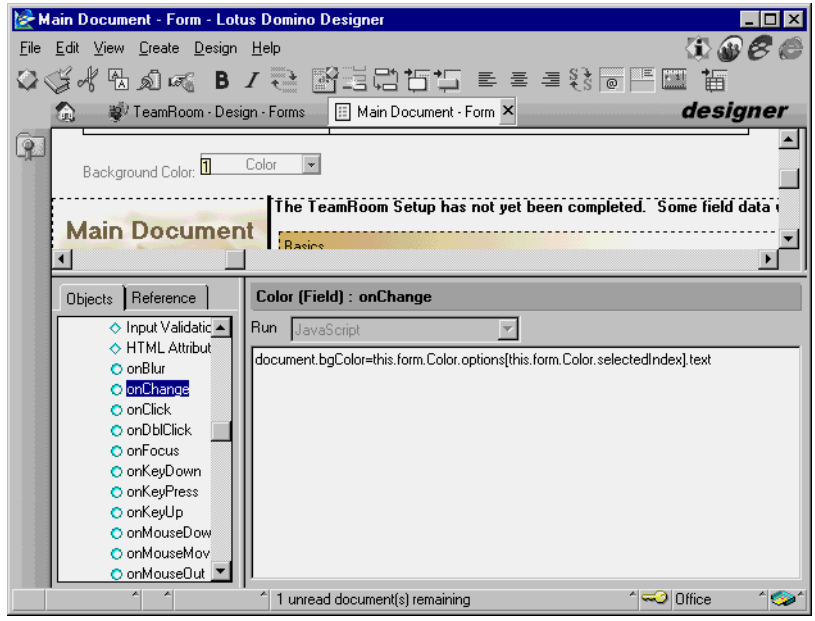
    OnChange event is a method which occurs when the value in the field is changed. The object *document* represents the current Web page, and it has a property *bgcolor* which is the background color of the page.

    ```
    this.form.Color.selectedIndex
    ```

    This returns an integer specifying the option selected in the field. Now that we know which of the options is selected, we can use it to return a text string with

    ```
    this.form.Color.options[this.form.Color.selectedIndex].text
    ```

16. After you have entered the code, the design should look like this:



Caution Unlike HTML and LotusScript, JavaScript is case sensitive.

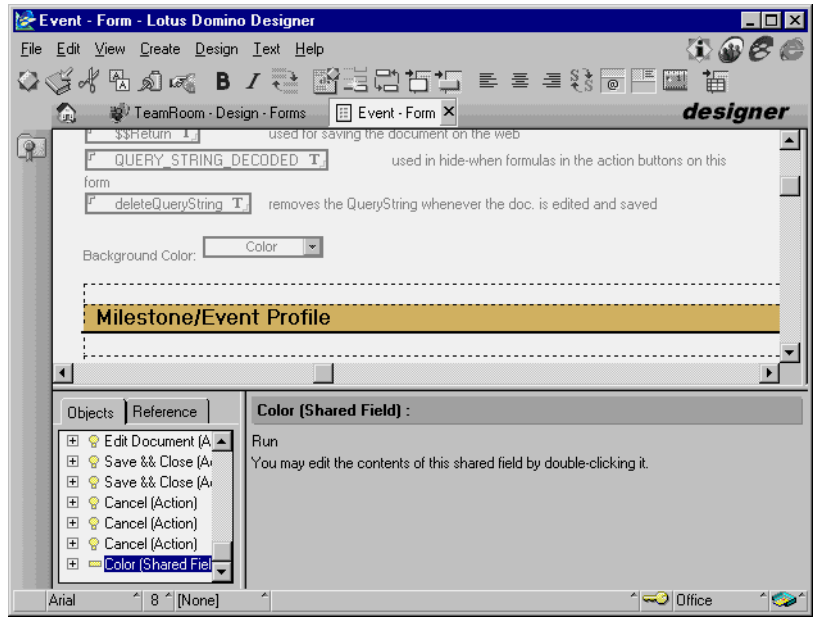
17. Preview the form in your browser.
18. When you select the value in the Color field, the background color of the page should change.

Sharing and Reusing a Field

You can only reuse fields in the database where the field has been defined as a shared field.

1. From Lotus Domino Designer R5.0, choose the Forms Design view. The list of forms is displayed in the view pane.
2. Double-click the Main Document form. The form is displayed.
3. Go to the Color field and select Design - Share this field.
4. Domino will automatically move that field to the Resources - Shared Field - View pane, where all database shared fields are stored.
5. In order to reuse the shared fields, choose the form where you want to add the field; for example, Event - Form. Choose Create - Insert Shared Field and a window will be displayed.
6. Select the Color and click OK. The field is now added into the form.

7. Press the ESC key and save the form. Now the design should look like this:



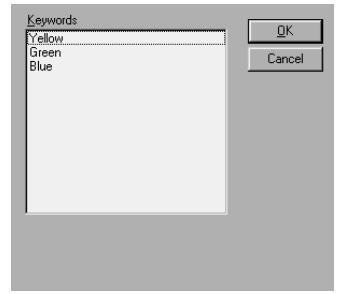
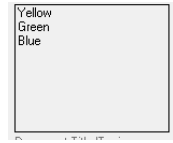
In this section we have only shown you how to create and run keyword fields, but all other fields work in a similar way. The following table lists the other field types and their explanations/declarations:

Field Type	*Declarations*	*Note*
Text	Normal text field, where user can enter text or numbers (if it is an editable field). Text can be string or variant type.	You can resize the field by using the field properties InfoBox and selecting the Use native control in Basic tab and changing the width and height in Options tab. Web doesn't support this option. The Designer can hide delimiters around the field by choosing the Hide Field Delimiter property.
Date/Time	Domino allows you to select different kinds of date and time formats. Time is date type. Field controls such as list boxes, and calendar controls are available on forms.	Domino supports four-digit year format and, with R5.0, Domino can also display a 4 character year field. Using calendar controls, enable "Use Native Control." Insert date and time in separate fields.

continued

Field Type	*Declarations*	*Note*
Number	Number field can count imported values. Number can be integer, float or double type.	The Designer can change the decimal symbols by changing On display preferences to Custom. The Currency option lets you select the right country currency, or you can customize it.
Combobox, Listbox, Dialoglist, Check box, Radio Button	These can be string or variant type.	These replace the keyword list fields in previous versions of Notes. By default the keyword list field type from R4 is automatically converted to the Dialoglist field type.
Rich Text	Domino allows a user to add text, attachments, Java applets, tables in this field. Rich text can be string or variant type.	Field value can be stored in MIME format.
Authors	Security field which allows designers to control form access. People, groups and servers who have been added in the field can open and edit documents. Author can be string or variant type.	The Choices option does not work on the Web.
Names	To help users enter names correctly in a document. Provides links to existing lists of names.	The Choices option does not work on the Web.
Readers	Security field which lets designers control form access. People, groups and servers who have been added in the field can open and read documents. Readers can be string or variant type.	The Choices option does not work on the Web.
Password	Users can add text. Each character is displayed with an asterisk (*).	Matches Web functionality.
Formula	Used for the Subscription feature. Provides a way to programmatically hold a formula that can be referred to by some other process.	

How keyword selections are displayed

Keyword	*Notes Client*	*Web Browser (IE 4.0)*
Dialog list		
Check box		
Radio Button		
Listbox **Note** Width and height do not work on the Web. It formats to the width of the longest string.		
Combobox **Note** Width and height do not work on the Web. It formats to the width of the longest string.		

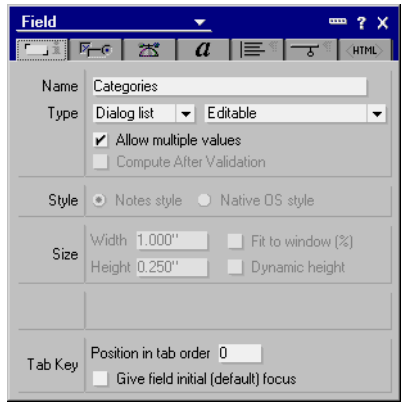
Note Choosing frame type has no effect on the Web.

Field Properties

Now that you have created a field, we are going to look at some of the properties of fields contained in the document form.

Let's first take a look at the Categories Field. We will look at the keyword field and explain the differences between this field type and other field types.

1. From the Lotus Domino Designer R5.0, choose Forms Design view.
2. Double-click the Main Document form in the view pane to open it.
3. Double-click the Categories field. The Field InfoBox is displayed and it looks like this:



The Field Info Tab

On the Field Info tab, Domino displays the field format. This field is of type Editable Dialog list field. There are several different ways of displaying the list of keywords from which users can make their selections. In our example, a View variable is given the "MissionLookup" view name (which is later used in @DBColumn) and an @DbColumn formula checks all the documents in the current database for categories and retrieves them for display in a keyword list.

The Field Info tab also shows how the data is actually put into the field. The following types of field are available:

- **Editable:** The user enters the data, or the data is created when the user selects a button performing a formula or script written by the developer.
- **Computed:** The field is computed each time the document is created, edited and saved.
- **Computed for display:** The field is computed each time the document is opened in browse or edit mode. The contents of the field are only visible while the document is open. It is not saved into the database and is not visible in a view.

  For example, this type of field is used to display the current time and date or work variables, such as the server name where the database is stored.

- **Computed when composed:** The field is only computed when the document is created. This type of field is especially useful for storing the name of the author of the document, the creation date or a document reference number.

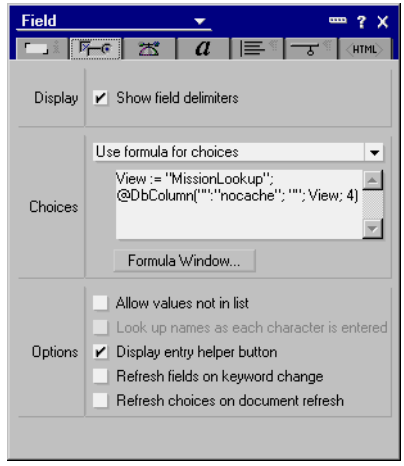
There is also a check box to allow multiple values to be selected at once.

Tab Order properties allow you to select the "time" when the user comes in to this field while editing the document and moving ahead from field to field using the Tab key.

Note Tab Order default value is 0 (zero).

Control Options Tab

The second InfoBox tab is Control options. It looks like this:



There are a number of check boxes:

1. **Show field delimiters.** This shows the field delimiters of the field.
2. **Allow values not in list**. By choosing this option you give the user a chance to enter a value which is not on the list.

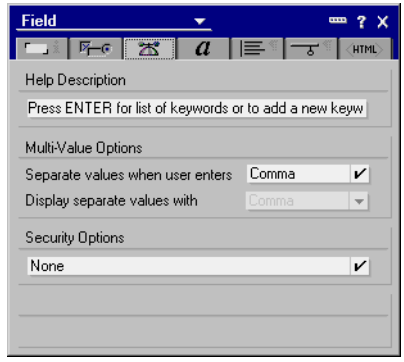
   Note This feature is not available for Web users.
3. **Display entry helper.** This displays the small grey down arrow in the bottom right hand corner of the field to tell the user that there are multiple options to select from.
4. **Refresh fields on keyword change.** This will force the document to refresh its fields if the value in the field is changed.

   Note This should be used sparingly as it can cause the user to become frustrated if the form contains a large number of fields.

5. **Refresh choices on document refresh.** This option is used to refresh the values in the field (usually based on a formula) if the document is refreshed.

Advanced Tab

The third tab is Advanced and it looks like this:



This tab enables you to specify:

- Field help: information is shown at the bottom of the Notes client screen.

  Note Field help is not available for Web users. If you require field help in a browser application, you could use the JavaScript onFocus event to update either the message area of the browser window or a separate field.

- Give field default focus: Here you can specify whether the entry field will have the initial focus when the form is opened. You must specify this option if you want to place the cursor in an entry field that is not the first one on the form. You can also use this option if you want to paste data in a particular entry field before placing the cursor in its final position.

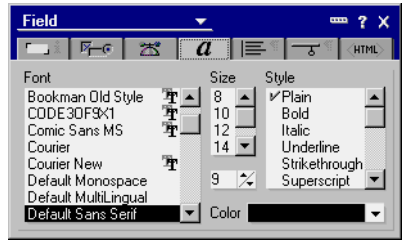
- Multi-value separators.

- Security options: for example, Enable Encryption for this field.

  Note Security options are not available for Web users.

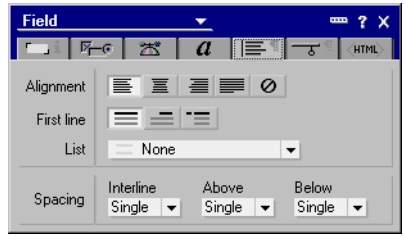
Fonts Tab

The fourth tab of the InfoBox lets you specify fonts and colors for the field data. The tab looks like this:



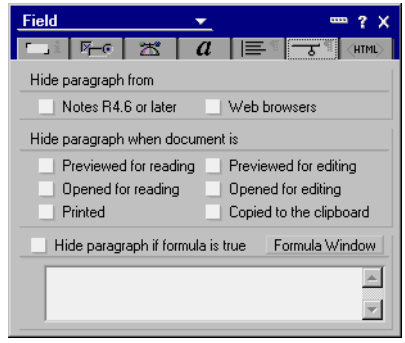
Alignment Tab

The fifth tab is the Alignment tab, which lets you specify the alignment of the paragraph containing the field. For example, you should use this option if you define a field to be used as the title. If you choose to align it in the center of the form, it will stay in the center irrespective of the screen resolution used. The Alignment tab looks like this:



Hide When Tab

The Hide When tab looks like this:



Example: You can use a formula to restrict a field so that only one group of people can see it:

- Create a Friend group in the server's Public Address Book.

- Select the Hide Paragraph if Formula is True option and add the following formula in the formula window:

```
!@Contains("Friends"; @UserNamesList)
```

The @Contains() formula checks if the Friends group exists in the list returned by @UserNamesList.

Note The exclamation point character (!) is the logical *not* operator.

The @UserNamesList formula returns a text list of all the users who are authorized to see that database. If the user has been added to the group named Friends, they can see the field and database.

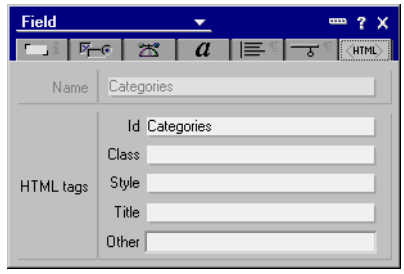
Several check boxes are available to hide the paragraph on predefined conditions. You can also specify other conditions using an @function.

Tip Whenever possible use the predefined conditions for better performance.

Note The InfoBox of each design element found in a form provides a tab that allows you to specify hide-when conditions.

HTML Attributes Tab

The last tab is the HTML attributes tab which enables you to name your field so that it can be accessed by JavaScript. By default its name is the same as the field name. The HTML attributes page looks like this:

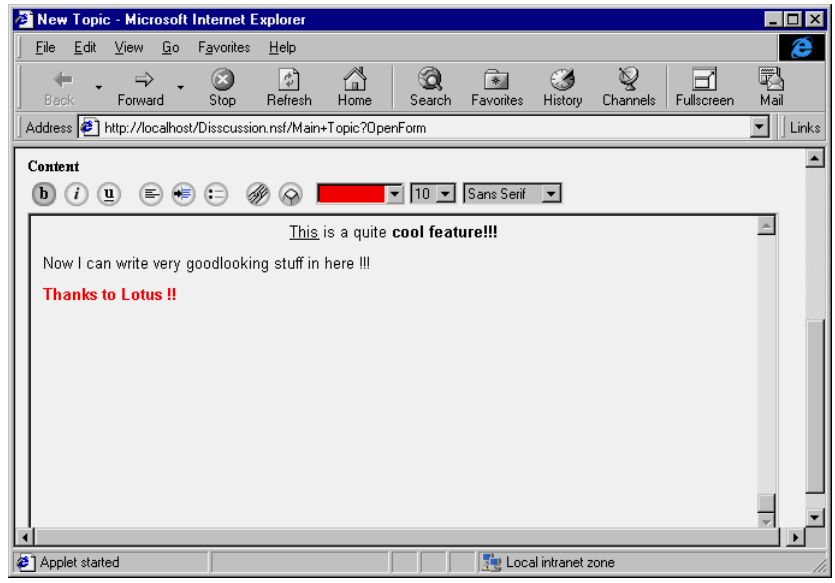


Rich Text Field (RTF) Applet

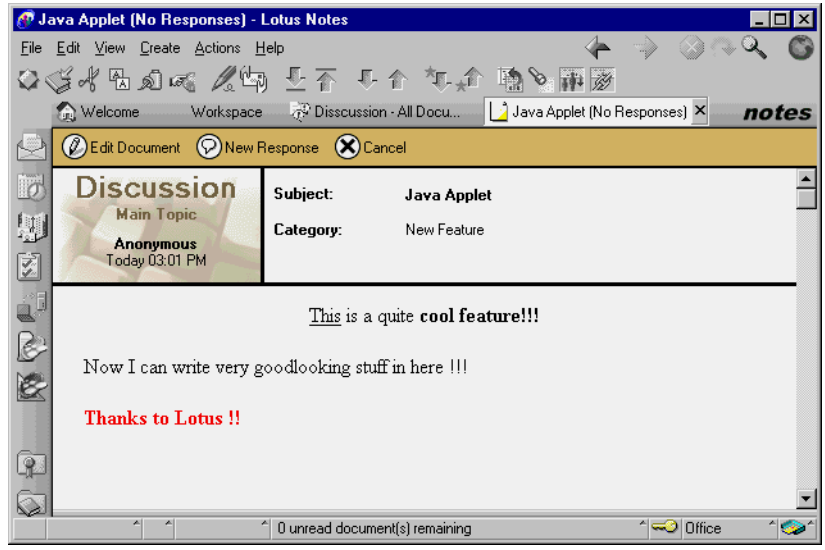
This is a good option when you want to give Web users opportunities to write different styles of text. One example where this field is very useful is in a feedback form. After the user has submitted feedback, Domino saves the document in the database. Using the RTF applet, Domino also saves the text format and style, which means that the text is stored in exactly the same format and style as when the user entered it. Let's see how this option works.

1. Open a Document form.
2. Go to the Body field and open the field's InfoBox. You can see that the field is RichText type and the Use Applet In The Browser option is enabled.

3. When you run the form in Notes it looks like an ordinary RichText field, but when you run the form in a Web browser it should look like this:



4. After the user has submitted the document, if you look at the document using a Notes client, it should look like this:



As you can see, the field contents are stored exactly as the user entered them.

Using the $$Return Field

The $$Return field is used among other things, for creating messages that will be displayed after the user has submitted a document on the Web. Without the $$Return field Domino responds with the default response "Form processed." To customize this response message, you can include HTML code as part of the formula for the $$Return field.

You can also use a $$Return field to run a custom CGI (Common Gateway Interface) program immediately after the user submits the form and Domino has created the document. For example, you can run a CGI program that uses the Notes API to further process the input data. The Web client displays the output of the CGI program to the user.

To run a CGI program, include the URL to the CGI program file and enclose it in brackets. Note that you can pass arguments, for example, values from fields in the form, to the CGI program.

Returning to Another Page

You can display another Web page to the user once a form has been processed instead of leaving a blank, gray screen. The following example displays the Lotus home page, but you can direct the user to the main view of the database instead, for example.

```
"[http://www.lotus.com]"
```

Adding a Link to Another Web Page

In the response message, you can include links to other Web pages. In this instance an HTML page will be created with a link to the Lotus home page. The user will see a blank screen with a link to the specified Web page once the form has been processed.

```
"<a href="http://www.lotus.com">Lotus</a>"
```

Adding links is useful if, for example, you want to provide the user with a choice of Web pages to visit once a form has been processed.

Personalized Messages

You can create a personalized message for the user who submits a form. For example, the following $$Return formula returns the response "Thank you for your document," and appends the user's name. <h2> </h2> is an HTML tag and means that text between those tags is a second level header. <hr> creates a horizontal rule.

```
"<h2>Thank you for your document, "+@Name([CN];@UserName) +"!
</h2><hr>"
```

Note For the @Name function to work, you need to authenticate with the Domino server when first opening the database. If you did, you would be classified as an Anonymous user of the application.

$$Return Example from TeamRoom Template

The TeamRoom template has a rather complex $$Return field, but going through the code gives you some good examples of how you could use this field. The $$Return field is a shared field and you can open the field by going to the Resources - Shared Fields Designer and double-clicking the field in the View pane.

You can use the REM command to add a comment for the $$Return field. For example:

```
REM "This $$Return field returns HTML as a result of the
successful form submittal.";
REM;
```

First, simple strings are assigned to variables to make it easier to write and to read formulas:

```
REM "Variables to translate";
PrevDoc := "Open the page you just submitted";
Another := "Create Another: ";
TRreturn := "Return to the TeamRoom";
Done  := "Done";
Subteam := "Subteam Profile";
Participant := "Participant Profile";
Event := "Event Profile";
MainTopic := "TeamRoom topic";
Mission := "Mission Page";
Response := "Response";
IntProfile := "Interest Profile";
ArcProfile := "Archive Profile";
TeamStatus := "Team Status";
SubteamStatus := "Subteam Status";
```

Next, the formula takes the current user name and adds the “Thank You, “ text into the ThankPerson variable:

```
ThankPerson := "Thank you, " + @Name([CN]; @UserName);
ThankYou := ThankPerson + ". The following page has been
successfully submitted: ";
REM "End variables to translate";
REM "Get the name of this database.";
```

The following formula gets the name of the current database, and replaces any spaces with the plus (+) character and replaces any backslash characters with the forward slash character (/). @DbName is a function which returns the name of the current Domino server and the name and the path of the database. @Subset with the -1 parameter returns just the database name and path:

```
DB := @ReplaceSubstring(@ReplaceSubstring(@Subset(@DbName; -1);
" "; "+"); "\\"; "/");
```

Next, you get the value from the webButtonPressed field:

```
FIELD webButtonPressed := webButtonPressed;
```

As the $$Return field is a shared field in this database, the formula first determines which form was used:

```
FormName := @If(Form = "MainTopic"; MainTopic;

Form = "Subteam"; Subteam;

Form = "ParticipantProfile"; Participant;

Form = "Event"; Event;

Form = "Response"; Response;

Form = "ResponseToResponse"; response;

Form = "Interest Profile"; IntProfile;

@Contains(form; "Archive"); ArcProfile;

Form = "Mission"; Mission;

Form = "Status"; TeamStatus;

Form = "SubteamStatus"; SubteamStatus; "");
```

Next, we format the message to return to the user.

```
Thanks := "<h3>" + ThankYou + Formname + "</h3><hr>";
```

Next, you create a link so that the user can return to this document. *@Text(@DocumentUniqueID)* returns the unique ID of this document:

```
existingdoclink := "<b><font size=2><a href=/" + db +
"/($All)/" + @Text(@DocumentUniqueID) + "?OpenDocument>" +
PrevDoc + "</a></b>   ";
```

The String variable contains a link which returns the user to the WelcomePage:

```
LinkTRReturn :=   "<a href=/" + db + "/WelcomePage?OpenPage>" +
TRReturn + "</a>";
```

Next, we create a button which closes the current window:

```
LinkDoneButton := "<FORM><INPUT TYPE=\"button\" VALUE=" + Done
+ " onClick=\"window.close(self)\"></FORM>";
```

The next three string variables contain links to the different forms:

```
LinkCRParticipant := "  <b><font  size=2><a href=/" +
DB + "/" + "ParticipantProfile?Openform>" +  Participant +
"</a></b> ";

LinkCRSubteam := "  <b><font size=2><a href=/" + DB +
"/" + "Subteam?Openform>" + Subteam + "</a></b> ";

LinkCREvent := "  <b><font size=2><a href=/" + DB + "/"
+ "Event?Openform>" + Event + "</a></b> ";
```

The *bkgd* variable contains a background color (#ffffff = white):

```
bkgd := "<body bgcolor=\"" + "#ffffff" + "\"+ >";
```

We then concatenate the variables into two text strings:

```
REM "Assemble the HTML to be returned";

OkMsg := bkgd + Thanks + existingdoclink + LinkTRReturn;

CancelMsg := mkgd + Thanks + LinkDoneButton + Another +
LinkCRParticipant + LinkCRSubteam + LinkCREvent;
```

Finally, an @If formula checks which of the two strings should be returned as a response to the user:

```
REM "Because the Part Profile, Subteam, and Event are created
in a smaller window, we\'re using a different msg. when
they\'re new docs.";

@If(webNewDoc = "1"; CancelMsg;OKMsg )
```

Note You can also use the onSubmit event in the form to control the forms submit process. To use the onSubmit event you will need to use JavaScript.

Tip Good JavaScript commands to use when the form has been submitted are *history.go() and history.back()*. For example, when you have submitted a form and you want the user to return to the view where they came from, and you don't want to use $$Return.field, you can type the following line into the onSubmit event:

```
History.go(-1) or history.back()
```

This JavaScript command allows the browser to step back one page in the history.

Field Events

Field events are functions that can be implemented using formulas, LotusScript, or JavaScript.

The following table lists all the Fields events:

Events	*Language*	*Description*	*Notes*	*Web*
Default value	Formula	When the document is loaded the contents of the Default Value event are displayed.	Yes	Yes
Input translations	Formula	Can be used to modify the data entered by the user, to trim blanks, and to change users' names into uppercase or propercase.	Yes	Yes
Input validation	Formula	Requires a choice from a list, which is done using an @If formula.	Yes	Yes
HTML Attributes	Formula	Allows you to add extra HTML attributes to the field tag that Domino generates.	No	Yes
onBlur	JavaScript	Runs when user exits the field.	Yes	Yes
onChange	JavaScript	Runs when the field value changes.	No	Yes
onClick	JavaScript	Runs when field is clicked.	No	Yes
onDblClick	JavaScript	Runs when field is double-clicked.	No	Yes
onFocus	JavaScript	Runs when user gives input focus to the field.	Yes	Yes
onKeyDown	JavaScript	Runs when a key is pressed down.	No	Yes
onKeyPress	JavaScript	Runs when a key is pressed.	No	Yes
onKeyUp	JavaScript	Runs when a key is released.	No	Yes
onMouseDown	JavaScript	Runs when a mouse button is pressed down.	No	Yes
onMouseMove	JavaScript	Runs when the mouse is moved.	No	Yes
onMouseOut	JavaScript	Runs when the mouse is moved out of an object.	No	Yes
onMouseOver	JavaScript	Runs when the mouse is moved over an object.	No	Yes

continued

Events	*Language*	*Description*	*Notes*	*Web*
onMouseUp	JavaScript	Runs when a mouse button is released.	No	Yes
onSelect	JavaScript	Runs when a user selects text in a text field.	No	Yes
(Options)	LotusScript	Applies to all scriptable objects and provides an area for statements (Use, option, Const, Def).	Yes	No
(Declarations)	LotusScript	Applies to all scriptable objects and provides an area where all global variables are declared.	Yes	No
Entering	LotusScript	When cursor enters a field.	Yes	No
Exiting	LotusScript	After cursor exits the field.	Yes	No
Initialize	LotusScript	When field is being loaded (user clicks the button, for example).	Yes	No
Terminate	LotusScript	When field is being closed.	Yes	No

Note The onFocus event is not triggered if the field gets the initial focus when a document is opened.

Sharing Design Elements With Subforms

Subforms provide a way to share fields between groups of design elements.

All design elements that are added to forms can also be used in subforms. These include:

- Static text and pictures
- Fields, whatever their type and format
- Hotspots as buttons or links
- Tables
- Action Bar
- Java applets
- Embedded elements
- Another subform

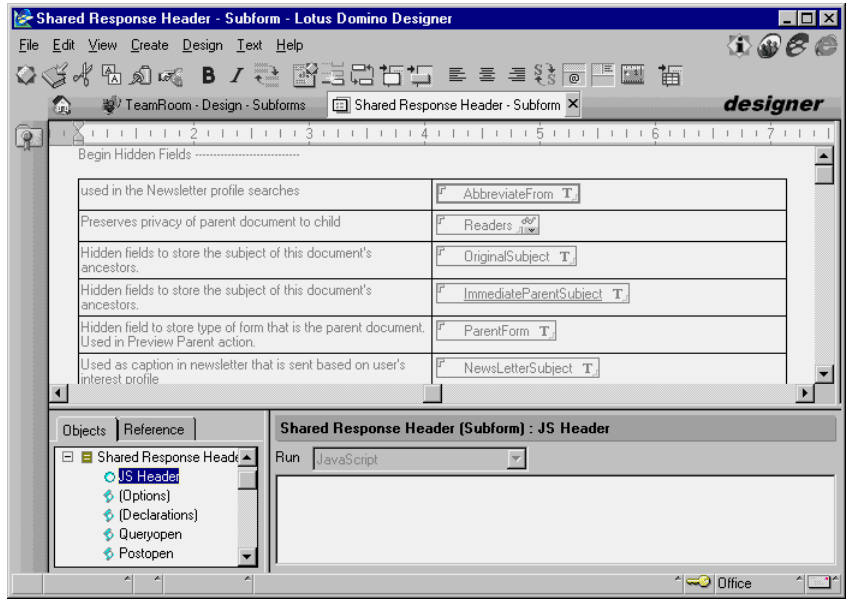
When you modify an existing subform, the changes are immediately reflected in all the forms that use the modified subform.

Note You can insert subforms into a table or even another subform.

A subform is provided with the TeamRoom template. You can work with its design in one of the following ways:

1. From Lotus Domino Designer R5.0, select the Subforms design view. The list of subforms is displayed in the view pane. Double-click the SharedResponseHeader subform. This subform is used to share hidden fields which are common to both Response and Response To Response forms. The Subform Builder window is displayed.
2. Alternatively, you can open the subform directly from the form. To do this, open the form that contains the subform (Response or Response To Response). Once the form is open, double-click the subform part of the form. The Subform Builder window is displayed.

   Tip You might have to scroll through the form to see the subform part. The following figure shows you that the Subform Builder window is identical to the Form Builder window:



It contains:

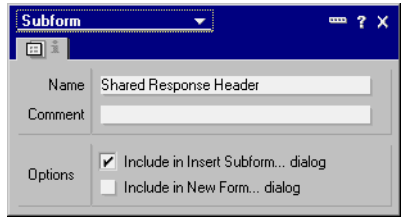
- The form in the design pane.
- The actions linked to the subform in the action pane. When a form and a subform are displayed, the action bars of both the form and the subform are shown.
- The field definition in the Programmer's Pane. In subforms, as in forms, @functions, LotusScript and JavaScript can be used.

Note The subform does not contain the Default title, onSubmit, onReset, HTML Head Attributes, HTML Body Attributes, WebQueryOpen, WebQuerySave, OnHelpRequest events, because the subform is always linked to a form and the form already contains those events.

Subform Properties

To display the subform properties:

1. On the Subform pane, click your right mouse button.
2. Select Subform Properties and the InfoBox is shown. It looks like this:



3. Close the InfoBox.
4. Close the subform.

Removing Subforms

You can remove subforms from the design of a form, or from the design of a database.

Removing Subforms From the Form Design

If the subform is no longer needed in a particular form:

1. Open the design of the form.
2. Click the subform area.
3. Choose Edit - Clear on the menu bar.

Removing Subforms From the Database Design

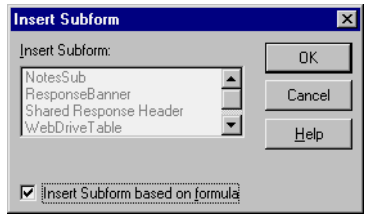
As for all design elements, you can remove subforms from the design of the database. For example, if all the fields contained in the subform are no longer needed in any of the database forms.

However, if the database contains documents using the deleted subform, you have to make sure that the users can still access the documents by having an empty subform that uses the same name as the deleted subform.

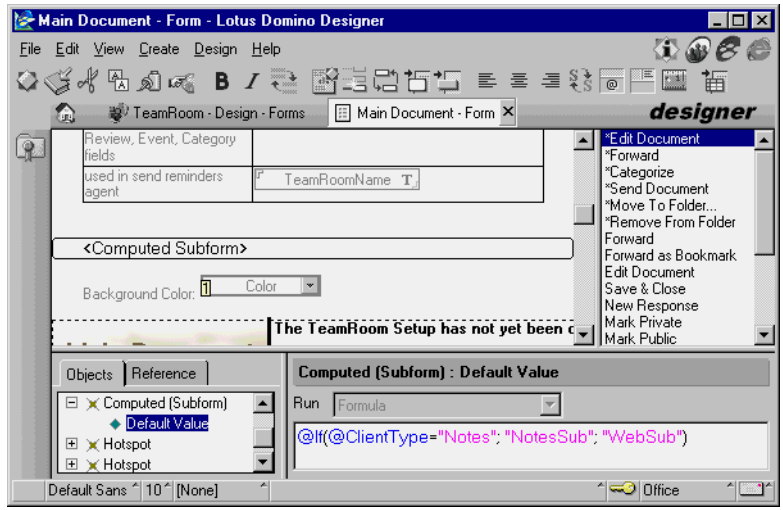
Computed Subforms

You can use computed subforms to show different elements to different users. Which subform is loaded is based on a formula, so you can load different subforms for different groups. You can also use different subforms for Web users than for Notes users. To do this:

1. First, you have to create some subforms, at least two.
2. When your subforms are ready, open the Main Topic form.
3. Create a computed subform by choosing Create - Insert Subform. A dialog box is displayed:



4. Select Insert Subform Based on Formula and click OK.
5. A computed subform is created on the form and it looks like this:



You have to specify a formula in the Programmer's Pane. The formula returns a text string which is the name of the subform to be loaded.

Looking at a Computed Subform

1. You should still be in the document form.
2. Click the Computed Subform once.

3. Add the following formula in the Computed(subform) - Default Value - Events Programming pane:

   ```
   @If(@ClientType="Notes";"NotesSub";"WebSub")
   ```

   @ClientType is a formula which determines whether the user is using a Notes client or a Web client. The result of the formula above is a text string, NotesSub or WebSub. If the user is using a Web client, the WebSub subform is loaded. If the user is using a Notes client, the formula returns the string NotesSub and the NotesSub subform is loaded.

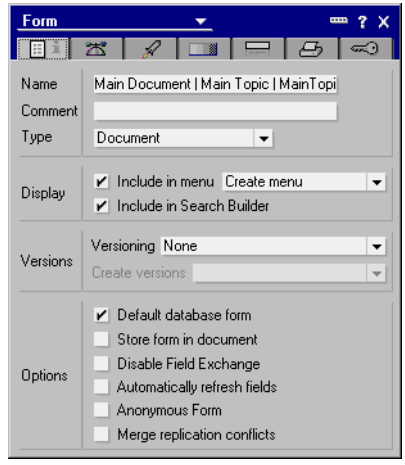
   Note If the formula returns an empty string, no subform is loaded.

Using a computed subform is a good way to show some elements only to one type of client, or to users who have different roles. For example, one user can read information, or provide content, to the Web site while another user may have the authority to approve information for the Web.

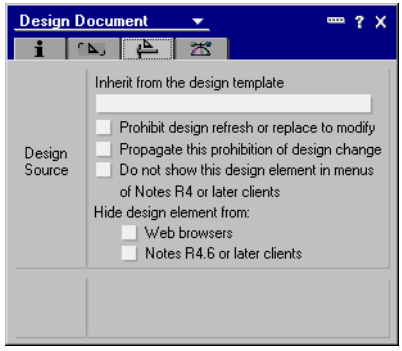
Displaying a Different Form to Web Users and Notes Users

Maybe the easiest way to show different things to different users is to use different forms. This is most useful when the information you want to show to Web users and Notes users differs considerably, or where Web browsers do not support the features used in your forms. To use different forms for Web users and Notes client users, do the following:

1. First of all you need to have two forms: one for Web users and one for Notes users.
2. Make sure that both forms have the same alias name. The form InfoBox looks like this:



3. The names of the forms can also be the same, but it's much more useful to give them different meaningful names. In this way, you can easily determine if the form is for use by Web users or by Notes users. For example, you could name them as Main Topic (Notes) and Main Topic (Web).
4. After creating the two forms you need to make them available only for Web users or Notes users. From the standard navigator, choose Design, then Forms. The list of forms is displayed. Click the form and click the Properties icon. The InfoBox is displayed. Go to the Design tab. It looks like this:



5. On the Hide Design Element From section, select Web browser if the form is only to be used in Notes, and select Notes R4.6 or later clients if the form is only to be used on the Web.

Note All Notes elements can be hidden from a Notes client or a Web client.

When the user opens a document, whether in Notes or on the Web, the correct form will then be used to display the document.

Working With Layout Regions

A layout region is generally used when creating dialog-like forms and in @Dialogboxes. It consists of a 32-bit graphic that contains several kinds of design elements, such as fields (except for rich text, password and formula), static text and buttons. However, Java applets, objects, attachments and embedded objects are not allowed.

Creating a Layout Region

To create a layout region, do the following:

1. From Lotus Domino Designer R5.0, choose the Forms design view.

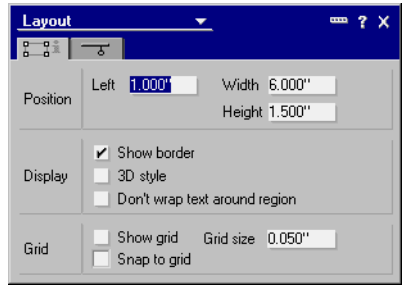
2. Open up a form listed in the View pane, for example, the Author profile form.
3. Position the cursor in an empty area of the form.
4. Choose Create - Layout Region - New Layout Region from the menu bar. An empty frame is built in the form.
5. If you want to add a picture, choose Create - Picture. The box is displayed where you can browse and import that picture.

   Note Domino supports .BMP, .GIF, .JPEG, .CGM, Lotus Pic, Tiff 5.0 and .PCX graphic files.
6. To add static text, choose Create - Layout Region - Text or use Create Static Text - SmartIcons (Create text box). A frame is created within the graphic. You can move the static text frame around. If you double-click the static text frame, you display the Control box.
7. To create fields, choose Create - Fields as in any other form.

Layout Regions InfoBox

To open the layout region that was used to create the Review Options form, follow these steps:

1. From the standard navigator of the Document Library database, choose Design, then Forms. The available forms are listed in the View pane.
2. In the View pane, double-click ReviewOptions to open up the form. It contains the layout region you were looking at earlier.
3. Click the bitmap.
4. Click the Properties icon. The following InfoBox is shown:



   The InfoBox has two tabs:

   - On the Basics tab, you can adjust the dimensions of the layout region. You can also display a grid to position the fields and static text within that region.
   - On the Hide tab, you can elect to hide the design element. Several options are available. You can also define an @formula to hide the design element.
5. Close the InfoBox.

Working With Collapsible Sections

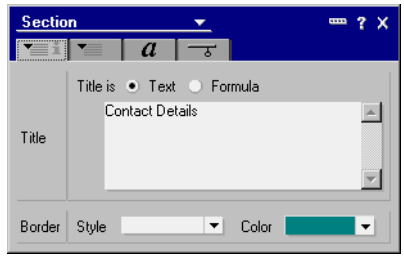
If the form design includes a long set of fields or fields that contain large amounts of data, it can be annoying for users to have to scroll up and down to find the information they are looking for. Collapsible sections can be a good solution to this problem.

Creating a Collapsible Section

To create a collapsible section within a form, follow these steps:

1. Open the design of a form.
2. Choose Create - Section - Standard if you want the section to be seen by all the users that have access to the document.
3. Alternatively, choose Create - Section - Controlled Access if you want to restrict access of the section to certain users defined in a formula.

   The InfoBox for a standard section looks like this:



4. The Title tab allows you to give a name to the section. You can also use formulas to name sections. Other properties that can be changed here are the border type and color.
5. The Expand/Collapse tab enables you to define expand and collapse rules when the document is previewed, opened for reading, opened for editing, or printed.
6. The Font tab enables you to set the text type and color.
7. The Hide When tab lets you specify when that section is displayed.
8. The HTML tab allows you to specify HTML options. For example, you can name your section using Title options in HTML.

Using Tables

There are many new features related to tables in R5.0, with a focus on increasing the variety of types of tables that can be used in Domino applications designed for use both by Notes clients and by Web browsers.

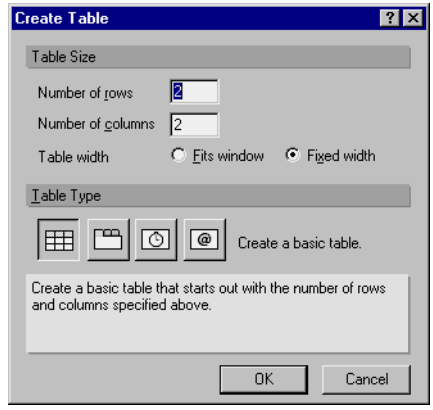
Tables are especially useful for applications on the Web. By using tables, you can be sure that fields are aligned correctly and that the images are positioned correctly.

With R5.0 you can include fields, graphics, buttons, subforms, hotspots, objects, sections, nested tables, attachments, Java Applets, and embedded elements inside a Domino table.

To create a table, follow these steps:

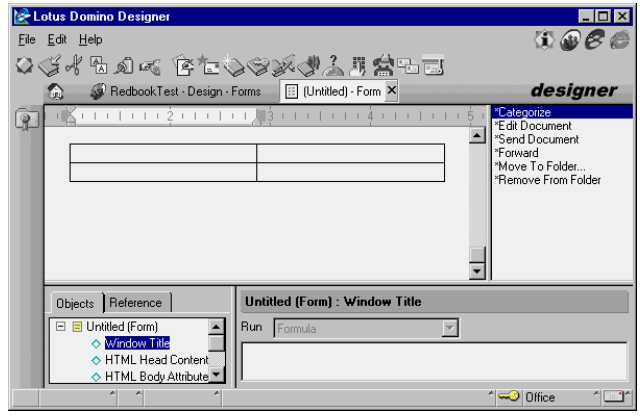
1. In your form, click where you want to create the table.
2. Choose Create - Table.

   The Create Table dialog box is displayed:



3. Select the type of table you would like to create.
4. Specify the number of rows in the Rows field.
5. Specify the number of columns in the Columns field.

The table is then created on the form and it will look like the one shown below:



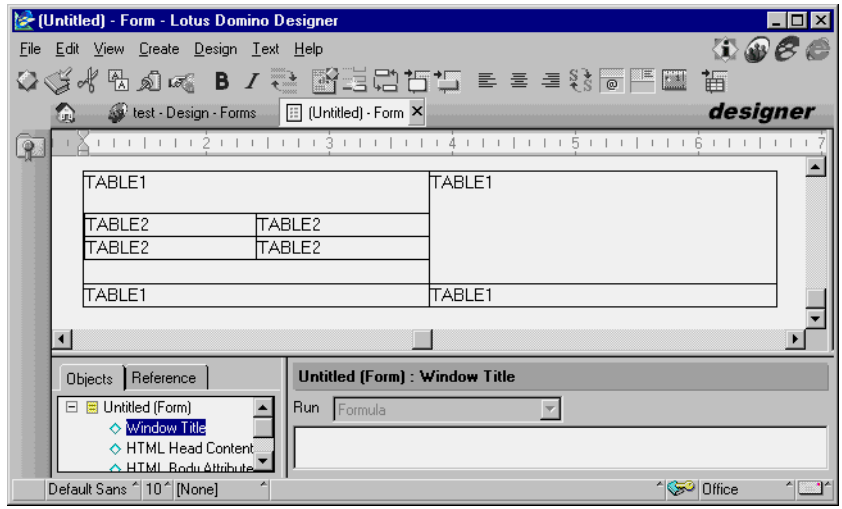
Create Tables Within Tables

Lotus Domino R5.0 also supports nested tables (up to four levels deep). This is useful for developing tabbed tables both for Notes client users and for Web sites when you want to be sure that fields and graphics are aligned correctly.

To create a nested table, follow these steps:

1. Select the table and position the cursor within the cell where you want to create a nested table.
2. Choose Create - Table.
3. Specify the number of rows in the Rows field.
4. Specify the number of columns in the Columns field.

The nested table is then created inside the table, and it will look like the one shown below:



If you want to use this nested table on the Web, you must highlight the whole table and choose Text - Pass Thru HTML. This causes Domino to translate the whole table into HTML, so that the nested table will be displayed correctly on the Web.

Note When you are using tables on the Web you must insert information in all the cells (even if it is only a dot), otherwise the empty cells will not be displayed.

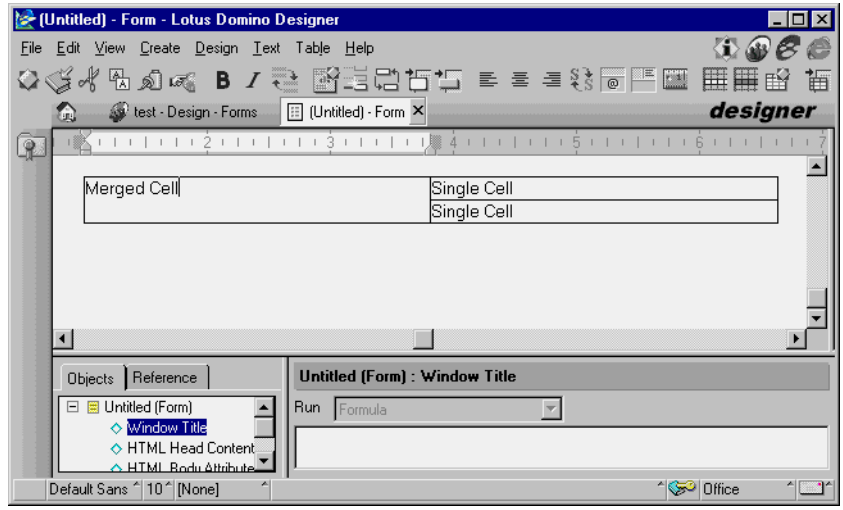
Merge and Split Cells

Table cells can also be merged into one cell. Domino translates this into the proper rowspan and colspan attributes.

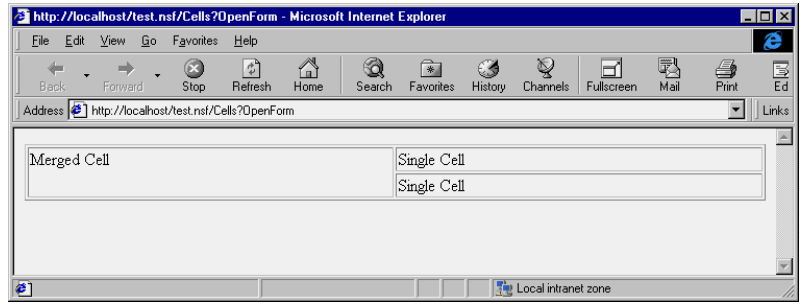
The following example demonstrates how to create a table of four cells and then merge two of them into one:

1. Create a blank form by choosing Create - Design - Form from the Lotus Domino Designer R5.0 client.
2. Create a table with two rows and two columns.
3. Highlight the two leftmost cells.
4. Choose Table - Merge Cells.

You can see that these two cells are now merged into one cell as shown below:



Let's preview the same form on the Web.



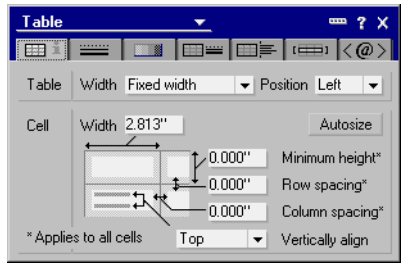
5. To split cells, click on the merged cell and choose Table - Split Cells.

Table Properties

There are seven available tab options where you can modify and concatenate tables:

Table Layout Tab

This is the first tab and it looks like this:



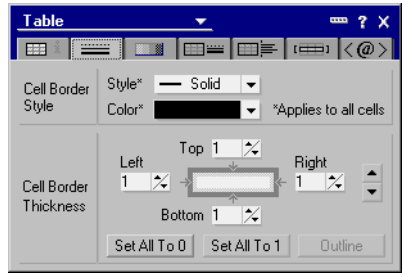
1. The Width option allows you to specify how the table appears to the user. There are three different ways to show the table:
   - **Fit to window:** The table uses the same width definitions as the current window.
   - **Fit with margins:** The table size changes when the user changes the window size. Fit with margins shows all the table cells at the same time so users don't have to use the horizontal scroll bar.
   - **Fixed:** The designer can manually determine the size of the table. The table width does not change if the user changes the window size.

   Note All of these options are supported on the Web.

   Note Only those cells that contain extra space will be resized. If the picture or field size is exactly the size of the cell, that cell cannot be resized.
2. If Fixed is selected, then you can select the table's alignment. Options are left, center, and right of the window.
3. Cell width: allows you to set the width of the cell.
4. The Space Between Columns and Space Between Rows options allow you to specify the distance between the cells or rows.

Cell Borders Tab

The second tab looks like this:

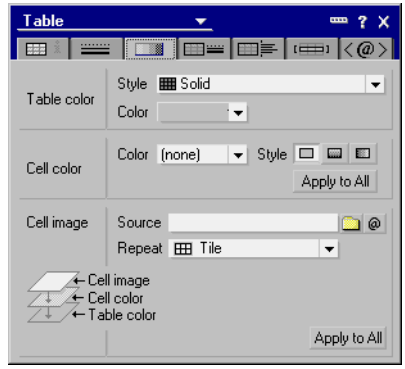


- Cell Border Style allows you to select the style of the cell borders. Available options are Solid, Ridge, and Groove.
- Color allows you to select the color of the border lines for the whole table.
- The thickness of the borders can be set from 0 to 10. You can set all the borders of selected cells to 0 or 1 by clicking one of the buttons at the bottom of the screen. If you have selected more than one cell, you can outline them by clicking the button at the bottom right of the section.

Note For the Web, there are only two options for borders, on or off. This is determined by the borders of the top left cell.

Table/Cell Background Tab

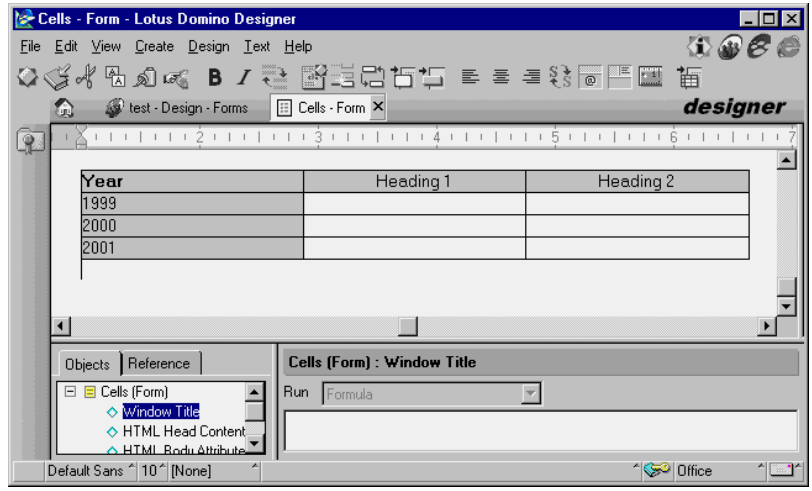
The Table/Cell Background tab looks like this:



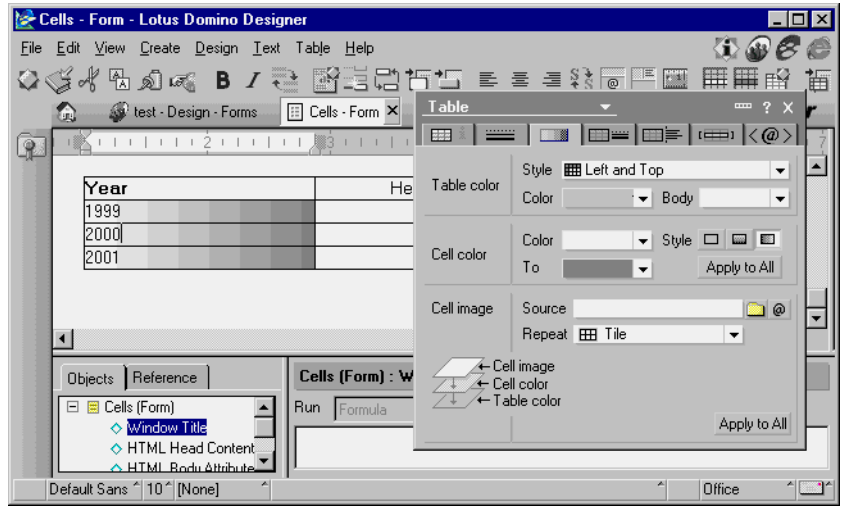
1. In the Colors tab, you can choose the background color of the cell. To have the same background color for the whole table, click Apply to All.
2. The Table Style option allows you to select different types of styles for the table. The available options are:
   - Solid
   - Alternating rows

- Alternating columns
- Left and top
- Left
- Right and top
- Right
- Top

Here is an example for table style — Left and Top:



The Style options give you the opportunity to use a particular color effect in the cells, as shown below:



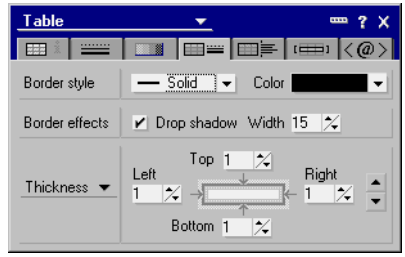
To do this, follow these steps:

1. Choose the background color for the cells.
2. Select the gradiant color.
3. For the direction option choose left to right.

Note This feature is not supported on the Web.

Table Borders Tab

The Table Border tab lets you specify the width and style of the table border. The Table Border tab looks like this:

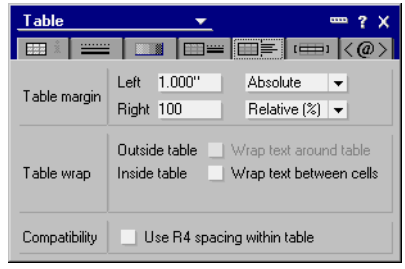


The Drop Shadow option shows a shadow around the table.

Note The Drop Shadow option is not supported on the Web.

Margins Tab

The Margins tab enables you to specify the left and right margins for the table. You can use either the percent sign (%) or inches. The Margins tab looks like this:

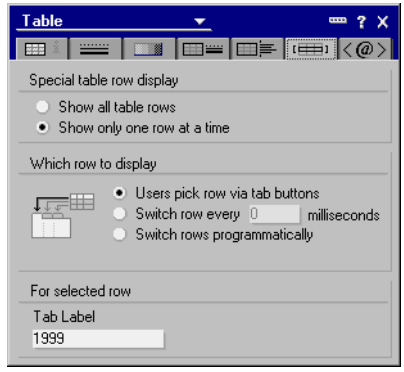


The R4 Spacing option converts the whole table into the R4 form.

Note The Margins option is not supported on the Web.

Table Rows Tab

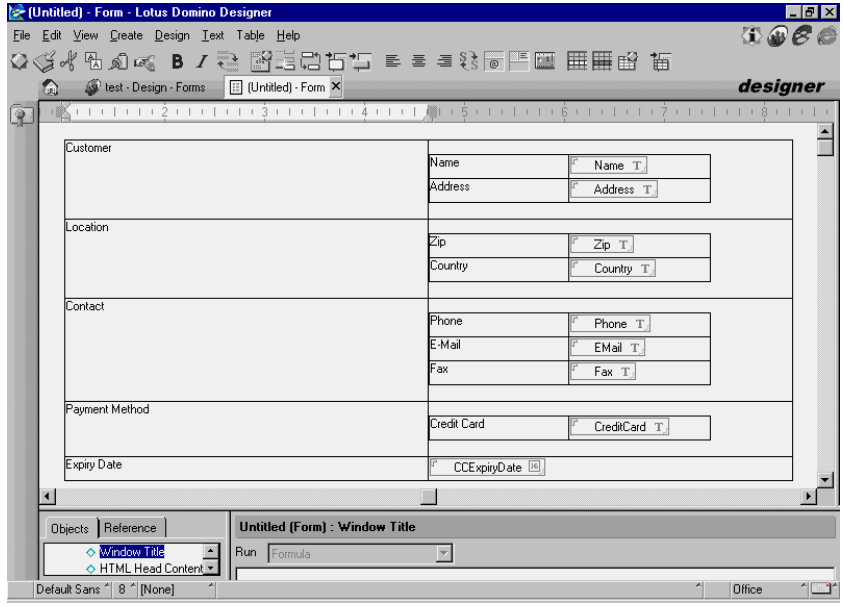
The Table Rows tab is a new feature in Domino R5.0 that lets you work more effectively with tables.



Select the Show only one row at a time option. You can use many different kinds of features.

Note Some collapsible features do not work on the Web.

The following figure shows a small table that we will use to demonstrate how the collapsible features work:

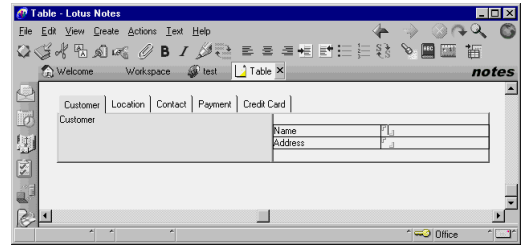


- There is one large table, which contains five rows and two columns.

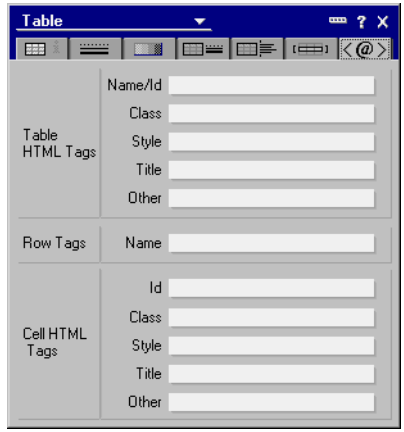
- In the second cell of each of the first four rows there is a nested table. The first and second nested tables each contain two rows and two columns. The third nested table contains three rows and two columns, and the final nested table in the fourth row contains one row and two columns.

The following table shows some further examples of these features:

Note All these table rows in the main table appear to the user so that they only see one row at a time.

Option	*Description*
Users pick rows via tabs	
Switch rows every *(n)*milliseconds — Advance on Click	This option allows you to display a single row from the main table that the user can cycle through to the next by clicking with the mouse.
Auto Start Timer	When the user clicks the table, Domino displays the second row of the tables.
Switch rows every *(n)*milliseconds — Once when opened	Domino runs (displays the rows one after another) the table once when the form is loaded. Interval option is allowed.
Switch rows every *(n)*milliseconds — Cycle once on Click	This is similar to the Once when opened option but is activated when the user clicks the first row.
Switch rows every *(n)*milliseconds — Continually	This is again similar to the Once when opened option but cycles through each row in the table continuously.
Switch rows programatically	Allows you to display a single row based on the value stored in special field $*table-name* where table-name is the value that you give your table in the Table-HTML tab, for example $MyTable. The value required in this field is the name of the tab or the row number (field must be a number type and rows start from zero), you need to have displayed.

HTML Tab



The HTML tab gives you an opportunity to specify options for your table. Available options are ID, Class, Style, and Title. Apart from the ID field, which is used in controlling the table programmatically, HTML options only take effect on the Web.

Embedded Elements

Using Domino R5.0 you can insert navigators, views, outlines, calendar controls, scheduling controls, folder panes, and file upload controls just by clicking the mouse button in your form. These elements are called embedded elements and they provide an easy way to enter and show information in both the Notes client and Web browsers.

The Notes client cannot use two of these elements, only Web users can. These elements are:

- Embedded Folder Pane
- Embedded File Upload Control

Web users cannot use Date Picker.

Note You can also use these embedded elements in any page and they will work as they do in a form.

Embedded Navigators

An embedded navigator is an element which provides you with an easy way to show a Notes navigator. You can have multiple navigators in one form. To insert an embedded navigator into your form, follow these steps:

1. From Lotus Domino Designer R5.0, open the TeamRoom database.
2. Choose Forms design view and create a new form by pressing the New - Form button in the Form view pane.
3. Create a table containing one row and two cells.
4. Go to the first cell in the table and choose Create - Embedded Element - Navigator and choose TeamRoom navigator in the dialog box.
5. The TeamRoom navigator is now inserted into the table, and you can see in the Programmer's Pane that the TeamRoom navigator is highlighted.
6. Try to change the embedded navigator by choosing another navigator in the Programmer's Pane to see what happens. As you will see, the navigator changes in your design pane when you select another navigator.
7. You can also embed a navigator based on a formula.

Embedded Date Picker

An embedded Date Picker is an element which shows the calendar view of the current month (default) and allows the user to browse other months.

Embedded Outline Control

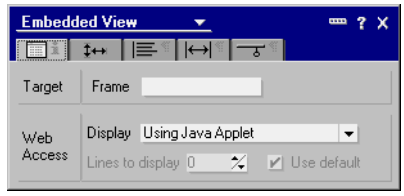
You can insert an Embedded Outline Control in your form to give more flexibility to your programming of the navigation pane than navigators provide. Navigators appear essentially as static image maps. If you need more flexible and programmable navigation, Outline control can be a good option. For more information on this, read Chapter 6: New R5.0 Design Elements.

Embedded View

An embedded view, also known as a UI Java view applet, is an element which provides you with an easy way to show a Domino view for Notes or Web users. The power of an embedded view is that when you load the site containing an embedded view through a Web browser, it actually behaves as if you were using a Notes client. For example, the Expand and Collapse options work fine, and it is not necessary for browsers to reload the site repeatedly, which can be slow. You can insert only one view in one form, subform or page.

To create a new embedded view, follow these steps:

1. Go to the second cell in the table and choose Create - Embedded Element - View.
2. A dialog box is displayed. Choose By Category and then OK.
3. The embedded view is inserted into the table and you can see a list of views in the Programmer's Pane.
4. By clicking with the right mouse button and selecting Embedded View, you can display the Embedded Views InfoBox, which will look like this:



- The Basic tab allows you to specify whether you want to use the view Java applet when looking at this form through a browser, and if not, lets you override the default number of lines to display as configured in the Domino server configuration.
- You can choose the color of the background and the size of the view in the Elements tab.
- The Fonts tab of the InfoBox lets you specify fonts and colors for the Embedded View.
- The Alignment tab lets you specify the alignment of the view.
- The Page Break tab allows you to control the pagination of the view.
- The Hide When tab allows you to set when to display and hide the view.

Embedded Group Scheduling Control

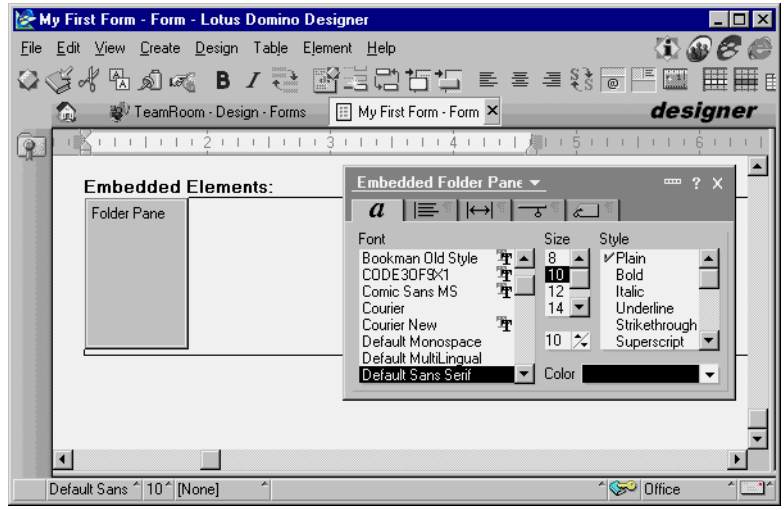
The group scheduling control is similar to the one found in the R5.0 mail template, and allows you to view the diaries/schedules of multiple people at the same time.

Embedded Folder Pane

An embedded folder pane is an element which provides you with an easy way to display a list of Notes views on the Web. You can have only one embedded folder pane within one form.

1. Open the form that you created earlier and delete the embedded navigator element.
2. To do this, click the embedded navigator element and press **DELETE**.

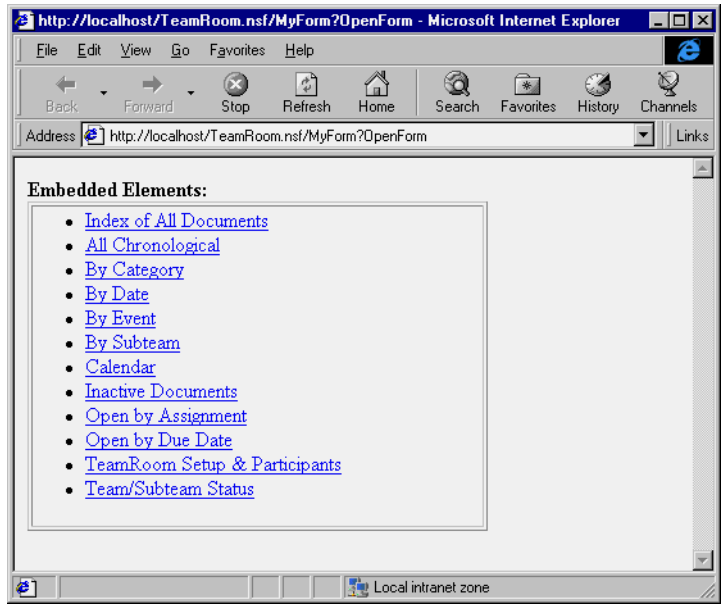
3. In the same place, choose Create - Embedded Element - Folder Pane. After embedding the folder pane element form, the layout will look like this:



You can look at the form in your Web browser by opening the database from the Web and typing the following address in the location field:

`http://"server name"/"database name"/"Form Name"?openForm`

where the “server name” is the name of your server, “database name” is the database name, and “Form Name” is the name of the form that you want to open.



You can also change the font, font size, color of the text, and alignment of the folder pane by opening the InfoBox.

4. Close the form and return to the standard navigator.

Embedded File Upload Control

A file upload control is an element which provides users with an easy way to upload file attachments to the Web. You can have multiple file upload control elements in one form.

Note Web browsers which support File Upload Control are Netscape Navigator 3.x or later and Microsoft Internet Explorer 3.02 or later. To be able to upload files with Microsoft Internet Explorer, you must also install the Microsoft Internet Explorer File Upload Control add-on.

Other Features of Forms

Horizontal Rules

It is not necessary to write HTML to insert and define horizontal rules. To create a horizontal rule, do the following:

1. From the standard navigator choose Design - Forms. A list of forms is displayed.
2. Open the Main Topic form.
3. Click the form where you want to create a horizontal rule.
4. Choose Create - Horizontal Rule.
5. A horizontal rule is created on the form. You can change the settings of the horizontal rule from the Infobox.
6. By default, the rule is set to fit to window. You can set the height and color, and either fit the width to the window or specify a width.

Note Experiment with your browser to see if your browser supports color and width attributes for rules.

Computed Text

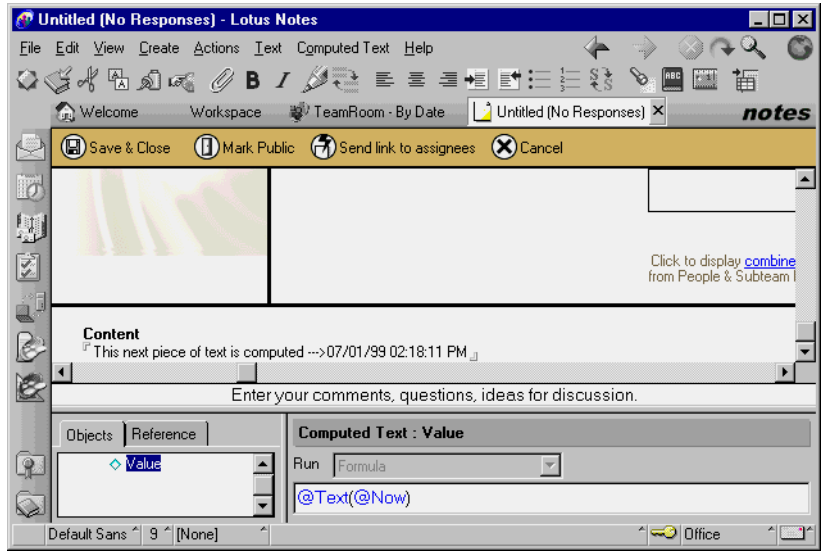
Computed text can be used to create text, on the form or in a rich text type field on a document, based on Notes @formulas. Creating computed text is similar to creating text fields which are computed for display.

Computed text is not stored in the document and it is computed every time the document is opened, reloaded or refreshed. Computed text is especially useful when used in documents.

Authors who don't have designer privileges, can create personalized Web pages using computed text. To create computed text in an existing field, do the following:

1. From the Notes, client open the TeamRoom database and create a new document choosing Create - Document - Document.
2. Go to the Content field.
3. Choose Create - Computed Text.

4. Computed Text is created on the field.



5. When Computed Text is highlighted, click the Programmer's Pane and type the desired formula. In this case the formula is:

   ```
   @Name([CN];@UserName)
   ```

   The formula returns the common name part of the username.

Note This is useful if you are not a designer but you want to add some functionality to the document. You can, for example, use computed text to show something to one user that other users will not see.

The following formula allows only Paul Revere to see the text; other users will not see it:

```
@If(@Name([CN]; @UserName) = "Paul Revere"; "Hello Paul, This
text is shown only for you, other users don't see it";"");
```

Tip If desired, you can type HTML into Computed Text.

Buttons, Action Bar Buttons, and Hotspots

Web browsers don't have a prebuilt Graphical User Interface (GUI) for Domino applications, so you must provide Web users with a way to perform common actions, such as creating new documents, saving a document, and changing to another view.

Multiple buttons can be created on Notes forms to be displayed on the Web. Button formulas are run when the user clicks the button. With buttons or view actions that display in the Action bar, Web users can perform equivalent functions to those that Notes users are able to perform by using menus and the Action bar.

To create multiple buttons that are displayed in a Web browser, perform the following steps:

1. The browser must support JavaScript.
2. Select the database property Web Access: Use JavaScript When Generating Pages. Without this property set, Domino recognizes only the first button in a document and treats it, by default, as a Submit button that closes and saves the document. If there are no buttons in the form, Domino places a Submit button at the bottom of the form.

Guidelines for Using Button Formulas

Notice the following guidelines and restrictions:

- On the Web Action bar, buttons and hotspots support only a subset of the available actions.
- You should create only one Submit button per form. You can customize this button, even if you haven't selected the database property Web Access: Use JavaScript when generating pages.
- Formulas on buttons are run when the user clicks the button, while formulas on hotspots and action buttons are run when the hotspot or action button is displayed.

Tip If your browser doesn't support JavaScript and you don't want any buttons on the form enter the following HTML code on the bottom of the form: *</form>*. Remember to mark text as Pass-Thru HTML and hide it from Notes users.

New Action Features

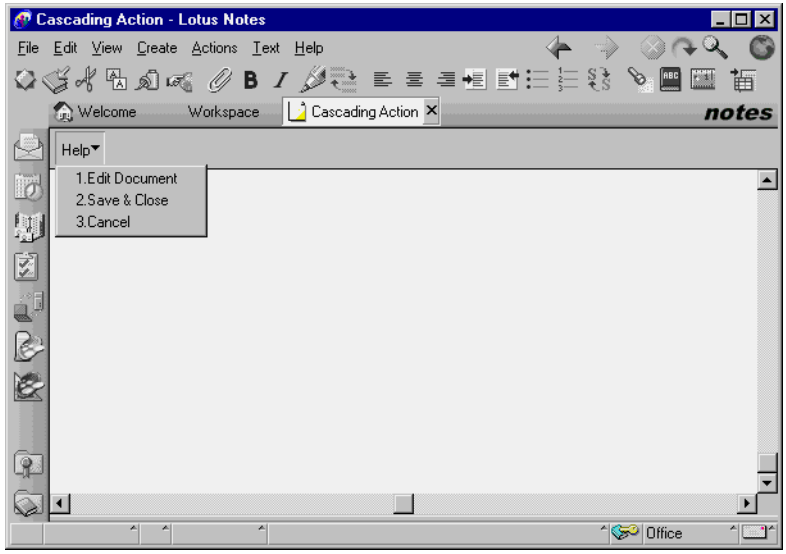
Domino R5.0 has brought new features to actions, such as drop-down action buttons.

- **Action Drop-Down Menu:** The application developer can now create a drop-down menu on an action bar by creating an action and giving it a name containing a backslash (\) in it.

  The following example creates action buttons for the following names:

  - Help\1.Edit Document
  - Help\2.Save & Close
  - Help\3.Cancel

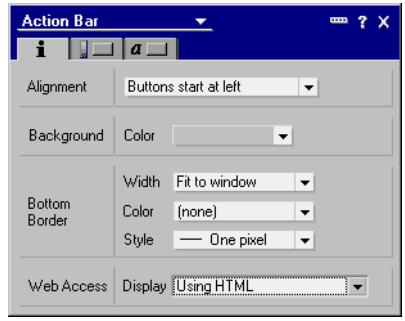
  When all the action buttons are created, save the document and preview the form by choosing Design - Preview in Notes. The action bar and the action button should look like this:



  As you can see, the drop-down menu opens when you click the Help button.

  Note You can add only one sublevel to the action.

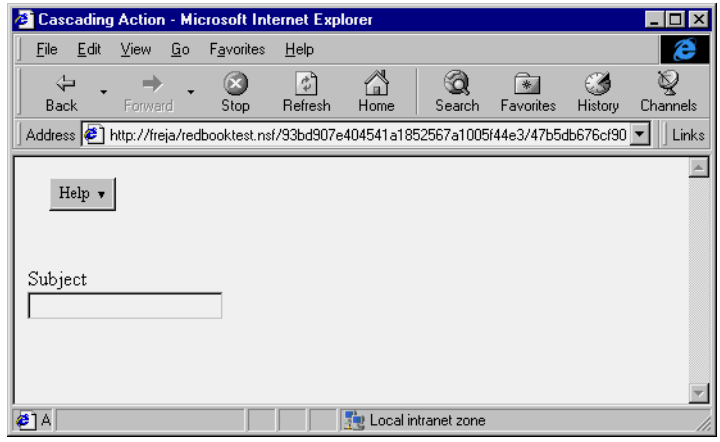
- **Action Bar InfoBox:** The Action bar InfoBox allows you to modify the properties of the Action bar. The Action bar InfoBox looks like this:



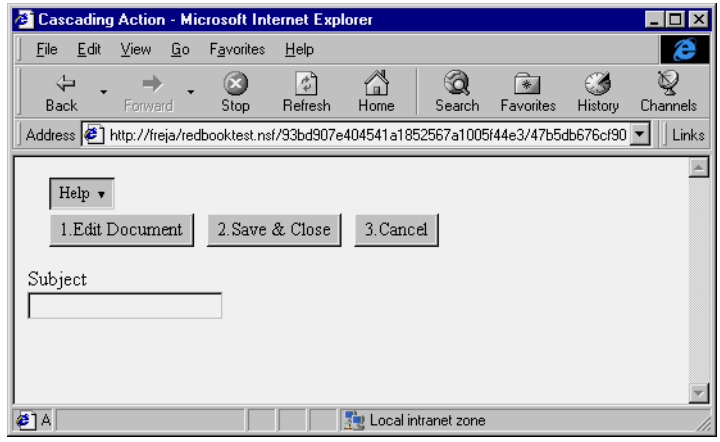
The Action bar InfoBox consists of three tabs:

1. **Action Bar Info:** on this tab you can specify
   - The alignment of the buttons
   - The background color of the action bar
   - The bottom border of the action bar
   - Whether to use HTML or the Java applet when viewing this page via a browser.
2. **Button Background tab:** This tab allows you to modify the button options; you can change the color of the button, when the user clicks the button, or when the user moves the mouse over the button. You can also add an image inside the button.
3. **Button Font tab:** This tab allows you to modify the text and the style of the buttons.

Using the Java applet to display the cascading action buttons we created earlier, the browser looks like this:



When you click the menu bar item it expands the menu choices below the item.



Images Within Forms

There are several ways to add images to your Web pages:

- Copying images through clipboard
- Importing pictures
- Image Resource

Copying Images

1. Copy your desired image to the clipboard.
2. Open the form in Design mode.
3. Click the form where you want to place the image.
4. Choose Edit - Paste.

Importing Pictures

1. Open the form in Design mode.
2. Click the form where you want to place the image.
3. Choose Create - Picture.
4. Select the type of file, click the file to import and click OK.

Tip Often the quality of the image is better when the picture is imported rather than pasted.

Using Image Resource

1. First, create a new image resource by opening Resource - Image - Design View.
2. Click the New Image Resource button, select the type of image file, and then click the file to import. Click OK.
3. Go to the form and move the cursor to the place where you want to place the image resource.
4. Choose Create - Image Resource, and select the current image for Insert Image Resource box.

Note Image Resource also supports animated images.

Alternate Text

Adding alternate text to graphical hotspots allows Web users with text only Web browsers to see text on the form where the graphic should be. Users with Web browsers that support graphics will see the alternate text while the Web browser is loading the graphic. To add alternate text:

1. Select the graphic and choose Picture - Properties.
2. In Alternate Text for Web and Deferred Loading boxes: enter the text to describe the graphic.

Using CGI Variables

Common Gateway Interface (CGI) is a standard for interfacing external applications with HTTP servers. When a Web user saves a document or opens an existing document, the Domino Web server uses CGI variables to collect information about the user, including the user's name, the browser, and the user's Internet Protocol (IP) address.

To capture this information in a Web application, you have two options:

- Create fields with the same names as CGI variables.
- Use LotusScript agents.

Table of CGI Variables Supported by Domino

The following table lists all the CGI (Common Gateway Interface) variables which are supported by Domino.

Domino captures the following CGI variables through a field or a LotusScript agent. You can also capture any CGI variable preceded by HTTP or HTTPS. For example, cookies are sent to the server by the browser as HTTP_Cookie.

CGI Variable	*Returns*
Auth_Type	If the server supports user authentication and the script is protected, this is the protocol-specific authentication method used to validate the user.
Content_Type	For queries that have attached information, such as HTTP POST and PUT, this is the content type of the data.
Content_Length	The length of the specified content as given by the client.
Gateway_Interface	The version of the CGI spec with which the server complies.
HTTP_Accept	The MIME types that the client accepts, as specified by HTTP headers.
HTTP_Referer	The URL of the page the user used to get here.
HTTPS	Indicates if SSL mode is enabled for the server.
HTTP_User_Agent	The browser that the client is using to send the request.
Path_Info	The extra path information (from the server's root HMTL directory), as given by the client. In other words, scripts can be accessed by their virtual path name, followed by extra information that is sent as PATH_INFO.
Path_Translated	The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.
Query_String	The information that follows the question mark (?) in the URL that referenced this script.
Remote_Addr	The IP address of the remote host making the request.
Remote_Host	The name of the host making the request.
Remote_Ident	This variable will be set to the remote user name retrieved from the server. Use this variable only for logging.
Remote_User	Authentication method that returns the authenticated user name.

continued

CGI Variable	*Returns*
Request_Method	The method used to make the request. For HTTP, this is "GET," "HEAD," "POST," and so on.
Script_Name	A virtual path to the script being executed, used for self-referencing URLs.
Server_Name	The server's host name, DNS alias, or IP address as it would appear in self-referencing URLs.
Server_Protocol	The name and revision of the information protocol accompanying this request.
Server_Port	The port to which the request was sent.
Server_Software	The name and version of the information server software running the CGI program.
Server_URL_Gateway_Interface	The version of the CGI spec with which the server complies.

For more information about CGI, see:

`http://hoohoo.ncsa.uiuc.edu/cgi/env.html`

Using a Field to Capture CGI Variables

When a field has the same name as a CGI variable, Domino copies the field value from the CGI environment and places it in the field. There are two things to note:

- You don't have to specify a formula if the field is editable.
- Make the field hidden when previewed for editing and opened for editing.

For example, you can create a field named HTTP_USER_AGENT. This field captures information about which browser the user is using. You can use this field to see if the user's browser supports features in your application.

The general format of HTTP_USER_AGENT variable is software/version library/version. The 11 leftmost characters basically tell you the browser version. You can evaluate this in formulas and display different things based on that. For example:

```
@If(@Left(HTTP_USER_AGENT;11)="Mozilla 3.0";"Netscape
Navigator"; @Left(HTTP_USER_AGENT;11)="MSIE 2.0";"Microsoft
Internet Explorer";"")
```

Some of the values returned by HTTP_USER_AGENT are:

Returned Value	*Browser*
MSIE	Microsoft Internet Explorer 4 or later
Mozilla 3.x	Netscape Navigator 3.x
Mozilla 4.x	Netscape Navigator 4.x

Using a LotusScript Agent to Capture CGI Variables

You can use the DocumentContext property of the NotesSession class to capture CGI variables. The property returns a Notes document that contains all CGI variables that are applicable to the session. You can use these values to collect or process information for the current session.

The following example demonstrates how to access CGI variables:

```
Dim session As New NotesSession
Dim doc As NotesDocument
Dim CGIValue As String
Set doc = session.DocumentContext
CGIValue = doc.HTTP_USER_AGENT(0)
```

The *CGIValue* string now has information about a user's browser.

Tip When submitting a document, CGI variables are calculated twice: Once when the form is displayed to the user in the browser, and again when the submit button is clicked and the form is posted back to the server. There are occasions when you need to capture the state of a CGI variable when the form is first displayed by the Web browser. For example, when using the HTTP_Referer field to capture the URL the user has just come from so that you can take them directly back to that point. In this case, you need to create a second field on the form that computes the value of the CGI variable: make the field hidden to the HTML form by using the HTML tag's <TYPE=HIDDEN>.

Summary

This chapter explained the main features and functions available to the application developer when creating forms in a Notes application. It outlined some of the areas that application developers need to consider when developing applications for use by both Notes clients and Web browsers.

Chapter 5
Views, Folders, and Navigators

This chapter describes the design elements: views, folders and navigators. It describes how to create and modify them and how to control the information that they display.

What is a View?

A view lists the documents stored in a Domino database and can be thought of as a "table of contents" of a database. Each row listed in a view represents data taken from a single document. Each column represents a field or a combination of fields taken from that document.

All Domino databases have at least one view, but most of them have more. A view can display all the documents in the database, or it can display a subset of the documents. Documents can be viewed by categories, such as creation date or author. Views can present documents sorted on different fields, for example, sorted by topic.

What is a Folder?

Folders enable you to store and manage related documents without putting them into a category, which requires a Categories field in the form used to create the documents. Folders are also convenient because you can drag documents to them.

What is a Navigator?

A database navigator is a graphical interface which allows the user to easily access views, Domino data, or other applications. Navigators can include graphic buttons or hotspots, which are programmed areas that a user clicks to execute an action

Creating Views

There are several ways to create a new view. If you know about a view in a different database than the one you are working in, you may want to copy this view over to your database and customize it to your needs. Alternatively, you can build a view from scratch.

Copying an Existing View

To copy an existing view:

1. Open the database where the view you want to copy is located.
2. Choose View - Design on the action bar to switch to Design mode.
3. Select Views from the Site Database List.

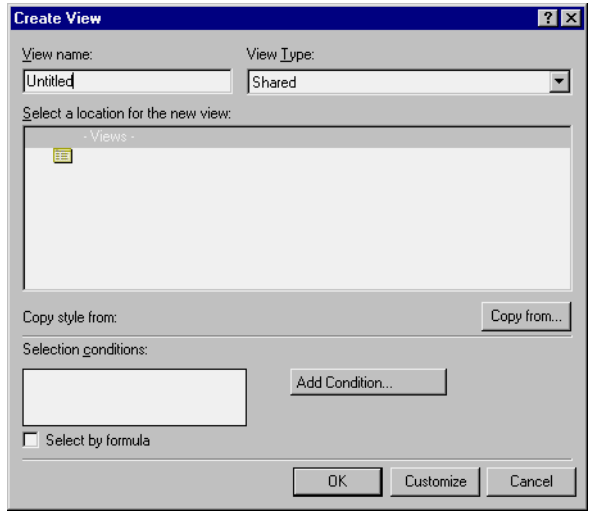
   Note If this database is already in your Site Database List, you may skip the first two steps.
4. In the View pane, select the view you want to copy.
5. Choose Edit - Copy to copy the view to the clipboard.
6. Open the database in Design mode where you want the view to be copied.
7. Click the View pane, then choose Edit - Paste to copy the contents of the clipboard. This creates the new view.

Tip Instead of choosing Edit - Copy and Edit - Paste from the action bar you may want to use the keyboard shortcuts.

Creating a New View

To create a new view:

1. Open the database in Design mode and switch to the View pane.
2. Click the New View button. (You may also choose Create - View from the action bar.) The Create View dialog box is displayed:



3. Change the view name from Untitled to a meaningful name of your choice. You also should provide a view alias at this time by entering the vertical bar, followed by the alias name after the view name.

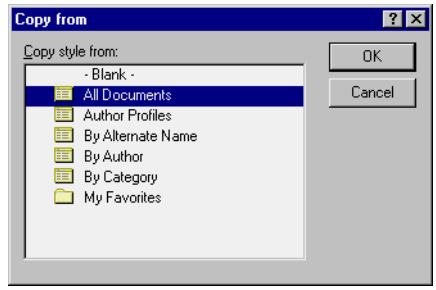
4. Select the view type from the pull-down list. The following options are available:
   - Shared
   - Shared, contains documents not in any folder
   - Shared, contains deleted documents
   - Shared, private on first use
   - Shared, desktop private on first use
   - Private

   Caution Shared, Personal-on-first-use views cannot be maintained by the designer of a database.

5. Select where the new view should appear in the View Menu. Selecting any position other than the top level will create a cascaded view.

   Tip You can also use the backslash character (\) to separate the levels to create a cascaded view.

6. Click the Copy From button if you would like the view's design copied from a different view to the one displayed. This will bring up the Copy from dialog box:



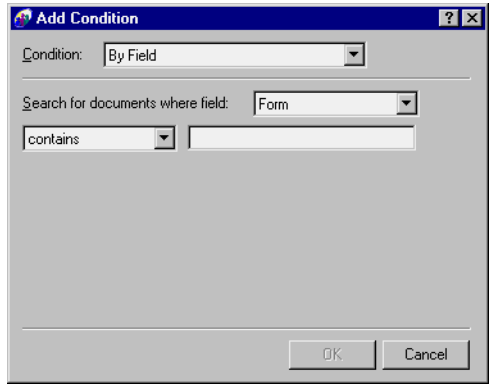
   You can now select a different view design as the default for your new view.

   Note With this method you can only copy the design from an existing view in the same database.

7. Decide if you want to specify the view selection using the Search Builder or the @function language.

   Note If you are creating a shared view which contains only deleted documents or documents not in any folder, you cannot specify any selection formula.

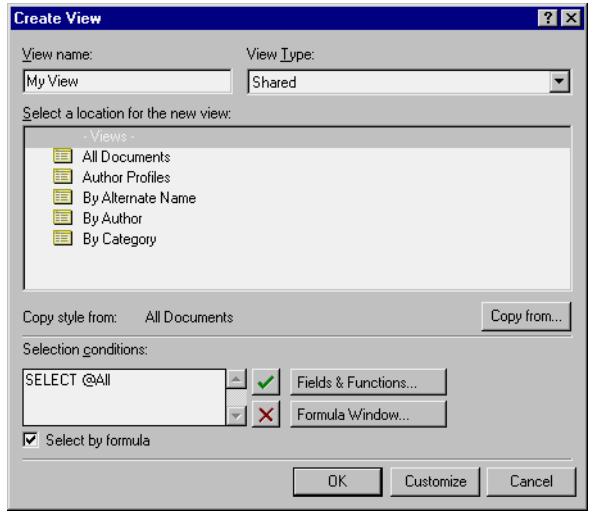
8. To use Search Builder, click the Add Condition button. This brings up the Search Builder dialog box:



9. Build your selection criteria and click the OK button.

   Note The layout and entry fields in this dialog box change depending on the condition you choose.

10. When using the @function language the layout of the dialog box changes to look like this:



11. Click the Fields & Functions button if you want to see a list of all the fields defined in the database and all the available @functions, along with their Help documents.

    Tip You can paste the field names and @functions from this list into the formula window.

    Tip Clicking the Formula Window button opens an formula input window where it is easier to enter more complex formulas.

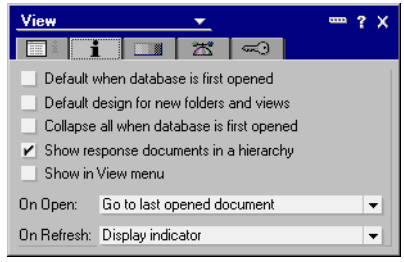
12. If you click the Customize button, the Design window of the new view is displayed so that you can start your customization. Otherwise, click the OK button.

Working With View Properties

To display the View InfoBox:

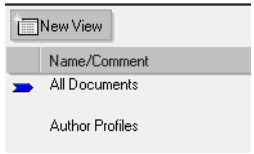
1. Open the view in Design mode.
2. Click the Properties icon to display the InfoBox. It contains five tabs.
3. On the View Info tab you may specify the name of the view. Including backslashes (\) in the name will cascade the views in the View menu. For example: Marketing\Lotus Domino.

   Note If you set the name of the view to the reserved word ($All), Domino will display this view with the name All Documents in the View menu. With the exception of the reserved names, for example ($All), ($Inbox), ($Trash), and ($Sent), enclosing a view in parentheses means that the view is hidden and is used solely for programming purposes. The user will not see it in the list of views.
4. Specify an alias. This is the name you will use in your code. The advantage of using an alias is that if the name of the view has to be changed, you need not change your code each time the view name is mentioned.
5. Specify a comment. The comment entry field is optional, but useful for maintenance purposes.
6. Choose a style. You can display the documents in a view as a calendar instead of a table. For example, a Calendar view can display a date, a meeting or appointment time, a duration, and optional text describing the entry. To display a view as a calendar the first column must be a Time/Date field. For more information on creating a Calendar view, see "Creating Calendar Views," later in this chapter.
7. Click the Options tab. It looks like this:



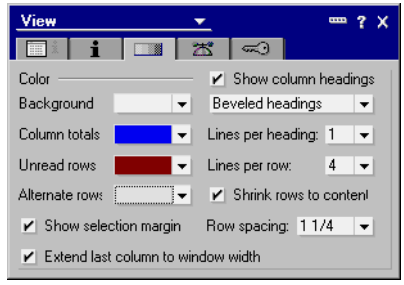
   - If you select Default when database is first opened, this view will be used to display the documents.

Note You should always make sure that a database has a default view. There can only be one view in a database having this attribute. In the View pane, such a view is marked with a dark blue arrow.



- If you select Default design for new folders and views, this view will be used as the template when the user creates folders or adds new views to the database.
- Collapse all when database is first opened will only show the headings for categorized documents.
- Selecting Show response documents in a hierarchy will show all response documents indented under their parent documents.
- If the Show in View menu check box is not selected, the view will be hidden from the View menu in the action bar.
- Furthermore, you may specify the behavior of the view when it is opened or refreshed.

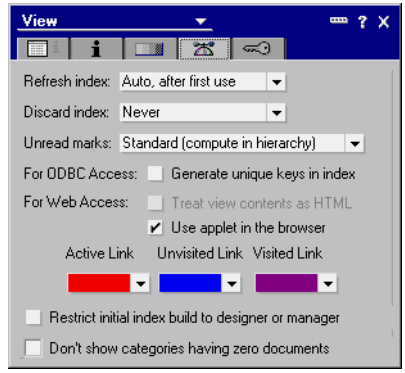
8. Click the Style tab:



9. In addition to customizing the Background Color of the view, Column totals and Unread rows, you can define a color to be used for Alternate rows.
10. (Optional) You can hide the selection margin.
11. You can have up to five lines for the column headings.
12. As far as rows are concerned, you can also have up to nine lines for each document, to enable you to store a large description. If you choose to use these lines, make sure that the check box Shrink rows to content is selected. It eliminates all the blank lines for documents that do not require the extra space.
13. Click the Advanced tab.

The Advanced tab provides information on the following topics:

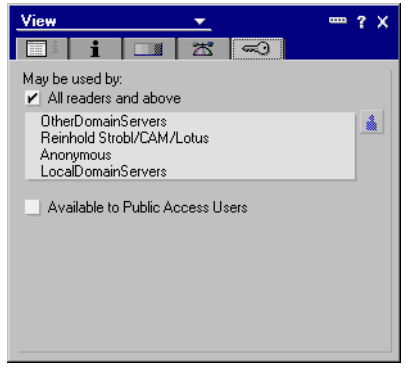
- About the index used to build the view; when it should be refreshed, and when it should be discarded.



- How users will be notified about documents that were added or modified since they last opened the view
- Whether unique keys are built in the view for ODBC
- How this view is displayed when viewed with a Web browser
- Restrict initial index build to designer or manager
- If empty categories are displayed or not. Select the Don't show categories having zero documents option if there are documents with a Reader field which may hide these documents from the user at runtime. If that is the case and there is no document to be displayed for a specific category, the category will also not be displayed to the user.

14. Click the Security tab.

In the example below, the view can be used by all users that have access to the database. If you want to restrict its use to only some users or groups of users, deselect the check box and add the users or groups that will be granted specific access.

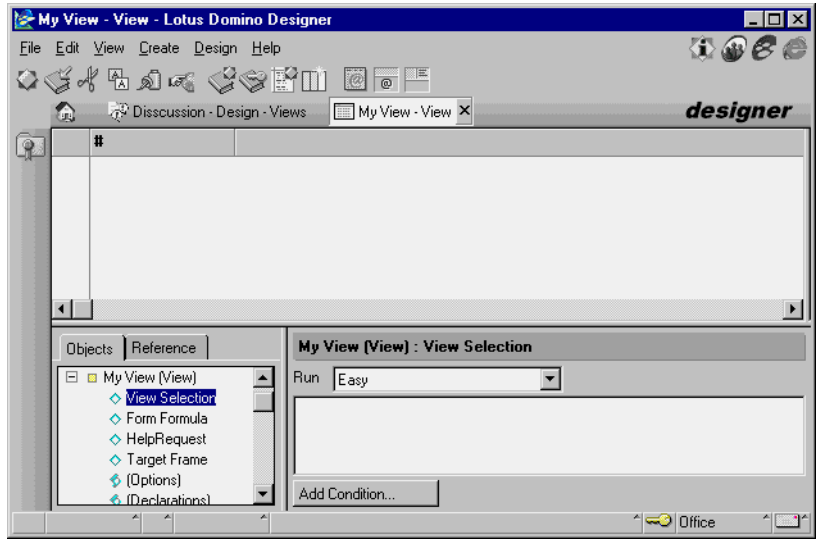


You may also select the Available to Public Access Users option to enable non-Notes users to access the view.

15. Close the InfoBox.

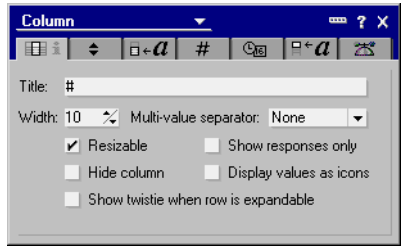
Editing View Columns

We will now take a closer look at column properties. To view the column properties of a view, open the view in Design mode. You will see a window like this:



To access the properties of a column, double-click the column heading or select the column heading and click the Properties icon. The InfoBox will be displayed.

The Column Info Tab



On the Column Info tab, you can specify the following:

- The title of the column to be shown in the column heading
- The width of the column
- If this column is related to a multi-value field, how this field will be displayed
- If the user is allowed to resize the column
- Whether or not the specification of this column is for response documents
- If the column should be hidden from the user

  This means that the column is used for programming purposes only.
- If icons should be displayed in the column

  If this box is checked, the width of the column should be 1. Furthermore, the Programmer's Pane must contain a formula which evaluates to a whole number. For example, the following formula determines whether a document has an attachment and, if so, displays the attachment icon (number 5):

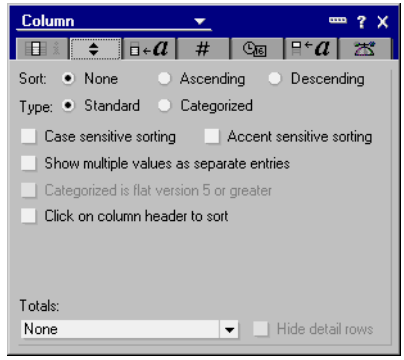
  ```
  @If(@Attachments;5;0)
  ```

  Use 0 as the "false" case when you want to leave the column blank. The formula above returns 0 when the document has no attachments, so nothing is displayed. A table of all icons and their associated numbers is available in the Help database for information. Note that you cannot add any icons to the predefined set.
- If a twistie will be shown if the row is expandable

  This option means that for a categorized row (see below), the value of this field is not shown on every row, but as a header to the documents in that category. When a user displays the view, an arrow will be shown next to those categories that can be expanded.

  Note The column width for such a row can be set to 1 because the value of the field is shown on a single line.

The Sorting Tab



On this tab, you can specify whether or not the documents displayed in the view should appear in a sorted order, and if yes, what the rules are for sorting. You do this by making the following selections:

- Select None, Ascending, or Descending as the sorting rule.
- Specify Standard or Categorized as the sorting type.
- Specify sorting rules for case and accent sensitivity.
- Choose options for multiple values and categories.
- Select if you want the user to be able to alter the sort order.

  Tip Using this option allows the user to specify a secondary sort column, as well.

- Select a value from the Totals pull-down menu if you would like to display some statistics in the view.

The Font and Style Tabs

- The Font tab lets you set the font and color used when displaying fields in this column.
- Use the Number tab to specify the display format of numbers in this column.
- Use the Time tab if you are going to display date/time values in this column.
- The Title tab allows you to set the font and color of the column title.
- Use the Advanced tab to force the values in this column to be used as document links when this view is displayed in a Web browser.

Creating Calendar Views

There are some rules that you must follow in order to create a Calendar view. You start creating like any other view described above, but on the View Info tab of the InfoBox you specify Calendar as the style. This tells Domino to display the documents in this view using the Calendar Outline.

Defining Columns

You must define the first two columns as described below in order for the view to function properly:

- The first column must be sorted and has to contain a Time/Date value or a list of Time/Date values. Also, make sure that the Show Multiple Values as Separate Entries option is selected in the Sorting tab of the InfoBox for this column. This causes documents containing a Time/Date value list to be displayed on more than one day in the calendar which you can use, for example, for repetitive events.

  Make this column hidden.

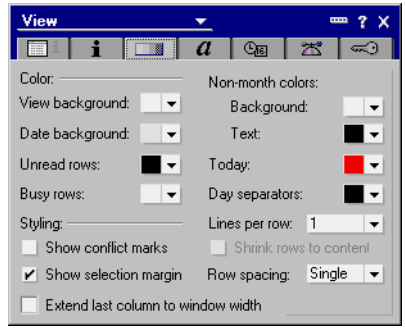
- The value for the second column must evaluate to a duration in minutes, for example, (EndDateTime - StartDateTime)/60. If the duration is not relevant for your view, set the value to zero.

  Make this column hidden.

Note Column headers are not displayed in Calendar views.

Customizing the View Layout

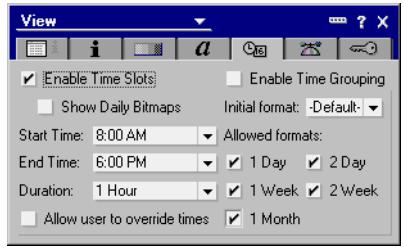
Use the Style tab of the View InfoBox to fine tune your settings:



If you select Show conflict marks, a vertical line is displayed in front of entries for appointments which are scheduled for the same time. Furthermore, you might not want to show the selection column in order to save space. The selection column will be shown automatically if there are selected documents in the Calendar view.

Note A document marked for deletion will appear with the strike-through mark.

Use the Time/Date tab to specify the defaults for time slots:



It is also a good idea to build your own column for displaying the start and end time of appointments. Make sure that you set the display to Time only in the column InfoBox. Additionally, you might want to truncate the seconds from the value in order to show only hours and minutes to the user.

Enhancing the Functionality of a Calendar View

To allow the user to easily add entries in the Calendar view, consider associating the following LotusScript samples with your view. They allow users to add an appointment by double-clicking in the Date/Time area.

Note The example assumes that the variables *ws* and *ClickedDate* are already declared.

1. Add the following to the Regiondoubleclick event of the view:

```
Set ws = New NotesUIWorkspace
If source.CalendarDateTime<>"" Then
    Call ws.ComposeDocument("", "", "Appointment")
End If
```

2. Add this LotusScript code to the QueryOpen event of the Appointment form:

```
Set ws = New NotesUIWorkspace
ClickedDate = ws.CurrentCalendarDateTime
```

ClickedDate is a global variable which is used to set the start date of the new appointment as default.

Summary

Below is a checklist with the most important things to consider when creating a view.

Checking View Design

- Is there a default view for the database?

  If not, double-click the view that should be the default and choose Design - View Properties. Click the Options tab and select the Default view when database is first opened option.

- Is there a view that is displayed by date?

  If not, make a Date column sortable or add a view that sorts documents by date, so that users can view documents in chronological order.
- Do all views appear correctly on the View menu? Are the appropriate keyboard shortcuts used? Do the views appear in the correct order?

  If not, verify the names and the Show in view menu selection in the View InfoBox.

  Tip You may consider numbering your views. This also helps Helpdesk members to identify the view when receiving a telephone call from a user.
- Is the information in the view easy to read?

  If the view appears cluttered or the columns are too close together, reset the column width and alignment.
- Are all the documents that should be in the view displayed, or are too many documents displayed?

  If the view is not displaying the correct documents, check the view selection formula.
- Are response documents indented?

  If they are not and you want to indent responses, select Show response documents in a hierarchy on the Options tab of the View InfoBox, and create a column for responses.
- Do response documents correspond to the correct main documents?

  If not, verify the view selection formula; also be sure that the responses-only column is placed directly to the left of the column that displays the main document information.
- If the view uses categories, do the categories appear correctly?

  If not, create a sorted, categorized column and use the name of the appropriate keyword field as its formula.
- If the view is used for programming purposes, make sure that it is a hidden view.
- Check workstation compatibility. Are all fonts used in this view available (or approximated) on all workstations? Are column widths sufficient for all workstations?
- If needed, does the view have a read access list?

  To create a read access list, select the Security tab in the View InfoBox.
- If the view is designed for Web access, keep it simple.

Checking Columns

- Is the information in each column correct?

  If not, verify the formulas in the column definitions.

- Is the column returning values of the appropriate data type?

  If not, verify the field formula. Columns display only simple text; therefore, you may need to use @TEXT in the formula to display values from numbers or time fields.

- Is each column displaying all the information that is contained in it?

  If not, you may need to adjust the column width and/or the font used to display the column.

- Are the contents of columns aligned properly?

  For example, numbers should be right-aligned; text should be left-aligned or centered. Verify the alignment for each column.

- Are documents in the right order?

  If not, make sure that you sort on the correct columns, and that you choose the correct sort order (ascending or descending).

Shared and Private Views

Views can be either shared views available to many users, or personal views used by one person. You designate the view type when you create it, and you cannot change it later.

Shared Views

Shared views are views that are available to any user with at least Reader access to the database. Most views that you design for databases are shared views. Users with Designer or Manager access can create shared views, and so can Editors when the database manager has selected Create personal Folders/Views for them in the access control list.

Shared, Personal-on-First-Use Views

Shared, Personal-on-first-use views are a convenient way to distribute customized personal views to multiple users. You usually create this type of view by using @UserName to customize the display for each user.

After a user saves a Shared-to-personal view, the user's copy of the view no longer inherits design changes. For example, if you add a column to the view, anyone using a personal version of the view won't see the new column. To obtain design changes, users must delete their personal versions of the view and open the Shared-to-personal view again.

Note Shared-to-personal views are not a security measure, because they do not protect data. If you create a Shared-to-personal view that omits certain documents, a user can still create a personal view that includes them.

Shared-to-personal views are stored in the database as long as they are shared. After the first use, Domino uses the Create Personal Folders/Views option to determine where to store the view.

Personal Views

Users can create personal views to organize documents in personalized ways by choosing Create - View.

If a user has rights to create personal views/folders in the access control list, personal views are stored in the database. If the user does not have the access control list right to create personal views/folders, personal views are stored in the user's personal workspace file.

Creating a Button on the Action Bar

You can create an Action bar in views and folders, as well as in forms. In general, the actions should do one of the following:

- Affect several documents or all the documents displayed in the view. You could store all documents created by your manager in a Manager folder, for example.
- Represent the actions that the user will perform most often.

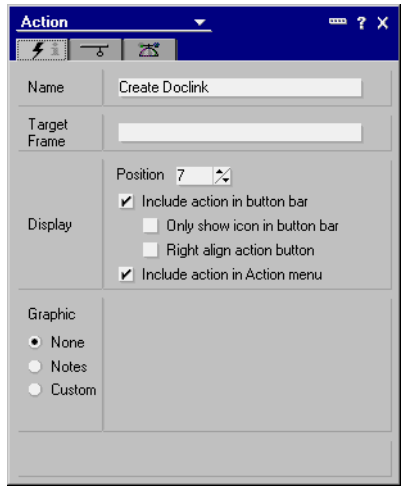
As in forms, you must make sure that the actions you create will fit in the Action bar, and you must also consider the screen resolution available to your users.

Creating a Document Link Using an Action Button

As an example, we are going to develop an action that creates a Document Link between two documents. The documents do not have a child-parent relationship.

To create a button in the Action bar:

1. Open the design of the ($All) view.
2. Choose Create - Action. The InfoBox for the Action properties is opened and you now have access to the Programmer's Pane. Fill in the Information tab as shown here:



3. Put the following @function statements into the Programmer's Pane:

   The document selected from the view is copied to the clipboard:

   ```
   @PostedCommand([EditMakeDocLink]);
   ```

   The form Document is created:

   ```
   @PostedCommand([Compose]; ""; "Document");
   ```

   The macro goes to the field Body:

   ```
   @PostedCommand([EditGotoField]; "Body");
   ```

   It pastes the doclink into the RichText field:

   ```
   @PostedCommand([EditPaste]);
   ```

It then positions the cursor back at the top entry field:

```
@PostedCommand([EditTop])
```

Testing the Formula

To test the formula:

1. Select the All Documents view.
2. Click one of the documents to highlight it.
3. Click the Link Documents button on the Action Bar. The document is opened for creation, the doclink is pasted, and the cursor is positioned in the first entry field.
4. Double-click on the DocLink to open the document that was selected in the view.

Properties of Actions and the Action Bar

Actions and Action bars have properties that you can display by selecting the action in the Action pane of the view or folder design window.

These properties are identical to the ones found in the Form Action bar.

Working With Views as a Developer

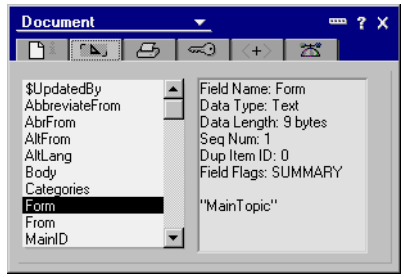
As an application developer, you may have additional requirements for a view in a database than a regular user. Therefore, you will probably build "administrative" views to be able to keep an eye on columns which are hidden to regular users, for example, create an All Documents view, or a view showing replication and save conflicts (using the $Conflicts field). These views are also a great help for administrators when the database is used in the production environment.

Tip There is an undocumented interface for displaying views. Hold down the **CTRL+SHIFT** keys and double-click the database icon. This opens the database showing the hidden views as well as the normal views.

While you are testing your application, you will probably need to inspect field values which are hidden on the form. Using the method described below, you can inspect a form's fields and their values without creating special views. To do so, look at the properties of a document:

1. Select any one of the documents displayed.
2. Click the Properties icon.

3. Click the Fields tab to see the list of fields for that document as well as their values.



When files are attached to a document, a field called $FILE exists. Scroll down the right listbox to see the file information. Here you can see the file name and size and the platform on which it was created. There are also other keywords, such as $Revision, $Links, $UpdatedBy, $Conflicts, and $Anonymous.

Note As you can see, all the reserved fields start with a dollar sign ($). Make sure not to prefix any of the fields you create with this character.

The field replication mechanism allows for a faster transfer of information across servers or between the servers and workstations. An indicator is attached to each field in all documents: Seq Num (or sequence number). If you have a replica of a server database, compare the values of the sequence number for fields of a replicated document. If their values are different, this means that the field containing the lower value will be modified at the next replication.

Tip To benefit from field replication, once you have finished developing your form, create a document using the form and check the values of this indicator for all the fields in the document. It will help you separate the fields that are frequently updated (and replicated) from the ones that are not. This could have a major impact on the replication throughput, especially if some of the fields contain large volumes of information such as graphics, large attached files or multimedia objects (video or sound).

Views and the Web

Domino dynamically creates Web pages from the views in a database, including URL links to the documents in the view. Using a Web browser, a user can navigate, expand, and collapse the view in much the same way as they can from a Notes client.

Using the Default Display

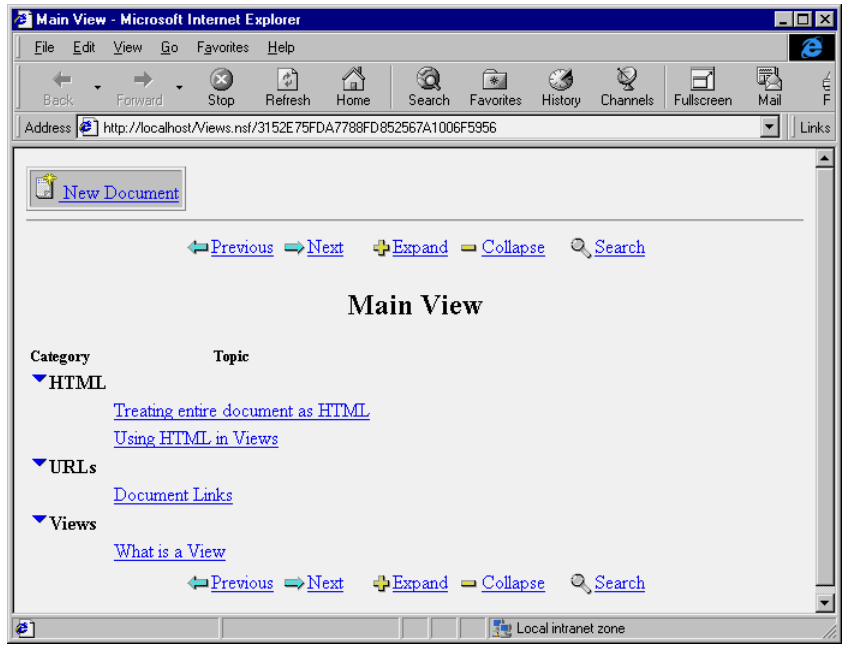
When a browser is used, the view is split into pages with 30 lines per page. This is to avoid having a view containing hundreds of documents presented as one page with all the documents. Limiting the lines per page in this way improves performance and makes navigation of the database more manageable.

Note The default of 30 lines per page can be changed in the HTTP section of your server document.

When Domino generates the HTML page for a view, it maintains the column and row format of standard Domino views. There are, however, a number of differences that you need to be aware of and to take into consideration when you develop applications for the Web.

Tip To improve Web performance (and performance on any other view for that matter), avoid time-sensitive column formulas with @functions such as @Now, @Created, @Modified, and so on. Since the Domino Web server generates Web views as HTML pages on the fly, time-sensitive formulas recalculate every time a Web user works in the view, for example, when they open, scroll, or expand the view. Instead, create a field in the form for that formula and refer to the field in your view.

The following figure shows an example of a categorized view as seen through a Web browser:



Notice that there is no outline of available views on the left. Domino does not include a view outline by default on Web pages. Furthermore, Domino also ignores any menu actions not supported on the Web. Also, notice that there is no selection column.

As you can see, Domino has automatically created a Web navigation bar at the top and bottom of the screen. This navigation bar contains buttons that users click to Expand, Collapse, Scroll, and Search the view.

On the Web, users open documents by clicking a document link column rather than clicking anywhere in the row, as you do in the Notes client. The application designer can specify which column(s) should include a URL link to the document. By default, it is the first non-categorized column.

Domino displays the width of a column in a view as the length of the longest entry in the column, regardless of what the column width is set to in the Column InfoBox.

Tip To force a column to be limited to a certain width when viewed from the Web, use a column formula to retrieve the field value, for example:

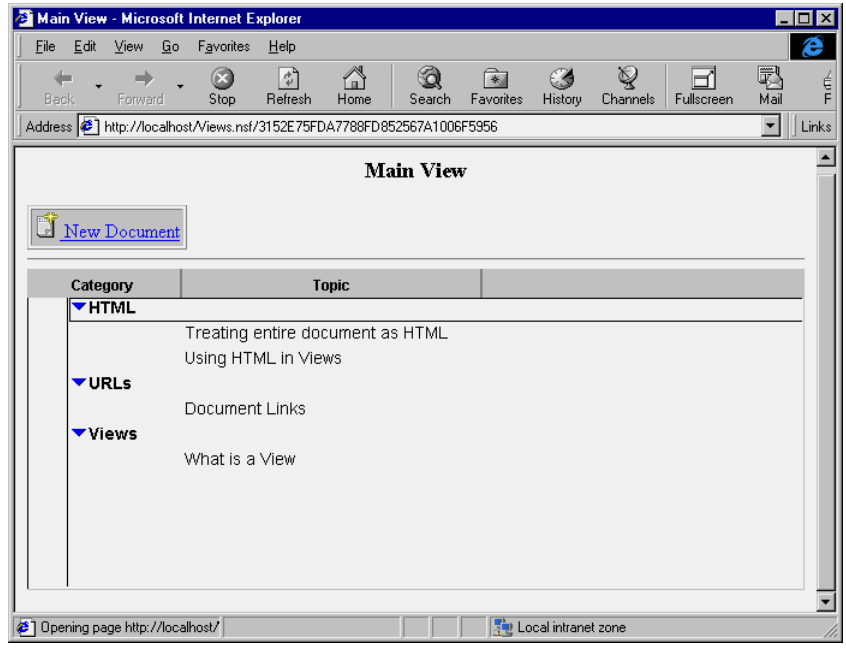
```
@Left(FieldName;50)
```

This formula will set the widest width of the column to 50 characters.

To avoid having long columns pushed to the right of the display, use the Style tab in the View InfoBox and specify a number greater than 1 in the Lines Per Heading setting. This causes the lines to wrap on the Web. If you specify 1, the lines will not wrap. The same guidelines apply to column headings.

Using Java Applets for Display

In Domino Release 5.0 there is a Java applet which allows you to display a view in a more Domino-like outline. You activate this applet by selecting the Use Applet in Browser option on the Advanced tab of the View InfoBox.



A view served as an applet supports such "Notes-like" features as expandable/collapsable categories, resizable columns, and multiple document selection.

The view applet is programmable via the following @Commands:

```
@Command([EmptyTrash])
@Command([ViewCollapse])
@Command([ViewCollapseAll])
@Command([ViewExpand])
@Command([ViewExpandAll])
@Command([ViewRefreshFields])
```

Using HTML Formatting for Views

You can add HTML to your column formulas to enhance the view when it is displayed on the Web.

Treating the View Contents as HTML

You can override the default row and column settings for a view by using HTML formatting attributes stored in a column.

In Domino, the view displays as a standard view. On the Web, the view uses the HTML formatting attributes that you specify in the column formula. You must include HTML that defines all formatting and document linking for the view.

1. Open the view in Design mode and choose Design - View Properties.
2. Click the Advanced tab and select For Web Access: Treat view contents as HTML.
3. Create a column.
4. In the design pane, click Formula and enter the HTML code in the edit window.

Enhancing the View Display Using HTML

You can also add some HTML coding to the view in order to enhance the display on the Web.

HTML embedded in views is a browser-only feature. If you embed HTML in a view, the view is accessible by a Notes client, but the client interface will be unattractive (because of the exposed HTML tags). There is no option to hide a column from Notes client users as in the case of forms and views. A column is either hidden or visible to both Web and Notes client users. Although you can see the HTML tags from the Notes client, the features that the HTML coding provides, such as linking, are not available to Notes clients.

Unlike forms and documents, there is no pass-thru-HTML option in the design interface. To write HTML in views, you need to include the HTML code in square brackets ([ ]). Domino treats everything between the square brackets as pass-thru HTML.

Here are some examples of how you might want to use HTML in views. The following section explains how to implement several of these examples.

- Insert a blank .GIF file between two columns in order to get a little more space between the columns on the Web.
- Add a horizontal ruler spanning the entire view for each document category.
- Add a couple of icons and a URL link to the Home Page in the column headers.
- Include an HTML statement in the formula for a column to display a blinking text string if it is a newly created document.
- Include a URL in a couple of documents and jump directly to the URL from the view.

Adding a Space Between Columns

1. Create a new column between two columns and set the column width to one (1). Deselect Show values in this column as links if not already done.
2. Select Formula in the Programmer's Pane and insert the following HTML code, including the quotation marks (this is a text string):

   ```
   "[<img src=/icons/ecblank.gif>]"
   ```

Note The "ecblank.gif" is one of the standard icons that come with Domino. It is located in the icons subdirectory on the Domino server.

Adding HTML and Icons to Column Headers

You can use the column Title field in the column properties box to add graphics and pass-thru HTML to your column headings. The only limitation is that you can only fit 64 characters into the column title.

1. Open the InfoBox of a categorized column, and in the Title field add the following line (no quotation marks):

   ```
   Sort [<img src =/icons/vwicn108.gif>]
   ```

   This will add a small green arrow to identify where to click to sort the column on the Web. Browser users might not be familiar with the little triangle if they are not familiar with the Notes client.
2. Open the InfoBox for any other column. In the title field enter:

   ```
   [<A HREF="../"><img src=/icons/vwicn069.gif
   border=0>Home</A>]
   ```

   This will add an icon with a DocLink to the home page from the column header.

   Click the Title tab and select Right Alignment.

Adding HTML to Column Formulas

1. Select a categorized column. In the Programmer's Pane, click the Formula button and enter the following formula for the column:

   ```
   Categories + "[<hr>]"
   ```

   This will display the value of the Categories field and then add a horizontal rule. Notice that the plus (+) sign is used to append values.

2. Add a column and enter the following formula:

   ```
   @If(@Now>@Adjust(@Created;0;0;7;0;0;0);"";"[<blink><font
   color=\"Red\"><b> New </b></blink>]")
   ```

   The formula checks to see if the document was created within the last week and if so, it will display a blinking "New" text string.

   Note The HTML blink tag is not supported on all browsers.

Creating URL Links at View Level

You can add HTML to the document fields displayed in the view columns in the same way. By adding a URL address to a document, you can display URL links in a view, enabling users to jump directly from the view level to a URL.

Note You could also calculate the HTML in the view, which is useful if you want to use the same document but different views for Web browsers and Notes clients.

1. Create a new document and include the following in a field that will be displayed in the view such as the Document title.

   ```
   [<A HREF=http://www.lotus.com>]LotusDevelopment Corp.[</A>]
   ```

2. Save your document and create a new one. Enter the following URL:

   ```
   [<A HREF=http://notes.net>]Notes.Net[</A>]
   ```

3. Save your document.

Both documents take you directly to their respective Web sites.

Domino View Properties Not Supported on the Web

Avoid using the following view and folder features in a Web application:

Views and Folders	*Reason*
Options properties	This feature is not supported using HTML in a Web browser.
Collapse all when database is first opened.	Web views do not expand or collapse all; instead they expand or collapse only one category at a time (equivalent to Expand/Collapse Selected Level).
Show in View menu	Web applications do not have a View menu. To exclude a view from the folders navigator, use the Design - Design InfoBox to hide the view from Web users or surround the view name in parentheses, for example (HiddenView).
On Open: Go To… options	This feature is not supported using HTML in a Web browser.
On Refresh options	This feature is not supported using HTML in a Web browser.
Style properties Unread rows Alternate rows Show selection margin Beveled column headings	These features are not supported using HTML in a Web browser.
Advanced properties Refresh index options Discard index options	These features are not supported using HTML in a Web browser. Views can be re-indexed at a Domino server.

Columns	*Reason*
Column Info properties	This feature is not supported using HTML in a Web browser.
Show twistie when row is expandable	Triangles are always shown.

Hints and Tips on Designing Views

The section below contains a collection of hints and tips. You may find them useful when you are going to design or change a view.

Naming Views

The name that you choose for a view is visible to Notes client users in the View menu, to Web users in the Views list, and in the Folders pane (unless the view is hidden). The name is case-sensitive and can be any combination of characters, including letters, numbers, spaces, and punctuation. The full name, including all alias names, can be up to 64 characters.

Naming Tips

Views appear in alphabetical order in menus and lists. To force names to appear in a different order, number or letter them. This also allows Helpdesk members to easily identify the view.

When possible, assign a name that indicates how the view sorts documents, for example, By Company Name or All by Category, or specifies which documents it includes, for example, New Customers.

Use consistent names across databases to make it easier for users to recognize views.

Alias Names

An alias is an internal name for a view. Usually you use this alias for programming purposes, for example, in @DbColumn formulas. Aliases follow the same naming rules as view names.

You can append more than one alias name by entering the | (vertical bar) symbol followed by the alias name. Always keep the original alias as the rightmost name.

```
Main View | View1
```

Changing a View Name

You can edit the view name or alias name in the View InfoBox when the view is open in Design mode. If you change the name of the view, copy and paste the previous name into the Alias box to the left of any other aliases, using the | as the separator.

For example, a movie database contains a view named By Screening Date. The name of this view is going to be changed to By Premiere Date. Here is how the name will look after it has been changed:

```
By Premiere Date | By Screening Date | DateView
```

Hidden Views
When you surround a name with parentheses, for example (All), the view does not appear to Notes client users in the Domino view menu, or to Web users or Notes client users in the Folders pane.

Cascading Views
If you don't want to overwhelm users with long lists, or if you have related views that should be grouped together, you can arrange them in a hierarchy so that a group of related menu items are organized under one item in the navigator pane. A user clicks on the higher-level name to display the cascaded list.

If you didn't specify a cascading view when you created the view, enter the name you want to appear on the Create menu followed by a \ (backslash), followed by the view name. For example, the Personal Address Book template has two views related to servers:

```
Server\Certificates
```

```
Server\Connections
```

Overview of Styles

The following tables give an overview of the most important settings:

View styles	*Description*	*Comments*
Color: Background	Determines the background color for the view.	White, light blue, and yellow are good choices.
Collapse all when database is first opened	Displays the top level category in categorized views or the main documents in a hierarchical responses view. Users click the category to see individual documents within the category.	Useful for large views with many categories or topics. Not applicable to Calendar views.

Row styles	*Description*	*Comments*
Lines per row (1-9)	Determines how many lines a row can contain.	"Shrink rows to content" and "Color: Alternate rows" are useful accompaniments to multi-line rows.
Color: Unread rows	Determines the color for unread documents.	Red is used for unread documents in template designs.
Color: Alternate rows	Determines the color that alternates with the background color to highlight every other row.	Useful for multi-line rows. Not applicable to calendar views.
Show selection margin	Shows the document selection margin. Deselect for cleaner-looking rows.	If you deselect "Show selection margin," users can still select documents by pressing and holding **SHIFT** as they click document names. The selection margin appears temporarily while documents are selected, and hides again when all documents are deselected.
Shrink rows to content	Keeps gaps from appearing below rows that are shorter than the number of lines per row you select.	
Row spacing (Single, 1-1/4, 1-1/2, 1-3/4, Double)	Determines how much space there is between rows.	More space makes each row easier to read. Less space condenses the view contents to make them useful for reports or Web users.
Show twistie when row is expandable	Shows a green triangle next to a column that displays categories or response documents.	Not applicable to Calendar views.
Extend last column to window width	Fills out the last column to avoid empty space in the view.	

Column styles	*Description*	*Comments*
Column width	Determines how many characters fit in one row of a column.	(Optional) Select "Resizable" to allow users to change the width as needed. With the view in Design mode, you can also click the column and drag the column divider line to the width you want.
Text style and color: Column title	Determines the font, size, color, and alignment of an individual column title at the top of the view.	Use the "Apply to All" button to change the text style for all titles in the view.
Text style and color: Column values	Determines the font, size, color, and alignment of values that display in this column.	Use the "Apply to All" button to change the text style for all columns that display text in the view.
Hide column	The column title and values do not display to users.	Useful for columns used for sorting that contain values users don't need to see. This is not a security feature.
Multi-value separator	For any documents that display multiple values in the column, separates each value with punctuation or a new line.	Useful for making columns more readable if they contain several values (usually generated by a multi-value field).
Show column headings (Beveled, Simple)	Shows a bar at the top of the view with column titles with either a beveled or flat look.	Beveled-background is gray. Simple-background matches view color.
Lines per heading (1- 5)	Determines how many lines a column title can wrap.	Useful for long column titles or instructions placed in a column title.
Color: Column totals	Determines the color of the Totals for any columns that contain totals.	

Identifying Unread Documents

To help users find new or modified documents, display the unread marks (asterisks) next to unread documents in the view. A set of unread marks are maintained for each user, so even if one person has read a particular document, the asterisk still appears for other users who have not yet read that document.

Choosing a Style for Unread Marks

These options are set as a Design property for a view. Open the Advanced tab of the View InfoBox and select an "Unread marks" option.

You can display unread marks as:

- **Standard (compute in hierarchy).** Displays asterisks for unread Main documents and Response documents, and for any collapsed categories containing unread Main or Response documents.
- **Unread documents only.** Displays asterisks only for unread Main or Response documents. Unread marks do not appear next to collapsed categories. This choice displays the view faster than the standard display, and is a good compromise between showing unread marks at every level and not showing them at all.

Choosing the option None omits unread marks. Users can still navigate to the next unread document using SmartIcons.

Choosing a Color for Unread Marks

To change the color of unread documents in the view from the default color red, click the Style tab and select another color for Unread rows.

Disabling Unread Marks for Modified Documents

If the unread status of modified documents is unimportant to users or if the database resides on a server that users don't access directly, turn off unread tracking for all documents in a database to conserve disk processing time. Click the Design tab of the Database InfoBox and select Do not mark modified documents as unread. This setting affects all views in the database. Users see only new documents as unread; modified documents do not appear as unread.

Using Categories in Views

A view that displays categories enables users to find related documents. A categorized view is neat and easy to scan. Users can collapse the categories to display only the category names and then expand categories individually, or expand the whole view.

To categorize a view, create a column to display categories and then select the Option Type: Categorized on the sorting tab of the column InfoBox. A categorized column groups documents with matching values and converts the value to a category name. The column is usually one that appears on the left side of the view. This column must always appear to the left of any sorted columns.

The setting of the following options is recommended:

- Style the column text with a different color and in boldface to make categories stand out.
- Select the Column Info property Show twisties when row is expandable to display a green triangle that users click to see categorized documents.
- Select the Options view property Collapse all when database is first opened to show only the category names when users open the view.

Creating an All by Category View

To allow users to categorize documents with Actions - Categorize, a database must have the following components:

- A form with a Categories field
  - Field name: Categories
  - Data type: Editable Text or Keywords field
  - Select Allow multi-values
  - Format: If you selected Keywords, leave the keywords list blank and select Allow values not in list
- A view named All by Category
- A Categories column, placed as the leftmost column in the All by Category view:
  - Width: 1 character
  - Column title: Leave the column title blank
  - Formula: Categories
  - Sort the column by selecting Sort: Ascending and Type: Categorized

Presenting Views to Users

Several options in the view and database InfoBoxes determine the initial display of a view.

Opening to a Particular Row in a View

To highlight a particular row when a user opens the view, select one of the following On Open options on the Options tab of the View InfoBox:

- Go to last opened document (the default choice)
- Go to top row
- Go to bottom row

Collapsing a View to Show Only Categories

If you have a view that displays categories, you can show the view in collapsed form every time users open it by selecting Collapse all when database is first opened on the Options tab of the View InfoBox.

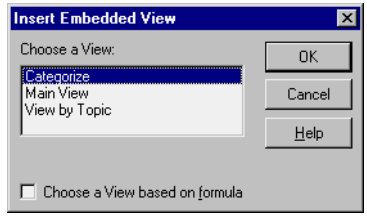
Displaying the Last-Used View

If you select Restore as last viewed by user (one of the On Database Open choices on the Launch tab of the Database InfoBox), Notes client users see the default view the first time they open a database, and afterwards they see the last view they opened. This option isn't available for views opened by Web browser users.

Embedding Views

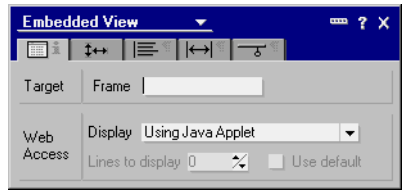
To embed a view on a form, do the following:

1. Open the form in Design mode.
2. Select Create - Embedded Element - View. The following window is displayed:



   Select a view from the list, or select Choose a View based on formula if you want to have a dynamic selection.
3. Click OK and the view will appear on your form.

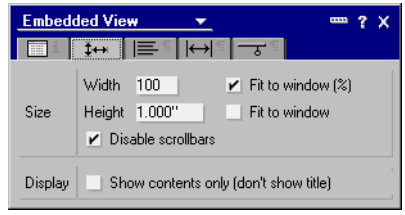
4. Click the Properties icon to bring up the InfoBox.



5. On the Information tab specify how this view is displayed through a browser.

   Note You can only change the numbers to be displayed in the browser if you select the Never option button.

6. Click the Element tab.



   - Check Fit to available width if you want Domino to determine the width of the view. Use the Percentage selection to refine your selection. Otherwise, provide a width value for the view.
   - Select a Background Color for the view.
   - Specify the Height of the view.
   - Check Disable Scrollbars if scrollbars are not to be available to the user.
   - Check Show Contents only if there should be no column headers displayed.

7. Use the Font, Alignment, Page Break, and Hide When tabs as you would for any other design element.
8. Go to the Object Browser and select Embedded View - Show Single Category, and specify a selection formula. This allows you to display only one category based on a formula.

Hiding Views

Hiding a View from Notes Client Users

There are several ways to hide a view:

- Deselect Show in view menu on the Options tab of the View InfoBox.

  Omitting a view from the View menu applies only to Notes client users, since Web users don't have access to Domino menus. The view still appears in the Folders pane when users choose View - Show - Folders.

- Open the Design toolbox, click the view name in the right pane, and choose Design - Design Properties. Select the Hide design element from Notes R4.6 or later clients option.

  Hiding a view from Notes clients is useful when you have a Web-only view or when you want to remove the view from both the View menu and the Folders pane.

- Give the view a name and enclose it in parentheses, for example, (Hidden).

Hiding a View from Web Users

- Open the Design toolbox, click the view name in the right pane, and choose Design - Design Properties. Select Hide design element from Web clients option.

  Hiding a view from Web clients is useful when you have a Notes-only view, or when you want to remove the view from the folders pane and the Open Database Views list.

- Give the view a name and enclose it in parentheses, for example, (Hidden).

Showing a View to Users of Older Notes Releases Only

When you have a view tailored to users of older Notes releases, open the Design toolbox, click the view name in the right pane, and choose Design - Design Properties. Click the Design tab and select Do not show this design element in menus of Notes R4 or later clients. The view will then not appear to anyone using Notes Release 4.0 or later.

Formatting Date and Time Columns

To format values that result in a time or date being displayed in a column, select a style in the Date and Time format tab of the Column InfoBox.

Option name	*Selections*
Show	Date and time Date only Time only "Today" and time
Date format	month/day month/year month/day/year
Time format	hour:minute hour:minute:second
Time zone	Adjust time to local zone Always show time zone Show only if zone not local

- "Today" and time show values resulting in the current date with the word "Today." Values resulting in the previous day display "Yesterday." All other values display the date.
- If you have an international date format set in your operating system, these choices change to suit the national conventions, for example, from month/day to day/month.
- If you have an international time format set in your operating system, these choices change, for example, from 02:30 to 14:30.
- You have three choices for time zone displays:
  - "Adjust time to local zone" displays the time relative to the time zone of the reader. A document created at 3:00 P.M. in New York that is read by a user in Los Angeles adjusts to Pacific Standard Time; the creation time is displayed as "12:00 PM."
  - "Always show time zone" displays the time zone where the document was created. With this option, the creator's time zone is always shown. If a document is created in New York at 3:00 P.M., a user in Los Angeles sees the creation time as "3:00 PM EST." A user in New York also sees the creation time as "3:00 PM EST."
  - "Show only if zone not local" displays the time zone where the document was created only when the document is read by someone in a different time zone. A document created in New York at 3:00 P.M. displays to all users in the U.S. Eastern standard time zone as "3:00 PM." Users in all other time zones see the creation date as "3:00 PM EST."

Formatting Numbers in Columns

To format values that result in a number being displayed in the column, select a style on the Number tab of the Column InfoBox. The following selections are available:

- **General**. Formatting displays numbers as they are entered; zeroes to the right of the decimal point are suppressed. For example, 6.00 displays as 6.
- **Fixed**. Formatting displays numbers with a fixed number of decimal places. For example, 6 displays as 6.00.
- **Scientific**. Formatting displays numbers using exponential notation; for example, 10,000 displays as 1.00E+04.
- **Currency**. Formatting displays values with a currency symbol and two digits after the decimal symbol; for example, $15.00. The currency symbol and thousands separator that appear are based on settings in your operating system.

The following additional formatting options also apply:

- For any formatting type other than General, select a number from 1 to 15 from the Decimal Places list.
- Select Percentage (value * 100)% to display values as percentages; for example, to display .10 as 10%.
- Select Parentheses on Negative Numbers to display negative numbers enclosed in parentheses; for example, (5) instead of -5.
- Select Punctuated at thousands to display large numbers with the thousands separator; for example, 1,000 in English, or 1.000 in French.

Indenting Response Documents

Indenting Response documents beneath Main documents is useful when readers want to see the progression of a discussion. You can display 32 levels of responses, with each level indented three spaces under its parent document.

Such a view requires that:

- Response forms are available to users with the types Response and Response-to-Response.
- The Show Response documents in a hierarchy option is selected on the Options tab of the View InfoBox, and the document selection formula uses SELECT @All or contains a formula that allows response documents to be included, such as:

```
SELECT Form = "Action Item" | @IsResponseDoc
```

- The view has a responses-only column.
- The responses column is created immediately to the left of the column under which responses are to be indented. Leave its title blank, make its width 1, and select Show Responses only on the Information tab of the Column InfoBox. Enter a column formula that displays information about the response documents shown in the column, such as their authors or creation dates.

Sorting Documents in Views

Every view needs a sorting method that organizes documents in a way that makes sense to users. For example, a By Date view sorts documents by their creation dates, and a By Author view sorts documents by author names. To achieve this effect, designate at least one column as a sorting column. You can then define it as a user-sorted column, an auto-sorted column, or both.

Views that display categories often use sorting methods to sort the category names into alphabetical order. If the sort column displays values from a multiple-value list, select Show multiple values as separate entries to show each value as a separate row. If you don't select this option, multiple values display as one entry and are sorted by the first value.

Ascending and Descending Order

Columns sort documents in either ascending or descending order:

- Ascending order sorts in increasing order (1 precedes 2, A precedes B, earlier dates precede later dates). For example, to display documents from the oldest to the newest, create a Date column that uses the Creation Date as its value and sorts documents in Ascending order.
- Descending order sorts in decreasing order (2 precedes 1, B precedes A, later dates precede earlier dates). For example, to display documents from the newest to the oldest, create a Date column that uses Creation Date as its value and sorts documents in Descending order.

Auto-Sorted Columns

To set up a sorting style in advance, select the option Sort: Ascending or Sort: Descending on the Sorting tab of the Column InfoBox. The sorting column is usually the one that appears on the leftmost side of the view.

User-Sorted Columns

Users see a triangle next to a column title where values can be re-sorted. Users click the column and choose a sorting method to see the documents in the order that they choose. To set up a user-sorted column, select the Click column header option to sort on the Sorting tab of the Column InfoBox. Next, select Ascending or Descending order, or select Both to allow users to cycle between ascending sort order, descending sort order, and no sort order for the column.

Multiple Sorting Columns

To create multiple levels of sorting, designate more than one column as a sorting column. For example, if a primary sorting column sorts entries by date, a secondary sorting column might sort entries by author. Then all documents created by one person on a particular date are grouped together.

Using an Auto-Sorted Column as the Secondary Sorting Column

To add a secondary sorting column, add a column to the right of the first sorting column and then choose Sort: Ascending or Sort: Descending. Documents and responses are sorted, then sub-sorted, in column order from left to right.

Designating a Secondary Sorting Column for a User-sorted Column

User-sorted columns override the sorting built into auto-sorted primary and secondary columns. If the view has a user-sorted column and you want to include secondary sorting, you can associate it with a secondary sorting column. In the Column InfoBox for a user-sorted column, click Secondary Sort Column and choose the secondary sort column and its sorting order.

Character Sorting Rules

Domino sorts characters in the following order:

1. Numbers
2. Letters
3. Accented letters
4. Punctuation/special characters

Sorting rules are also governed by these options:

- Case-Sensitive Sorting. Displays uppercase letters before lowercase letters. For example, A sorts before a.
- Accent-Sensitive Sorting. Sorts accented letters in the sorting order appropriate to the language being used at the workstation.

Overriding Alphabetical Sorting with a Hidden Column

The sorting column does not need to be visible. Sometimes you may want to use a hidden column that selects documents according to criteria that you specify in a formula as your sorting column. For example, a Service Request form contains a Priority field, which uses the following keywords list:

- Urgent
- High
- Medium
- Low

You want the By Priority view to sort documents by the value in the Priority field, but you don't want them to appear in ordinary alphabetical order (High, Low, Medium, Urgent). You want users to see Urgent-priority documents at the top of the view, High-priority documents next, and so on.

To do this, create a column that has the following characteristics:

- Is hidden
- Has no title
- Is one character wide
- Uses this formula to determine the order of each priority:

  ```
  @If(Priority="Urgent";"1";Priority="High";"2";
  Priority="Medium";"3";"4")
  ```

- Is sorted in ascending order

Add a column to the right of the hidden column that:

- Is not hidden
- Has the title "Priority"
- Is 10 characters wide
- Displays the value of the Priority field
- Is not sorted

Exporting and Importing Views

You can export a view to a worksheet, a tabular text file, or a structured text file.

Exporting to a Worksheet

When you export a view to a worksheet, each document becomes a row in the worksheet. Each field becomes a column in the worksheet. The original field contents become cell contents. Column titles become labels in the worksheet.

When you export a view to a new worksheet file, Domino exports the file in 1-2-3 .WK1 format. Although you can specify any extension with the file name, it's best to keep the extension .WK1. When you open the exported file in a newer release of 1-2-3, 1-2-3 automatically converts the file to its new format and adds the appropriate file extension.

If you change the exported file to .WK2, .WK3, or .WK4 before opening it in a release of 1-2-3 that uses that extension and then open the file in 1-2-3 and attempt to add and save attributes, the new copy of the file will be in conflict with the original, and you won't be able to save your changes.

Exporting to a Tabular Text File

Tabular text is ASCII text arranged in rows and columns and separated by space characters. When you export a view to a tabular text file, each document becomes a text row (line). Each field becomes a text "column," separated by space characters. The original field contents become the "cell" contents. To see the separation between columns in the exported view, use a monospaced font for the columns in the original view.

The result of a tabular export may not look exactly like the Domino view. Views use a proportionally spaced font, but exports use a monospaced font. Export a few documents as a test. You can then, if necessary, adjust the column width settings in the view before you perform the whole export.

To display or edit the contents of a Domino view in a Domino document, export the view to a tabular text file. Next, import the tabular text file into a Domino document.

Exporting to a Structured Text File

Structured text is ASCII text that retains its structure in fields and values. Exporting a view to structured text creates a file containing the text of all the documents in the view, minus any rich text attributes.

File Formats You Can Export and Import

No matter what type of data you want to import into a view, it's best to import a small test file first. You can quickly determine whether the import is working correctly and make any necessary corrections before you import the entire file.

The following table lists the types of files that you can import from and export to:

File type	*File extensions*	*File description*
Lotus 1-2-3® and Symphony® Microsoft Excel*	.WKS, .WK1, .WRK, .WR1, .WK3, .WK4	Entire worksheet or named range
Tabular text	.TAB, .TXT, .PRN, .RPT	ASCII text arranged in rows and columns; limit of 1536 characters per record, total
Structured text	.LTR, .CGN, .STR	ASCII text that retains its structure in fields and values; shown as one record per page and one field per line; limit of 256 bytes per simple text field

*To import a Microsoft Excel file, use Excel to save the file as a 1-2-3 worksheet, and then import it.

Exporting a View

Below are the steps to export a view:

1. Select the database and open the view that you want to export.
2. (Optional) Select specific documents to export only a subset of the view.
3. Choose File - Export.
4. Select the file format to which you are exporting.
5. Do one of the following:
   - To replace an existing file, select a directory and file name from the list.
   - To create a new file, enter a new file name.
   - To append to an existing tabular text file, select a directory and file name from the list.
6. Click Export.
7. If you selected an existing file, do one of the following:
   - To replace the file, click Replace.
   - To append to an existing file, click Append.
8. Click Select All Documents unless you selected specific documents in Step 2.
9. Select Include View Titles to export the view column headings to a worksheet or tabular text file. (This is optional for 1-2-3 worksheet or tabular text formats).
10. Change the inter-document delimiter and word-wrap settings if you are exporting to a structured text file. (Optional for structured text format).
11. Click OK.

Note Choose Form Feed to use a form-feed to separate records in the file, or choose Character Code to separate the records in the file with the ASCII code for a delimiter other than the default, ASCII 12 (form-feed). The word wrap option represents the number of characters at which each line wraps. The default is 75 characters.

Importing a View

Below are the steps to import a view:

1. Select the database and open the view that will receive the source data.
2. Choose File - Import.
3. Select 123 Worksheet.
4. Select the name of the source file.
5. Click Import.

6. Select a form to use for the imported source data from the Use Form list.
7. Leave Main Document(s) selected in the Import As list, unless you are creating response documents.
8. Select a Column Format.
9. If you select Format File Defined, enter the name of the .COL file, including the complete operating system path.
10. To import part of a worksheet, type a range name or range address in the WKS Range Name box. You cannot import a 3-D range.
11. (Optional) Select Calculate fields on form during document import.
12. Click OK.

After importing, change the column font to a proportional space font, such as Courier, to improve the display of worksheet data.

Note If you are importing a multiple-sheet worksheet file, Domino imports only the sheet that was open when the file was last closed, or, if you are importing a specified range, imports only the range from that sheet.

Designing a Folder

Folders have the same design elements as views. You design folders in much the same way as views using the Create - Design - Folder command.

The difference between folders and views is that views always have a document selection formula that collects and displays documents automatically. A folder remains empty until users or programs add documents to the folder.

When you create a folder, its design is automatically based on the design of the default view of the current database. You can choose to base the folders design on a different existing view, or to design the folders from scratch. Designing folders is useful when none of the existing views of a database show information in the way that you want to see it.

After you create a folder, it appears in the navigation pane until you delete it.

You can keep a folder personal, or share it with other users of a database. No one else can read or delete your personal folders. To create personal folders in a database, you must have at least Reader access to the database. To create shared folders in a database, you must have at least Editor access, and the option Create shared folders/views must be enabled for you.

When you create a personal folder, Domino stores it in one of two places:

1. If the Manager of the database has allowed it, your folder is stored in the database, allowing you to use the folder at different workstations.

   Note To see whether a database allows you to store personal folders in it, select the database, choose File - Database - Access Control, select your name, and see whether the Create personal folders/views option is enabled.

2. If the Manager has not given you the option to create personal folders in the database, Domino stores your folder in your desktop file.

   Note If a folder is stored in your desktop file, you can use the folder only from your workstation, and you can't use Full text search in the folder.

Managing Access to Views and Folders

If you only want certain users to see a view or folder, you can create a Read access list. Users who are excluded from the access list will no longer see the view or folder on the View menu. A view or folder Read access list is not a true security measure. Users can create private views or folders that display the documents shown in your restricted view, unless the documents are otherwise protected. For greater security, use a Read access list for a form.

You can add users to the Read access list for a view or folder as long as they already have at least Reader access in the database access control list.

Creating a Read Access List

1. Open the view or folder in Design mode.
2. Choose Design - View Properties or Design - Folder Properties.
3. Click the Key icon (Security tab).
4. Deselect All Readers and Above.
5. Click each user, group, server, or access role that you want to include. A checkmark appears next to each selected name.
6. Click the Person icon to add person or group names from a Personal Address Book or the Domino Directory.
7. To remove a name from the list, click the name again to remove the checkmark.
8. Check Available to public access users if you want documents in this view or folder available to users with public access Read or Write privileges in the access control list for this database.

9. Save the view or folder.

Important Do not create a Read access list for the default view of a database.

Note It is necessary to provide server access to views that are Read-restricted when a database must be replicated.

Creating a Write Access List

To allow only certain users to update documents in a folder, create a Write access list for the folder. You can add users to the Write access list for a folder as long as they already have at least Author access in the database access control list. To grant access to users, do the following:

1. Select the database and choose View - Design.
2. In the navigation pane, click Design - Folders.
3. Double-click the view or folder.
4. Choose Design - Folder Properties.
5. Click the Key icon (Security tab).
6. In the Contents can be updated by: section, deselect All authors and above.
7. Click each user, group, server, or access role that you want to include. A checkmark appears next to each selected name.
8. Click the Person icon to add person or group names from a Personal Address Book or the Domino Directory.
9. To remove a name from the list, click the name again to remove the checkmark.
10. Save the folder.

Note Web users cannot drag documents into folders.

Using Navigators

A database navigator allows the user to easily access views, Domino data, or other applications. It is like a roadmap that guides the user through the application using a graphical interface. Most navigators include graphic buttons or hotspots, which are programmed areas a user clicks to execute an action. A hotspot can be text, graphics, or a combination thereof.

Note You might also consider using framesets and pages, as they offer more flexibility when creating applications for both Notes clients and Web browsers.

Navigator Objects

You create a navigator by combining objects. These might include a background graphic for display only, and some combination of graphic buttons and text objects. To create navigator objects, import or paste objects from another application, or use the drawing tools that are supplied by Domino. The drawing tools include hotspot tools that you use to define a clickable area in a navigator.

Navigator Actions

A navigator action determines what happens when users click an object. You can add actions to all navigator objects except those that were pasted or imported as graphic backgrounds.

Lotus Domino Designer provides the following simple actions that you can attach to navigators:

- Open another navigator.
- Open a view.
- Serve as an alias for a folder.

  Clicking the object displays the contents of the designated folder in the view pane. Dragging and dropping a document to the folder object adds the document to the actual folder.
- Open a database, view, or document link.
- Open a URL.

In addition, a navigator can perform the following functions:

- It can run an @function formula. This requires knowledge of the Notes macro language, but offers more choices than the simple actions supplied by Domino. Clicking the object runs the formula associated with that object.
- It can run a LotusScript program. This is a more complex function to create, but offers the most flexibility. LotusScript programs can perform tasks that are not possible with @function formulas, such as the ability to manipulate a database access control list (ACL). Clicking the object runs the LotusScript program associated with the object.

Creating a Navigator

In the following section we will create a simple navigator. We will create a graphic background and add a button. The button will have an action associated with it.

You can add a navigator to your database in one of three ways:

- Copy an existing navigator from the same database.
- Copy an existing navigator from another database.
- Create a new navigator.

Whichever way you choose, you need Designer access or higher to the database. In our example, we will create a navigator from scratch.

To create a navigator, do the following:

1. Open the database where you want to create the navigator in Design mode.
2. Go to the navigator pane.
3. Click the New Navigator button. This will bring up the Programmer's Pane for navigators.

Creating a Background

There are two options for creating a background image:

- Copy any available graphic image to the clipboard and use Create - Graphic Background to paste the graphic in as the background.
- Use the File - Import dialog box to import a graphic as a background to your navigator.

Tip Whenever possible, use the File - Import method to create a background because this gives better color fidelity when the graphic is displayed.

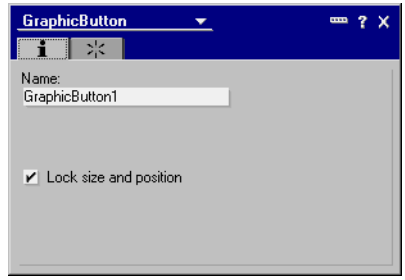
Note You can only have one graphic background for each navigator. You cannot move the graphic background once it has been pasted. Therefore, when you create the graphic object, consider the position and size of its components before you use it as a graphic background.

Tip To remove a graphic background, choose Design - Remove Graphic Background.

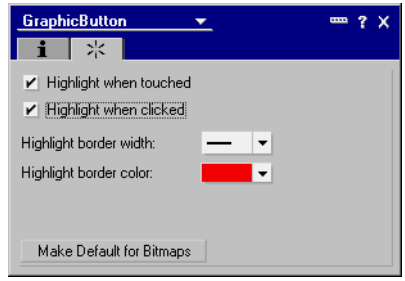
Creating a Graphic Button

Creating a button is done in the same way as creating a background:

1. Use cut and paste or the Import dialog box to build the graphic button.
2. Move the button to the desired position by dragging it.
3. Choose Design - Object Properties to display the InfoBox.



4. Select Lock size and position.
5. Click the Highlight tab.



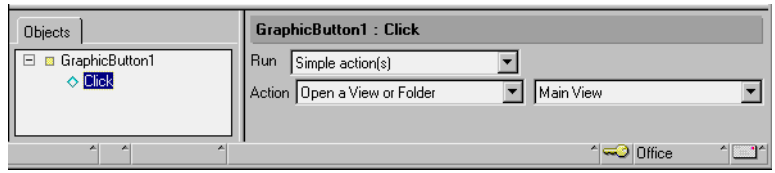
6. Select Highlight when touched and Highlight when clicked.
7. Close the InfoBox.

Tip You can remove a graphic button by selecting it and pressing **DELETE** on your keyboard.

Adding an Action to a Navigator Object

It is very easy to add a simple action to a navigation object. For example, if you want to add an action that opens a view, follow these steps:

1. Select a graphic button or create a hotspot.
2. In the bottom pane, select Simple action(s).
3. From the Action drop-down list, choose Open a View or Folder.
4. From the drop-down list next to the Action drop-down list, choose a view. It now looks like this:



5. Save your changes.

Adding an Action Using @Functions or LotusScript

If you require a more complicated action to be added to a graphic object, you can create the action by using an @function or a LotusScript program. You do this in the same way as for Simple Actions, except that you select the Formula or LotusScript option button in the bottom pane.

Note Make sure that Click is selected in the Event area. This ensures that the LotusScript program is run when the user clicks the object.

Testing a Navigator

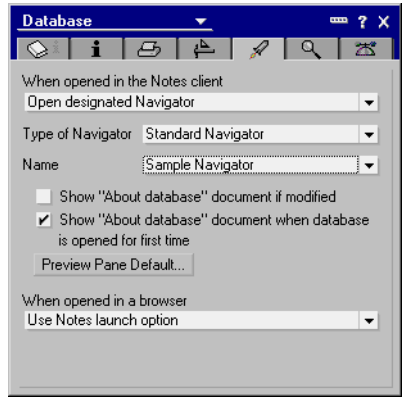
To test a newly designed navigator, follow these steps:

1. Open the new navigator and choose one of the following options:
   - Preview in Notes
   - Preview in Web Browser
2. Select and click each object to see if the highlighting and the actions occur as expected.
3. If the test is not satisfactory, deselect the Preview tool to return to Design mode and make changes as required.
4. Next, test the navigator using some documents. For navigators with actions that perform multiple steps or complex tasks, split the process into several smaller tasks and create an action for each task. Test and fix each small task first. When everything is working correctly, combine the formulas into one, and then test the navigator again.

Including a Navigator in the View Menu

To display a navigator when a database is opened, follow these steps:

1. Open the database InfoBox.
2. Click the Launch tab.
3. To display the navigator in the navigation pane, choose Open designated navigator under On Database Open. If you want to display the navigator in a full-screen window, select Open designated navigator in its own window.
4. From the Navigator drop-down list, select the navigator that you want displayed in the view:



5. Close the InfoBox.

When the database is opened, the navigator should launch.

Summary

Views are the entry point to the data stored in a database. When users open a view, a list of documents in the database is displayed, each row presenting pieces of information from a document. As such, views give users a logical and organized overview of information available in Domino databases.

The dynamic nature of views allows application developers to design highly flexible entries for databases and Web sites, based on user requirements or access levels.

Navigators are graphical image maps containing several active areas called hotspots or buttons. These hotspots or buttons can be links to other Domino objects or Web pages located outside Domino, and can also launch agents to perform specific tasks on documents stored in Domino databases.

Chapter 6
New R5.0 Design Elements

This chapter describes the new design elements that are available in Domino R5.0; what they are, and when and how you use them. In summary, the new elements are:

- Pages
- Framesets
- Outlines
- Resources

After you have read this chapter, you will understand what pages are and where and when you can use them. You will also learn how to create and manage the pages.

Before reading this chapter, you should be familiar with the functions described in Chapter 4: Forms, as that chapter describes how to create and use various elements in a form. These elements will also work in pages.

This chapter also covers outlines. After you have read this chapter, you will understand when it is beneficial to use outlines and how you can create, modify and insert outlines into your form or page.

Pages

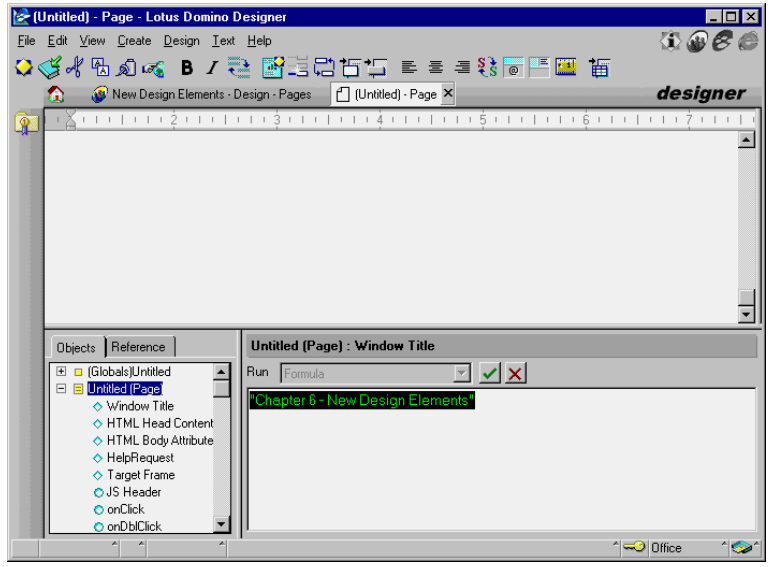
Pages are a new type of design element. They are a cross between a standard form and a "special" form (Help About, Help Using). Pages are a part of the design collection, which implies that they are inherited from a template when that template's design is inherited.

A page can be thought of as a form without fields, but action buttons, LotusScript objects and events work in the same way as they do on forms. All rich capabilities (images, attachments, buttons, hotspots, OLE, animated .GIFs, and so on) are fully functional. Pages are stored in the Design Collection (Pages) and are named as are other design elements. You can also use aliases to identify your pages, and use those aliases within your code.

Pages provide application developers with a much improved level of control over the layout of Web pages on Domino sites. Application developers not familiar with traditional Notes development, but with experience designing Web pages, now have a WYSIWYG HTML authoring tool that provides support for a broad range of browser technologies including HTML 4, image file formats, Java applets, Active X components and multimedia objects.

Creating a New Page

You can create a new page by choosing Create - Design - Page or by clicking the New Page button in the Design Collections Page View pane. The new page looks like this:



Elements that are not enabled in pages are Create Field, Layout Region, Actions (you can use buttons though), Subforms, and a few Embedded Elements. Everything else is fully functional.

Specifying Page Properties

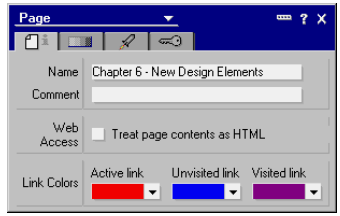
The Page InfoBox contains all of the information related to pages.

To look at the page properties, do the following:

1. Click the Properties icon.
2. In the InfoBox displayed, click the triangle in the top middle of the InfoBox and select Page. An InfoBox is displayed which allows you to set the properties of the page. There are three tabs:
   - Page Info
   - Background
   - Launch

Page Info Tab

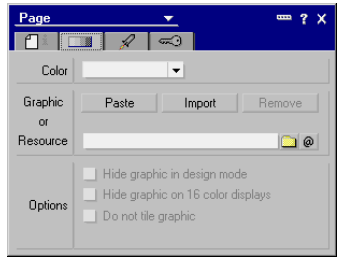
The Page Info tab stores general information about the page.



1. In the Name field, specify a name for the page.
2. The Comment field allows you to enter some informational text about this page. This comment will show only in the page designer view and therefore should contain information relevant to the designer rather than the user.

Background Tab

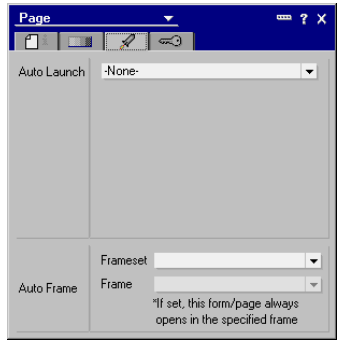
The Background tab allows you to specify the background color for the page. You can also paste in a graphic using the clipboard or import a graphics file.



Note Although Domino supports RGB colors, the quality of the image tends to be better when the image is imported rather than pasted.

Launch Tab

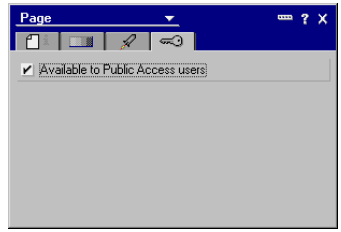
The Launch tab specifies what happens when the page is loading:



1. The Auto Launch option determines which object is launched first.
2. The Frameset option allows you to select the frameset which will open when the page is loaded.
3. If you select Frameset, you need to decide in which frame the page is opened. You can select the current frame by using the Frame drop-down list.

Security Tab

On this tab you specify whether this page is available to Public Access users or not.



Page events are just like form events. For more information about events, see Chapter 4: Forms.

Using Pages

You may be wondering why you would use a page element rather than a form. On the surface, it seems to have the same utility as a form, but without the option to create any fields or subforms within it. Why not use forms instead? The answer is that pages are designed to replace the $$ViewTemplate, and for that reason they are very useful. Furthermore, you can determine that when the database is opened, it launches the page first. Therefore, it can appear as the About This Database form did in earlier Domino releases. However, using pages you have more options than you had with the About This Database form, and you can also insert the pages into the framesets.

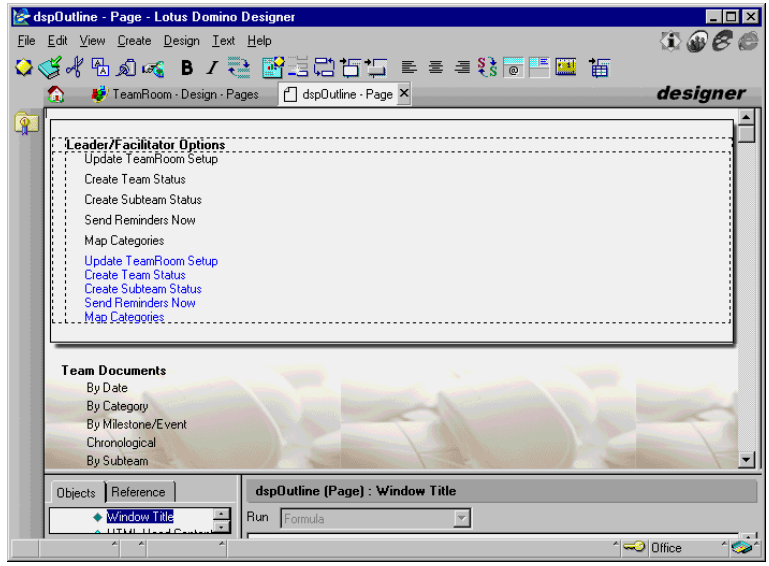
Pages are intended more to display information rather than to gather information from the user. They can include views and image links to other pages or forms.

TeamRoom Database

We will now open the TeamRoom database to see how the pages are used in that application.

1. Create a new database using the TeamRoom template.
2. Open this database in Design mode.
3. Go to the Pages design collections. The view pane shows all of the pages in the database.

4. Open the dspOutline page by double-clicking it in the view pane. The page should look like this:



5. The dspOutline page contains one embedded outline element and several action hotspots.

   Note Although the dspOutline contains only the embedded outline element and several hotspots, you can use many more elements and objects in your page.

Launching Pages

There are many options for launching pages, some of which include the following:

- **Inserting the page into a frameset.** Select the frame where you want to put the page, and then click the right mouse button and select Frame Properties. Select the Named Element in Type option, and from the Kind options, select Page.

  Note You can also select a page from another database.

- **Using existing framesets.** Open the Pages InfoBox and select the Launch tab. Select the Frameset option and choose the framesets where you want to insert the page. Select Frame options and select the page or form that you want to replace.

- **Launching the page from another site.** If you want to open the page from a form using action hotspots, choose Create - Hotspots - Action Hotspots and enter the following in the Programmer's Pane:

  ```
  @URLOpen("http://" + @Subset(@DBName; -1) +
  "TeamRoom5.nsf/WelcomePage?OpenPage");
  ```

- **When database is opened.** Open the database InfoBox and select the Launch tab. Choose Open Designed Navigator in the On Database Open dialog list. Next, select the page from the Type of Navigator dialog list and select the current page name in the next field.

  Note In the On Web Open options, select the Use Notes Launch option if you want to open that page for Web users as well as Notes client users.

Framesets

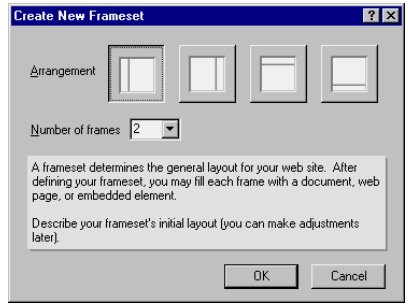
Framesets are a standard technique used to present a multi-pane interface to the user. Using the Lotus Domino Designer, you can create framesets and then associate specific pages, views, forms, Java applets, ActiveX components, or a URL with each frame.

Creating a Frameset

To create a new frameset, do the following:

1. Open the database in Design mode.
2. Select Framesets from the list of design elements.
3. Click the New Frameset button.

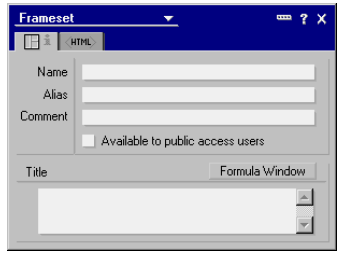
   A window will be displayed where you can choose your initial configuration. Select the layout you want and click the OK button.



   This will bring up the frameset.

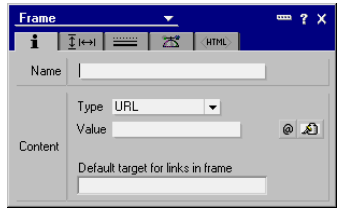
4. Click your right mouse button within the frameset.

5. A window will pop up. Click Frameset Properties to display the frameset InfoBox. It looks like this:

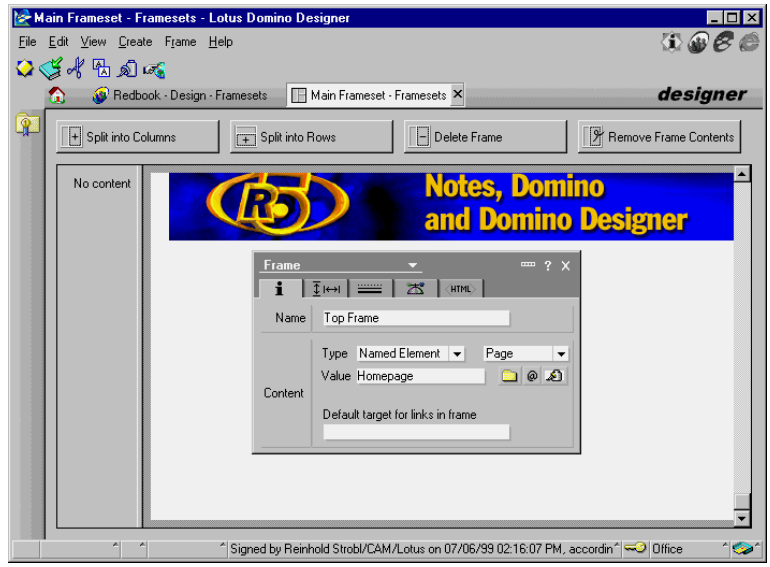


6. Enter the name, the alias, and a comment for the frameset. After you have done so, you may close the InfoBox.
7. Adjust the size of the frames to your needs. To do so, click with the left mouse button on the border line between the frames, hold down the mouse button, and drag the border line to its new position.
8. Select the frame that you want to start working with and bring up the InfoBox for this frame. Again, you do this by clicking the right mouse button within the frame and selecting Frame Properties.

   The Info tab of the InfoBox looks like this:

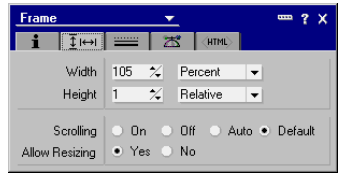


9. Specify a name for the frame. In our example, choose Named Element as type and select an already defined page. The result looks like this:



You may also specify a URL or a link as the type for this frame. All other input fields on this tab will change accordingly.

10. Switch to the size tab of the InfoBox.

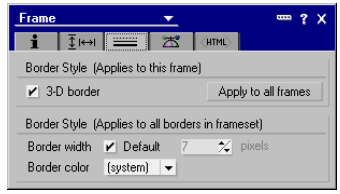


On this tab you can manually adjust both the width and height of the frame. The following options are also available:

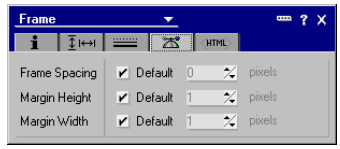
- **Relative.** This means that you specify the width and height relative to the frames in this set. For example, if you have two frames and set the width for the first one to 1 and the second to 2, the second frame will be twice as wide as the first.
- **Pixel.** With this selection you provide an absolute value measured in pixels.
- **Percent.** This option allows you to specify the width and height values as a percentage of the window.

You can also select scrolling settings as well as whether or not the frames may be resized.

11. Switch to the Border tab of the InfoBox.



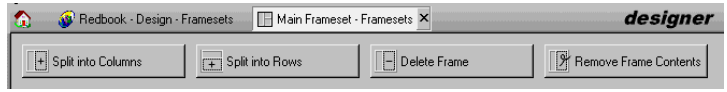
12. Here you can specify the style of the borders around the frames as well as the width and color of the borders.
13. Switch to the Advanced tab of the InfoBox.



14. On this tab you can fine-tune your layout by specifying the spacing between the frames and the margins within the frame.

Changing the Layout of a Frameset

You can change the layout of the frameset whenever there is a need to do so. Simply select the frame that you want to be altered, and select the desired action. Using the buttons shown in the following figure, you can add, delete and alter the contents of the frame. You can then size the frame by dragging its border.

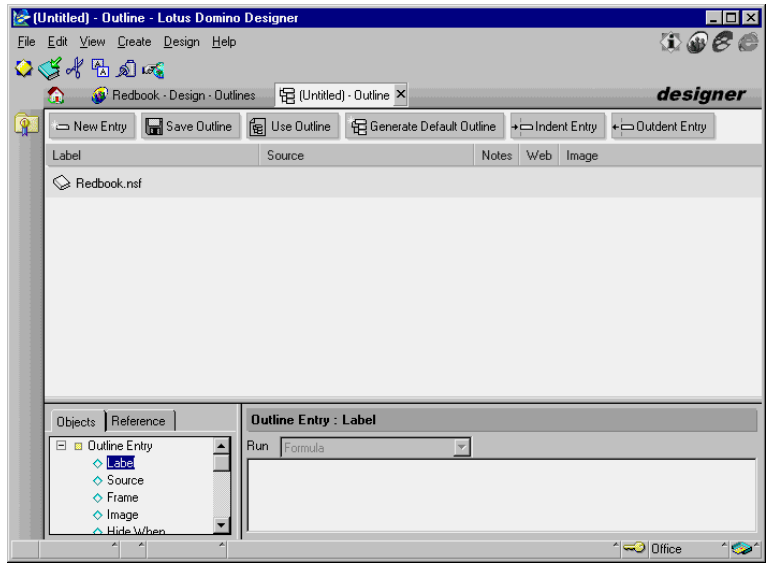


Outlines

The Outline Designer enables you to create a hierarchical tree structure of links and elements for your site, similar to a high level site map.

Because outlines are a new feature and the work area is different than that for other design elements, we will treat outlines separately in this next section.

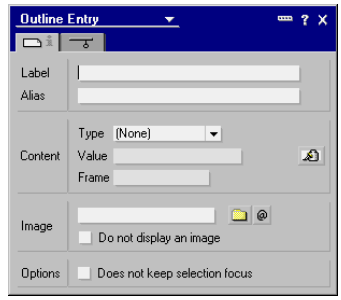
When you create a new outline, the work area looks like this:



The Outline Design work area consists of buttons, the Outline view, and the work area.

Buttons

1. **New Entry:** You can add a new entry to your outline. When you click this button, the new entry InfoBox is displayed:



- You can name your Entry in the Label field. This is shown to users, so make it as descriptive as possible.
- The Type field allows you to select the Link type. The available options are Link, Named Element, and URL.
- If you select Link or Named Element as type, a new field appears that allows you to specify in more detail what you are linking (page, form, database, and so on). This field gives you a combobox where all the available elements are shown.
- The Value field is only used for online entries of the URL type. In this field you type the URL address for this entry or you can use a formula to compute it dynamically.
- Frame is the target frame in which to display the link. This setting is optional.
- The Image value indicates which icon to display to the user, to the left of the label value. Images are stored in the Image Resources part of the database.

2. **Save Outline** saves the outlines.
3. **Use Outline** creates a page and then displays a dialog list where you can select the current outlines.
4. **Generate Default Outline** creates a default outline. This outline will be based on the folders and views in the current database.

   Tip It is generally quicker to create a default outline first and then modify it, rather than creating an outline by adding one entry at a time.
5. **Indent Entry** enables you to indent the outlines.
6. **Outdent Entry** enables you to outdent the entries.

Outline Pane

This is the place where you can modify your outlines by dragging and dropping the entries from one place to another. You can also change the entry settings and options by opening the Entry InfoBox.

Programmer's Pane

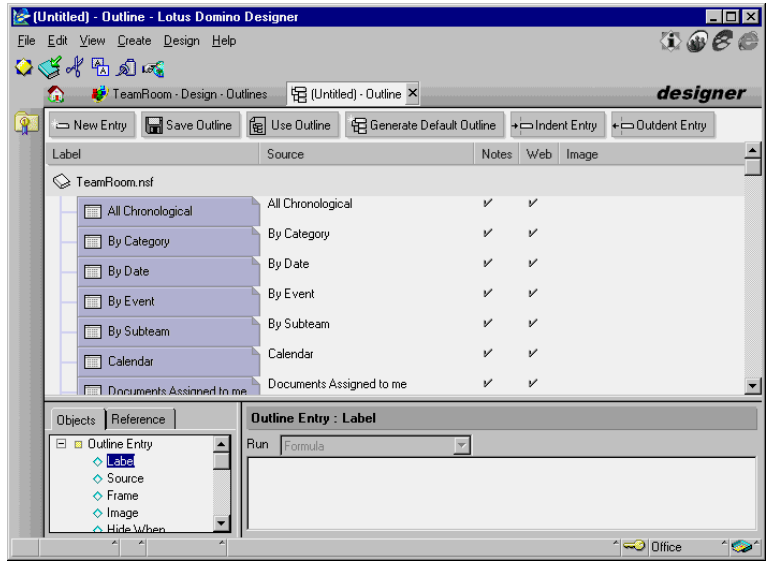
This contains the Entries Events so that you can dynamically generate their values at runtime. The available events are:

- **Button Label:** You can name your Entry.
- **Destination:** Defines the place where the page, form, HTML-document, and so on are stored.
- **Frame:** The frame name that should open.
- **Icon:** The icon number that you are using in front of the entry.
- **Hide When** Defines when the entry is displayed.

Creating a New Outline

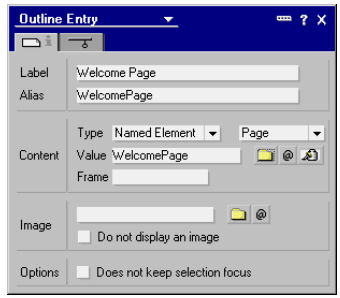
We are now going to create a new entry which is linked to the page called WelcomePage in the TeamRoom database. To create a new outline, follow these steps:

1. Open a database in Design mode.
2. Go to the Outline design view and click the New Outline button.
3. Click the Generate Default Outline button. Designer generates the outline using the folders and view pane. The outline might look like this:



4. Add a new entry by using the New Entry button.
5. Name the entry, for example, Welcome Page.

6. Link the page by selecting Named Element in the Type field, selecting Page in the Kind field, and adding the page name, WelcomePage, in the Value field.
7. When you have inserted all of the required information, the entry InfoBox should look similar to the one below:



Note You can always come back and modify each of the entries by opening the InfoBox.

8. Save the outline.

To delete the entry, right-click and select Clear.

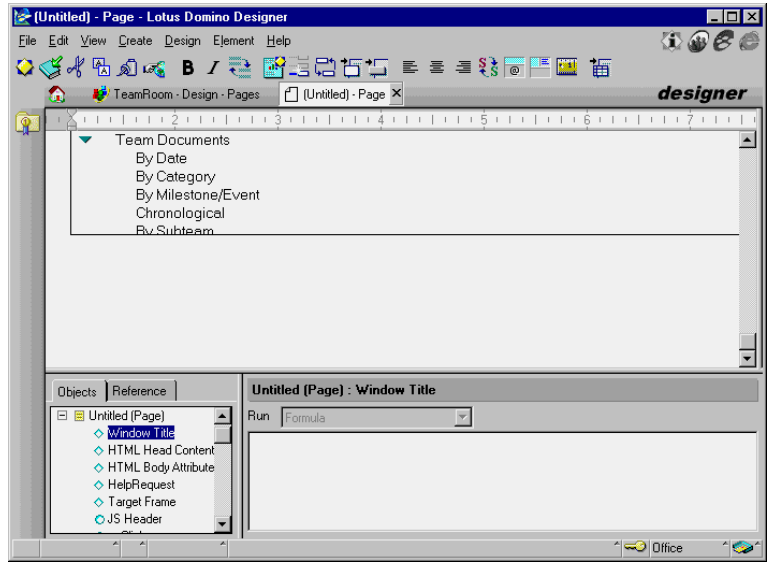
Embedded Outline

The Embedded Outline is one of the embedded elements in Domino Designer, and it allows you to insert the outlines into your form, page, or subform. After you have inserted the outlines into your page, you don't need to reinsert or modify the current page if you make any changes to the outlines. Domino already has this capability. This can save time when maintaining your application.

To insert the outlines, do the following:

1. Create a new page or go to the existing page.
2. Move your cursor to the place where you want to insert the outline.

3. Choose Create - Embedded Element - Outline to insert the outline into the page.



Note You can also insert the outline into a table or nested table.

Embedded Outline InfoBox

The InfoBox looks like this:

Below are some of the features that you can control from this InfoBox:

Info Tab (shown)
Here you can identify your outlines by name. Furthermore, you can select the style of the outline, the vertical or the horizontal order, and the size. You can also specify the position of titles, as well as top and sub levels. Web access allows you to use the Java Applet (which works as it does in the Notes client) or to use standard HTML.

Font Tab
The Font tab allows you to specify the font type, style, color, and size for the whole outline.

Background Tab
The background tab allows you to specify the background color of the outline or allows you to add an image as a background.

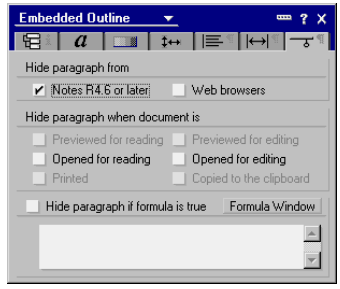
Alignment Tab
The Alignment tab enables you to customize the spacing and alignment of the outline entries, images, and background.

You can also specify the tab settings for the fields. You must use this option if you have multiple fields on the same line and want to keep them in the same place that you have placed them in the form.

Although you can set the tabs manually, it is better to set them directly in the form using the ruler.

Note Printing properties do not have any effect on printing from the Web.

Hide When Tab
The Hide When tab looks like this:



Several check boxes are available to hide the embedded outline on predefined conditions. You can also specify other conditions using an @function.

In the figure above, the outline control is hidden when the document is accessed from Notes (4.6 or later).

Resources

The Resources view consists of five different views:

- Images
- Applets
- Shared Fields
- Script Libraries
- Other

The advantage of resources is that you can store the resource elements and objects in one place, and then reuse them later. For example, when you want to use the same LotusScript code in several different places, you can create a new Script Library and put the code in it. You can then call that Script Library wherever you want.

This enables you to control and manage your applications more easily because you can store images and applets within resources and then share these elements, rather than copy and paste the images or applets into several different locations.

Images

The images resource allows you to store all of the images that you are using in your database. This is a very useful feature because the database can now contain all the pictures that it needs and you don't need to worry about sending all the images along with the application when you send an application to another server, for example.

Creating an Image Resource

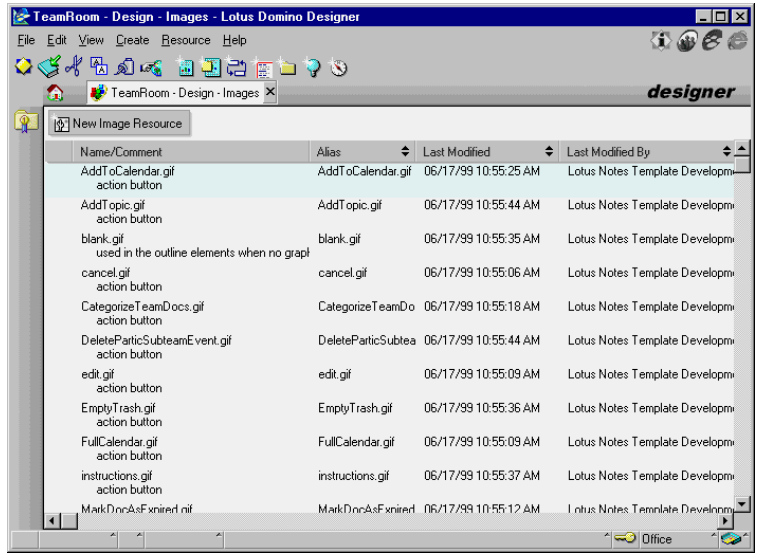
To create a image resource:

1. Open the database in Design mode.
2. From the Resource design, click the New Image Resource button in the view pane.

3. Select Insert Image, and click Open.

   Note Domino R5.0 supports .GIF, .JPEG, and .BMP graphics files.

   When the image is inserted in the view pane, it should look like this:

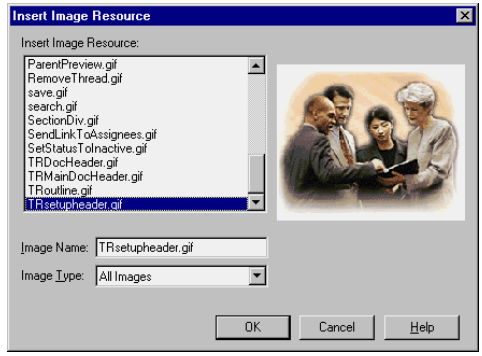


4. You can specify your Image by opening the InfoBox and entering the alias name and the comments for that image.

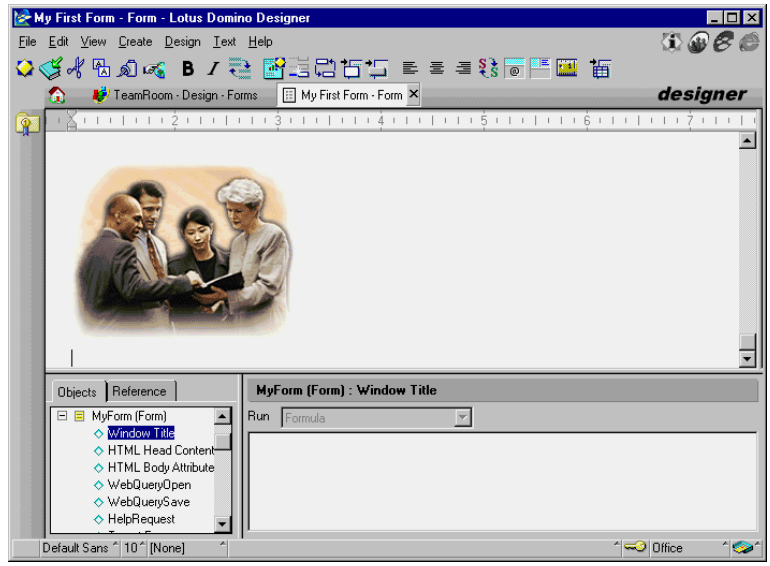
Adding the Image to Your Form

Next, place the image that you have created onto the form.

1. In the database design, go to the Forms design view.
2. Click the New Forms button in the view pane. The new form will be displayed.
3. Go to Create - Image Resources in the Main menu.
4. The Insert Image Resource window displays:



5. Select the Image that you want to insert.
6. Click OK.
7. The image is now inserted into the form; it will look similar to the one shown below:



Note Domino now stores .GIF images in their native format. When Domino displays an animated .GIF in the image selection dialog box and on the page the .GIF is actually moving.

8. Save the form.

Note You can preview the form by choosing Design - Preview in Notes.

Applets

Java applets that you are using in many places can be added to a Resource. Then, if you make any changes to the applet later, you only need to copy the revised applet into one place and those applications that use the applet will then have access to the new applet.

Shared Fields

The Shared Fields resource contains all the shared fields of the database. If you want to make any changes in the shared fields, you can easily open the field by double-clicking the current field name in the design view. For more information about Shared fields, read Chapter 4: Forms.

Script Libraries

The Script Libraries resource contains all the LotusScript and Java code segments that you want to use from other scriptable objects. This allows you to handle and manage your code more easily.

Other

The Other resource contains the icons, Using Database document, About Database document, database script, and actions. The first four are the same as in Release 4; actions is a new feature.

Actions

This new feature in R5.0 allows you to store shared actions that you may be using in many forms and views in one place.

Summary

In this chapter we looked at the new functions in Domino R5.0: pages, framesets, outlines, and resources, and how to use them in your applications.

Chapter 7
Agents

This chapter describes Domino agents: what they are, what they do, where to use them, and how to create them. The chapter also covers Web agents, WebQueryOpen and WebQuerySave, and how to make agents available for Web users.

You will also learn how to write agents using LotusScript, how to access CGI variables, and what access you need in order to run an agent.

About Agents

Agents allow you to automate many tasks within Domino. They are standalone programs that perform a specific task in a database for the user, for example, filing documents, changing field values, sending mail messages, deleting documents, or performing more powerful actions, such as interacting with external applications. Agents are the most flexible type of automation because they can be run by users or in the background, and they are not tied to a specific view or form.

Agents can either be *private*, created by the user and used only by the user, or *shared*, created by a designer and used by anybody who has sufficient access to the application. Both private and shared agents are design elements stored in the database for which they are created. They can be run manually by the user, automatically when certain events occur such as mail arriving or documents being changed or added to the database, or scheduled to run at certain intervals. They can contain Notes simple actions, @function formulas, or a LotusScript or Java program.

Access Control

In the Access Control List (ACL) for the database there is an option to Create Personal Agents. Since personal agents on server databases take up server disk space and processing time, the database manager may deselect this option to prevent users from creating personal agents.

The following options in the database access control list affect agents:

- To create a shared agent, a user must have Designer access or higher.
- To create a personal non-LotusScript agent that is stored in a shared database, a user must be assigned the Create Personal Agents authority.
- To create a shared LotusScript or Java agent, a user must be assigned the Create LotusScript Agents option in the access control list. To store the agent in a shared database, a user must also be assigned the Create Personal Agents authority.

When agents run, they automatically check the identity of a Domino user against any server document or ACL restrictions. Manually run agents run with the identity of the Domino user; scheduled agents run with the identity of the person who created or last modified the agent.

Note A Domino administrator can also specify restrictions in the server document to prevent users from running agents on a server. Users denied this server access cannot create personal agents on the server regardless of the ACL setting.

Note Since Web users do not have a Notes ID that identifies them, you must choose Agent - Agent Properties, click the Design tab, and select For Web Access: Run Agent as Web User.

Restricted and Unrestricted Agents

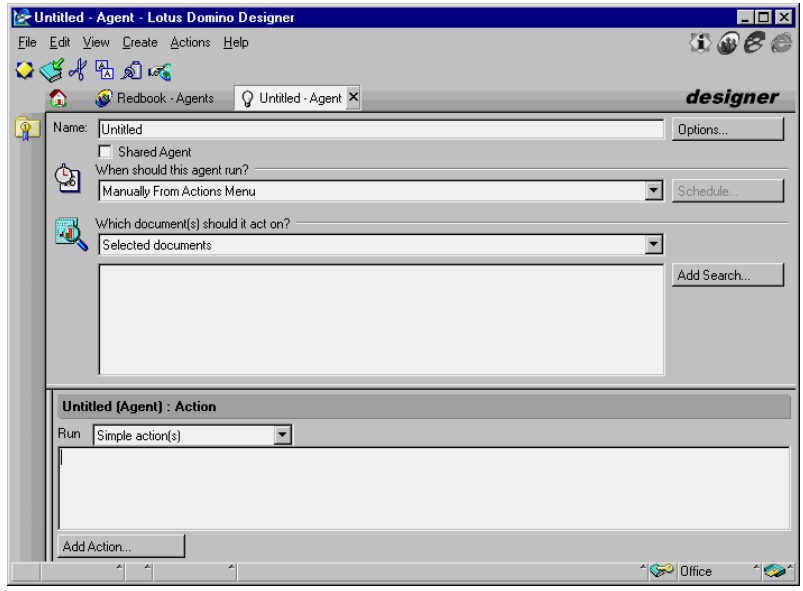
In the server document of the Domino Directory you can determine who can run unrestricted and/or restricted agents. Using unrestricted agents, users have full access to the server's system. By using certain LotusScript methods, properties, and classes in an agent, you can determine that the agent can only run if the user has the authority to run unrestricted agents. However, be aware that this does not apply on the Web. Web users can run any agent as long as the agent is not hidden from Web users.

Creating an Agent

There are a couple of ways to create an agent:

1. Open the database in Design mode, go to the Agents pane, and click New Agent.
2. Copy an existing agent by opening the database you want to copy from in Design mode and using cut and paste.

Whichever method you use, the following Agent Builder window is displayed:



Naming the Agent

The first thing to do is to give the agent a name. A descriptive name is especially important for an agent that you are designing for users to select from the Action menu. Also try to keep the first character unique. This is because, as with forms and views, Domino will use the first unique character as an accelerator key. Alternatively, you can force Domino to use a letter of your choice as an accelerator key by putting the underline character in front of it. It is also good programming practice to provide an alias for the agent.

Click Shared Agent if you want this agent to be used by other users.

Using the Options button will allow you to set the agent to be available for public access. Creating agents enabled for public access allows users with No Access or Depositor access to view and use any manually-run agents.

Caution Once you have saved an agent you cannot change a shared agent to a private agent or vice versa.

Note Personal agents should not be created in design templates because their changes cannot be distributed in design replace or refresh procedures.

Scheduling the Agent

Next, select when the agent will run. The following list of options is available:

- **Manually From Actions Menu**

  Select this option for all user-activated agents or those triggered by WebQuerySave or WebQueryOpen agents or formulas. It is the only choice that allows users to see the agent in the Actions menu.

- **Manually From Agent List**

  Use for hiding agents that are run by another agent or that are still being developed. If the agent is called by another agent (the main agent), the document selection is ignored. The main agent always determines the document selection.

  Note The name of a hidden agent is surrounded by brackets.

- **Befor New Mail Arrives**

  Use for processing mail before it is listed in the mail database, for example, to move incoming mail to a folder.

- **After New Mail Arrives**

  Use for processing incoming mail to respond to it, forward it, or file it.

- **If Documents Have Been Created or Modified**

  Use for workflow tasks where an action is performed based on new or changed documents.

- **If Documents Have Been Pasted**

  Use if documents are pasted into the database routinely and you want to modify them as they are being pasted.

- **On Schedule More Than Once a Day**

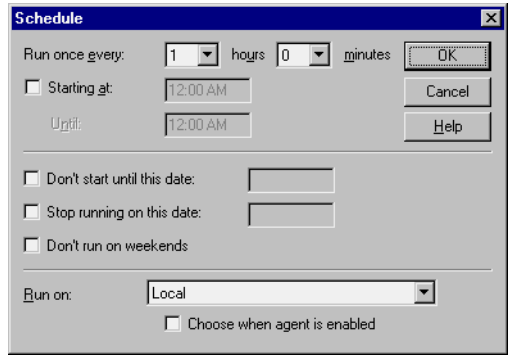
  Use this option for high-priority databases, such as those critical to a business process and for databases that replicate several times a day.

  Caution If you schedule very frequent runs, for example every 5 minutes, the server's performance could be affected.

- **On Schedule Daily/Weekly/Monthly/Never**

  Use to schedule agents to run at regular intervals.

If you have chosen to run the agent on an interval basis you may click Schedule to bring up a dialog box to schedule the specific run time. The figure below shows an hourly-scheduled agent:



To set up an interval-scheduled agent:

- Specify Run Once Every in a range between 5 minutes and 11 hours 55 minutes.
- Specify the start and end time of each day.
- Specify the start and end date for the agent to run, and whether it should run on weekends or not.
- Specify the server on which the agent is to run.

  The following options are available:

  - **Running the agent on the original server**
    To run the agent on the same server where you create the agent, leave the default setting in the Run box.
  - **Running the agent on another server**
    To run the agent on another server, select a server in the Run list box or enter a server name.
  - **Running the agent from any server**
    If you cannot specify a server in advance, select -Any Server- as the server name in the Run box. This wildcard entry allows any server to run the agent. Choosing this may result in replication conflicts if several servers run the same agent and change documents.
  - **Allowing users to choose which server runs the agent**
    If you select Choose When Agent is Enabled, users are prompted to select a server when they enable the agent. This is useful for distributing agents in ready-to-use applications.

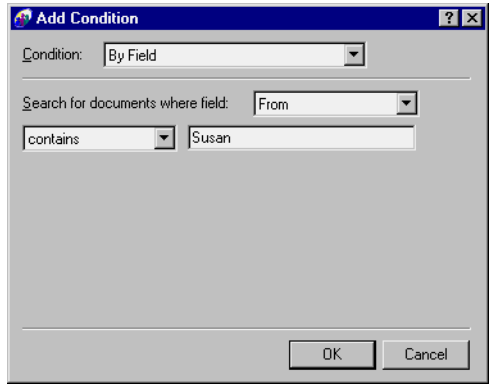
Tip If the agent is modifying data in a database, it should run just once on one server. The changed data is then replicated to the other replicas of the database.

Selecting Documents to be Processed

This selection is set depending on the option selected for scheduling the agent. For example, if the agent is scheduled to run if new mail has arrived, this option is set to Newly Received Mail Documents, and it cannot be changed.

You can further select which documents are processed by specifying search criteria. For example, if you want to process only documents in your mail box which have been sent by a specific person, click the Add Search button to activate the Search Builder window.

1. From the Condition drop-down box, select By Field.



2. In the Search for documents where field box, select From.
3. Select Contains.
4. Type in the search criteria.
5. Click OK to save your settings.

The following table shows which document options are allowed with which run options:

Run options	*Document options**
Manually From Actions Menu	All documents in database. All new and modified documents since last run. All unread documents in view. All documents in view. Selected documents. Run once.
Manually From Agent list	All documents in database. All new and modified documents since last run. All unread documents in view. All documents in view. Selected documents. Run once.
Before New Mail Arrives	Each incoming mail document.
After New Mail Has Arrived	Newly received mail documents.
If Documents Have Been Created or Modified	Newly modified documents.
If Documents Have Been Pasted	Pasted documents.
On Schedule More Than Once a Day	All documents in database. All new and modified documents since last run.
On Schedule Daily, Weekly, Monthly, Never	All documents in database. All new and modified documents since last run.

* To narrow down the document collection, specify search conditions. Click Add Search when the agent is in Design mode.

Specifying What the Agent Should Do

There are five ways of specifying what the agent should do: simple actions, formulas, LotusScript, imported Java, and Java.

Simple Actions

These are predefined actions which allow you to define a sequence of actions without requiring any programming knowledge. They are ideal for the end user who wishes to automate some routine tasks. The simple actions available are:

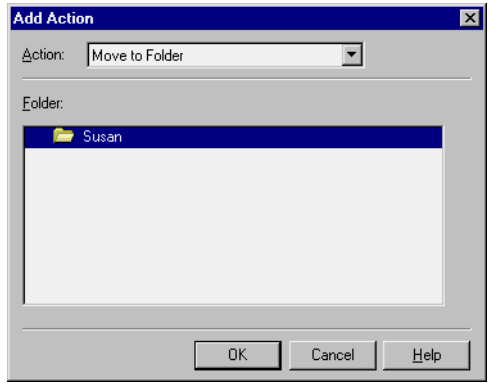
- Copy to Database
- Copy to Folder
- Delete from Database

- Mark Document Read
- Mark Document Unread
- Modify Field
- Modify Fields by Form
- Move to Folder
- Remove from Folder
- Reply to Sender
- Run Agent
- Send Document
- Send Mail Message
- Send Newsletter Summary
- Run @Function Formula

In our previous example, we used the Search Builder to select only those documents sent by Susan. Now we can easily write a simple action to store these documents in a folder.

To do so:

1. Return to Agent Builder window.
2. Click the Simple Action(s) option button.
3. Click the Add Action button.
4. In the Action field, choose Move to Folder.



5. Select the folder where you want to move the document.
6. Click OK.
7. Save your changes.

We have now built an agent which stores selected documents in a folder.

Note You can combine simple actions in one agent to build more complex functions.

Caution Simple Actions refer to the displayed name of design elements. Therefore, changing the name of such an element will cause the agent not to work.

Formulas

Formulas can use the full range of @functions available with Domino. You can write an @function formula that runs by itself or with a simple action. You cannot combine an @function formula with a LotusScript program.

- To write a stand-alone formula, click Formula in the design pane and type the formula in the panel.
- To combine a formula with a simple action, click Simple action(s) and then click Add Action. Choose @Function formula from the Action list and type the formula in the editing window.

  Click Fields & Functions to select an @function or field and paste it into your formula. Click @Commands to select and paste an @command into your formula.

Formula-based agents operate on each of the documents in turn and run the complete formula on a document before proceeding to the next document.

You can apply search criteria through the agent interface to specify which documents in the database are to be processed. A SELECT statement in the formula further limits the search. If you do not include a SELECT statement in the formula, Domino appends a SELECT @All statement. Except for SELECT @All, a SELECT statement must be the first statement in the formula to be effective.

For example, if you want to forward documents only if they do not have attachments, do the following:

1. In the Agent Builder window, click the Formula option.
2. Enter the following formula in the Programmer's Pane:

```
@If (@Attachments>0
    @Return(" ");
    @MailSend("Gina"; " "; " "; Subject; _
        "Please handle this" + @Newline; "Body" ; " " )
 );
```

LotusScript

Agents can also be written in LotusScript.

1. In the Agent Builder window, select LotusScript as programming language.
2. Enter the LotusScript code in the Programmer's Pane.

Script-based agents run once and must therefore process all documents selected. You supply the search criteria and the processing order through the language constructs. Search criteria applied through the agent interface are effective only through the UnprocessedDocuments property of the NotesDatabase class. This property contains all documents not yet processed by the agent or the result of the search specified to the agent builder, depending upon how you create the agent.

Note Control is always passed to the agent using the Initialize event, so this is where your program should begin.

Imported Java

To attach a Java program to an agent, first write the program in a Java development environment. In Domino Designer, click Imported Java and then click Import Class Files to import the files into the agent.

Java

Use this selection to create your Java written agent using the Programmer's Pane of Domino Designer.

Displaying the Agent Pop-up Menu

1. To display the pop-up menu of an agent, click with the right mouse button on the agent listed in the agents pane. The agent pop-up menu is shown:

2. From this menu you can:
   - Display the agent InfoBox.
   - Cut and copy to the clipboard, and paste from the clipboard.
   - Clear the agent, which means deleting it.

- Edit the agent, which is the same as double-clicking the agent's name in the Agent pane, to display the Agent Builder window.
- Run the agent.
- Test the agent, which tells you how many documents the agent will process.
- Enable or disable the agent.
- Look at the agents log.

Testing an Agent

Test agents during development and again before copying the agent to a server database.

Testing an Agent During Development

You can quickly test an agent by simulating a run without affecting documents.

1. Select the database and go to the Agent pane.
2. Select the agent and choose Actions - Test.
3. Read the Test Run Agent Log, which describes how many documents would be processed and what action would be taken if the agent were actually run.

Testing an Agent Before Copying it to a Live Database

For agents that have multiple steps or complex tasks, split the process into several smaller tasks and create an agent for each. Test and fix each smaller agent first. When everything is working correctly, combine the agents into one and then test the agent again.

1. Choose File - Database - New Copy to make a test copy of the database with documents. For all agents except those that act on mailed documents, the test copy can be local.
2. If the agent works on mailed documents, the test database must be on a server, and a Mail-in Database document must exist in the Domino Directory. Mail a few documents to the test database.
3. If the agent works on pasted documents, paste a few documents into the database before running the test.
4. If you don't need to run the agent from a view, select the database and go to the Agent pane, select the agent you're testing, and choose Actions - Run; otherwise, open the database, select the view, and choose Actions - <Agent Name>.

5. Make any required changes to the agent to fix any problems that the test run shows. If necessary, create a new copy of the database to run the agent again.
6. When the test shows no problems, copy the tested agent to the live database.

Checking the Agent Log

Every time an agent runs, it writes a report that includes when it ran, how many documents it ran on, and what actions it took on those documents. Each new run of the agent writes over the previous log report. Domino stores the Agent Log with the database.

To view the most recent Agent Log:

1. Select the database and choose View - Agents.
2. Select the agent whose log you want to check and choose Agent - Log.
3. Click OK to close the Log window.

If there is no Log (because the agent has never run), you will see the message, "This agent has never been run before."

Debugging Agents

To debug your agents you can use the debugging features of Domino Designer. However, it is sometimes difficult to debug agents on your workstation when these agents are expected to run on a server or on the Web. Use one of the following techniques to debug such agents:

- Include MESSAGEBOX statements to print out statements.

  This method is convenient, but can clutter the server log if you have many statements, or run the agent repeatedly. You either need your own testing server or permission to populate a server log with debugging information.

- Use the AgentLog class to write out your statements.

Disabling Scheduled Agents

If you have Designer access or above you can disable any agents, except manually run agents, in order to prevent servers from running them automatically. This is useful for debugging a problem with an agent. Designers can still run disabled agents by selecting an agent at the agent pane, and choosing Actions - Run. After you re-enable them, scheduled agents resume their schedule.

To Disable Individual Agents

1. Select the database and go to the Agent pane.
2. Click the enabled agent and choose Actions - Enable. The checkmark will disappear next to the agent name, indicating that it is disabled.

To enable a disabled agent, click the agent and choose Actions - Enable.

To Disable All Automated Agents in a Database

Disabling all agents is useful for debugging a problem with an agent running on a server.

1. Select the database and choose File - Database - Properties.
2. Click Disable Background Agents for this database.

Troubleshooting Agents

The Agent Manager runs background agents — scheduled, mail-activated, and change-activated agents — according to the schedules that you specify in the agent manager section of the server document in the Domino Directory. If an agent is not executing correctly, or if you are experiencing performance difficulties, there are a number of areas that you can examine to correct problems.

Agent is Not Running

Check the access for the agent to make sure that the agent can run on all specified databases. For example, if you design an agent to copy documents from database A to database B, but you don't have access to database B, the agent will not be able to execute the task. To check for access problems, view the Agent Log after the agent runs.

Make sure that the agent is completing its task. If the task exceeds the amount of time allotted in the "Max LotusScript execution time" setting in the server document, the Agent Manager terminates the agent before the task is complete. Increase the allotted time for execution, or rewrite the agent as several smaller agents.

Agent Manager is Not Working

Make sure that the Agent Manager is scheduled to run when agents are scheduled. In the Agent Manager section of a server document in the Domino Directory, make sure the Start time and End time parameters are set for Daytime and Nighttime to cover the hours when agents are scheduled to run. Adjust these parameters as necessary.

Agents are Running Slowly

Check to see if you have too many agents competing for server resources. Reschedule agents for the nighttime hours, when system demand is lower. Alternatively, allocate additional system resources to the Agent Manager by increasing the "Max concurrent agents" setting and the "% of polling period" setting in the Agent Manager section of the server document. Be aware that shifting resources to the Agent Manager might slow down other server processes.

Agent Will Not Run on a Particular Server

Examine the Agent Manager section of the server document in the Domino Directory to make sure that the user who created the agent has the access to run the agent on the server. Also, check the access control list for the server to make sure that the user who creates an agent has access to the server. An agent inherits access rights from its creator, so an agent cannot run on a server to which the creator does not have access.

Debugging with NOTES.INI Settings

If you cannot find out why your agent is not running using the procedures described above, you can turn on the debugging flag for the Agent Manager in the NOTES.INI file on the server. Add the following line to the server NOTES.INI:

```
Debug_AMgr = flag
```

Flag can be one or a combination of the following:

Flag	*Output*
c	Show output agent control parameters
e	Information about Agent Manager events
l	Agent loading reports
m	Agent memory warnings
r	Agent execution reports
s	Information about Agent Manager scheduling
v	Verbose mode, showing information about agent loading, scheduling, and queues
*	All of the above information (same as turning on all the flags)

The output is written to the console log and the Notes Log.

Caution Debugging affects the server performance.

Additionally, you can turn on agent execution logging. You do this by adding the following line to NOTES.INI (You can also do this in the server configuration document in the Domino Directory.)

```
Log_AgentManager = value
```

Value can be one of the following:

Value	*Output*
0	No logging
1	Show partial and complete success
2	Show complete success

This setting gives you only a subset of information compared to what the Debug_AMgr generates, but it has less of an impact on the performance of the server.

Debugging at the Server Console

Use the following server commands for troubleshooting the Agent Manager:

- Tell amgr schedule
- Tell amgr status
- Tell amgr debug

Tell amgr schedule

Issuing this command on the server console shows the Agent Manager schedule of all the agents scheduled to run for the current day. Use it to see if your agent is waiting in one of the Agent Manager queues.

There are three queues:

- A queue for agents that are eligible to run (E)
- A queue for agents that are scheduled to run (S)

  When the time they are scheduled to run arrives, they are moved into the "Eligible to run" queue.
- A queue for event-triggered agents waiting for their event to occur (V)

  Event-triggered agents (new mail and document creation/modification agents) are queued in the "Event" queue until an event they are waiting for occurs. When the event occurs, the agents move into the "Scheduled" queue and then into the "Eligible to run" queue.

Here is an example:

```
E S 09:33 AM Today agent_a TEST_R5.NSF

S S 09:54 AM Today agent_b TEST_R5.NSF

V U agent_c TEST_R5.NSF
```

The first column contains the Agent Manager queue type. The second column contains the agent trigger type (S means the agent is scheduled, M represents a new mail-triggered agent, and U represents a new/updated document-triggered agent). The following columns show the time that the agent is scheduled to run, the name of the agent, and the database name.

Note It will take up to a minute for an agent to appear in a queue or to move from one queue to another.

Tell amgr status
The "Tell amgr status" command shows a snapshot of the Agent Manager status. It gives you information about all settings that are in effect.

Tell amgr debug
Use this command to set or change the Agent Manager debug settings. You need to use the same debug values as shown above for the Debug_AMgr setting in the NOTES.INI file.

Agents and the Web

Agents are also used in the Web environment to perform several functions. In this section, we will concentrate on agents which can be activated by a user on a Web browser.

As an application developer, you will most likely create two sets of agents in order to perform the same operations from both a Notes client and from a Web browser. The main reason for this is the difference in the way in which an application interacts with the user in the two environments. In Domino, the applications can interact with users using message boxes or by prompting information, for example, to change the values in the fields of the currently open document. On the Web, the only way to show information to users without using JavaScript is by using HTML to create Web pages. If you want to change the current document on the Web, you can only do it before the document is loaded, using the WebQueryOpen event, or before it is saved, using the WebQuerySave event.

Agents for Web users are most often written using LotusScript or Java, since simple actions are not available on the Web and @formulas do not allow you to return information to users.

Running Multiple Instances of an Agent

When Domino is being used as a Web server, add the following line to the NOTES.INI file on the Domino server.

DominoAsynchronizeAgents=1

This enables an agent to be run by more than one person at the same time. By default, the Domino server only runs one copy of an agent at a time and queues other requests.

WebQueryOpen and WebQuerySave Agents

There are two special events in all Domino forms: WebQueryOpen and WebQuerySave.

WebQueryOpen Event

A WebQueryOpen event runs the agent before Domino converts a document to HTML and sends it to the browser. Domino ignores any output produced by the agent in this context.

Examples for using this agent include performing large computations that aren't possible with @commands, or collecting statistics about who opened documents and when.

WebQuerySave Event

A WebQuerySave event runs the agent after field validations and before Domino saves the document in the database. The agent can run any operations with document data or modify the document.

An example of a WebQuerySave agent could be an agent which creates another document in the Notes database but doesn't save the current document.

To perform error checking, field validation, and other processing before Web users open or save documents, create a shared agent that runs manually. You can then write a formula that uses @Command([ToolsRunMacro]) to run the agent and attach it to the WebQueryOpen or WebQuerySave form events. This simulates the LotusScript QueryOpen and QuerySave form events that are not supported on the Web.

Using the @URLOpen Command to Call Agents

The @URLOpen function allows you to reference an agent within a formula. You can associate such a formula with a button to invoke agents from a Web browser. These agents run when the user clicks the button. The following example shows how to invoke an agent called WebDelete. Note that the agent is passed the document's UNID as a parameter.

```
UNID:=@Text(@DocumentUniqueID );

@URLOpen("/"+@ReplaceSubstring(@Subset(@DbName; -1);"
";"+")+"/WebDelete?OpenAgent&"+UNID)
```

Using a LotusScript Agent to Capture CGI Variables

CGI (Common Gateway Interface) is a standard for interfacing external applications with HTTP servers. When a Web user saves a document or opens an existing document, the Domino Web server uses CGI variables to collect information about the user, including the user's name, the browser, and the user's Internet Protocol (IP) address.

You can use the DocumentContext property of the NotesSession class to capture CGI variables. The property returns a Notes document which contains all CGI variables that are applicable to the session. You can use these values to collect or process information for the current session.

The following example demonstrates how to access CGI variables:

```
Dim session As New NotesSession

Dim doc As NotesDocument

Dim CGIValue As String
```

To create an instance of the DocumentContext:

```
Set doc = session.DocumentContext
```

Once you have access to this object, you can access every CGI variable and store it in a LotusScript variable. For example:

```
CGIValue = doc.HTTP_USER_AGENT(0)
```

CGIValue (string) now has information about the user's browser.

Examples:

- To find out the Web identity, use:

  ```
  Set webUserName=docContext.remote_user(0)
  ```

- To read the arguments passed, that is, the string followed by the ampersand (&) that ends some URLs, use:

  ```
  Set args=docContext.query_String(0)
  ```

Creating a Web Page Counter

To keep track of the number of times a Web page is accessed, you may build a LotusScript agent into a form to monitor hits from Web browsers. To do so, create a Web page document from the form.

This method is recommended for a single Web page created from a form. If you create multiple Web page documents from the same form, their counters are stored together in a single profile document.

Note This tip was provided as a Tip of the Week at the Lotus Developers' Central Web site **(http://www.lotus-developer.com)**.

The Agent

Create an agent named Counter. Select Shared and specify that it should be Run Once and Manually from Agent List. Use the following script:

```
Sub Initialize

Dim Session As New NotesSession

Dim ProfileDoc As NotesDocument

Dim doc As NotesDocument

Dim num As Double

Dim NumStr As String


Set db = Session.CurrentDatabase


'The following line gets a handle to the current document

Set doc = Session.DocumentContext


'The following line creates a Profile document called 'Domino'
'the first time

'it is executed and from then on modifies the existing 'Domino'
'document

Set ProfileDoc = db.GetProfileDocument("Domino")


NumStr = ProfileDoc.num(0)

If NumStr = "" Then

num = 1

Else
```

```
num = Cdbl(NumStr) + 1
End If

ProfileDoc.num = Cstr(num)
Call profiledoc.save(False,False)

doc.Number = num

End Sub
```

The Form

1. Create a number field (named Number) to store the counter value. Add static text to surround the field value, for example:
   This page has been accessed [Number] times.
2. Specify the Counter agent as the WebQueryOpen form event.

The Security

Agents are protected by security features in the database access control list and in the server document in the Domino Directory. To make the Web page counter agent work, do the following:

1. Write down the name of the person who created or last modified the agent. This is the agent owner. The name is the Notes ID name, for example, Mark Hunt/Acme.
2. Add the agent owner's name to the database access control list and give at least Editor access.
3. Give access to the agent owner in the server document.
   - Open the Domino Directory (NAMES.NSF) and open the Server\Servers view.
   - Find the server document for the server that stores the database and open the document in edit mode.
   - Add the owner's name to the Run Restricted LotusScript Agents field in the Agent Manager section.
   - If the database design has been signed using the server administration Sign a Database tool, the signer's identity must also be added to the Run Restricted LotusScript Agents field in the server document.
   - Close and save the document.

Troubleshooting

If you see the following message:

```
Error 401 HTTP Web Server: Lotus Notes Exception - You are not
authorized to perform that operation.
```

there is a problem with the entry in the access control list. Make sure that the person who created or last modified the agent is listed with at least Editor access. In addition, make sure that the agent property For Web Access: Run Agent as Web User is not selected in the Agent InfoBox.

If you see the following message:

```
Error 500 HTTP Web Server: Lotus Notes Exception - Error
validating user's agent execution access.
```

there is a problem with the Run Restricted Agents field. Make sure that the person who created or last modified the agent is listed in that field. In addition, make sure the agent property For Web Access: Run Agent as Web User is not selected in the Agent InfoBox.

Using Agents — Advanced Topics

Agents are very useful if you need to change the design of a database. They can help you to keep the data in the database consistent with the design. For example, you can use an agent to update all documents which are affected by a form change. Usually you will create a private agent which selects the documents affected by the form changes and run it manually.

The following is a list of examples of where agents can be very useful after changes are made to the design of a database:

Editing and Resaving Documents

To save the step of editing and resaving documents manually, create an agent that uses the following formula:

```
@Command([ToolsRefreshAllDocs])
```

Adding a Field

If you create a new field, insert the new field into existing documents by creating an agent that uses the following formula:

```
FIELD New_field_name := value;
```

where New_field_name is the name of the field, and value is the value you want the field in these documents to have. The value can be the field default value, a formula that calculates the value, or a null value (“”) that inserts the field into the documents, but does not give them any initial value.

Removing Field Data from All Documents

If you delete a field, existing documents continue to store the obsolete field and its values. This unnecessary storage can affect disk space. To remove the obsolete field, create an agent that uses the following formula:

```
FIELD Field_name := @DeleteField;
```

After you run the agent, compact the database to reduce its actual file size.

Renaming a Field

If you rename a field, existing documents continue to refer to the old field name. To update documents to refer to the new name, create an agent that uses the following formula:

```
FIELD New_field_name := Old_field_name;

FIELD Old_field_name := @DeleteField;
```

where New_field_name is the new name for the field, and Old_field_name is the original name for the field.

Reassigning Documents to Another Form

If users attempt to open documents created with a form that has since been deleted, they see a message indicating that the form cannot be found. To prevent users from seeing this message, use these agent options to reassign existing documents to another form:

1. Under Which document(s) should it act on, select All Documents in Database and click Add Search.
2. Select By Form Used, select the name of the obsolete form, and click OK.
3. In the design pane, click Formula and enter:

```
FIELD Form := "Reassigned_form_name";
```

where Reassigned_form_name is the name of the form that the documents should use.

Removing the Stored Form from Documents

Selecting the form property Store Form in Documents is useful for mail-enabled applications in which users need to see a document and don't have the original form stored in their mail databases. This form property is permanently attached to all documents created with the form. To remove the stored form, remove all internal fields connected with that form by creating an agent that uses the following formula:

```
SELECT $TITLE="Old_form_name";

FIELD $TITLE:=@DeleteField;

FIELD $INFO:=@DeleteField;
```

```
FIELD $WINDOWTITLE:=@DeleteField;
FIELD $BODY:=@DeleteField;
FIELD $ACTIONS:=@DeleteField;
FIELD FORM:="New_form_name";
```

This formula removes all internal fields attached to the documents where Old_form_name is the name of the form used to create the documents. The last line creates a FORM field where New_form_name is the form that will display the documents in the future.

After you run the agent, compact the database to reduce its actual file size.

Summary

Agents allow you to automate many tasks within Domino. They can operate in the background to perform routine tasks automatically, and in the foreground when called by the user. They can easily be created without programming knowledge by using Simple Actions, but very complex algorithms can also be implemented using LotusScript or Java.

On the Web you can also use agents to perform operations before a document is opened or before it is saved. You are also able to access CGI variables to capture information about the user.

Chapter 8
Securing Your Application

As a database designer you can control who has access to an application you create at every level in the application. Domino provides a variety of security mechanisms to enable you to do this. This chapter will help you understand how these mechanisms fit together to secure your application.

The designer of an application and the administrator of the system should work closely together to define security for an application, because the choices you make have an impact on system performance.

Even though user authentication and creation are normally administrative tasks, we will briefly discuss them here as these tasks may have an impact on defining security for different types of clients and users in your application. You may also have to integrate with, or even develop, a user registration application for Web users. For a detailed description of Domino's system security and authentication features, please refer to the redbook: *Lotus Notes and Domino R5.0 Security Infrastructure Revealed,* IBM form number SG24-5341, Lotus part number CT6TPNA.

This chapter will help you decide how to:

- Set up an access control list (ACL).
- Create roles to manage access for groups of users.
- Restrict access to database elements.
- Control document access.
- Develop a plan that provides the required security for your data and appropriate access for each user.

Note This chapter is in part based on the Lotus Education Learning Byte: Securing Your Application.

Controlling Access to Domino Data

There are a number of ways of approaching access to data in a database; you can *secure certain design elements and information* so that that users cannot access them at all. Alternatively, you can *hide* certain fields and information from the user. In this instance, the information is still accessible to the

experienced user but it is hidden from the casual user. This is more a usability issue than a security feature.

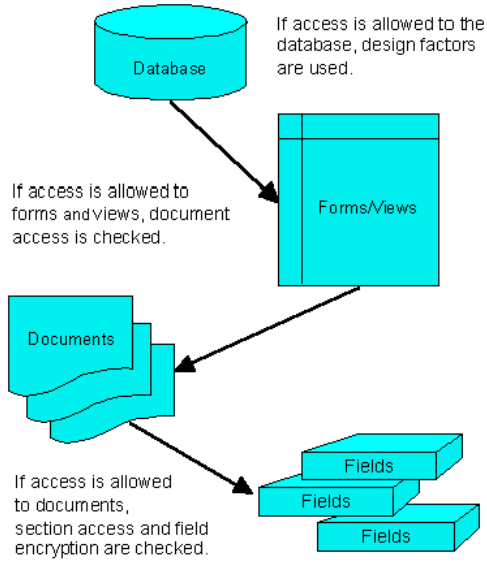
By using both the database access control list (ACL) and the encryption features provided by Domino, you can achieve true security for your application. Creating access lists, hiding design elements, and using such features as computed subforms, hide-when features, and collapsible sections lets you hinder access, are good usability features but they are not true security features.

Overview of Domino Security Architecture

The Domino environment is made up of several components, all of which can be secured. If access is allowed to:

- The network, server tests are applied
- The server, database tests are applied
- The database, design factors are tested
- Design elements, encryption is checked

The following figure illustrates the places in the database structure where access tests are applied. These are the elements you will be concerned with in securing your application at the database level:



Design Elements for Controlling Access

Setting up the ACL establishes who has access to the database as a whole. You can further restrict access to database elements by using the following Domino design elements:

- Access lists for documents, forms, and views
- User roles in the ACL
- Authors and Readers fields in a document
- Hide-when capabilities for fields, actions, sections and outlines
- Controlled access sections

To control user access to Domino data, consider the following situations:

Security Requirement	*Solution*
Allow anonymous users access to your site.	Create an Anonymous entry in the database ACL. See "Anonymous Access to Databases" in this chapter.
Define server authentication at the user level for Web users.	Create Web users and passwords in the Domino Directory. See also "Planning for Web User Access" in this chapter.
Restrict access to database elements to specific users.	Create access lists for documents, forms, and views, and consider creating user roles in the ACL. Start with "Using the Access Control List to Control Access to an Application" in this chapter.
Control Web user access to parts of your site.	• Add a group containing the names of registered Web users to the ACL. • Choose which databases can be accessed by Web users and what level of access to provide for each database. • Authenticate any Web client accessing a Domino server, database, view, or document.
Restrict access to specific documents.	Create Authors and Readers fields in a document, or create a document access list.
Control display of database elements within forms.	Use hide-when capabilities for fields, actions, and sections, or create a controlled-access section.
Secure field information.	Apply encryption techniques.
Display different information for Web users and Notes users.	Use @ClientType to enable a computed subform.
Provide an extra layer of security.	Add encryption to HTTP transactions by activating Secure Sockets Layer (SSL) at the server. (See the Domino Administrator's Help for more information on SSL.)

Using the Access Control List to Control Access to an Application

Every database includes an Access Control List (ACL) which Domino uses to determine the level of access that users and servers have to that database. When a user opens a database, Domino classifies the user into an access level that determines privileges. The access level for a user may vary in different databases.

The access level assigned to a user determines the tasks that the user can perform in the database. The access level assigned to a server determines what information the server can replicate within a particular database.

Only someone with Manager access can create or modify the ACL of a database located on a server.

This section covers:

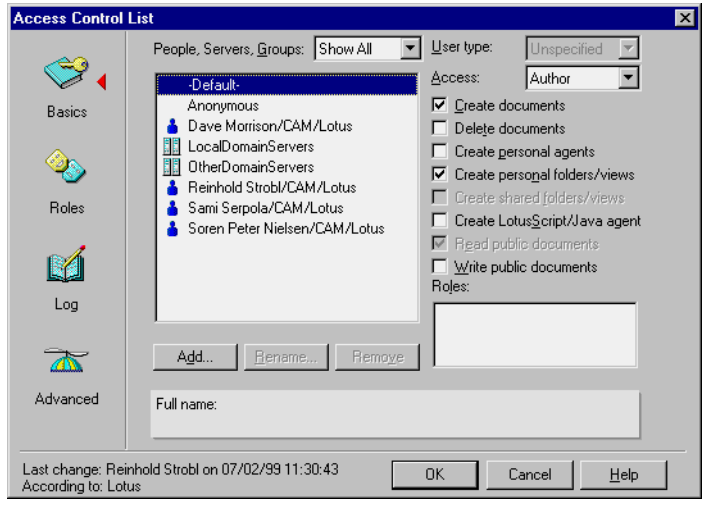
- Displaying the ACL
- User and server access levels

Displaying the ACL

The access control list of a database lists all the servers, groups, and users who have access to the database.

To display the access control list of a database:

Choose File - Database - Access Control and the following panel will be displayed:



User and Server Access Levels

A database ACL determines the level of access that users, groups, and servers have. Someone with Manager access to the database assigns levels to the users, groups, and servers listed in the ACL.

With Domino Release 5.0 there are seven main levels of access that a database administrator can assign to a person, server, or group:

Level	*Users with this access can…*	*Servers with this access can…*
No Access	Not access the database at all.	Not access the replica at all.
Depositor	Create documents, but cannot read, edit, or delete documents, including those they create.	Not receive changes; not relevant for servers.
Reader	Read documents, but cannot create, edit, or delete them.	Pull changes from the replica but not send changes to it.
Author	Create and read documents, but can only edit their own documents if they are listed in an Authors field on that document.	Replicate new documents.
Editor	Create, read, and edit all documents unless there are restrictions on specific documents.	Replicate all new and changed documents.
Designer	Have Editor access to documents, except where restrictions exist for specific documents, and they can modify the database design, but they cannot delete the database or modify the ACL.	Replicate design changes as well as all new and changed documents, but not ACL changes.
Manager	Perform all operations on the database, including modifying ACLs and deleting the database.	Replicate all changes to the database and the ACL.

Server access levels are often the cause of databases failing to replicate as expected. Keep the following points in mind:

- Servers not specified in the ACL have the access level that is assigned to the -Default- group.
- Listing a server with Manager access in the ACL lets people know which server has Manager access.

To allow a replica to receive changes made by people with Author access, assign the server Editor access or higher in the replica ACL.

Setting Up and Refining the ACL

When you set up the access control list, you can refine the access for users in several ways, beyond simply specifying an access level:

- **Select User Type to specify Users, Groups, and Servers**

  When you enter users in the ACL, you can specify whether they are users, groups, or servers.

- **Access Options**

  Assigning access options allows you to further refine user access.

- **User Roles**

  Roles allow you to define responsibilities in the application and refine access rights to database elements.

Users, Groups and Servers

A group is a list of users and/or servers which have something in common. Using a group helps simplify many administration tasks. For example:

- A group of users can be given access to a database in the ACL.
- A group of servers can be designated as permitted to replicate with a database.
- A group of users can be denied access to a resource.

Note Groups you specify in the ACL must be listed in the Domino Directory.

There are two default server groups in the ACL:

- LocalDomainServers are servers in the local domain.
- OtherDomainServers are servers in other domains. These are usually servers in other companies with whom users in your company need to communicate.

User Types

The ability to specify user types lets you clearly indicate whether a name is that of a person, server, or group. See the table below for descriptions of the available user types:

User type	*Assign for this type of user*	*Allows you to . . .*
Person	An individual user; this includes a user on a server workstation.	Control access for an individual user.
Server	A single server; this includes a server console, and server workstation.	Prevent someone from accessing the database from a Notes workstation using the server ID.
Server Group	A group of servers.	Identify a group of servers that will host replicas of the database.
Person Group	A group of individual users.	Grant the same access to all users in a group without listing each user name in the access control list.
Mixed Group	A group of servers and individual users.	Grant the same access to a group of users and servers.
Unspecified	In the Advanced Access Control List window, click Lookup User Types for "Unspecified Users." Notes looks up an unspecified user type in the Address Book.	If you leave type as Unspecified Domino will not check whether the access is given to a user or a server.

Assigning User Types for Additional Security

Assigning user types can provide additional security. Specifying names in the ACL as a person, server, or server group prevents someone from either:

- Creating a group in the Domino Directory with the same name and adding his or her name to it to access the database through the group name.
- Accessing the database from a Notes workstation using the server ID.

Note Designating a name as a server or server group is not a foolproof security method. It is possible to create a Domino add-in program that gains access to the database from a workstation through the server ID, since the add-in program behaves like a server.

Access Options

When you add users and groups you can specify individual options that further refine user access. For each ACL entry, you can specify slightly different options:

Enable this option…	*To allow…*	*This option is assigned by default to…*
Create documents	Authors to create documents.	Managers, Designers, Editors, and Depositors
Delete documents	Managers, Designers, Editors, and Authors to delete documents. Authors can delete only documents they created.	No one
Create personal agents	Designers, Editors, Authors, or Readers to create personal agents.	Managers
Create private folders/views	Editors, Authors, and Readers to create personal folders and views in a database on a server.	Managers and Designers
Create shared folders/views	Editors to create shared folders and views.	Managers and Designers
Create LotusScript/ Java agents	Readers, Authors, Editors, and Designers to create LotusScript and Java agents.	Managers
Read public documents*	Users to read documents created with forms, and use views and folders, designated as "available for public access user."	Readers and above
Write public documents*	Users to create and modify documents with forms designated as "available for public access user."	Authors and above

* Enabling users to read and write public documents lets you give users with No Access or Depositor access the ability to access specific forms, views, and documents without giving them Reader or Author access in the database. Public documents are useful for calendar applications in which one user might delegate the ability to read or create appointments on his or her behalf to another user.

You can specify the availability of a database element for public access at the bottom of the Security tab in its InfoBox.

Documents created with a form where public access is enabled will have the field $PublicAccess with a value of "1" added by Domino.

Anonymous Access to Databases

You can handle anonymous users in one of the following two ways:

- Define an anonymous entry in the ACL and specifically define access privileges for anonymous users.
- Allow anonymous users the same access as the Default entry in the ACL.

Note Any application that will be deployed on the Web should have an Anonymous entry in the ACL.

If you allow anonymous access to a server, you can still control access to databases. To control database access for anonymous users, follow these steps:

1. Add a user with the name Anonymous in the Add User dialog box of the ACL.
2. Click OK.
3. In the Access drop-down box, select either:
   - No Access to prevent access by anonymous users.
   - Reader to allow access to an information database.
   - Author to allow access to an interactive database.

Caution If the database ACL does not contain an Anonymous entry, all anonymous users receive the Default access.

To protect the databases from unregistered users you can establish the Default as No Access. If Default access needs to be higher, create an Anonymous entry in the database ACL and grant it No Access.

When granting access to unauthenticated Web clients, you will want to grant anonymous users the least access that still allows them to use the database effectively. For example, you might grant anonymous users:

- Reader access for an information database
- Author access for an interactive database

Differentiating Default and Anonymous Access

If Anonymous is not listed in the ACL, Domino grants the user access based on the default database access level. This may be a higher access level than you want for anonymous users.

Access Level definitions:

- Default: a user not specified in the ACL.
- Anonymous: a user without a valid Notes ID for that organization.

Roles in the ACL

When a group you want to add to the ACL does not exist in the Domino Directory, you may want to create a special group or role for users of the database. Roles let you define responsibilities in the application and further define access to database elements.

What is a Role?

A role is a subset of the ACL that is controlled by the database manager. A role can be used anywhere that a group or user name can be used. Users and groups are assigned roles to refine access to particular views, forms, sections, or fields of a database. Instead of assigning access to a design element to users and groups, you assign access to the role.

Some advantages of using roles are that they:

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

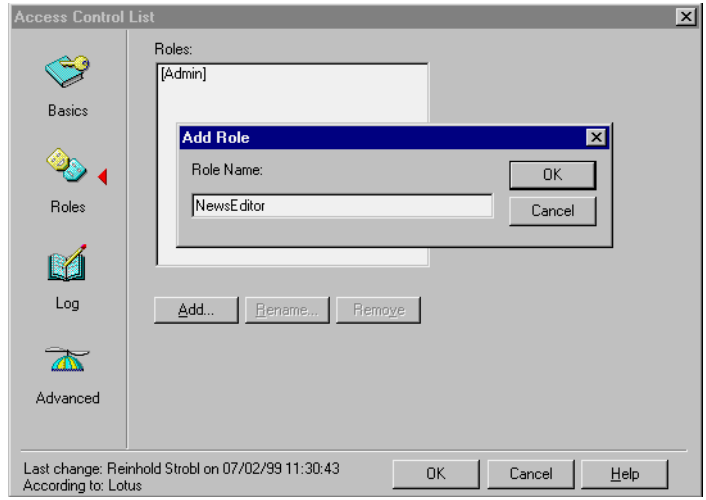
To use a role in an application, assign roles to users and groups in the ACL. Include the role in access lists, just as you do with users and groups (or actually *instead* of adding specific users and groups).

Adding Roles to the ACL

To add roles to an ACL, follow these steps:

1. Open the database ACL.
2. Click Roles in the Contents pane.

3. Click Add. The Add Role dialog box appears:



4. Enter a role name no longer than 15 characters and click OK. The role name appears in brackets in the Role list.

Assigning Roles to Users

To assign a role to a user:

1. Open the database ACL.
2. Select the user name in the list of people, servers, and groups.
3. Click one or more role names in the Roles list.
4. Confirm roles by highlighting a user. A checkmark appears next to the user role or roles.

Enforce Consistent ACL

You can ensure that the ACL of a database remains the same on all replicas. You do this by selecting the advanced access control list option "Enforce a consistent Access Control List across all replicas of this database." Selecting this option ensures not only that the ACL remains consistent across server replicas, but also that the ACL is enforced on replicas of the database made on workstations or laptops; if you do not select this option, users have Manager access to local replicas of server databases, which allows them to make changes their access levels on the server replica, although they can't replicate such changes back to the server.

Enforcing a consistent access control list as it applies to ACLs on workstation or laptop replicas is not a security feature. Data in the local replica is not secure unless you physically secure the workstation or laptop or you encrypt the database using the local security feature. Also, a Domino add-in program can bypass an ACL enforced on local workstations.

To keep the ACL the same across all server replicas of a database, you must select this setting on a replica whose server has Manager access to the other replicas; otherwise replication will fail because the server has inadequate access to replicate the ACL.

Maximum Internet Name and Password Access

When working with advanced ACL options, you can also specify a maximum access level for users that have been authenticated with the internet name and password setting (browser users). This setting overrides individual settings in the ACL. No browser user can get higher access than specified for Maximum Internet Name and Password Access.

Check this setting if you are experiencing problems with Web users not getting the access they have been granted in the ACL.

Changing the ACL Programmatically

You can change the ACL programmatically using these Domino classes.

The classes in the Domino Object Model to use when working with the ACL are:

- NotesACL
- NotesACLEntry

Refer to the descriptions of the classes in the Domino Designer documentation for detailed descriptions of the possibilities.

Example
This example illustrates how to obtain all entries in the ACL that are associated with a given role:

```
'Declare Variable session as a new Notes session

Dim session As New NotesSession

'Declare db as a Notes Database

Dim db As NotesDatabase

'Declare acl as the Notes Database ACL

Dim acl As NotesACL

'Declare aclentry as ACL Entry type
```

```
Dim aclentry As NotesACLEntry
'Declare RoleName as type String
Dim RoleName As String
'Set db to the currently selected database
Set db = session.CurrentDatabase
'Set acl to the ACL of the current database
Set acl = db.ACL
'.....
'Get Rolename from somewhere
RoleName = "[NewsEditor]"
'.....
'.....Here You would check that role exists in ACL using
'.....'Forall RNames In acl.Roles'
'.....
'Set aclentry to the first name in the ACL
Set aclentry = acl.GetFirstEntry
'Continue looping until you run out of names in the ACL
While Not ( aclentry Is Nothing )
   If ( aclentry.IsRoleEnabled( RoleName ) = True ) Then
   'If that ACL name is in the selected role
   'Display that name to the user
      Messagebox aclentry.Name
    End If
   'Move to next name in list
   Set aclentry = acl.GetNextEntry( aclentry )
Wend
```

Using Outline Control to Hide Parts of an Application

You can control which parts of an application are visible to the user depending on whether they are a Notes user, a Web user or on the role the user has, by using outlines.

For each outline entry, you can use the InfoBox to specify hide from:

- Notes R4.6 or later
- Web browsers
- Depending on a formula (that, for example, checks on assigned roles)

Note Using this method only controls which navigational aid the user is offered. The underlying objects must also be secured if the user should not be allowed access to them. For example, if you have a view that only Notes users should see, then you can hide it from Web users in the outline, but you must also limit Read access for the view; otherwise Web users can access the view by specifying its exact URL.

Using Directory Link Files to Control Access to an Application

The System Administrator can control access to all databases in a given directory by creating a directory link file. A directory link file must be named in the format xxxxxxxx.dir where xxxxxxxx is the name that will appear as a directory in the user's Open Database dialog.

The file is a flat text file where the first line holds the path to the actual directory holding the databases and the following lines hold the names of the people and groups that are allowed to access that directory.

Example
A directory link file called projecta.dir has the following content:

```
d:\projects\projecta

ProjectAMembers

#Admin

CN=Soren Peter Nielsen/OU=CAM/O=Lotus
```

This means that the databases the Notes user can see in the *projecta* directory are physically stored on the server in the *d:\projects\projecta* directory. Access to this directory through Domino is allowed only to people in one of the groups *ProjectAMembers* and #*Admin* and the person *Soren Peter Nielsen/CAM/Lotus*.

Note The System Administrator can control whether Web browsers are allowed to access databases using directory links through the NOTES.INI variable *DominoNoDirLinks*. A value of zero (0) will allow Web browsers to access directory links while a value of one (1) prevents it.

Tip The group name for administrators #*Admin* starts with the character #. This is to make it one of the last entries in a sorted list. When a user brings up an address dialog from the Domino Directory the list will not be cluttered with system groups if they are named so they appear last in the list.

Controlling Access to Views and Forms

Domino provides a number of mechanisms that can refine the ACL in granting or denying access to individual design elements or data for specific users. In this section we will discuss:

- Access to views using view read access lists
- Access to forms using form create access lists

Controlling Access to Views

To control which views each user has access to when he or she opens the database, create a view read access list. The list can contain any users, groups, servers, and roles that are in the ACL for the database.

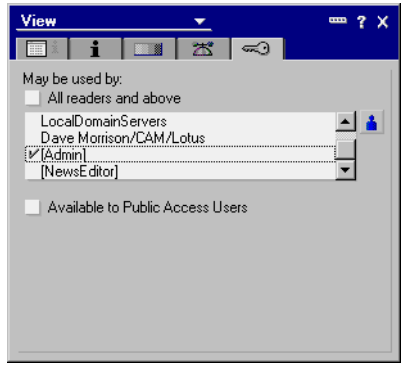
Caution By default, when a user opens a database for the first time, the default view is displayed. Therefore, never restrict access to the default view for the database. Users will not be able to open the database if they are restricted from the default view.

Creating a View Access List

To create a view read access list:

1. Open the view in Design mode.
2. Select Design - View Properties to open the InfoBox for the view.
3. Click the Security tab (key icon).

4. Deselect All readers and above (the default). The list in the window displays the contents of the ACL:



5. Click one or more of the users, groups, servers, and roles that you want to have access to the view. A checkmark appears next to the names you select.
6. Click the blue person button to add names, roles and groups to the list from the Address books that you have access to and make sure that they are added to the ACL.

To deny access to the view, deselect by clicking the name to remove the checkmark.

Controlling Access to Forms

You can control access to a form in several ways:

1. Exclude the form from the Create menu and make it available to a select set of users with a View action button.
2. Create a form access list that specifies who can create documents with the form.
3. Create a form for Public Access users with Read or Create rights in the ACL.

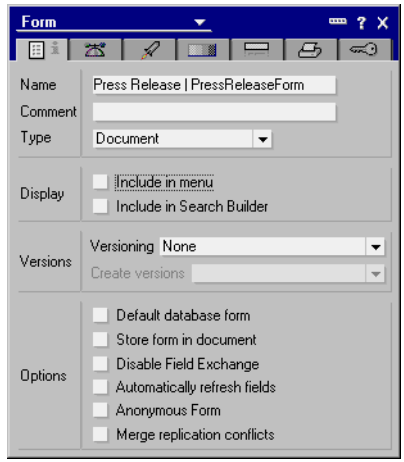
Making a Form Available to a Select List of Users

This method has two parts:

- Exclude the form from the Create menu.
- Create a View action button that is available to a select set of users.

To prevent a form from appearing on the Create menu:

1. Open the form in Design mode.
2. Select Design - Form Properties to open the InfoBox for the form.
3. On the Information tab, deselect Include in: Menu option.



To create the action button:

1. Open a view that displays the form in Design mode.
2. Create a view action using the formula @Command([Compose];"formname").
3. Open the Action InfoBox and click the Hide tab.
4. Enter a formula to hide the view from everyone except the users and groups you specify.

Using a Form Access List

Form access lists override the ACL and allow only those on the list access to the form or documents created with the form:

A form Create access list allows only those on the list to create documents using the form.

A form read access list allows only those on the list to read documents created with the form.

To create a form Create access list:

1. Open the form in Design mode.
2. Select Design - Form Properties to open the InfoBox for the form.
3. Click the Security tab (key icon).
4. In the Who can create documents with this form section, deselect All Authors and Above (the default).

   The list in the window displays the contents of the ACL.
5. Click one or more of the users, groups, servers, and roles that you want to have the ability to create documents with the form. A checkmark appears next to the names you select.
6. Click the blue person button to add names, roles, and groups to the list from the Address books that you have access to and check to see that they are added to the ACL before you make the database available to users.

To deny access to the form, deselect by clicking the name to remove the checkmark.

Note A user without access to a restricted form may still be able to create documents that look like they have been created with that form by using an agent or by pasting a document from another database.

Creating a Form for Public Access Users

A public access list works with the database ACL to expand user access to specific views, forms, and documents. Creating forms and views enabled for public access allows you to provide users with No Access or Depositor access the ability to view specific documents, forms, and folders without giving them Reader access to the entire database. Users who have this access level in the database ACL will see only documents, folders, and views specified as available for public access in the form/folder/view InfoBox.

Public documents are useful for calendar applications where one user might delegate the ability to read or create appointments on his or her behalf to another user.

To create a form for public access:

1. Choose Design - Form Properties.
2. Click the Security tab.
3. Select Available to Public Access users.
4. Create a field and open its InfoBox.

5. In the Name field, enter $PublicAccess.
6. In the Type field, select Text and Computed when Composed.
7. In the Design pane, enter "1" as the default value for the field and click the green button to accept the value.
8. To hide this field from users, select the Hide tab and specify hide-when conditions in the Field InfoBox.
9. Save the form.

To sum it up: You must both specify that the form is for public access using the InfoBox AND add a hidden computed field called $PublicAccess with a value of "1".

Note The views the Public users will use must also be marked for Public Access. This is done using the View InfoBox.

Preventing Printing, Forwarding, and Copying of Documents

You can discourage users from printing, forwarding, or copying documents created with a form. This feature helps to prevent accidental distribution of confidential information, but it is not a true security feature since the settings can be manipulated with the appropriate Design and Document access rights or circumvented by using a screen capture program.

1. Open the form in Design mode.
2. Choose Design - Form Properties.
3. Click the Security tab (the key icon).
4. Select Disable printing/forwarding/copying to clipboard.
5. Close and save the form.

Controlling Access to Documents

Individual documents can contain sensitive information. Domino security provides several mechanisms that can restrict access to a document. You can control both Read and Editor access to documents:

- Restrict Read access to documents:
  - Create a read access list for all documents created with a form.
  - Use a Readers field.
- Restrict Editor access to documents:
  - To those named in the Authors field.
  - Use a Controlled Access section.

Read Access

If you want to control read access at the document level you can you can do it for all documents created with a specific form or you can do it for each document. In the following sections we will describe:

- Controlling Read access to documents based on the form used to create them
- Controlling Read access to individual documents using a Readers field.

Read Access List for a Form

A read access list for a form refines the ACL by allowing only those named in the list to read documents created with the form.

The $Readers Field

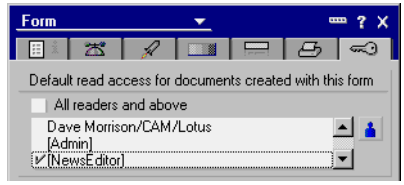
When you create a read access list for a form, Domino adds the internal field $Readers to the form. The value of the field is the form read access list. Each document that users create with the form contains the $Readers field list of readers. If there is no read access list for the form, the documents do not have a $Readers field.

Note The Author or an Editor of a document can change the read access list of a document by going to the document InfoBox and changing the selection in the read access list of the Security tab.

Creating a Form Read Access List

To create a read access list for a form:

1. Open the form in Design mode.
2. Select Design - Form Properties to open the InfoBox for the form.
3. Click the Security tab (key icon).
4. Disable the default option (All readers and above). The list in the window displays the contents of the ACL:



5. Select specific users, groups, servers, and roles to which you want to give Read access for documents created with the form. A checkmark appears next to the names you select.
6. Click the blue person button to add names, roles, and groups to the list from the Address books that you have access to and check that they are added to the ACL before you make the database available to users.

7. To deny Read access for documents created with the form, click a name to remove the checkmark.

Note Creating a form read access list lets you assign read access to a specified set of users, groups, and roles for all documents created with that form. However, if you need to be able to assign readers rights dynamically, you must use a Readers field as follows.

Readers Field

A Readers field is a field data type that restricts readership for the document to those users and servers that are listed in the field. There are two ways to create a Readers field in a document:

- The Designer places a field with the Readers Data Type on a form.
- The Author or an Editor of the document opens the document properties and sets the Read access in the security tab. This automatically creates a $Readers field in the document.

Readers fields have the following characteristics:

- Readers fields are an excellent means of restricting replication, as only the documents for which a user is listed in the readers field will be replicated.
- If a document contains multiple Readers fields, all entries from all Readers fields have read access to the document.
- Readers fields restrict reader access to individual documents only; access to each document depends on the contents of its Readers field.
- Editable Readers fields allow authors and editors to enter names of authorized readers.

Caution If you allow users to enter names of authorized readers, you should also have a separate, hidden, computed Readers field that contains the names of any servers that should replicate the document. Without the server names in a Readers field, the document will not be replicated. It is recommended to add a role for replicating servers to a Readers field and then assign the relevant servers to that role in the ACL.

Creating a Readers Field

To create a Readers field:

1. Add a field to a form.
2. Select Readers as the field data type.

3. Specify readers by using one of the following methods:
   - Enter user names, roles, or groups directly.
   - Use a formula to compute user names, roles, or groups.
   - Make the field editable so Authors and Editors can select and change readers.

Note Before Domino R5.0, even though the reader was not allowed to see a certain document in a categorized view, the category for that document could still be seen, so you had to take care not to display sensitive information in the categories. Using Domino R5.0 the reader will only see the categories in a view for documents that he or she is allowed to read. This allows you to include sensitive information (for example, social security numbers) in the document category.

Editor Access

Authors Field

An Authors field is a Notes reserved field that lets you give users Editor access to their documents when they have Author access to the database.

An Authors field:

- Works only in a database located on a server, or on a local replica when the Enforce a Consistent ACL option has been selected.
- Refines the ACL but does not change it.

To allow users with Author access to edit documents they create, you must include them in the documents' Authors field.

Users with Editor access can edit a document even if they are not in the Authors field. (Use Readers fields to prevent users with Editor access from reading the document, since if Editors cannot read the document, they cannot edit it.)

Users with No Access, Depositor access or Reader access cannot edit the document even if they are listed in the Authors field.

If you add only one Authors field to a document and it contains a null value, then only an Editor or above can edit the document.

Creating an Authors Field

To create an Authors field:

1. Add a field to the form.
2. Select Authors as the field data type.
3. Specify the authors using one of the following methods:
   - Entering user names, roles, or groups.
   - Using a formula to compute user names, roles, or groups.
   - Making the field editable so that users with author or editor authority can select and change authors themselves.

Combining Readers and Authors Fields

Use this table as a quick reference to determine how Readers and Authors fields can protect your document:

Assume that Jane Brown and Roberto Sanchez both have Author access in the ACL and that there is no form read access list.

Readers field	*Authors field*	*Who can read*	*Who can edit*
None	None	Everyone with ACL Reader access	Everyone with ACL Editor access and above
None	Roberto Sanchez	Everyone with ACL Reader access or above and Roberto Sanchez	Roberto Sanchez and everyone with ACL Editor access and above
Jane Brown	None	Jane Brown	No one
Jane Brown	Roberto Sanchez	Jane Brown and Roberto Sanchez	Roberto Sanchez

Note Do not hardcode group or people names in Readers and Authors fields as this will make maintenance harder. Use roles instead as this will allow the database Manager to assign the desired access to different groups and people by assigning the right role to them in the ACL.

Caution When programming with the Domino classes you cannot use the extended class method to assign values for Readers and Authors fields as you can when working in the formula language. Suppose that you have a

Readers field in your document called DocumentReaders and that you want to assign the value "Cecilie Vibe/Asgård" to this document. Using the formula language you can do as follows:

```
FIELD DocumentReaders := "Cecilie Vibe/Asgård"
```

You might also expect to be able to do it in a similar way using the Domino classes by treating the item name "DocumentReaders" as a property of a NotesDocument object (assigned to the variable 'doc' in this example):

```
...
doc.DocumentReaders = "Cecilie Vibe/Asgård"
...
```

This will NOT work because it changes the type of the DocumentReaders field to Text thereby removing the read protection from the document. When working with Readers and Authors fields in the Domino classes you must use the NotesItem class as follows:

```
...
Dim newValues( 1 To 2 ) As String
newValues( 1 ) = "Cecilie Vibe/Asgård"
newValues( 2 ) = "[ReplicatingServers]"
Dim authorsItem As New NotesItem(doc, "DocumentReaders",  _
newValues, READERS)
...
```

Note that the role [ReplicatingServers] was also added to the Readers field in the last example. Documents protected by Readers fields should always include read access to a role that can be assigned in the ACL to the servers that need to replicate the database.

Field Editor Access Security Option

In a database where some users have Editor access while others only have Author access you can combine the use of Authors fields with the individual field security option Must have at least Editor access to use. Thus, even though you are giving users with Author rights in the ACL access to a document, by using an Authors field you can hinder them in editing certain fields by using the Must have at least Editor access to use field security option.

Controlled Access Sections

In addition to having the same properties as the standard access, controlled access sections also have a separate list of allowable editors. To users with Editor access, fields behave as normal editable fields. To users who are not listed as Editors of the section, the fields are read-only.

In addition to limiting Editor access to the fields within a section, sections also provide a means of attaching multiple signatures to a document (up to one signature per section). This is a useful feature for workflow type applications.

Note Sections are not to be used as a security feature. An access-controlled section does not physically protect data because a user can modify the section through a different form. To make a section truly secure, encrypted fields must be used within the section. Sections should be used when documents require multiple signatures validating the Composer or Editor of the data.

Use of Hide-When Formulas

You can control whether an action button, a paragraph or a cell in a table is to be shown to a user by a hide-when formula. Select the object that you want to hide and display its InfoBox. Click the Hide When tab and select Hide Object if Formula is True. Then enter a formula for when the selected object should be hidden.

See the Using @UserRoles section later in this chapter for examples of hide-when formulas.

Note Hide-when formulas are not a true security feature on the Notes client as all hidden fields in a document can be seen through the document InfoBox. It is also not secure for a Web client if Generate HTML for all fields is selected in the Form Properties box.

Using Encryption for Field Security

Encryption allows you to secure information at the field level. You can encrypt the contents of any field so that only readers who have the encryption key can access the message or field.

Note The database Managers can encrypt an entire database.

Users who need to:

- Create and save documents with a form must have at least one of the encryption keys you selected in the default encryption keys list.
- Read the encrypted fields must also have at least one of the encryption keys used to encrypt the fields.

Caution Encryption does not work with Web browser users because the encryption key is held in the Notes ID. Do not rely on encrypted fields to provide security if Web users are authorized to read documents.

How Encryption Works for Fields

Encryption prevents unauthorized access to critical data in selected fields and is enabled using encryption keys. The System Administrator distributes

the encryption keys to authorized users when deploying the application by mailing the key or giving it to users in a file. When users receive an encryption key, they must merge it with their user ID files.

Encryption Methods

You need to choose an encryption method and design for it. There are three ways you can apply encryption:

- **Automatically:** You can design a form to automatically encrypt all encryptable fields whenever someone saves a document composed with that form.
- **Manually:** Authors and Editors can encrypt the document by selecting an encryption key in the document InfoBox.
- **Manually or automatically:** You can create a field that generates a list of encryption keys from which the Author or Editor can choose a key, or you create a field that contains a formula that chooses the key.

Creating an Encryption Key

A document can be encrypted only if it contains at least one field designated as encryptable.

To encrypt a document:

1. Create an encryption key.
2. Enable encryption for a field.

To create an encryption key:

1. Choose File - Tools - User ID.
2. Click the Encryption icon.
3. Click New.
4. Enter a name that describes the purpose of the key.
5. (Optional) Enter a comment. Include the names of the database, forms, and fields that use the encryption key in case you need the information later.
6. Click North American if users are going to use the key only in Canada or the U.S. Click International if users are going to use the key in other countries.
7. Click OK.

Protect the encryption key by specifying a password for the key when you export it. In this way, only those who know the password can import the key into their user IDs. Additionally, you can specify that a user who receives the encryption key cannot give it to another user.

Enabling Encryption for a Field

You can enable encryption for a field manually or automatically. To allow Editors and Authors to specify keys to encrypt their documents, you need to manually enable encryption on the field.

To manually enable encryption on a field:

1. Create a field in a form.
2. Open the fieldInfoBox.
3. Click the Options tab.
4. Select Security Options: Enable Encryption for this field.

The Editor or Author must then specify which encryption key to use on the Key tab on the InfoBox for the document.

To enable automatic field encryption:

1. In the form InfoBox, click the Key tab.
2. From the Default Encryption Keys list, select one or more encryption keys in your ID. If you select more than one encryption key, all the encryptable fields will be encrypted with all the keys.

Authentication on the Web

HTTP Basic Authentication

The communications protocol used by the World Wide Web is the Hypertext Transfer Protocol (HTTP). HTTP includes a simple user ID and password-based authentication scheme known as basic authentication. The implementation of basic authentication is server-specific, but in general they all use it for two purposes:

- As a mechanism to identify which user is accessing the server
- To limit users to accessing specific pages (identified as Uniform Resource Locators, URLs).

How Basic Authentication Works

Basic authentication uses a challenge mechanism to prompt users to authenticate themselves. The user ID and password block is constructed by creating a string of the form: userID:password and then encoding it using the base64 algorithm.

You may wonder, why you are not repeatedly prompted for a password every time you access a new restricted page. The reason is that the browser caches the user ID, password, server name, and realm name in memory. In

fact, most browsers go one stage further than this and send a user ID and password for any URL that is likely to need it.

Is Basic Authentication Secure?

There are two obvious loopholes in HTTP basic authentication:

The user ID and password are included in the packet header, which means that they can be captured by anyone with a network sniffer or trace tool at any place in the session path.

The user ID and password are cached in the browser, so if you leave the machine unattended anyone can use your ID to access restricted information.

The second loophole is no different from any other situation where a machine is left unattended. The solution is one of user education: always lock the screen when you leave your desk. Note that the caching is in memory, so the user information is lost once the Web browser has been shut down.

The first loophole is more significant. The user ID and password are not encrypted when they are placed in the packet header, but instead are encoded with base64. Base64 is an algorithm that forms part of the Multipurpose Internet Mail Extensions (MIME) protocol. It is a mechanism that turns any bit stream into printable ASCII characters. (It is described in RFC1521.) In fact, the objective of base64 is not for masking data at all, but to provide a method to send binary data through a mail gateway that can only handle character data.

The result of this is that by capturing the Authorization: Basic header from an HTTP request, an attacker can easily extract the user ID and password.

How serious is this exposure? Within a corporate network it may not be a big problem. In fact, base64 offers protection of user IDs and passwords that is superior to many older protocols that send them as clear text. On the Internet it is a different story. Here you have to assume that someone, somewhere is tracing everything you send. Clearly HTTP basic authentication should not be used as the sole method of protection for any critical resource.

You can make basic authentication secure by providing an encrypted connection for it to operate in. SSL is a good example of a protocol that encapsulates HTTP data in this way.

Secure Sockets Layer (SSL)

The SSL protocol was originally created by Netscape Inc., but now it is implemented in World Wide Web browsers and servers from many vendors. SSL makes use of a number of cryptographic techniques, such as public key

and symmetric key encryption, digital signatures and public key certificates. SSL has two main objectives:

1. To ensure confidentiality, by encrypting the data that a client and server send.
2. To provide authentication of the session partners, using RSA public key methods. Most current implementations only require the server to be authenticated in this way, although the protocol does allow for client authentication.

There are two parts to SSL: The handshake, in which the session partners introduce themselves and negotiate session characteristics, and the record protocol, in which the session data is exchanged in an encrypted form.

SSL and Certifying Authorities

Authentication in SSL depends on the client being able to trust the server's public key certificate. A certificate links the description of the owner of a key pair to the public part of the key. The validity of a certificate is guaranteed by the fact that it is signed by some trusted third party, the certifying authority (CA). But how does a certifying authority become trusted? In the case of an SSL-capable browser, the certificates of trusted authorities are kept in a key database, sometimes called a key ring file. The list of top-level authorities, for example VeriSign, is pre-installed when you get the browser.

This approach has the benefit of being very simple to set up; a browser can authenticate any server that obtains a public key certificate from one of the CAs in the list, without any configuration or communication with the CA required.

SSL Client Authentication

Both client and server use digital signatures to identify themselves and those signatures use public keys that are validated by the existence of a shared hierarchy of certificate authorities. You may think that SSL could apply a similar approach to allow Web browsers to authenticate themselves, simply by having the client implement a mirror image of the server authentication process. In fact, this is exactly what the SSL protocol specifies, by adding to the handshake a server challenge that the client must encrypt using its private key.

Unfortunately life is not that simple. Notes can support a symmetrical authentication scheme because you have control over the CA hierarchy and therefore you can ensure a region of shared trust. On the Web you do not have that luxury. The monolithic certification scheme used by servers is not flexible enough to support the large number of certificates that client authentication would require.

Domino and SSL

Domino Release 4.5 added support for Domino to obtain a certificate from an external certificate authority to provide for Domino server authentication and session encryption (SSL V2).

The Domino 4.6 server added support for Internet client authentication (SSL V3), using client certificates obtained from an external certificate authority. The client certificate had to be registered in the user's person record in the Domino Name and Address book. Domino 4.6 also added support for Domino to be a certificate authority to issue X.509 certificates for servers (although the Domino CA could not yet issue client certificates).

The X509 certificate is a standard certificate format for the Internet. Certificates verify the user's identity and bind the public key to the user's name. X.509 certificates are used on the Internet/intranet for authentication and encrypted sessions (SSL), and encrypted mail and digital signatures (S/MIME).

Domino Release 5.0 adds SSL 3 Client Authentication and the ability to create X509 certificates that can be issued to Notes clients.

Previous Notes clients used their own internal certificates (similar concept to X.509, but different format) and did not support X.509 certificates. The Notes R 5.0 client will have the ability to request a certificate from any certificate authority, including a Domino certificate authority, and store the X.509 certificate in the Notes ID file. To obtain an X.509 certificate from a Domino CA, Notes users will use the Domino CA Web site (CA application) just as a browser user does today. The X.509 certificate can be used for encryption and digital signatures between Notes clients and Internet clients using S/MIME and for access to secure web sites using SSL.

When to Use Internet Security

There are three methods of Internet security that can be used, Basic Authentication, SSL Server Authentication, and SSL Client Authentication with X509 certificates. The table below describes when, typically, to use each of these levels of security in your applications:

Security Feature	*When to use it*
Basic Authentication	In a closed network application such as a company intranet where the level of risk from outside attack is low.
SSL Server Authentication	In an Internet application where the risk of outside attack is greater and you wish to protect the information and data on your Web server to a greater extent.

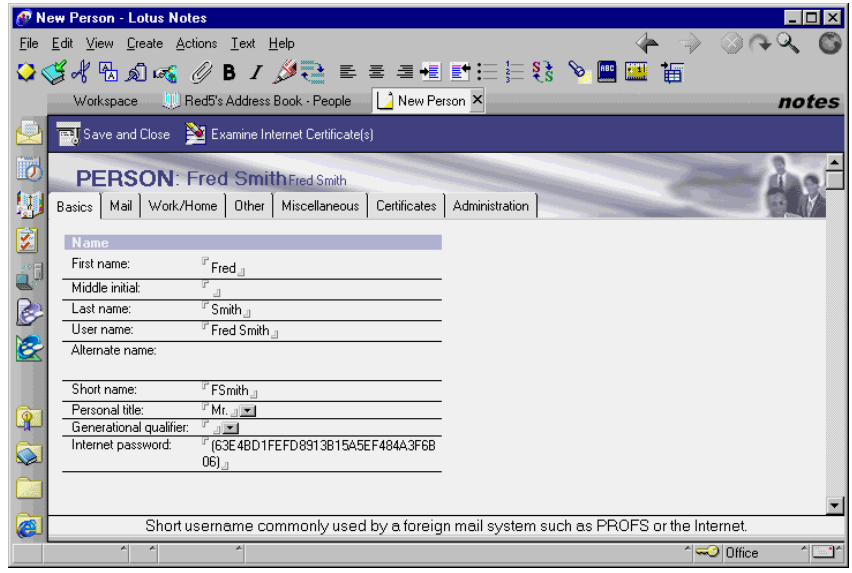
continued

Security Feature	*When to use it*
SSL Client Authentication with X509 certificates	When security is at a premium and you are exposed to potential attacks from the Internet.

Defining Web Users

You may register new Web users from scratch, or let existing Notes users have access to the Domino Web server. To create a new Web user follow these steps:

1. Open the Domino Directory and select the Person view.
2. Select the Add Person action button to display the Person document.
3. Enter the required information. The User Name and Internet Password fields are the only fields used for Domino Web authentication. The other fields are optional with the exception of the Last Name field, a required field without which the form cannot be saved.
4. The following figure shows a Person document being created:



For more information on Internet Security, see the IBM redbook: *Lotus Notes and Domino R5.0 Security Infrastructure Revealed,* IBM form number SG24-5341, Lotus part number CT6TPNA.

Programming Considerations

Domino provides several @functions that help you control application behavior based on the user, or on the client type. We will discuss the following functions:

- @UserRoles: returns a list of roles for the current user.
- @UserName: returns the user name or server name.
- @ClientType: returns a text string to differentiate Notes and Web clients.
- @UserNamesList: returns a text list containing the current user name, any group names, and any roles.

Using @UserRoles

Use @UserRoles in formulas to either:

- Determine what to do for a particular set of users, without needing to use the user names in the code.
- Direct one set of users to one page, and another set of users to another page when the user clicks a button.

The @UserRoles function has no arguments:

```
@UserRoles
```

and returns a text list whose value is the role or roles of the current user. You can add code to perform an action based on the returned value.

Note @UserRoles only works on a server-based database unless the Enforce a Consistent ACL option is selected in the Advanced section of the ACL settings.

Examples

To display an action only to people in the NewsEditor role, enter the following hide-when formula for the action:

```
!@IsMember( "[NewsEditor]"; @UserRoles )
```

To hide a database element from Web users who have not registered in the Domino Directory, create a Registered role in the ACL and enter the following formula on the element's Hide When tab:

```
!@IsMember("[Registered]" ; @UserRoles)
```

Using @UserName

@UserName returns the current user name. Using @UserName allows you to make the current user name available to formulas. You can use it to:

- Restrict the Edit action in documents created with a particular form based on whether @UserName is equal to the author of the document.
- Hide portions of documents in hide-when formulas based on the user name.

Example
The following view selection formula selects only documents created by the current user to display in a private view:

```
SELECT @UserName=Author
```

@UserName has been extended to take a new parameter, index, which returns the user name, alternate user name, or server name indicated by the parameter.

Alternate user name is new in Domino R5.0. It allows the administrator to specify two names when creating a user: their real name and their simplified name, so that people in other countries, with other keyboard setups, will be able to type the name anyway.

Using @ClientType

@ClientType returns a text string to differentiate Notes and Web clients. Use @ClientType in formulas for which the outcome is different depending on client type.

Example
Used in a computed subform formula, the following formula inserts the subform "WebHead" if the form is to be displayed on the Web, and the subform "NotesHead" if the form is to be displayed on a Notes client:

```
@If(@ClientType = "Web"; "WebHead"; "NotesHead")
```

Using @UserNameList

@UserNamesList returns a text list containing: the current user name, any group names and any roles. It also works on local databases when the Enforce Consistent ACL option is selected. Returns "" for a local database where Enforce ... is not in effect.

@UserNameList will allow you to combine the functionality of @UserRoles and @UserName.

Example

```
@If(@IsMember("NewsEditors"; @NamesList); "Editor Head";
"Generic Head")
```

Note Unlike @UserRoles, this function does not append "$$WebClient" to the list. This is because @ClientType is now available for this purpose, so @UserRoles overloading is no longer necessary.

Password Field

A new field type called Password has been introduced in Domino R5.0. You can create a Password field that displays only asterisks when a value is entered into the field. This field allows users to enter a password in privacy.

However, you must either make sure that the password is passed on and not saved in the document or protect it in the document by using a Readers field or an encrypted field to avoid other users seeing the password through the InfoBox for the document.

Controlling if Users Paste Documents into Database

You must also consider whether users should be allowed to paste documents into your appplication database. If this is not controlled, users may be able to compose a document in another database in a way not allowed in your application and then paste the document into your application database.

To control the pasting of documents, create an Agent that runs when documents are pasted into the database and then apply the appropriate controls in that agent.

Hiding the Design of a Database

The developer can protect the design of a database by hiding it. Hiding the design of a database will not hide the data in that database.

Consider carefully before hiding the design of a database. It may be more difficult for the System Administrator to maintain while it is deployed. For example if you hide the design of a database the Agents view is also hidden and the System Administrator cannot enable or disable server agents unless you have added action buttons or similar functions.

Other Security Options and Considerations

Using Signatures for Security

Domino checks the signatures in design elements for two reasons:

- To allow execution of the application in the Notes client.

  Signers allowed to execute in the Notes client are listed in the client Execution Control List (ECL).

- To allow execution of server-based agents in the application.

  Signers allowed to execute agents on the server are listed in the Security (Agent Restrictions) section of the server document in the Domino Directory.

To make management of Execution Control Lists and the Security section of the server document easier, it is recommended that you create a common ID for your development organization that is used to sign all deliverables before handing them over to the System Administrator for deployment.

The following sections will explain:

- Execution Control Lists
- Server-based security for agents, Java and Javascript
- Signing a database
- Signing an applet

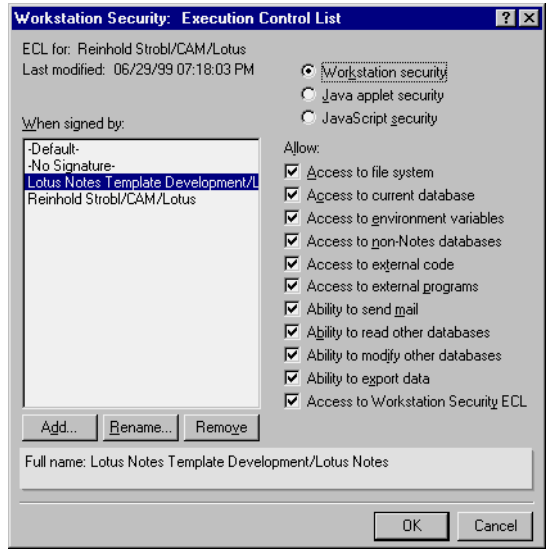
Execution Control Lists

Execution Control Lists (ECL) have stemmed from the concern that a Notes user does not have much control over what a Notes application is doing to their document, database, or system.

ECLs are a means by which the Notes user can now specify what level of access an executing formula or LotusScript program created by another person can have to their system.

By default, no scripts or formulas, whether signed or unsigned, can execute on your workstation without displaying a warning message.

ECLs are stored on a user's workstation. To work with ECLs choose File - Preferences - User Preferences. Click the Security Options button. The Workstation Security: Execution Control List dialog box is displayed:



The dialog is split into three areas. These are the developers' signatures and the levels of access that documents signed by these developers are granted on your workstation.

In the upper right corner you specify whether the level of access you specify relates to code in Domino databases (Workstation) or to Java applets executing in the Notes client Web browser or to JavaScript.

For example, suppose you are trying to create a document from a design that has been signed by the ID Domino Development/Asgård. You have specified in your ECL list that you do not want to give this ID access to your environment variables.

When you open up the document and the program tries to perform an @Environment command, a dialog box will appear telling you that the system is trying to access an environment variable when it does not have the authority to do so. If you want to allow the command to continue you can do so. You can either allow it to run this one time only, or you have the ability to change the ECL permanently to give Domino Development/Asgård the ability to access any of your environment variables in the future.

Note By default every template that comes from Lotus is signed by Lotus Notes Template Development which is given full access to your system.

Central Maintenance of the ECL
When new Notes clients are installed a default ECL is pulled from the server. The System Administrator also has the option of not allowing users to modify their ECL.

If the System Administrator updates the default ECL the new settings can be distributed to existing clients by mailing them a memo that includes a button that executes the function @RefreshECL and asks them to refresh their ECL.

A complete list of LotusScript and @functions that are affected by the ECL can be found in the Notes Help database.

Agent, Java and Javascript Security on the Server

The System Administrator can decide the access rights for agents and code utilizing the Domino Object Model based on the ID that the agent/code is signed with.

In the Security part of the server document there are entries for specifying:

- Agent Restrictions: options to specify who can:
  - Run personal agents.
  - Run restricted LotusScript/Java agents.
  - Run unrestricted LotusScript/Java agents.

- NOI Restrictions: options to specify who can:
  - Run restricted Java/Javascript.
  - Run unrestricted Java/Javascript.

Here the System Administrator can enter specific IDs, for example:

```
Domino Development/IBM
```

or give access to a whole organization, for example:

```
*/IBM
```

You can read more about the different levels of restrictions for agents and the Domino object interface in the Domino Administration Help.

Signing a Database

You can sign templates and databases developed by your organization so that you can then add the associated name to the Administration ECL. To do this:

1. Launch the Domino Administrator client. One way to do this is through the menu. Choose File - Tools - Server Administration.
2. If the ID you want to use to sign is not the current one, switch to the correct ID.
3. Click the Files tab.
4. Select the server that stores the databases or templates that you want to sign.
5. Select the databases or templates that you want to sign by highlighting them in the listbox.
6. In the Tools pane, expand the list under Database.
7. Click Sign. A dialog is displayed.
8. Select which elements to sign. To sign every design element in a database or template, select Sign every design note.
9. Click Sign. A dialog box shows the number of databases processed and the number of errors that occurred (if any). See the log file for details.
10. Click OK.

Access Control for HTML and Other Files

Domino extends its access control to files in the file system. This is particularly useful for HTML and other types of files used in Web sites. When using the Domino native HTTP stack, you will be able to configure access control lists for files (HTML, GIFs, etc.) in the file system. This gives you complete security and access control for your Domino Web sites, no matter how they are built or where their assets are stored.

File system protection does not apply to CGI scripts, servlets, or agents that access files on the system. The scripts, servlets, and agents have full access to any files accessed. File system protection does apply, however, to files that access other files, for example, HTML files that open image files. If a user has access to the HTML file, but does not have access to the JPEG file that the HTML file uses, Domino does not display the JPEG file when the user opens the HTML file.

A File Protection document needs to be created for each directory that contains files that you want to protect. The directory is relative to the HTML directory, set in the Server document or, if you have virtual servers, set in the Virtual Server document.

1. Open the Domino Directory.
2. In the Server/Servers view, highlight the Server document for the server on which you want to protect files.
3. Choose Actions - Create File Protection, fill out the following fields, and save the document:

Tab	*Field*	*Value*
Basics	Applies to	If you have virtual servers on this machine, select whether you want this setting to apply to all virtual servers on this machine or only to the virtual server you specify.
	IP Address	If you selected Virtual Server, enter the IP address of the server to which the file protection applies.
Path	Path	The drive or directory that you want to restrict. The path is relative to the HTML directory specified in the Server document or, if you have virtual servers, the Virtual Server document. If you want to restrict access to specific files, enter the name of the file or use the wildcard characters * or ? to specify a group of files.

continued

Tab	*Field*	*Value*
Access Control	Current access control list	The users who can access the files you specified and the type of access they are allowed. To add users to this list, click Set/Modify Access Control List. Select a user name from the Public Address Book in the Name field and select Read-Execute, Author, or Full access. Read-Execute lets the user open files and start programs in the directory. Author lets the user create new files in the directory in addition to the Read-Execute privileges. Full access lets the user delete files in the directory in addition to the Author privileges. If users connect to the server using Anonymous access, enter Anonymous in the Name field and assign the appropriate access.

4. Create a File Protection document for every path that you want to restrict on the server.
5. At the server console, type **`tell http restart`** to refresh the file protection settings.
6. To display the File Protection document, open the Server\Web Configurations view. Notes displays the File Protection document as a response to the Server document.

APIs for Customized Authentication, Encryption, and Signing

Domino R5.0 offers new APIs for:

- Authentication of Domino users by another system.

  This is part of the Domino Web Server API.

- Authentication of user for other applications.

  This is part of the support in Domino R5.0 for the Common Data Security Architecture interface.

Customizable Authentication Through Domino Web Server API

By using a new API provided with Domino R5.0 you may now create your own authentication schemes for Domino that can leverage external directories or security systems. This interface provides you with greater flexibility in integrating Domino into your existing environment and building "single-sign on" log in capabilities. Refer to the Domino documentation and release notes for more information.

Common Data Security Architecture (CDSA)

Domino provides a common, multivendor interface for managing various security services including a standards-based interface to Domino security for application developers. This interface exposes Domino encryption, signing, authentication, and other elements to application developers. CDSA makes the process of adding security to Domino applications easier and provides for interoperability between different applications that use security. Refer to the Domino documentation for more information.

Backup and Restore

Backup and restore is also part of securing your application.

You need to work with your System Administrator on how to implement backup and restore of application data during production. In larger organizations there will often be a standard mechanism for these important processes.

During development it is also important that you secure your work against loss through user or system error by making a backup of your development databases on a regular basis.

Developing a Plan for Securing Your Application

Securing a Domino application is a joint effort. The database Designer must work closely with the System Administrator and the database Manager to successfully design, create and deploy a secure application.

Depending on the company, the database Designer, the database Manager, and the System Administrator may be one person or three separate people. The System Administrator often takes over the responsibility of database Manager when the database is launched.

The following table shows a guideline for the tasks that are the normal responsibility of each participant in an application:

Database Designer	*Database Manager*	*System Administrator*
Design a security scenario	Implement the ACL	Set up and authorize users and groups
Design the access control list (ACL)	Update the ACL	Update the Domino Directory
Implement Design tools for security		Manage the server
Implement Design changes Implement the ACL		Propagate replication
		Manage user issues

Database Manager and System Administrator

Securing servers and controlling access to a domain is usually the responsibility of the Server Administrator. In addition, in a production environment, it will be the System Administrator who is assigned the role of database Manager in the ACL for purposes of setting up and maintaining the ACL. The database Manager should receive the following kinds of information from the Designer:

- A list of users, groups, and roles in each database.
- A comprehensive security plan for each database, so that it can be maintained on the server.
- All changes that are made in user and group access.
- Updates to access levels and restrictions.

Database Designer

While the Designer must design the security plan for the application, it is usually the System Administrator who has responsibility for implementing and maintaining the security plan. Therefore, it is essential that you work out your plan carefully, so that you can document it for the database Manager. Designers need to:

- Work with the Server Administrator while designing the application so that access levels and replication can be set up properly.
- Let the Server Administrator know whether there are any applications that require anonymous access.
- Determine which users and groups have access to which parts of the application.
- Decide what roles need to be added to the ACL for each database in the application.
- Document design changes for the Server Administrator so that they can replicate appropriately.

Note All users and groups need to be listed in the Domino Directory before they can be added to the access control list in a database.

Key Design Issues

Before setting up the ACL for an application, you need to create an access scenario. Use the following questions to create the scenario:

- Who is responsible for setting up and maintaining the ACL?

  The System Administrator usually has overall responsibility for the ACL, and the Designer would document the security mechanisms to be used for the Administrator.

- Which users need what kind of access?

  The Designer needs to inform the System Administrator about which access levels to set for users, servers and groups.

- Can you determine groups of users who need the same kind of access?

  The System Administrator needs to make sure that the groups are listed in the Domino Directory before adding them to the ACL.

- How is the database distributed: by direct replication or by design template?

  Design templates allow you to change and replicate the design without disturbing the production database. Design changes happen automatically through the designated template on the same server.

- Is there a hub server responsible for replication?

  If so, you should set up replication so that changes are added to the hub replica, then the hub adds the changes to other servers.

- Will this application be deployed on the Web?

  If so, it is advisable to always have an Anonymous entry for Web users so that you can specify exactly what Web users can do without registering.

Note The backup and restore of application data is part of securing the application. You also need to consider this during the design of your application.

Server Access

Server documents in the Domino Directory contain restrictions that are used to control access to a server. The database access control list refines these restrictions, but cannot override them.

Caution You cannot use server lists to control access by Web clients. The clients are not authenticated until they try to access a database.

Servers in the Database ACL
If there are replicas of a database, add server names and server groups to the ACL. Server access levels affect what information can be exchanged between the replica databases.

It is important to understand which design changes replicate and which do not, and how the database ACL and other replication settings affect the distribution of design changes:

- Servers need to have Editor access at minimum, so that they can replicate data changes.
- Servers must have Designer access to replicate design changes.

Planning for Web User Access

Users are granted access to the Domino Web server through basic authentication; the standard for Web security that is based on a challenge/response protocol.

The Web user can be challenged upon initial access to the Web site if restricted, or upon request to open any database that is restricted by the default entry in the ACL of No Access, or an Anonymous entry in the ACL of No Access.

All Web users have access to any database on the server that has a default access of Reader. If the database is restricted, a Web user must be listed in the Domino Directory with an HTTP password.

You can define additional access privileges and refine the ACLs for an authenticated user for a server, database, document, and so on.

Controlling Web Access to Domino Data
To set up Web access to your Domino data, you:

- Authenticate any Web client accessing a Domino server, database view, or document.
- Define server authentication at the user level by creating Web users and passwords.
- Choose which databases can be accessed by Web users and what level of access to provide for each database. It is a good practice to create a separate database for the home page and use ACL restrictions to control access to all other databases.
- Determine how to handle anonymous users.
- Optionally add encryption to HTTP transactions by activating Secure Sockets Layer (SSL) at the server.

Planning for Anonymous Users

You can control the level of access to a database for users who are not recognized by the system. These include both Web users and Notes users who do not share a certificate in common with the server. Such users are considered anonymous users.

Anonymous Access to Servers

Before anonymous users can be granted access to a database, they must be allowed server access. In the security section of the server document, the Administrator defines the security settings in the following ways:

If the Compare Notes Public Keys Against Those Stored in the Address Book option is:

- Yes; then the Notes user must have an entry in the Domino Directory to access the server.
- No; the Notes user does not have to exist in the Domino Directory to access the server.

If the Allow Anonymous Notes Connections option is:

- Yes; then all Notes users in the world can access the server.
- No; then only Notes users who have a certificate in common with the server can access the server.

Distinguishing True Security Features

As you have gathered from reading this chapter, it is important to distinguish between true security features and access control features that are used to make it more or less easy for users to use the application, and where the access control can be circumvented. Use this table to distinguish between the security abilities of the different database elements:

Is this a true security feature	*In Notes…*	*On the Web…*
ACL	Yes	Yes
Public Access to documents	Yes	Yes
Reader Access Lists and Readers fields	Yes	Yes
Author name fields	Yes, for users that have Author access in the ACL. However, users with Editor access will be able to edit the document even without being in the Reader list.	Yes, as in Notes

continued

Is this a true security feature	*In Notes…*	*On the Web…*
Field Encryption	Yes	Not Applicable
Signing	Yes, verifies ownership.	Not Applicable
Controlled Access Sections	No, users can modify the section through a different form, but combined with signing the section non-authorized updates can be discovered.	No, Given the document ID a Web user can get around the control in a section (this is not for the casual Web user though).
View restrictions	No, users may create private views and will be able to see all document content that isn't protected by Readers fields or encrypted fields.	Yes
Form restrictions	No, users can always see field content through the Document Properties dialog box.	Depends: No - when "Generate HTML for all fields" is selected. Yes - when "Generate HTML for all fields" is deselected.
Hide-when formulas	No, users can always see field content through the Document Properties dialog box.	Depends: No - when "Generate HTML for all fields" is chosen. Yes - when "Generate HTML for all fields" is deselected.
Prevent Copying and Forwarding	No	No (Not Applicable)

Summary

In this chapter we discussed the various levels of security available to Domino Server Administrators and application developers to secure information within a Domino database. We also discussed how Domino implements standard Internet-based security protocols and how these can be used.

Chapter 9
Searching

This chapter will cover setting up Web searching with Domino. Topics covered include:

- Search related URLs
- Full text indexing of databases
- Customizing search forms
- Customizing search results
- Setting up a site search database

Adding Search Capabilities to Your Web Site

As the number of databases making up a Web site increases, and the amount of data grows, you will want to add search capabilities to the site. Enabling your users to search for text throughout a Web site, or within a view, helps them to quickly and easily find the information they are looking for.

Domino provides a Full text search engine that acts on text within documents in a Domino database as well as searching the PDF, Word, Word Pro™, 123®, Excel, Freelance®, PowerPoint, and Ichitaro file attachments within those Domino documents. To do this the Domino server requires the HTTP server to be installed.

You can set up your site to allow text searching within a single view or in multiple databases. Creating an index for multiple databases requires creating a search site database on a server. The person creating the search site database specifies a search scope that includes all the databases to be included in the desired index, plus the type of information to be included. Each database in the search scope must have the "Include in multi database indexing" design property enabled.

You should then include buttons or links within your views, pages, or forms to open a search form when users click them. The users fill in the search criteria on the form and Domino returns a list of links to the documents that match the criteria.

Search-Related URLs

You can use the @Command([ViewShowSearchBar]) in the formula for your buttons or links to bring up the search form, or you can define a URL to display the search form. The search form can be either a customized search form or the default search form. Search-related URLs are available for both view searches and search site searches.

Search View URLs

You use search view URLs when you want to limit a search to documents displayed in one database view.

The search view URL enables you to control the search results. If you want users to be able to search on all documents in a database your search view URL points to a view displaying all documents, or you can limit the search, for example, to a view that shows only documents whose status is Completed.

When using the search view URL you can specify the number of documents you want to return from the search, and where in the returned list you want to start counting.

To display a search form for user-defined searches, include the $SearchForm object in the URL:

```
http://<site>/<database>/<view>/$SearchForm?SearchView
```

And if you want to specify the return documents, you can use the *Start* and *Count* arguments:

```
http://<site>/<database>/<view>/$SearchForm?SearchView&Query=st
ring_to_find&Start=start_number&Count=count_number
```

Where the *string_to_find* is the string you want to find, *start_number* is the number at which to begin counting the results, and *count* is the number of results to return.

The next example returns the 1st through to the 10th result that contains the word "CD-Records" in the "By Product" view:

```
http://www.lotus.com/Millenium.nsf/By+Product?SearchView&Query=
CD-Records&Start=1&Count=10
```

Search Site URLs

You can use search site URLs for text searches in multiple databases. The URL requires the name of a search site database.

To display a search form for user-defined searches, include the $SearchForm object in the URL:

```
http://<site>/<search site database>/$SearchForm?SearchSite
```

It is also possible to use the *Start* and *Count* arguments to return the number of documents:

```
http://<site>/<database>?SearchSite&Query=string_to_find&Start=start_number&Count=count_number
```

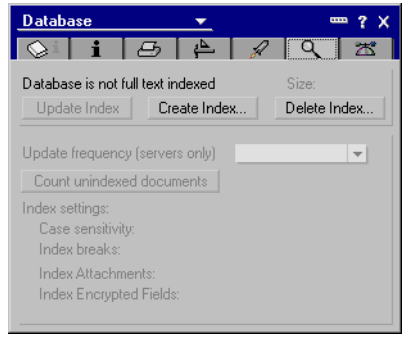
Where the *string_to_find* is the string you want to find, *start_number* is the number at which to being counting the results, and *count* is the number of results to return.

The next example returns the 20th through to the 29th result that contains the word "Milk":

```
http:/www.lotus.com/groceries.nsf?SearchSite&Query=Milk&Start=20&Count=10
```

Full Text Indexing

It is a prerequisite for Web searching with Domino that the databases you want to act on are Full text indexed. Use the Full Text tab in the database properties box to create a Full text index on a database.



A Full text index is a collection of files that lets a user search one or more databases for information. After creating a Full text index, a user can enter a word or phrase and locate all the documents containing that word or phrase.

To create an index for a single database stored on a server, you must have Manager or Designer access. You can index any database stored on your workstation without regard to access levels.

The Full text index for a search site database lets a user search all the databases in the search scope. Creating and maintaining a search site database can greatly affect system resources and performance. The system administrator should administer search site databases. As a database manager, you should know if any of your databases are associated with a search site database.

Security Issues

If the default Index Encrypted Fields option is selected when the index is created, the following three things happen:

- Any Notes user with access to the database can search for phrases within the encrypted fields without the encryption key. For example, the Employee form in the Personnel database contains the encrypted field Salary. With the Full text index, any user can search on "50,000" and documents containing that figure are returned; however, the user cannot read the contents of the field without the encryption key.
- The Full text index file is unencrypted plain text, and anyone with access to the server can read the file. Encrypted text may now be unencrypted.
- The encryption key is part of the server ID, and is active for all databases on that server. If you index a different database and do not deselect Index Encrypted Fields, any fields using that encryption key are compromised.

Location of a Full Text Index

When you create a Full text index, Domino creates a subdirectory in which to store the index files. Domino names this subdirectory using the name of the indexed database and the file extension .FT. When indexing multiple databases, the search site database is the indexed database.

For example, if you index a database named LOG.NSF, Domino creates the subdirectory LOG.FT. Domino places the subdirectory in the same directory as the database, usually the Notes data directory.

Size of a Full Text Index

The size of a Full text index depends on two factors:

- The ratio of text to non-text elements (such as bitmaps, buttons, and agents) in the indexed database(s). The percent of text typically ranges from 25% to 75%. If you choose to index attachments (including OLE objects) or encrypted fields, the text associated with them increases the percentage of text available for Full text indexing.
- The index options chosen. The Word Breaks Only option creates a Full text index that is about 50% of the space used by text in the database. The Word, Sentence, and Paragraph Breaks option creates an index that is approximately 75% of the space used by text.

Full text indexes take up a significant amount of disk space and you need to be sure adequate system resources are available to store an index.

Full Text Index Updates

Since the contents of most databases change over time, Full text indexes associated with them need to be periodically updated to keep in sync with the changes. You must manually update indexes on databases stored on your workstation. Domino automatically updates indexes on server databases according to frequencies selected by the database Manager for individual databases or System Administrator for a search site database.

Note Documents added to the database since the database's Full text index was last updated will not be returned by the search.

Customizing Search and Result Forms

You can customize searching at your site by adding you own design touches to search-input and search-result forms. A customized search form must be named $$Search, either as its actual name or as an alias. When a user clicks a link to open up a search form, Domino looks in the current database or in a search site database for a form of that name. If the form exists, Domino opens it. If it doesn't find it, Domino uses the default search.htm file found in the icons directory on the Domino server.

You can either customize the search.htm file, create a new search form of your own, or copy & paste the search forms provided in the Search Site database. There are two Web search forms in the Search Site database:

- **Web Search Simple.** This is used for simple searches. It includes a link to the Web Search Advanced form.
- **Web Search Advanced.** This form allows users to create more advanced searches.

The forms are designed to work in a Search Site database. To use the forms in a database that allows view level search you must change the formula in the $$Return field.

1. Copy the Web Search Simple form to the database where you want to use it.
2. Open the form in Design mode.
3. Go to the $$Return field. The beginning of the original formula looks like this:

   ```
   DBName:=@Subset(@DbName;-1);

   "[[/"+DBName+"?SearchSite&Query="+Query+"&SearchOrder="
   ```

4. You need to change ?SearchSite to ?SearchView and add a viewname after the DBName. The start of the new formula should look like this:

   ```
   DBName:=@Subset(@DbName;-1);

   "[[/"+DBName+"/ViewName/?SearchView&Query="+Query+"
   ```

   where ViewName is the name of a view in your database. The rest of the formula should remain the same. Since you are creating a URL, the viewname must start and end with a forward slash (/), and spaces in the viewname must be replaced with plus signs (+).

Make the same changes to the Web Search Advanced form if you want to include the advanced search in your database.

Creating a TeamRoom Search

In the following example we will create a simple-to-use customized search form. The form allows users to write a free-text search or use appointment pre-defined values to create queries. The form also enables the user to set the number of documents to be returned. Based on user selections, the form generates the search view URL and returns a list of documents found.

To avoid forcing users to learn the syntax for creating complex queries like:

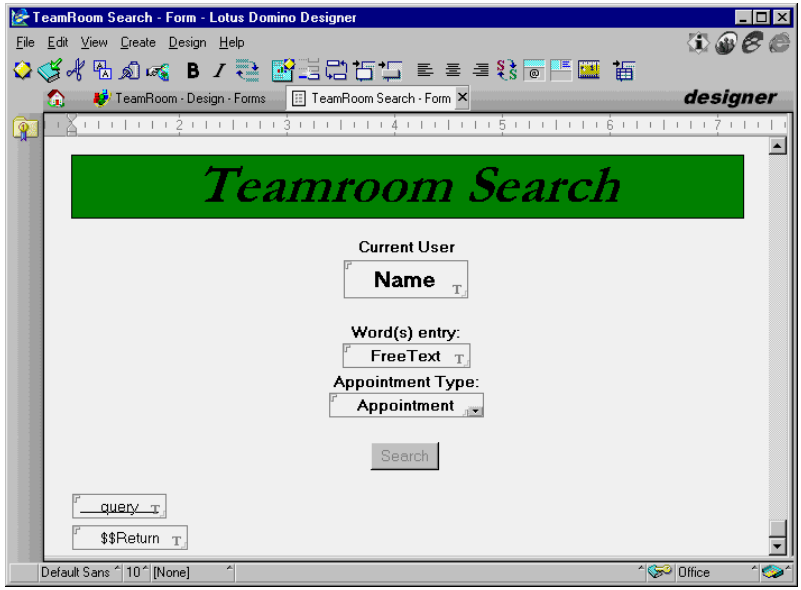
```
(<FreeText>) AND (FIELD Remote_User=Jane Woody) AND (FIELD
Subject=<Document
Subject>)&SearchOrder=<Value>&Start=<Value>&Count=<Value>&Searc
hWV=<Boolean>&SearchThesaurus=<Boolean>
```

You can create a simple-to-use interface for building complex queries.

We will use the TeamRoom database as an example, where we have created a few documents. We want to enable users to search their documents where the Subject field is one of the available values in the Appointment fields of the Search form and the searchable documents have to contain the Search Form FreeText field's value (If the value is entered).

Note You have to authenticate first that you have been recognized by the browser.

1. Start by creating a new form and giving it an alias of $$Search.



2. Create the following non-hidden fields:

   - Name.

     The Name field sets the current user name that we are using to search for the user's documents.

     Select the Field type Text and computed, and in the Design pane (Default value) enter the following formula:

     ```
     @Name([CN]; @UserName);
     ```

   - FreeText.

     FreeText is the field where the user can enter the search string.

   - Appointment.

     The Appointment field is a keyword field what contains the following values:

     ```
     Customer meeting
     Internal meeting
     Team meeting
     Reminded
     Untitled
     ```

3. Create the following hidden fields:

   - The Query field is computed. It is used to generate our query arguments. Enter the following formula for the field:

     ```
     user:="FIELD Remote_User="+Name;

     app:="FIELD Subject="+Appointment;

     user + @If(FreeText!=""; " AND (" + FreeText + ")"; "")+

     @If(Appointment!=""; " AND (" + app + ")"; "")
     ```

     The user variable gets the "FIELD Remote_User="+Name field, which is the name of the current user. The app variable gets the "FIELD Subject="+Appointment. This is used to determine the search query. The rest of the @If-functions seek what the user has entered; if both field values are empty, then Domino uses only the Name field for creating the search query.

   - We are using the computed $$Return field to generate our search URL with the search arguments, and have the browser immediately return the result of our query. To return a URL, the $$Return field must specify a URL in square brackets:

     ```
     DBName:=@Subset(@DbName;-1);
     "[[/"+DBName+"/OpenView/?SearchView&Query="+Query+"]]"
     ```

     When a user enters data in a form, like our search form, Domino normally acknowledges the submittal. To overrule the submittal functionality of the $$Return field and go directly to the results page we enclose the URL in double square brackets.

   - Normally, when a Web user submits a form, Domino will create a document in the database. In our case we do not want to create documents with the query form. To prevent this from happening we use a special Notes field called SaveOptions. If the field value for the SaveOptions field is "0" Domino will not save the document.

We have now accomplished what we wanted. Our users can now construct complex queries in the database by writing the free text query and simply pointing and clicking, without having to worry about the correct syntax.

Note We have used the (OpenView) view for showing all the databases documents, where the Search query is searching the documents.

Note All that is left to do is adding a link from our Web site to open the customized search form using the URL:

```
[</<database>/<view>/$$Search>]
```

To enhance your search form you may want to add fields to gather the following search arguments:

- SearchMax

  This field determines the maximum number of documents you want to return.
  The Syntax is: SearchMax=[n]

- Start

  Use this to determine the document that appears first in your list of results. Start=0 means that paged results will not be returned.
  The Syntax is: Start=[n]

- Count

  Use this to specify the number of results to display on each page until the SearchMax has been reached.
  The Syntax is: Count=[n]

- SearchFuzzy

  This is a keyword field which specifies whether you would like to return only documents which contain exact matches to the current word.
  The Syntax is: SearchFuzzy=[TRUE,FALSE]
  The default is FALSE.

- SearchWV

  This is a keyword field and it finds word variants as defined by thesaurus.
  The Syntax is: SearchWV=[TRUE,FALSE]
  The default is TRUE.

- SearchOrder

  The Sort field determines in which order you want to return the documents.
  The Syntax is: SearchOrder=[1,2,3] where 1="Sort by relevance", 2="Sort by date ascending", 3="Sort by date descending"
  The default is 1.

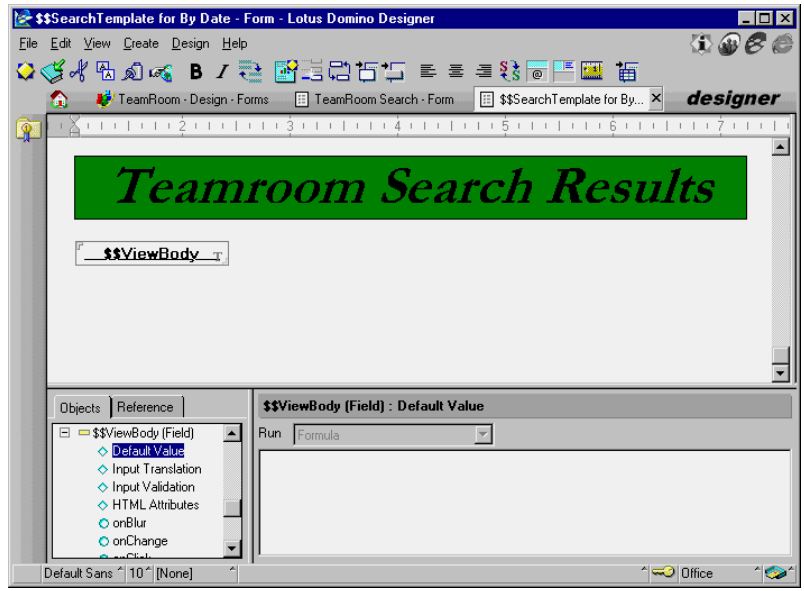
Customizing Search Result Forms

To customize the Search Results page, create a form and assign it one of the form names below:

- **$$SearchTemplate for <viewname>**
  Associates the form with a specific view. The form name includes <viewname>, which is the alias for the view, or, if no alias exists, the name of the view.

- **$$SearchTemplateDefault**
  This form is used as the default search result form for all Web searches that aren't associated with a specific form.

Both forms require a $$ViewBody field on the form. The value of the field is ignored by Domino, but the field is mandatory on a search result form.



Note The following features in the default Search results form provided with Domino cannot be built into a customized Search results page:

- A count of the number of documents found
- The ability to restate the search query
- A search bar that allows users to reset search results

Search Site Databases

You can create search site databases that enable users to search multiple databases for information. To set up a search site database, you should configure a search scope which defines the databases and the information in them that users can search. The databases included in the search scope can span more than one server and more than one domain, but must have the Include in Multi Database Indexing option selected.

You then create a Full text index that indexes all the databases in the configured search scope. You don't need to create Full text indexes for individual databases included in the search scope unless you want users to be able to perform view searches as well.

You can create more than one search site database, each indexing a group of related databases. For example, you can create a search site database for marketing databases, another for sales databases, and another for customer service databases.

To search the databases for information, users fill out a search form in the search site database and specify search criteria. Domino returns each document found in a search results form, as in the case of single database view searches.

Creating a Search Site Database

1. Choose File - Database - New.
2. Select server location.
3. Enter a title and file name for the search site database.
4. Select Show Advanced Templates.
5. Select Search Site (SRCHSITE.NTF) as the template.
6. Click OK.

Note The default access of the database created is Author. There is a role named [SearchSiteAdmin] in the access control list, which is associated with the Search Scope Configuration form and all of the views except the Private Searches view. Database Managers should assign this role to anyone who is authorized to create configuration documents in the database.

Defining a Scope

Create Search Scope Configuration documents:

- To specify a domain as a search scope, click Domain and enter the name of the Domain.
- To specify a server as a search scope, click Server and enter the name of the server.
- To specify a directory on a server as a search scope, click Directory then enter the name of a server and a directory on the server relative to the Notes data directory.
- To specify a specific database as a search scope, click Database and enter the name of the server and the file name for the database.

Refining a Search Scope

When you modify a database entry to refine a search scope, for example when you change the Full text index option to No Index to exclude the database from searching, Domino create a search scope configuration document for the database that reflects the change.

A search scope may be refined automatically according to which databases have the design property Include in multi database indexing. For example, if this design property is selected for only ten databases on a server, you can create a search scope configuration with the server as a scope, but only those ten databases are indexed.

Caution When you change configuration options, you have to delete the Full text index on the search site database and recreate it for the changes to take effect.

Database Views

You use any of the following views to see the databases included in a search scope:

- All By Server
- Databases By Category
- Databases By Replica ID
- Databases By Server
- Databases By Title

Selecting an Indexing Option

The index option you select affects the size of the Full text index for the search site database.

- To exclude databases from the selected scope of a search, click No Index. This option excludes a database or databases and therefore adds nothing to the index size.
- To index only summary data that appears in views and not rich text or attachments, click Index Summary Data (No RTF). If a search scope is wide-ranging, use this option as a way to limit the size of the Full text index.
- To index only summary data and rich text but not attachments, click Index Full Document.
- To index summary data, rich text, and attachments, click Index Full Document and Attachments. This indexing option increases the index size the most. Notes indexes only the ASCII text in attachments and OLE objects. If an attachment or OLE object is compressed, it is not indexed.

Multi-Database Full Text Indexes

A Full text index created for a search site database is just like a Full text index created for a single database except that it indexes multiple databases. The index is stored in a subdirectory to the Notes data directory that stores the search site database. The subdirectory name is SEARCHSITE.FT, where SEARCHSITE represents the name of the search site database without the .NSF extension.

You should plan carefully which databases and data to include in a search scope, and make sure the server that stores the search site database has adequate space. Including many databases in a search scope may require a server which stores only the search site database. You can help keep the index size to a minimum by selecting the index options Word Breaks Only and Exclude words in stop word file, when you create the index, and by selecting the option Index Summary Data when you configure a search scope, which prevents rich text from being indexed.

Because updating the Full text index of a search site database can take some time, you should select Daily as an update frequency so that the index updates once a day during off-hours.

How Users Search Using a Search Site Database

Incorporate SiteSearch URLs in your site by adding them where you want them. The search site database contains two Web forms you can use as is or modify:

- Web Search Simple.
- Web Search Advanced.

The search criteria the forms use is similar to the criteria used for a single-database search. You can search for a specific word or phrase or perform advanced searches which allow searches for multiple words, word variations, synonyms defined by the thesaurus, and documents by date and category. The search results can also be sorted.

Domino returns each document found in a Search Results document. Each document entry includes a document link and summary information about the document from the default view of the database the document comes from.

Caution All people and groups in the ACL for the databases included in search scope must have at least Reader Access (including the OtherDomainServers group). If not, the document entry in the search results form will only show the document link as the database name; it will not display any summary information about the document. If you are using ReaderNames fields in a database, those documents will not appear in a search.

Note If the property Show in Open Database dialog, has not been selected for a database in the search scope, then a Search Results document for that database shows a document link but no text.

Customizing Search Site Result Forms

To customize the Search Results page in a search site database, create a form and assign it the following name: $$SearchSiteTemplate. This form is used as the default search result form for all Web searches in the search site database.

The form requires a $$ViewBody field. The value of the field is ignored by Domino, but the field is mandatory.

Summary

Domino provides a powerful Full text search engine that acts on text within documents in Domino databases. With Domino you can combine text searching within a single view or in multiple databases.

To enable view level searching, databases must be Full text indexed. The search site database indexes all databases in the search scope. A database included in the search scope does not need to be Full text indexed itself, unless you want to allow view level searching on the database as well.

Search and result forms are fully customizable, enabling you to utilize all the forms design features such as graphics, buttons, tables, and navigators when you design search facilities for your Web site.

Chapter 10
Programming for Domino

This chapter will cover the available methods to use when programming in Domino. We will briefly discuss Simple Actions and the Formula language, but will also focus on the LotusScript and JavaScript languages. The chapter will explain how, when, and where you can use them and also introduces the new formulas and LotusScript functions that are available in Domino R5.0.

Hints and tips on how to manage your code more efficiently using the Template database are also included.

Programming in Notes

The following section explains the differences between the three existing integral programming interfaces to Domino: Simple Actions, LotusScript and the Formula language. It will give you a short overview of Simple Actions, and it will compare LotusScript and @functions. The examples are written in LotusScript, which is one of the programming languages you can use to develop your applications.

Simple Actions

These are predefined actions which allow you to define a sequence of actions without requiring any programming knowledge. They are ideal for the end user who wishes to automate some routine tasks. The simple actions available are:

- Copy to database
- Copy to folder
- Delete from database
- Mark document read
- Mark document unread
- Modify field
- Modify fields by form
- Move to folder

- Remove from folder
- Reply to sender
- Run agent
- Send document
- Send mail message
- Send newsletter summary
- @Function Formula

Note Simple Actions only cover basic functions. To implement more complex functions you might consider using @formulas or a programming language, such as LotusScript or Java.

Formula Language

Domino formulas are expressions that have program-like attributes. For example, you can assign values to variables and use a limited control logic. The formula language interface to Domino is provided through calls to @functions. If you are familiar with the macro language in other products, such as 1-2-3, then you will quickly become proficient in the @functions in Domino.

@Functions are a powerful tool when you want to manipulate the current Domino document in an application, since the developer need not obtain the context for the document.

```
@SetField(@Subset(TmpVar; 1));
```

or, you can use only one function at the time:

```
@UserName;
```

Error Handling Using Formulas

New to Domino R5.0 are improved formula error messages. Formula error messages now give specific information as to where the error has occurred, which helps developers when debugging their applications.

For example, if you are entering a syntactically incorrect @formula expression for field validation, Domino will now give you an accurate message as to where it has detected the error.

New Functions

The following table summarizes all the new functions and commands in Domino R5.0:

Functions & Commands	*Description*
@AddToFolder	Adds current document to one folder while removing it from another. NULL string can be substituted for either argument to skip the action.
@BrowserInfo	Returns information about the user agent, including version, HTML/JavaScript support, language, etc.
@FontList	Brings you the list of available fonts.
@FormLanguage	Returns the language or the current form in a format based on RFC1766.
@HardDeleteDocument	Only works in an agent running a formula. Physically deletes a document from the database, as opposed to @UnDeleteDocument and @DeleteDocument which now have options to soft delete documents.
@IsAppInstalled	Checks whether an application like "Designer" or "Admin" is installed.
@LanguagePreference	@LanguagePreference returns a user's preferred language for database contents. If a database has language-dependent contents like design elements or data, the preferred language is used to select suitable contents from the database.
@LaunchApp	Launces one of the following Domino applications: Notes, Designer, Admin
@Locale	Returns the language name for the specified Notes internal language code.
@NameLookup	Internally @NameLookup functions calls the NAMELookup Notes API, specifying "($Users)" view for view name to be searched.
@Narrow	Converts full-pitch alphanumeric character (Double byte characters — DBCS) in the specified string into half-pitch alphanumeric characters (Single byte characters — SBCS).
@PasswordQuality	Returns a number that indicates the quality level of a password string.

continued

Functions & Commands	*Description*
@SetTargetFrame	This feature allows application designers to specify a target frame when opening Notes design notes.
@UndeleteDocument	In a database that has soft deletes enabled (a new database property), this command removes the soft deleted status if a document has been soft deleted.
@UserNameLanguage	@UserNameLanguage returns language tag associated with your user ID. It is a corollary function to enhanced @UserName. @UserNameLanguage retrieves language tags from your ID file that are expected to be the same value within the AltNameLanguage field in the Domino Directory.
@UserNamesList	Returns a text list of all the names by which the user is authorized: user name, group names, and roles. This is the same list that appears in the "Groups and Roles" dialog when you click the "What's my access" button on the status bar in R4.
@V4UserAccess	Indicates which access the user has to a given database on a given server. Must be used in databases that are on Domino R4.x format.
@ValidateInternetAddress	This function validates the Internet address.
@Wide	Converts half-pitch alphanumeric characters (Single byte characters - SBCS) in the specified string to full-pitch alphanumeric characters (Double byte characters - DBCS). This function works in Japanese, Korean, Simplified Chinese, and Traditional Chinese environments. In the Japanese environment, this function can convert half-pitch Katakana as well.
@Command([AddBookmark])	Adds a bookmark with the specified URL or current object.
@Command([EditArchiveSettings])	Displays the archive settings dialog.
@Command([Directories])	Opens the Directories windows. Allows user to add found name to personal address book.
@Command([OpenFrameset])	Opens the defined frameset.
@Command([OpenHelpDocument])	Opens the defined Help document.
@Command([OpenPage])	Opens the Defined Page.

continued

Functions & Commands	*Description*
@Command([RefreshParentNote])	Used only in dialog boxes. This formula sends the values entered in the dialog box to the parent document. A Designer can update a parent note and close the dialog box without having to use the OK button on the dialog box.
@Command([ReloadWindow])	Reloads or refreshes the contents of the current window. If the window is a frameset the contents of all frames will be reloaded.

LotusScript

Despite all the attention being given to Java and JavaScript languages in Domino, LotusScript is not going away; in fact, a new version of LotusScript is included in Domino R5.0, LotusScript R4. LotusScript offers the application developer the wide variety of features expected of a modern, fully object-oriented programming language. Its interface to Domino is through predefined object classes. Domino oversees the compilation and loading of user scripts and automatically includes the Domino object class definitions. This allows you to code your programs in an efficient way.

While @functions are ideal for coding simple logic, for example, input translation or input validation of a field, LotusScript provides the ability to code loops, select (case) constructs, and a lot more. Also, the Integrated Development Environment (IDE) performs automatic indentation, which follows the program logic in IF-THEN-ELSE and loop constructs and makes your programs readable and easy to maintain.

Furthermore, the hierarchy of the Domino object classes represents the flow of control you follow in the user interface if you step down from a database icon to a view, and further on to a document, and to a specific field within this document. For example, if you are coding in LotusScript, you will start with the UIWorkspace class and go down to the UIDocument class which represents the currently open document. Once you have set this object variable, you have access to the fields of the document.

The same principle applies if you are working with the back-end classes of Domino, which represent the objects you might wish to work with that are not in the user interface. You will start at the Notes Session class and go down through the Notes Database class to the Notes Document class. The front-end and back-end classes are described in the section about Domino Object Models.

Here is a short summary of the benefits offered by LotusScript:

- **Superset of BASIC**

  Since LotusScript is a superset of the BASIC language, it is easy to learn, especially for Visual Basic users. You can write sophisticated scripts by using conditions, branches, subroutines, while loops, and others.

- **Cross-platform**

  LotusScript is a multi-platform scripting language. You can create just one application, which can be used on any supported platform.

- **Object-oriented**

  Domino objects are available to LotusScript. You can write scripts to access and manipulate these objects. The scripts are event-driven, such as by an action, clicking the object or button, opening a document, or opening a view.

- **Included in Lotus applications**

  Since LotusScript is supported by all the Lotus products, these products are able to access Domino objects using a Domino-supplied LotusScript extension. Another advantage is that you only need to learn one language to become proficient in writing scripts in other Lotus products.

- **OLE support**

  Domino can be the perfect container for SmartSuite documents and other OLE-enabled applications, such as Microsoft Office. You can use external OLE 2.0 automation objects by scripting them, such as 1-2-3 worksheet objects.

  Domino registers itself as an OLE automation server. External applications can use these objects in scripts to create and reference them. LotusScript is able to combine all the parts and provide the means for controlling and manipulating objects.

- **Coexistence with Notes @functions**

  Lotus continues to support @functions. LotusScript can work with them.

- **Integrated development environment**

  The Domino R5.0 Integrated Development Environment (IDE) provides an easy-to-use interface to create, edit, and debug scripts, and to browse variables and properties of the Domino Object Model. This allows you to write more complex scripts in Domino.

- **Extendable through LSXs**

  You may extend LotusScript by writing your own classes, which are called LotusScript eXtensions (LSXs) in C or C++, as a Dynamic Link Library, DLL. Creating your own LSXs allows you to expose custom functionality to LotusScript developers in precisely the same way as Domino functionality is exposed. You might use this, for example, if you have customer processing logic, such as a proprietary pricing process, that you wanted to make available to Domino developers.

- **Connecting to external databases**

  You can connect your application to use another database, for example DB2, by using the LS:DO. The benefit is that you can use the existing database so that data is stored in only one place.

The Domino Object Model

We will now take a look at the Domino Object Model (DOM). Using this model, you have access to Domino databases and application services. The Domino Object Model is mapped to a set of object-oriented classes available for building applications. You can access the Domino Object Model from a broad range of languages, including Java, LotusScript, and Visual Basic. In Domino R5.0 the classes have also been exposed as CORBA objects to enable the creation of distributed applications. For more highly customized applications, you can directly access Domino services using the C++ APIs.

If you are going to write your own application you can use the objects, methods and properties defined in this model to work with Domino objects, for example, databases, views, and forms. Normally, you use the properties of an object to get information about the object, for example, you use the ReplicaID property of the database object to query the ReplicaId of a database. On the other hand, you use methods to perform actions on an object, for example, the CreateDocument method of the database object creates a document in a given database.

Conceptually, there are two types of objects. They are:

- Front-end UI (user interface) objects
- Back-end (server) objects

Domino Front-End UI Objects

Front-end UI objects are used to manipulate the current user interface. They are typically used for programming events and give you access to the Domino object that the user is currently working on. The following front-end UI objects are available:

- **NotesUIWorkSpace** represents the current Notes workspace window.
- **NotesUIDatabase** represents the currently used database.
- **NotesUIView** represents the currently used view.
- **NotesUIDocument** represents the document that is currently open.

The following objects have only events associated with them:

- **Button** represents a button.
- **Field** represents a field.
- **Navigator** represents a navigator.

Domino Back-End Objects

Domino back-end objects are used for manipulating Domino data. They do not support any event or user interface interaction. Nevertheless, you can combine back-end objects with front-end objects in UI scripts. For example, the NotesUIDocument object has a property called *Document* which provides access to the underlying document.

The following back-end objects exist:

- **NotesSession**

  Represents the Domino environment of the current script, providing access to environment variables, Domino directories, information about the current user, and information about the current Domino platform and release number.

- **NotesDbDirectory**

  Represents the Domino databases on a specific server or local machine.

- **NotesDatabase**

  Represents a Domino database.

- **NotesACL**

  Represents the Access Control List (ACL) of a database.

- **NotesACLEntry**

  Represents a single entry in an Access Control List. An entry may be for a person, a group, or a server.

- **NotesAgent**

  Represents an agent.

- **NotesView**

  Represents a view or folder of a database and provides access to documents within it.

- **NotesViewColumn**

  Represents a column in a view or folder.

- **NotesDocumentCollection**

  Represents a collection of documents from a database, selected according to specific criteria.

- **NotesDocument**

  Represents a document in a database.

- **NotesItem**

  Represents a piece of data in a document. All of the items in a document are accessible through LotusScript, regardless of what form is used to display the document in the user interface.

- **NotesRichTextItem**

  Represents an item of type rich text.

- **NotesRichTextStyle**

  Represents the rich text field attributes.

- **NotesEmbeddedObject**

  Represents embedded objects, linked objects, and file attachments.

- **NotesDateTime**

  Represents a date and time. Provides a means of translating between the LotusScript date-time format and the Notes format.

- **NotesDateRange**

  Contains a range of NotesDateTime. An object of type NotesDateTime represents a given date and time.

- **NotesLog**

  Enables you to record actions and errors that take place during a scripts execution. You can record actions and errors in a Notes database, a mail memo, or a file (for scripts that run locally).

- **NotesNewsLetter**

  Represents a document that contains information from, or doclinks to, several other documents. All of the NotesItem properties and methods can also be used on a NotesRichTextItem.

- **NotesForm**

  Represents a form in a Notes database.

- **NotesInternational**

  This object contains properties which provide information about the international settings, for example, date format, of the environment in which Domino is running.

- **NotesName**

  Properties of this object contain information about a Domino user name.

- **NotesTimer**

  Objects represent a timer in Domino.

- **NotesRegistration**

  Represents the creation or administration of an ID file.

- **NotesOutline**

  Represents the Notes Outline attributes.

- **NotesOutlineEntry**

  Represents an entry in a Notes Outline.

- **NotesReplication**

  Represents the replication settings of a database.

- **NotesRichTextParagraphStyle**

  Represents RichText paragraph attributes.

- **NotesRichTextTab**

  Represents RichText tab attributes**.**

- **NotesViewEntry**

  Represents a view entry. A view entry represents a row in a view.

- **NotesViewEntryCollection**

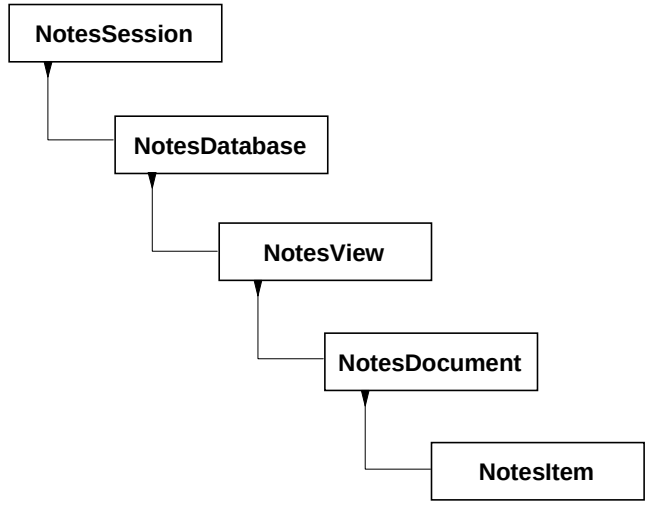
  Represents a collection of view entries, selected according to specific criteria.

- **NotesViewNavigator**

  Represents a view navigator. A view navigator provides access to all, or a subset of, the entries in a view.

Object Hierarchy

There is a hierarchical relationship for Domino objects. Higher hierarchical objects contain the lower ones. The figure below is an example of the hierarchical relationship between a few of the Domino objects:



Each object has defined members, properties and methods. Using these members, you can access other objects. The relationship of containment and access means that the higher object has the property or the method to access the lower one.

For example, you can see all the views when you open the database. This means that the opened database(object) in the workspace includes the views(object). Furthermore, you can see the documents when you select one of the views. This means that your selected view(object) contains the documents(object). This hierarchy is important when using Domino objects. NotesSession is the top level object in the Domino Object Model. You can work your way to any Domino object if you start from NotesSession.

Using Domino Objects from LotusScript

We will now look at some examples of code which use objects in LotusScript.

Example 1: Getting the Text of the Subject Field

```
Dim session As New NotesSession
Dim db      As NotesDatabase
Dim view    As NotesView
Dim doc     As NotesDocument
Dim item    As NotesItem
Set db   = session.CurrentDatabase
Set view = db.GetView( "Main View" )
Set doc  = view.GetFirstDocument
Set item = doc.GetFirstItem( "Subject" )
```

First, we declare the variable *session* as types of *NotesSession* object, and *New* is used to create an instance of that object.

We declare the variables *db, view, doc, item* as types of *NotesDatabase, NotesView, NotesDocument, and NotesItem* objects, respectively.

To get the text of the subject field, we need to follow the hierarchical path from the top to the lower one. In this example, we go from *NotesSession* object to *NotesItem* object:

NotesSession - NotesDatabase - NotesView - NotesDocument - NotesItem.

We initialize the variable *db* with the property *CurrentDatabase* of the higher level object.

We set the object variable *view* using the *GetView* method, giving it the name of a view.

The next statements are the same as before: we use the methods *GetFirstDocument* method to give us the first document from the view, and *GetFirstItem* to get the subject field from the document.

Example 2: Disabling a Role for a Person

```
Dim session As New NotesSession
Dim db      As NotesDatabase
Dim acl     As NotesACL
Dim entry   As NotesACLEntry
Set db    = session.CurrentDatabase
Set acl   = db.ACL
Set entry = acl.GetEntry("Susan Preissler")
Call entry.DisableRole("Auditor")
Call acl.Save
```

To access the personal ACL (access control list) data for a database, you need to follow the hierarchical path from the top class to the lower one. This example steps from the *NotesSession* object to the *NotesACLEntry* object:

NotesSession - NotesDatabase - NotesACL - NotesACLEntry.

The object that you would like to manipulate has methods or properties to handle its own data. The first seven lines of this example are similar to Example 1. The eighth line uses the *DisableRole* method of the *NotesACLEntry* object to disable the role [Auditor] for "Susan Preissler."

Example 3: Getting the Subject Field of All Documents

```
Dim db    As New NotesDatabase("Server","db.nsf")
Dim dc    As NotesDocumentCollection
Dim doc   As NotesDocument
Dim item  As NotesItem
Dim subject As String
Set dc = db.AllDocuments
Set doc = dc.GetFirstDocument()

While Not(doc Is Nothing)
  Set item =  doc.GetFirstItem("Subject")
  subject = item.text
  Set doc = dc.GetNextDocument(doc)

Wend
```

The earlier two examples start at the *NotesSession* object, but to access an existing database when you know its server and file name, you can get the database object directly as shown in the first line. This illustrates a unique feature of writing Notes applications in LotusScript, as opposed to the formula language: you can access any database from within a script, and perform *any* function upon it.

The following sequence is the same as in the earlier examples. The *NotesDatabase* object contains the *NotesDocumentCollection* object, which contains *NotesDocument*:

NotesDatabase - NotesDocumentCollection - NotesDocument - NotesItem.

We use the *AllDocuments* property of the *NotesDatabase* object to get all the documents in the database.

Next, we use the *GetFirstDocument* method of the *NotesDocumentCollection* object to get the first document in a collection.

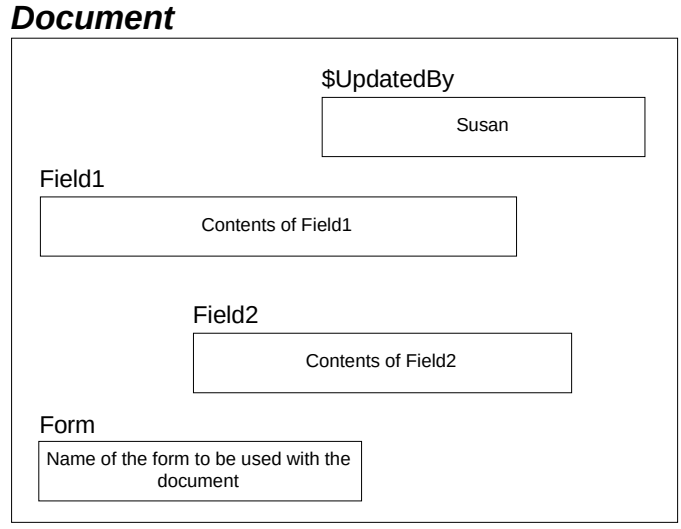
We then use the *GetNextDocument* method of the *NotesDocumentCollection* object to get the document immediately following the earlier document in a collection. If a document does not exist in a collection, the *GetNextDocument* method returns *Nothing*.

Understanding Front-end and Back-end Classes

First of all, you need to consider how data is stored in Domino. You can think of a document within a Domino database as a record, but a Domino document is more sophisticated than a typical database record. It may contain rich text, pictures, objects, and many other types of information. For example, if you access a Domino document using the back-end classes, you may manipulate the contents of the fields, add new fields to the document, or remove fields from the document. However, if you make such changes, the input translation and input validation formulas contained in a form are not executed.

On the other hand, if you work with front-end objects of Domino, your changes to the fields are visible to the user. For example, if you invoke the Refresh method of the NotesUIDocument class, the input translation and input validation formulas are carried out.

The following figure represents a back-end document and shows how data is stored in a Domino database:



The fields *Field1* and *Field2* have been defined in the form which was used to create this document. The name of this form is stored in the field *Form*. If you change the value of the field *Form* using an agent or LotusScript, the document will be presented to the user using the other form when it is opened the next time.

Note If there is no form field within a document, Domino will display such a document using the database default form. If there is no default form, the document cannot be displayed.

Field *$UpdatedBy* is an internal field created by Domino and contains a list of users who have worked on this document.

Tip New for Domino R5.0, you can now specify the maximum amount of user entries kept in this variable in the InfoBox of the database.

Note Mostly, field names starting with $ are used and maintained by Domino.

Using Domino Objects From Java

You can also access the Domino back-end objects from Java. This allows you to write parts of your application in Java. The Java program runs on the machine where Domino is installed. For example, Java agents can be written that will manipulate Domino objects.

Note The Java classes are not a port of the LotusScript classes to Java. Actually the same C++ code is executed, only the syntax of the interface is different.

Programming With LotusScript

When you program in Domino, you write your LotusScript code to affect Domino objects. Your code is executed by the occurrence of an event to the objects; such as clicking a button, opening a document, closing a document, or entering data in a field. Using the Objects view, you can easily see the events that are available for an object.

For example, you can write a very simple script for an object such as a button:

```
Sub Click( Source As Button )
   MessageBox( "I'm learning LotusScript!" )
End Sub
```

This script just shows a message box when you click the button.

The Event Model

Each programmable object in Domino has a list of associated events that it responds to; for example, a button responds to an event called "click" that is executed when the user clicks the button.

Another example of this is the postopen event of the form, which is triggered when a user displays a document on their screen. The postopen event occurs once all data has been loaded from the backend database into the form, and just before it is displayed to the user.

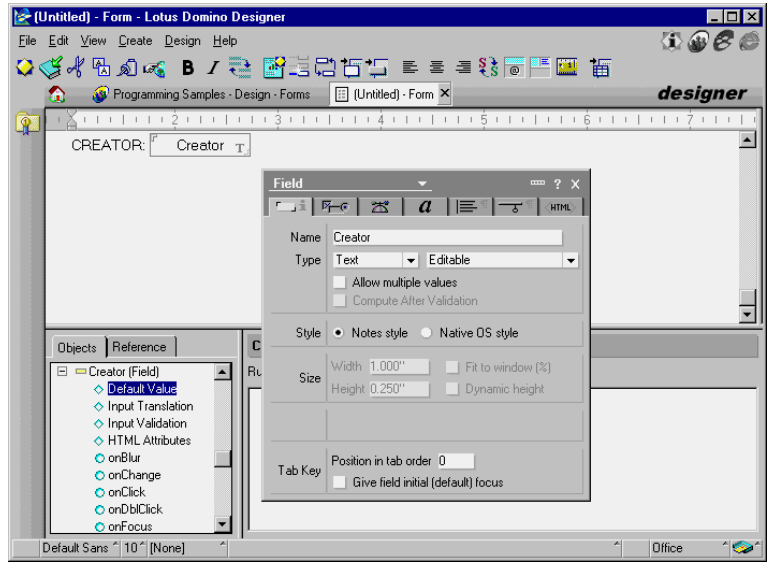
Here, we will show you how to use the postopen event.

Using the Postopen Event

This next example adds an action to the Form (document) object and uses the *Postopen* event.

Note You should try the following example by creating a temporary database based on the Blank template, so as not to corrupt any existing databases.

1. After you have created a blank database, open it in Design mode.
2. Select the form design view and click New Form. The new form is displayed.
3. Enter CREATOR: at the cursor blinking position.
4. Choose Create - Field. The InfoBox used to set the properties of a field appears.



5. Enter Creator in the Name: field of the InfoBox and close the box.
6. Go to the Untitled (Form) in the Objects view.
7. Choose the Postopen event.
8. Edit the LotusScript so that it looks exactly like this:

```
Sub Postopen(Source As Notesuidocument)
  Dim session As New NotesSession
  If source.EditMode Then
    Call source.FieldSetText("Creator",
session.CommonUserName)
  End If
End Sub
```

9. Choose File - Save.
10. Enter LotusScript1 as the form name and click OK.

Running the Example

There are two alternative options to test the form. You can do either of the following

- Click Design - Preview in Notes.
- Domino opens the current form and shows its contents.

Note This only lets you run the form you are working on. If you want to test the whole application, it is better to open the current database.

or

- Select the database to which you added the script.
- Choose Create - LotusScript1.

The new form LotusScript1 appears and your name is set in the creator field.

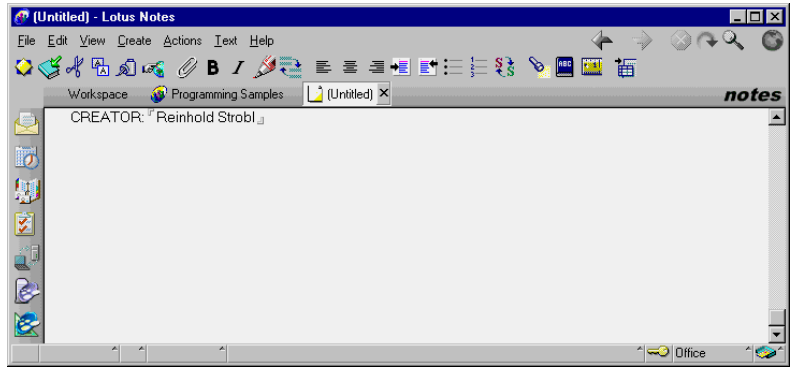
This script runs after the user opens the document. If the document is new, the Creator field is set to the name of the creator. You can select any events mentioned earlier and write a script. For example, you can select the QuerySave event to check whether or not every field has a value entered in it before the document is saved.

Event Type and Sequence

Below, we describe the event for each object in Domino.

Field Object

The Field object has the following events where you can write LotusScript. Other events are available for JavaScript or Formula:



- **(Options)** (Provided area for LotusScript options)
- **(Declarations)** (Declare all global variables)

  Note In the above two events, you can't write executable LotusScript statements
- **Initialize** (When it is being loaded)
- **Entering** (When the cursor is moved to the field in edit mode)
- **Exiting** (When the cursor is moved out of the field edit mode)
- **Terminate** (When it is being closed)

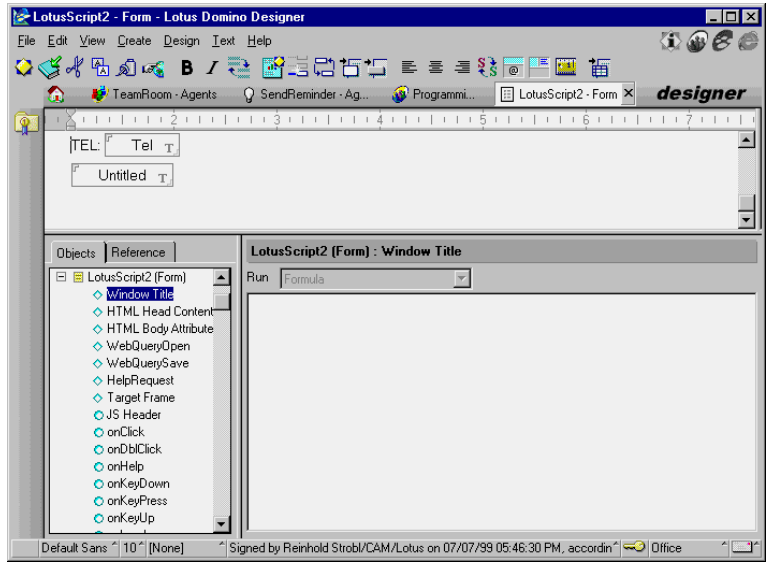
Example

We will now add an action to a Field object and use the Exiting event.

Note You should try the following example by creating a temporary database to avoid corrupting any existing databases. You could use the database that you used in the first example.

1. Select the database and then the Form design view. Next, click New Form in the programming view. The form design window is opened and the cursor is blinking at the top left-hand side.
2. Enter TEL: at the cursor blinking position.
3. Choose Create - Field. The InfoBox appears.
4. Enter TEL in the Name: field of the InfoBox.
5. Create one more field. You don't need to change the name in the InfoBox — you can leave it as Untitled. When we go to this field later, we will exit from the TEL field, which will cause the exiting event to occur.
6. Choose TEL (Field) from the Objects view and choose Exiting event.
7. Edit the LotusScript so that it looks exactly like this:

```
Sub Exiting( Source As Field )
  Dim ws    As New NotesUIWorkspace
  Dim uidoc As NotesUIDocument
  Set uidoc = ws.CurrentDocument
  tel = uidoc.FieldGetText("TEL")
  If tel = "" Then
    While tel = ""
      tel = Inputbox("Enter your telephone number")
    Wend
    Call uidoc.FieldSetText("TEL", tel)
  End If
End Sub
```



8. Choose File - Save. Enter LotusScript2 as the form name and click OK.

Running the Example

1. Select the database to which you added the script.
2. Choose Create - LotusScript2. The new form LotusScript2 appears.
3. Select the second field without entering any data. A message box appears which asks you to enter your telephone number.

This script runs when the user exits from the TEL field. The script makes sure that the user enters a telephone number.

Button Object

The Button object has the following events:

- **(Options)** (Provided area for LotusScript options)
- **(Declarations)** (Declare all global variables)

  Note In the above two events you can't write executable LotusScript statements.

- **Initialize** (When it is being loaded)
- **Click** (When it is selected)
- **ObjectExecute** (See note below)
- **Terminate** (When it is being closed)

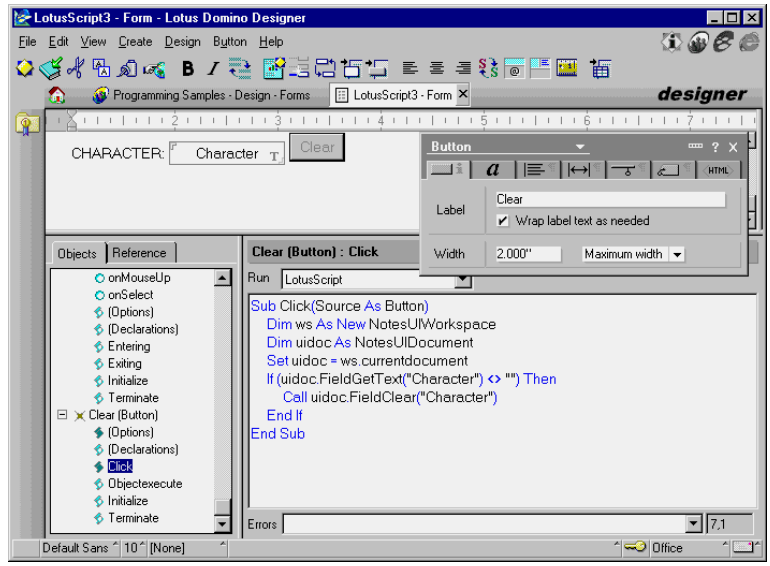
Note The ObjectExecute event is primarily used in external applications and should not be used in the Notes environment.

Example

We will now add an action to a Button object and use the Click event.

Note You should try the following example by creating a temporary database so as not to corrupt any existing databases.

1. Select the database and then the Form design view and click New Form. The form design window is opened and the cursor is blinking at the top left-hand side.
2. Enter CHARACTER: at the cursor blinking position.
3. Choose Create - Field. The InfoBox appears.
4. Enter Character in the Name field in InfoBox.
5. Set the cursor position just to the right side of the Character field.
6. Choose Create - Hotspot - Button. A button is placed on the form, and the InfoBox for the button appears.
7. Inside the InfoBox, enter **`Clear`** in the Button label.



8. Choose Clear (Button) from the Objects view.
9. Edit the sub so that it looks exactly like this:

```
Sub Click(Source As Button )
     Dim ws    As New NotesUIWorkspace
     Dim uidoc As NotesUIDocument
     Set uidoc = ws.CurrentDocument
     If ( uidoc.FieldGetText( "Character" ) <> "" ) Then
          Call uidoc.FieldClear("Character")
     End If
End Sub
```

10. Choose File - Save. You are asked to specify a form name for the new form.
11. Enter LotusScript3 as the form name and click OK.

Running the Example

1. Select Design - Preview in Notes. The new form LotusScript3 appears.
2. Enter some characters in the field, then click Clear. The characters you entered are cleared.

Action Object

The Action object has the following events:

- **(Options)** (Provided area for LotusScript options)
- **(Declarations)** (Declare all global variables)

  Note In the above events you cannot write executable LotusScript statements.

- **Initialize** (When it is being loaded)
- **Click** (When it is selected)
- **ObjectExecute** (See note below)
- **Terminate** (When it is being closed)

Note The ObjectExecute event is primarily used in external applications and should not be used in the Notes environment.

Using LotusScript in Web Applications

Domino also allows you to run your LotusScript code in Web applications, but there are a few limitations. Usually you use LotusScript to develop agents which you will call from the *WebQueryOpen* and *WebQueryClose* events. LotusScript agents can only run on the Domino server, not within the Web browser.

How Scripts and Formulas Are Executed

If your application contains a combination of LotusScript and the formula language, it is useful to know the order in which the events and formulas in a form are executed.

The following example lists the order in which LotusScript events and Domino formulas in a single forms design are executed, during a number of activities. The list was generated by embedding message box commands or @prompt formulas into all the possible events and formulas on a test form containing different field types. The form does not include all the possible field types or evaluation combinations. By studying the results in this example, however, you may be able to better understand the order of execution in the forms of your own application.

The test form contains five fields from top to bottom, in the following order:

- Subject - Editable/Text Field - (with Default Value, Input Translation and Input Validation Formulas).
- From - Computed When Composed/ Authors Name Field - (with Value Formula).
- Counter - Computed/Number Field - (with Value Formula).
- DisplayNum - Computed For Display/Number Field - (with Value Formula).
- Body - Editable/RTF Field - (with Default Value Formula).

The following tables show you different activities, such as composing a document, and the order in which the LotusScript events and Domino formulas are executed for each activity.

Composing a Document

Object	*Formula or Event*
Form	Initialize Event
Form	Window Title
Form	Query Open Event/WebQuery Open 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	Value Formula
Body Field	Initialize Event
Subject Field	Entering Event
Form	PostOpen Event

Saving a Document Using @Command([FileSave]) or File - Save

Object	*Formula or Event*
Form	QuerySave Event
Subject Field	Input Translation Formula
Counter Field	Value Formula
DisplayNum Field	Value Formula
Subject Field	Input Validation Formula

Closing the Window using @Command([FileCloseWindow]) or File - Close

Object	*Formula or Event*
Form	QueryClose Event / WebQuery Close Event
Form	Terminate Event
Subject Field	Terminate Event
From Field	Terminate Event
Counter Field	Terminate Event
DisplayNum Field	Terminate Event
Subject Field	Terminate Event

Reopening the Document in Read Mode

Object	*Formula or Event*
Form	Initialize Event
Form	Window Title Formula
Form	Query Open Event / WebQuery Open 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

Object	*Formula or Event*
Form	QueryModeChange Event
Subject Field	Entering Event (depends on cursor)
Form	PostModeChange Event

Toggling from Edit Mode to Read Mode with Document Open (No Changes)

Object	*Formula or Event*
Form	QueryModeChange Event
Form	PostModeChange Event

Toggling from Edit Mode to Read Mode with Document Open (Saving Changes)

Object	*Formula or Event*
Form	QueryModeChange Event
Same sequence as for saving a document	
Form	PostModeChange Event
Form	QueryClose Event / WebQueryClose Event
Same sequence as for closing a document	
Same sequence as for reopening a document in read mode	

Moving Cursor From One Editable Field to Another

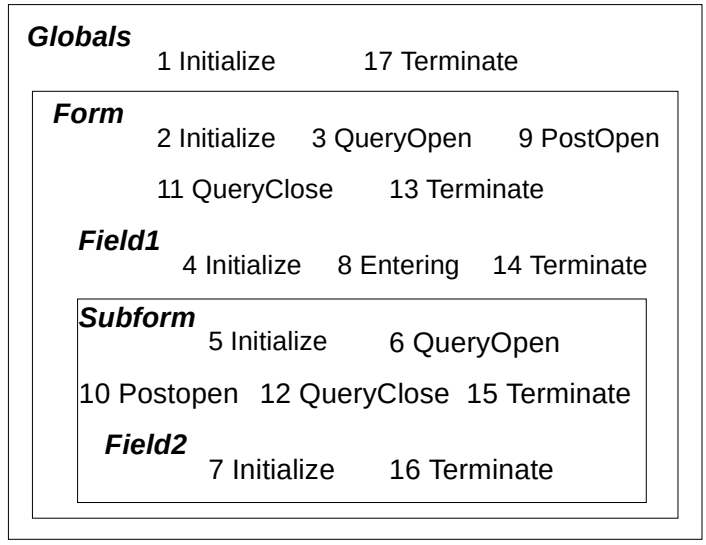
Object	*Formula or Event*
First field	Exiting Event
Second field	Entering Event

Refreshing Fields While in Edit Mode (F9)

Object	*Formula or Event*
Subject Field	Input Translation formula
Counter Field	Value Formula
DisplayNum Field	Value Formula
Subject Field	Input Validation Formula
Form	PostRecalc Event

Sequence of Events in a Complex Example

The following picture shows the sequence of events in a form which contains a subform:



Here is the sequence of events when the document is opened:

1. Initialize of Globals
2. Initialize of Form
3. Queryopen of Form
4. Initialize of Field1 (contained in Form)
5. Initialize of Subform
6. Queryopen of Subform
7. Initialize of Field2 (contained in Subform)
8. Entering of Field1
9. Postopen of Form
10. Postopen of Subform

This is the list of events when the document is closed:

1. Queryclose Form
2. Queryclose Subform
3. Terminate Form
4. Terminate Field1
5. Terminate Subform

6. Terminate Field2
7. Terminate Globals

LotusScript Programming Tips and Considerations

The following section will give you some help in structuring your LotusScript code for event programming within Lotus Domino.

General Suggestions

Do any or all of the following to improve your scripts:

- Declare all variables in the global definitions for an object and use the *Option Public* statement. Next, instantiate the variables in the PostOpen event or in a subroutine that you can call from either the QueryOpen event (for an existing document), or the PostOpen event (for a new document). Your variables will be easier to find and maintain, and you'll be able to use them in any script for the object. You also might consider using Option Declare to make certain that you have declared all the variables in your application.
- Store subroutines and functions in the global definitions for a form or navigator. Then you can use the subroutines or functions with any object on the form or navigator.
- To reuse a segment of script in multiple scripts, put the segment into a function or subroutine or use script libraries (see section on script libraries further on in this chapter).
- Try not to nest subroutine calls or conditionals deeper than three levels. Nesting to too many levels makes scripts hard to follow.
- To debug a script that runs on a shared field, insert the field into a temporary form so that you'll have a place from which to run the debugger.
- In Initialize and Terminate events in forms, fields, actions, and buttons, avoid using the uidoc variable for the NotesUIDocument class. A document object may not be available to access (for example, a document window may not be open) at the time the script runs.

For complete information on LotusScript, see the online Lotus Notes Help information or the *Programmer's Guide*.

Use Consistent Variable Names

The Domino templates use a set of standard variable names, as shown in the table below. For example, in the Domino templates the variable *note* always refers to the current back-end document.

Class Name	*Object Variable*	*Comments*
NotesSession	session	
NotesDatabase	db	
NotesView	view	
NotesViewColumn	column	
NotesDocument	note	Refers to the data associated with the current document.
	parent	The parent of the current document.
	child	A child of the current document.
	profile	A profile document from which you are retrieving processing parameters.
NotesItem	item	
NotesRichTextItem	rtitem	
NotesEmbeddedObject	embobj	
NotesDocumentCollection	documents	
	responses	Use if you are working within one collection of responses to the current document.
	children	An alternative to using the variable name responses. Use if you're using child as the NotesDocument object variable.
NotesDateTime	date1, date2, …	Consider using for comparing dates.
NotesAcl	acl	
NotesAclEntry	aclentry	
NotesAgent	agent	
NotesDbDirectory	dbdir	
NotesLog	log	

continued

Class Name	*Object Variable*	*Comments*
NotesUiWorkSpace	ws	
NotesUiDocument	source	Already an argument to the form events — using this name keeps your scripts consistent.
	uidoc	To use, set uidoc = source in PostOpen. Then you can use this object variable in field and action scripts in the form.

Using these names in your own scripts makes them easy to read and understand, keeps them consistent, helps you maintain them more easily, and may help to share the code with other developers.

Consider using all lowercase for object variables and a combination of lowercase and uppercase, for example VariableName, for other variables.

When passing values to a subroutine or function, use the same variable names in the called routine as in the calling routine. For example, don't call something StatusNumber in one and StatNo in the other. Consistent naming ensures that others can easily read and understand the script.

Reserved Fields

There are some reserved fields in Domino that you can use to automatically add functionality that you need to otherwise program yourself. If you try to use a reserved name differently, or redefine the field, Domino displays an error message.

The following table displays all the reserved fields for mailing documents:

Reserved Field Name	*Description*
MailOptions	Gives users the option of mailing a document.
Sign	Sign creator's name to prevent tampering.
SaveOptions	Controls whether documents are saved when mailed.
Encrypt	Encrypts mail.
SendTo	Sends mail to users listed in field.
CopyTo	Sends copy to users listed in field.
BlindCopyTo	Send blind copy to users which are listed in field.
DeliveryPriority	Delivers mail high, medium, or low priority.
DeliveryReport	Returns a report when mail is delivered to recipient.
ReturnReceipt	Returns a receipt when recipient reads mail.
MailFormat	Routes mail via cc:Mail.

The next table displays all the reserved fields for general use:

Reserved Field Name	*Description*
Categories	Categorizes documents.
$VersionOpt	Controls version tracking for documents.
FolderOptions	Puts new documents in folders.
SecretEncryptionKeys	Encrypts documents with secret, rather than public, encryption keys.

Example

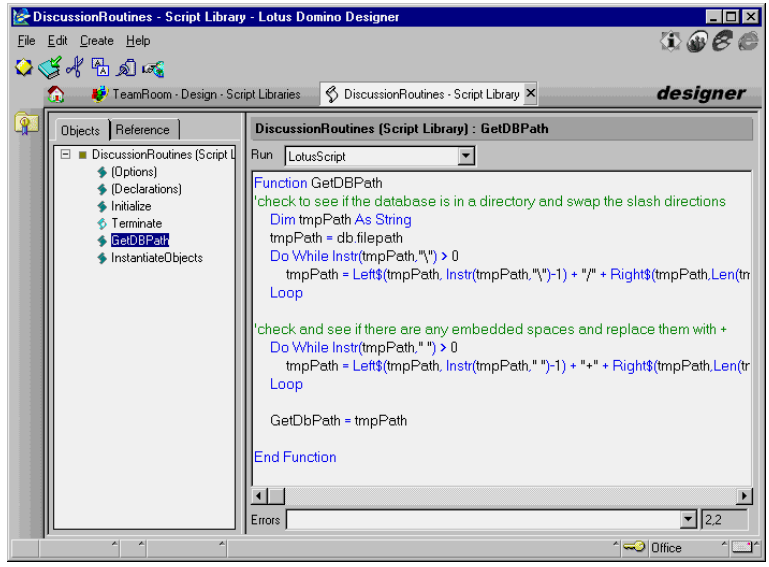
When a document with its contents changed is being closed, a dialog box is presented to the user asking whether the changes should be saved or not. Setting SaveOptions to "0" before closing the document prevents the dialog from being displayed. Thus, the document is closed with all changes discarded.

Using Script Libraries

A script library is a place where you can store code segments that you want to use from other scriptable objects. You may code options, declarations, an initialize subroutine, a terminate subroutine, and user scripts.

To create a new Script Library, choose Resources - Script Library design view in Domino Designer and click New Script Library. The new Script Library is created. You can write the code inside the initialize and terminate subroutine, or you can create your own subroutines. To do so, write a statement such as Function or Sub in an existing script. The editor automatically creates a new script and transfers your code there.

The figure below shows the Discussion Routines script library in the TeamRoom database, which contains GetDbPath and InstatiateObjects subroutines:

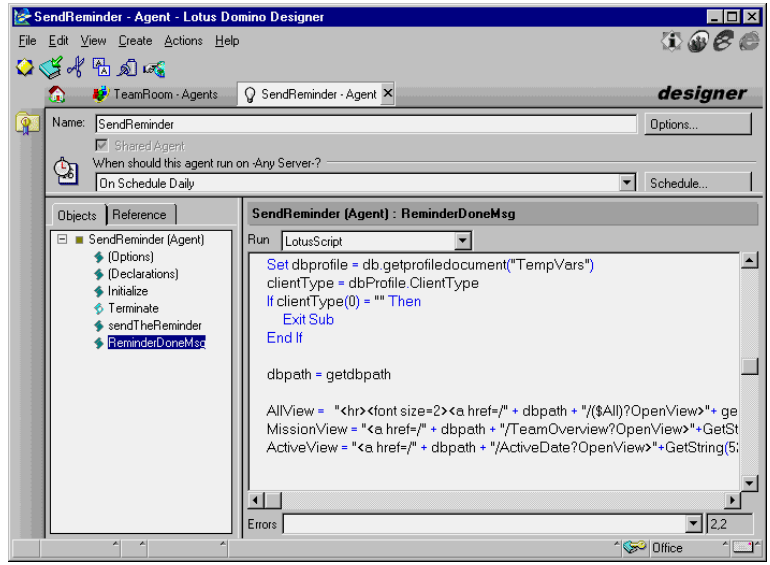


To incorporate a script library into a scriptable object, enter a *Use* statement in the (Options) script for the object or for the (Globals) object. You can see how script libraries are used in the TeamRoom database by opening the SendReminder agent.

```
Use "DiscussionRoutines"
```

The name is not case-sensitive and should not contain spaces. Specify the name as a character literal or named constant.

The picture below shows how the SendReminder agent (ReminderDoneMsg subroutine) uses the GetDBPath routine of the DiscussionRoutines script library:



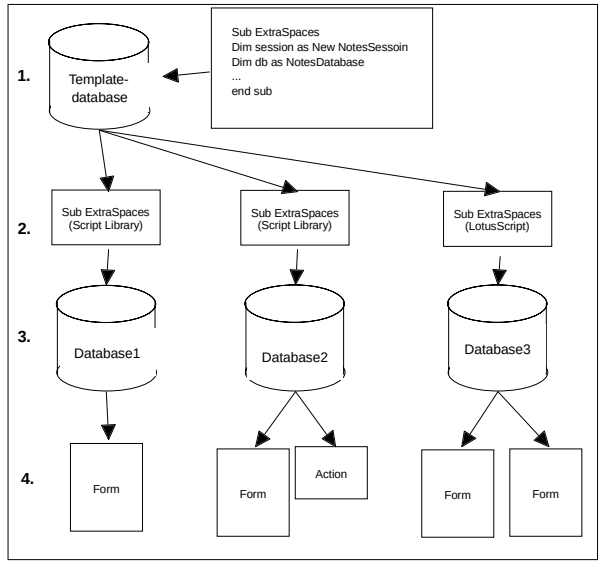
The code in the (Options), (Declarations), Initialize, and Terminate scripts of the library becomes available as though it were in the current objects corresponding scripts. User scripts in the library become available as though they were in the current object.

Using a Template Database

When you are developing a Domino application, you usually use a lot of the same code that you have used before for performing standard procedures. Many different applications contain the same procedures, for example, removing extra spaces in a field, creating a chart in the field, and so on.

It is very useful to use the Template databases where you can store the most commonly used Lotus Script, Java Script and Java code, actions and buttons, and other commonly used functions. It is better than using cut and paste because if you need to change the code, for example, you need to add new Lotus Script code to manipulate a field, you only need to add the code in one place and then the updated code is used in each application that references it.

The next figure shows this process diagramatically:



1. The Template database contains a lot of commonly used code, actions, shared fields, and so on. Sub ExtraSpaces is one Script Library code segment which removes all the extra spaces in the current field.
2. The ExtraSpaces class is used in three databases and it is copied to each database.

   Note When you use the copy and paste options, Domino automatically asks you if you want to inherit the changes from the Template database when the ExtraSpaces Script Library changes.
3. The ExtraSpaces Script Library is now stored in each database.

Updating the ExtraSpaces Subroutine

When you want to update the elements or code segments, you only need to make changes in the Template databases. For example, you need to add some more code into the ExtraSpaces Script Library.

1. Open the Template database and go to the ExtraSpaces code segment.
2. Make the required changes and save the ExtraSpaces Script Library.
3. If you have selected the Inherit options, and the Database1, Database2, and Database3 are on the same server as the Template database, Domino automatically runs the inherit program. The server will refresh the databases overnight. You can also run the refresh commands manually by selecting the database that you want to refresh and then selecting File - Database - Refresh Design.

4. After the refresh, Database1, Database2, and Database3 will have the new updated version of ExtraSpaces Script Library.

Catching Errors at Compile Time

Specifying *Option Declare* in the *(Options)* event of the object forces you to declare variables explicitly. With this option in effect, any undeclared variables will be flagged during compile time. This is useful if you design large applications and it prevents you having to search for typing errors.

Improving Form Performance

A form that performs well is one that Domino can calculate quickly for display, so that documents created with the form are more likely to open quickly.

To improve form performance, do any or all of the following:

- Avoid overusing hide-when formulas on forms. Each formula that Domino must calculate when opening a form slows the application down. Before you use a hide-when formula, try using a computed subform or a hide-when condition, such as "Hide when editing" or "Hide when reading."
- If you must use hide-when formulas to hide buttons on an Action bar, use @Command([RefreshHideFormulas]) or the LotusScript RefreshHideFormulas method in the action formulas or scripts to force calculation of the hide-when formulas. This closely correlates the appearance of different buttons with users' button clicks, and allows each calculation to occur only when needed.
- If a form has keyword fields (for example, in a layout region), and you want formulas to calculate based on changes in those fields (for example, hide-when formulas that progressively disclose items in the layout region) select the "Refresh fields on keyword change" option instead of the "Auto refresh fields" option. Domino performs more calculations when "Auto refresh fields" is enabled; for example, it refreshes all formulas every time a user moves between keyword fields, instead of just when values in keyword fields change.
- Remove infrequently used items from a form. For example, redesign your application to display infrequently used items in a dialog box.
- Consider limiting, or eliminating entirely the use of shared fields or subforms on any form that must open quickly.
- Minimize the number of fields per form, because each field is calculated when a document is opened, refreshed, or saved. After your design is complete, run an agent to remove any blank, unused fields.

- Consider putting field formulas into form events rather than into the fields themselves, so that you can more easily control which formulas are calculated at each event. Don't use hidden fields for processing events unless you have to.

If your application was created in Release 3.x, it may include forms with hidden fields containing formulas that process a document when it's opened or saved. To improve the performance of the application, convert the formulas to LotusScript, and use the PostOpen and QuerySave form events.

When to Use Formulas and LotusScript

In general, formulas are best used for working within the object that the user is currently processing, for example, to return a default value to a field or to determine selection criteria for a view. Scripts are best used for accessing existing objects, for example, to change a value in one document based on values in other documents. Scripts provide some capabilities that formulas do not, such as the ability to manipulate RichText fields. Formulas provide better performance in some situations and may be more convenient for simple applications.

When you're ready to use both, deciding whether to use LotusScript or the Domino formula language for a given task usually depends on the complexity of the task. Consider these questions when making your decision:

- **Do you need to process a quantity of data?**

  A formula that "touches" many databases or documents using @functions must rely on the Notes user interface to access each document, whereas LotusScript accesses the documents more efficiently and quickly.

  For example, LotusScript is a good tool for creating an agent that scans all the databases on your workspace and returns information such as size of database, percent used, number of documents, and so on. LotusScript is also a good tool for running a full-text search on multiple documents and performing an action with the results of the search.

- **Are you using Domino object model front-end or back-end classes?**

  Domino object model (front-end classes) use the same Domino code as their equivalent @commands, so LotusScript won't perform better than the formula language when you use these classes. The database (back-end) classes, however, use different code, and perform more quickly than the equivalent @functions.

  For example, avoid using the front-end class NotesUIDocument to perform many field updates. The back-end class NotesDocument 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, as @Command([EditInsertText]) does.

In addition, the front-end classes will not work in scheduled agents run by a server, only in agents run from a user's workstation (for example, from the menu).

- **Are you using CORBA applets?**

  Domino object model (back-end classes) support the CORBA applets. This is useful if you are planning to develop Web applications which use the CORBA applet to display information to browser users.

- **Do you need to manipulate the currently selected object?**

  Use the formula language instead of LotusScript.

- **Do you need to program buttons on an Action bar?**

  Consider using the formula language instead of LotusScript. Button actions are usually simple and perform tasks usually accomplished directly through the Notes user interface, such as saving or closing a document.

- **Do you need to return the default value to a field?**

  Use the formula language instead of LotusScript.

- **Do you need to return the title of a window?**

  Use the formula language instead of LotusScript.

- **Do you need to control a work flow process from a form?**

  LotusScript is best for controlling workflow with form events, especially the QuerySave event, because it can handle the more complex tasks you may want to accomplish, such as looping and setting multiple variables.

  For example, you can require a user to fill out fields on a form in a predetermined order by manipulating enter and exit field events, or you may prevent a user from opening, saving, or editing a form until certain conditions are met.

- **Are you including too many @functions in one formula?**

  If a formula includes many @functions in sequence, try changing the formula to LotusScript. However, formulas that need only a single @function, such as @Command[FilePrint], are more efficient and perform better than scripts that do the same thing.

- **Do you have to use a lot of if-then-else or for- and while-loops ?**

  Use LotusScript instead of formulas because your code will be easier to maintain. Furthermore, formulas don't support loops.

Using the Evaluate Function to Combine LotusScript and Formulas

Use the Evaluate function in LotusScript to combine pieces of formula language with LotusScript. This allows you to make your scripts leaner wherever @functions do something in fewer lines than LotusScript does. Keep in mind that including formulas in scripts may make the scripts easier to write, but won't necessarily improve performance.

You can use Evaluate to include any @functions except the ones that directly interact with the Notes user interface, such as @Prompt, @DialogBox, @PickList, and @Command. A couple of particularly useful @functions to combine with LotusScript are:

- @Name, which lets you manipulate hierarchical names.
- @Replace, which pulls a value from a text list without requiring the looping that LotusScript would demand.
- @Unique, which removes duplicates from a text list.
- @Subset, which reads the list from left to right.

You can also combine LotusScript and formulas in an application by using them in different parts of the same form.

Evaluate Function in LotusScript

The Evaluate function executes a LotusScript formula.

Syntax:
`Evaluate(`*`macro`* **`[,` *`object`*`])`**

Elements:
macro

Mandatory. A string expression specifying the text of the Notes macro, in the syntax that Domino recognizes. Refer to the Domino documentation for the correct syntax of the macro.

Note If the macro text is in a constant or string literal, Domino needs to do only initial processing of the macro once at compile time, while variable strings incur that processing each time the Evaluate Function is called.

object

Optional. (If the macro requires a Notes object)

Example:

```
result = Evaluate("@Sum(Numlist)", ...)
```

or

```
Const NotesMacro$ = "@Sum(NumList)"
result = Evaluate(NotesMacro$, ...)
```

Return Value
If the macro being run returns a value, the Evaluate function returns a Variant containing that value. Otherwise, the function does not return a value.

Sample Code
This script runs when the user exits from the Subject field and changes the characters to proper case.

```
Sub Exiting (Source As Field)
   Dim ws    As New NotesUIWorkspace
   Dim uidoc As NotesUIDocument
   Dim doc   As NotesDocument
   Dim eval  As Variant
   Set uidoc = ws.CurrentDocument
   Set doc = uidoc.Document
   eval = Evaluate( "@ProperCase(Subject))", doc)
   Call doc.ReplaceItemValue("Subject", eval)
End Sub
```

In this example, we use the *Evaluate* function to get *@ProperCase* carried out. Parameters to the Evaluate function are the string containing the @function and the field name, as well as the object that contains the field.

Tip Since in the above example, the variable *uidoc* is only used to get the object of the next lower class, you may also write *Set doc=ws.CurrentDocument.Document* to initialize variable *doc*.

Making Field Value Changes Effective

There are two ways of making changes to field values effective in your LotusScript programs.

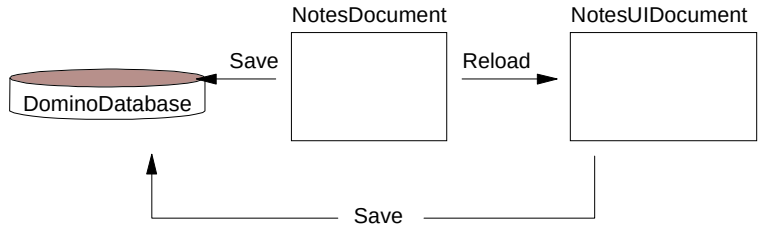
You can use the Refresh method of the NotesDocument class. It has the same effect as using the Refresh key on the Lotus Notes user interface.

When you modify "ReplaceItemValue" or remove "RemoveItem" fields in a document in your LotusScript program, you need to use the Reload method of the NotesDocument class to make the changes effective in the Lotus Notes user interface. The following statements are examples to show the Reload method.

```
Postopen(Source As Notesuidocument)
  note.RemoveItem("Action")
  note.RemoveItem("SaveOptions")
  note.ReplaceItemValue("Action","Approve")
  source.Reload
  .....
```

Note You usually add the following statement at the initialization stage of your program to improve performance, as it stops the screen refreshing each time you update a field. Remember to manually invoke the Reload method in your programs when you use this statement.

```
source.AutoReload = False
```



Using Validation Formulas and QuerySave

If you are using Input Translation and Input Validation formulas along with QuerySave, be sure to do a refresh (source.Refresh) at the beginning of the script for the QuerySave event. The reason for doing this is that the QuerySave event occurs before Notes refreshes the document when saving.

You want QuerySave to have the properly validated data to process. For example, you don't want QuerySave to process an empty field, because a validation formula that would have flagged the field as empty hasn't yet run.

Working With a Rich Text Item and Rich Text Style

The NotesRichTextItem class inherits from NotesItem class. Therefore, you may use GetFirstItem of the NotesDocument class to access the rich text item. Of course, you also may use methods of NotesRichTextItem, for example GetFormattedText. However, when a rich text item is the return value of a method such as GetFirstItem in NotesDocument, do not declare it with a Dim statement as NotesRichTextItem or NotesItem. If you declare it as a NotesRichTextItem object, a "type mismatch" error will occur during runtime. If you declare it as NotesItem object, you cannot use the NotesRichTextItem properties and methods; a "not a member" error occurs at compile time. The solution is to declare it as a variant, and then check to see what type of object is retrieved. If it is a rich text item then you can perform rich text functions on it.

In the following example, you will create a document using the back-end classes and import the first 300 characters of a file stored on your workstation into the document body field. Furthermore, you will set the font for the body field to Courier and the text size to 14 points.

To do this, follow these steps:

1. Create a file, for example Readme.txt, and save it in the root directory of your C-drive.

   Note Create the Readme file with at least 300 characters in it.

2. Create a new Blank Database.
3. Create a form where you want to insert this information.
4. Create a page containing a button (code will be written into the button).
5. Create a view which contains all new documents.
6. Modify the database properties so that the page opens when the database is opened.

Next, create a blank database.

7. From the menu, choose File - Database - New.
8. Give the Title of the Database: Rich Text Style. Give the name of the database: RichStyle.nsf
9. Set the Servers field to Local (where the database is about to be saved) and the template to be a -Blank-.
10. Click OK. The new database is displayed. Select the database in your workspace if it is not already selected, and choose View - Design. The database is now shown in Design mode.

Next, you must create a form where you want to put that information.

1. Go to the Forms Design view and click New Form. A new form is displayed.
2. Name the form Sample1 (You will use that name in LotusScript when you are creating a new document).
3. Type something in the form (don't use the Courier 14 font type).
4. Insert a Rich Text field and name it Body.
5. Save the form.

Next, create a page to contain a button.

1. Go to the Page design view and click New Page: the new page is displayed.
2. Create a button, open the InfoBox if it is not open, and enter a label for the button: "Create Document".
3. Go to the Objects view and select the click event.

   Now, add the following code in the click event:

```
Sub Click(Source As Button)
'**First we read and import the contents of
Readme.txt-file, first 300 characters
        '**Freefile() returns an unused file number
        fileNum% = Freefile()
        '**Opens a readme-file
        Open "C:\Readme.txt" For Input As fileNum%
        '**When file is opened we read first 300 characters
in txt-temporary variable
        txt$ = Input(300, fileNum%)
'**Second code declares the Notes session and RichTextStyle
        Dim session As New NotesSession
        Dim richStyle As NotesRichTextStyle
        Set richStyle = session.CreateRichTextStyle
        '**Document uses the following styles: Courier, 14
        richStyle.NotesFont = FONT_COURIER
        richStyle.FontSize = 14
'**Third code declares the database and create a new
document.
        Dim db As NotesDatabase
        Set db = session.CurrentDatabase
        Set doc = db.CreateDocument
        '**Give the name for the form
        doc.Form = "Sample1"
        doc.Creator = session.CommonUserName
        '**Declares a new Rich Text Item named Body
        Set notesRichTextItem = New NotesRichTextItem( doc ,
"Body" )
        '**RichTextItem rendered using that style's
attributes until another style is appended.
        Call notesRichTextItem.AppendStyle( richStyle )
        '**Set the txt$ value in the RichTextitem
        Call notesRichTextItem.AppendText( txt$ )
        Call doc.Save(True, False)
        '**Message box tells that document is saved
        Messagebox("The Sample Document is created")
End Sub
```

4. Save the page.

Next, create a view which contains all the new documents that you are about to create.

1. Go to the Views design view and double-click the (untitled) view.
2. Open the views InfoBox and give the view a name, for example, ViewSamplel.
3. Next, go to the first column and select the Creator in the field on the Programmer's Pane.
4. Save the View.

Next you must modify the database InfoBox launch tab. You want the page to open when a user opens the database. To do this, follow these steps:

1. Open the database InfoBox and click the Launch tab.
2. On the database open field, select Open Designated Navigator.
3. Type of Navigator field select Page and then in the Name field select the PageSample1.
4. Close the InfoBox.

You have now created a small application, which contains a page, button, view, and form.

Note The drawback of using the NotesRichTextStyle class with existing rich text fields is that updates to such fields are not visible to the user immediately. In order to see the rich text that has been added, the user must first save the document, then quit out of the document and then reopen it.

Running the Sample

To run the sample all you need to do is open the database. When you do that, the page you created opens and the view will be displayed on the right side of the page (ViewSample1).

Click the button to start your LotusScript code, which displays a message box to tell you that the application has created a document.

Close the message box and press **F9** to refresh the view, and you will see the new document.

Double-click the document and you will see that the Body field now contains the text of the readme file and the style is Courier 14.

New Domino Objects in Domino R5.0

The next table summarizes the new LotusScript libraries and constants in Domino R5.0:

Domino Objects	*Type*	*Libraries/Constants*
LIBRARIES		
NotesSession	properties	NotesBuildVersion
	methods	CreateRichTextParagraphStyle
NotesDocument	properties	FolderReferences IsDeleted
NotesDatabase	properties	FolderReferencesEnabled MaxSize ReplicationInfo
	methods	InitIMAP4Mail GetFirstProfileDoc GetNextProfileDoc
NotesView	properties	IsHierarchical IsConflict RowLines HeaderLines Spacing IsModified BackgroundColor IsCategorized ColumnCount TopLevelEntryCount GetAllEntries
	methods	GetFirstEntry GetLastEntry GetNextEntry GetPrevEntry GetNthEntry GetNextSiblingEntry GetPrevSiblingEntry GetParentEntry GetChildEntry GetEntryByKey GetAllEntriesByKey

continued

Domino Objects	*Type*	*Libraries/Constants*
NotesViewColumn	properties	DataType ListSep Alignment HeaderAlignment IsSortDescending IsHideDetail IsIcon IsResize IsResortAscending IsResortDescending IsShowTwistie IsResortToView IsSecondaryResort
IsSecondaryResortDescending		IsCaseInsensitiveSort IsAccentInsensitiveSort FontFace FontStyle FontColor FontPointSize NumberDigits NumberFormat NumberAttrib DateFmt TimeFmt TimeZoneFmt TimeDateFmt Flags1 Flags2 IsField IsFormula
NotesAgent	properties	Trigger Target IsNotesAgent IsWebAgent
NotesDocumentCollection	methods	AddDocument DeleteDocument GetDocumentByID Contains
NotesACL	properties	InternetLevel

continued

Domino Objects	*Type*	*Libraries/Constants*
NotesACLEntry	properties	UserType Flags CanCreateLSOrJavaAgent IsServer IsPerson IsGroup CanCreateSharedFolder IsAdminReaderAuthor IsAdminServer
NotesRichTextItem	methods	AppendParagraphStyle AddPageBreak
NotesRichTextStyle	properties	PassThruHTML
NotesRichTextParagraphStyle	properties	Alignment InterLineSpacing SpacingAbove SpacingBelow LeftMargin RightMargin FirstLineLeftMargin Pagination Tabs
	methods	SetTab ClearAllTabs SetTabs
NotesRichTextTab	properties	Position Type Clear
NotesViewEntry	properties	IsDocument IsCategory IsTotal IsConflict IsUnread SiblingCount DescendantCount IndentalLevel FTSearchScore noteID UniversalID ColumnValues Document Parent
	method	GetPosition

continued

Domino Objects	*Type*	*Libraries/Constants*
NotesViewEntryCollection	properties	Count IsSorted Query Parent
	method	GetFirstEntry GetLastEntry GetNextEntry GetPrevEntry GetNthEntry RemoveAll UpdateAll StampAll PutAllInFolder RemoveAllFromFolder FTSearch AddEntry DeleteEntry GetEntryByID Contains
NotesSetup	properties	KnownNetworkPorts ModemList TCPHostName TCPDomainName TCPFullName ProgressTitle ProgressEnabled ProgressLevel StatusMessage ErrorMessage ErrorPath
	method	New ExecuteSetup

continued

Domino Objects	*Type*	*Libraries/Constants*
NotesReplication	properties	CutoffInterval CutoffDate Disabled IgnoreDeletes HideDesign DoNotCatalog CutoffDelete NeverReplicate Abstract DoNotBrowse NoChronos IgnoreDestDeletes MultiDbIndex Priority
CONSTANTS		
NotesRegistration	constants	NOTES_LIMITED_CLIENT NOTES_DESKTOP_CLIENT NOTES_FULL_CLIENT
NotesViewColumn	constants	VC_DTNUMBER VC_DTTIMEDATE VC_DTTEXT VC_ALIGN_LEFT VC_ALIGN_RIGHT VC_SEP_SPACE VC_SEP_COMMA

continued

Domino Objects	*Type*	*Libraries/Constants*
NotesAgent	constants	TRIGGER_NONE TRIGGER_SCHEDULED TRIGGER_NEW_MAIL TRIGGER_DOC_PASTED TRIGGER_MANUAL TRIGGER_DOC_UPDATE TRIGGER_SYNCHRNOUS_NEW_MAIL TARGET_NONE TARGET_ALL_DOCS TARGET_NEW_DOCS TARGET_NEW_OR_MODIFIED_DOCS TARGET_SELECTED_DOCS TARGET_ALL_DOCS_IN_VIEW TARGET_UNREAD_DOCS_IN_VIEW TARGET_PROMPT_USER TARGET_UI_SELECTABLE_OBJECT
NotesRichTextParagraphStyle	constants	ALIGN_LEFT ALIGN_RIGHT ALIGN_CENTER ALIGN_FULL ALIGN_NOWRAP SPACING_ONE_POINT_50 SPACING_DOUBLE PAGINATE_DEFAULT PAGINATE_BEFORE PAGINATE_KEEP_TOGETHER PAGINATE_KEEP_WITH_NEXT TAB_LEFT TAB_DECIMAL TAB_CENTER TAB_RIGHT
NotesReplication	constants	DB_REPLICATION_PRIORITY_LOW DB_REPLICATION_PRIORITY_MED DB_REPLICATION_PRIORITY_HIGH DB_REPLICATION_PRIORITY_NOTSET

Error Handling

Ideally, you would not need to write anything to handle run-time errors; however, some errors may occur at run-time, such as running out of disk space or dividing by zero, causing the script to stop unexpectedly. To avoid this situation, you can include error-handling procedures in your script.

Using On Error and Resume Statements

By using On Error and Resume statements in your script, you can handle run-time errors that may occur. These statements are built-in functions provided by LotusScript. The script needs the following steps to handle the error:

1. Trap the error using an On Error statement and specify where to go to handle the error.

   For example, if the error occurs, you can go to the label ERRORPROC.

```
     Dim x As Integer, y As Integer, z As Integer
     x = 3
     y = 0
     On Error GoTo ERRORPROC
     z = x/y
Exit Sub
ERRORPROC:
```

2. Script the error-handling process. For example, at the ERRORPROC: label.

```
ERRORPROC:
    MessageBox("Divide error")
    y = CInt( InputBox("Enter new number") )
```

3. Complete the error-handling process using a Resume statement to go back to the statement where the error occurred.

```
     Dim x As Integer, y As Integer, z As Integer
     x = 3
     y = 0
     On Error GoTo ERRORPROC
     z = x/y
Exit Sub
ERRORPROC:
     MessageBox("Divide error")
     y = CInt( InputBox("Enter new number") )
     Resume
```

Creating an Error Handler for Debugging

It is useful to have an error handler to help debug your programs, as the LotusScript debugger ends when errors occur. To prevent this from happening, you can create an error handler like this:

```
On Error Goto ErrorHandler
ErrorHandler:
  Messagebox "Error:" & Error(Err), 0+64, "Error!!"
  Print "Error No. : " Err
  Print "Description : " Error(Err)
  Print "Line No. : " Erl
  Resume Next
  Exit Sub
```

Note If you include the constant definition file (%Include "LSCONST.LSS", you can use constant symbols (MB_OK, MB_ICONINFORMATION and so on) instead of values 0 and 64 in the Message box statement.

If you want to catch all errors in your programs, you need to write the above error handler in all event routines that you describe (for example, "initialize," "postopen" and so on), but not in the subroutines. Once you have written an error handler in a specific event routine, it can be referred to in the subsequent subroutines.

Using the Debugger

While you are writing scripts, you will find some errors which will require fixing. Domino recognizes two kinds of LotusScript errors: compile errors and run-time errors.

Compilation of your script takes place when you save it. Compile errors are reported and the script cannot be saved. Because Domino does not allow you to save your script with compile errors, you need to correct all the compile errors first.

Tip If you need to save your LotusScript program without correcting the compile errors at that time, you can exclude the statements with compile errors using the %REM and %ENDREM statements, as shown in the following example:

```
%REM
"Your program with errors"
%ENDREM
```

A run-time error is an error which cannot be detected during compilation. Run-time errors are found while Domino is running the script, causing the program to stop. The script may have the correct syntax, but, for example, division by zero is not allowed. Here is a simple example of this run-time error:

```
Dim x As Integer
Dim y As Integer
Dim z As Integer
x = 5
y = 0
z = x / y
```

During execution of the above code, LotusScript will stop and issue an error message because dividing 5 by 0 is not a valid operation.

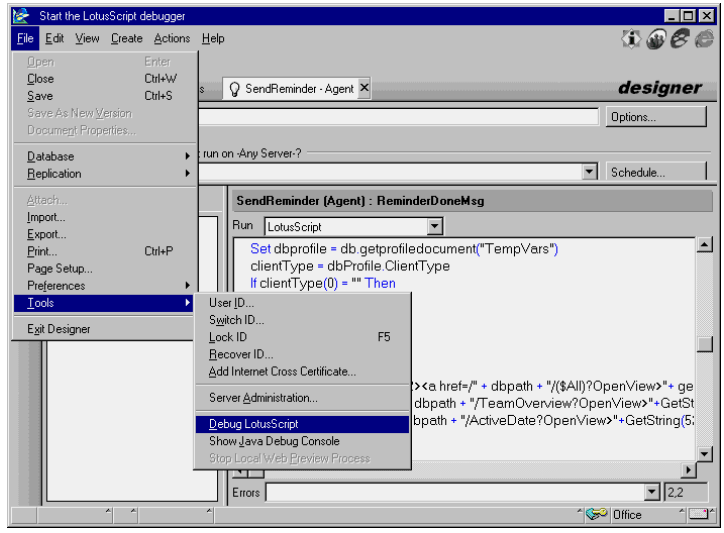
There is one more error type, which is a logical error. You might be able to run your script without errors, but the result is not as intended.

The Debugger helps you to detect run-time errors and logical errors.

How to Enable the Debugger

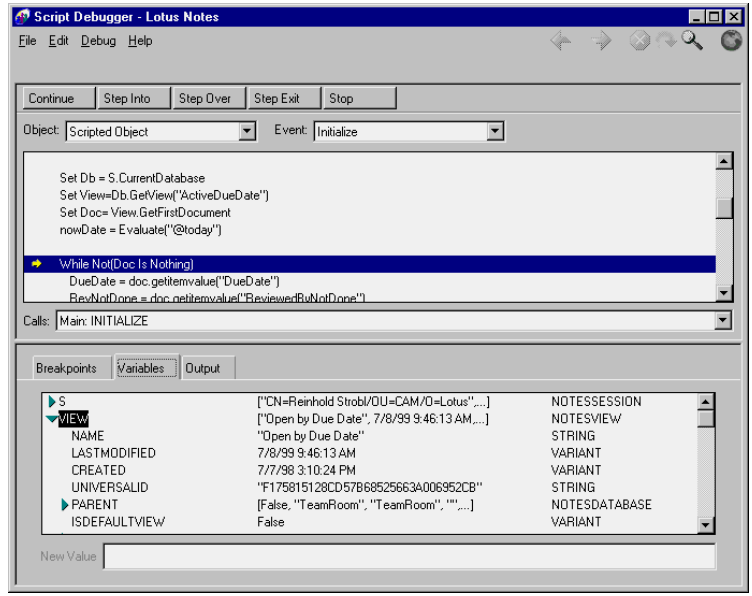
It is easy to enable debug mode. Before running your script, do the following:

1. Choose File - Tools - Debug LotusScript.
2. To check if the Debugger is enabled, choose File - Tools. If the Debugger is on, there is a checkmark next to the menu option, as shown in the following figure.



If you click the Debug LotusScript menu again, debug mode is disabled.

If the Debugger is enabled when you start running any LotusScript, the debugger is launched and the script stops at the first line. The debugger is shown as follows:



In this example, the script is in interrupt mode.

When you run a script in debug mode, the script shows one of three states:

- When a script is interrupted at a breakpoint, the debugger has control.
- When a script is stepping, control passes to the script and then back to the debugger after a single statement in the script is performed.
- When a script is continuing, it runs uninterrupted until a breakpoint or the end of the code is reached.

While the script is in interrupt mode, you can do one of the following:

- Inspect the script.
- Inspect the value of variables and properties.
- Control which is the next statement that will be performed.
- Inspect other defined objects, events and the scripts related to them.

You can control which statement is the next to be performed in interrupt mode. Click:

- Continue

  To continue until a break point is reached.

- Step Into

  To perform the current statement and step to the next statement.

- Step Over

  To perform the current statement and step to the next statement, stepping over the subprogram if the current statement calls a subprogram.

- Step Exit

  To continue executing the current subprogram and stop in the subprogram that called it at the line following the call.

Making Breakpoints

If you find a run-time or logical error, inspect your script and make breakpoints at the statement (or around it) where you suspect the error is occurring. You can then run your script, and it will stop at the breakpoint. In interrupt mode, you can inspect the value of important variables and properties.

One-Step Execution

During one-step execution, only the current statement is performed before the code is stopped. You can then inspect the values of variables or properties before and after performing the statement.

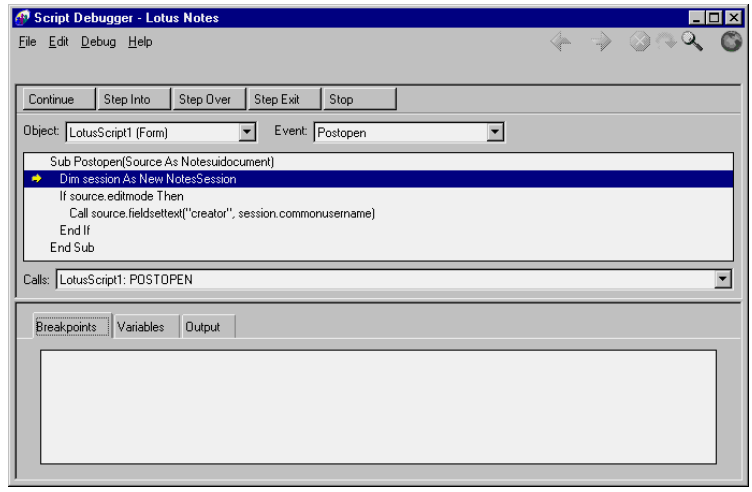
Instance Inspection

1. Click the Variables tab in the bottom pane to access the variables window. The variables defined for the procedure appear in a three-column display, showing the name, data type, and value of each variable.
2. To view array or type members, click the arrow to the left of the variable name.

A Simple Example

To show you how to use the debugger, we will take the database we used earlier as an example for the PostOpen event.

1. Choose File - Tools - Debug LotusScript to enable debug mode.
2. Open the database and create a new document. The debugger window will be displayed. The execution of the script stops at the first statement.

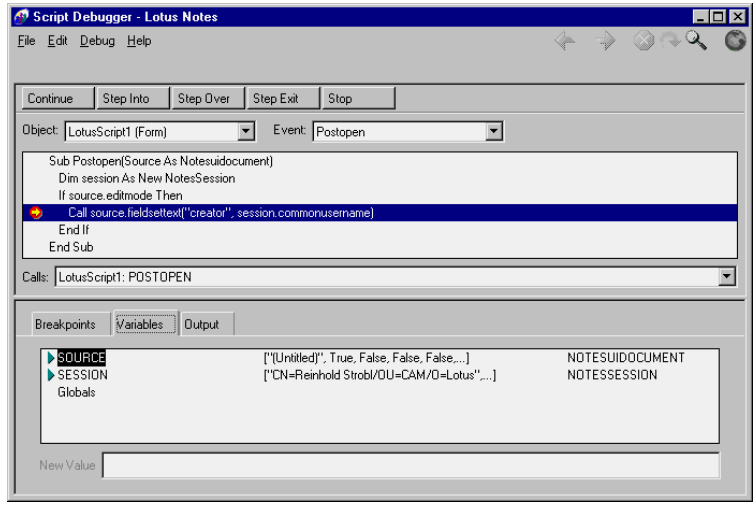


   The script added to the form object has been launched by the Postopen event, and the execution stops at the marked sentence.

   Go through the debugger.

3. Double-click or click on the statement Call Source.FieldSetText.… and press **F9** in the upper pane. This creates a breakpoint.
4. Click the Variables tab. In this pane you can see instances or variables.
5. Click the green triangle next to Source in the bottom pane. You can see properties of the Source instance, which is of type NotesUIDocument. This class represents the document that is currently open in the Notes workspace.
6. You can see that the variable session does not yet have values.

7. Click the Continue action button.



8. The script runs and stops at the breakpoint that you made. The session variable now has a value.

   Note For objects containing a data structure, the values of the data items are also shown on the top level.

9. Click Continue to run the script to its end. This will then close the debugger.

This very simple example shows how easy it is to control the execution flow of the program, and to inspect variables.

Tracing Programs Without a Debugger

There are several ways to trace programs without a debugger, though you will need to add some statements into the programs to use them. For example, you can use the PRINT and MESSAGEBOX statements to look at variables in your programs.

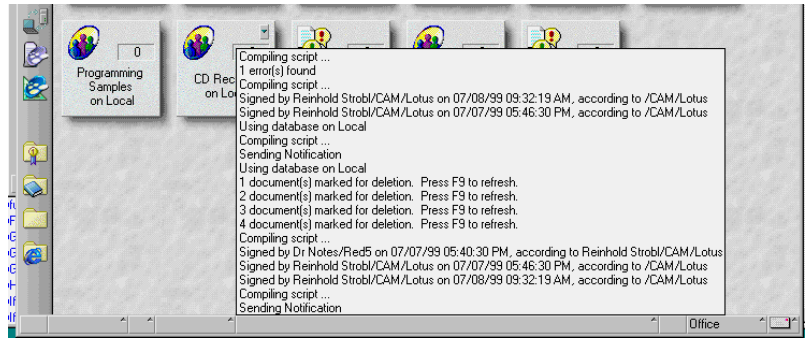
PRINT Statement

The Print statement displays constant values, and the contents of variables on the status line at the bottom of the Notes interface.

```
Print "Sending Notification"
```

This statement results in the following:

When you click the status line, you will see the following message list box, which contains the Print message history.



To clear the status line, simply issue the Print statement with no arguments. This clears the status line. However, you can still click the cleared area to display the message box.

You can also see the messages created by the Print statement by clicking the Output button when using the Lotus Notes debugger:

Messagebox Statement

The Messagebox statement displays a dialog box with some buttons to show messages.

```
%INCLUDE "LSCONST.LSS"
Dim twoLiner As String
twoLiner = |This message
is on two lines|
MessageBox twoLiner, MB_OKCANCEL, "Demo"
```

The following message box is displayed:

Note The vertical bar (|) is the string delimiter for multi-line strings.

Using JavaScript

Before Domino R5.0, the main use of JavaScript within Domino was to modify the standard behavior of Web pages by adding some "client side" functions. Client side means that no "server side" activity is requested by JavaScript functions, so that their result is presented quickly to the end user. With Domino R5.0, you can use JavaScript to write applications which will support both the Notes client and the Web browser. All events associated with an object are programmable, using JavaScript, LotusScript or even simple @functions, and can be easily accessed within the Programmer's Pane.

JavaScript allows you to handle events such as *onLoad* (for a Web page), *onClick* (for an input button on form), *onChange*, *onBlur*, *onFocus* (for input fields), and so on. You can use these events to trigger JavaScript functions that can also perform some complex operations. In a Web browser, JavaScript functions can access all elements on a Web page (like input fields), as well as properties and methods that control the status and the behavior of the Web browser window itself. Adding JavaScript to Domino forms and fields is particularly useful, as it allows you to create forms with a more dynamic behavior without adding workload to the Domino server. For example, with JavaScript, field values can be validated locally on the browser, instead of on the Domino server, after submitting.

Using JavaScript in Domino Design Elements

To use JavaScript in your application, you add JavaScript code to events as you do with LotusScript. The table below shows some of the supported JavaScript events for forms and pages:

Notes Form Event Handlers	*Description*
onLoad	Similar to PostOpen event.
onUnLoad	Before document is cleared.
onSubmit	Window event, before Document saved.
onReset	Window event, before Document reset.
onBlur	When the forms loses focus.
onClick	An object on a form is clicked.
onDblClick	The user double-clicks a form element or a link.
onFocus	The form receives focus.
onHelp	Triggered when the user presses the **F1-HELP** key.
onKeyDown	The user presses a key down.
onKeyPress	The user presses or holds down a key.
onKeyUp	The user releases a key.
onMouseDown	The user presses a mouse button down.
onMouseMove	The user moves the cursor.
onMouseOut	The cursor leaves the form or page.
onMouseOver	The cursor moves over the form or page.
onMouseUp	The user releases a mouse button.

For form elements, for example, fields, you can provide JavaScript for the following events:

Notes Form Element Event Handlers	*Description*
onFocus	Entering the object.
onBlur	Exiting the object.
onChange	When object is changed.
onClick	When object is selected.

If you want to add JavaScript code for other Window events that are not handled by the Notes client (such as onFocus, onResize, or onMove), you can do this in the HTML Body Attributes object for the form. Likewise, if you want to add an event for a form element, you can do it in the HTML Body Attributes Field event.

Code in pass-thru HTML and the HTML body attributes fields are passed to the browser, but ignored in the Notes client.

Note JavaScript must be enabled in the User Preferences in order to be executed by the client.

Caution If you enter JavaScript code into a formula, keep the following rules in mind:

1. Within the text string that you are going to put in the formula, every double quote (“), single quote (‘), and backslash (\) must be preceded by a backslash (\).

   For example:

   ```
   <a href="http://www.ibm.com"> click here </a>
   ```

   must become :

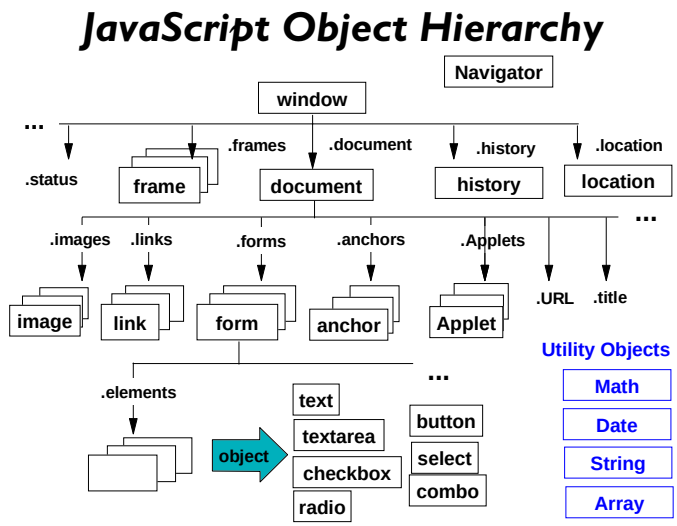
   ```
   <a href=\"http://www.ibm.com\"> click here </a>
   ```

2. The same text string must be included between two double quotes before pasting it into the formula. For example:

   ```
   "<a href=\"http://www.ibm.com\"> click here </a>"
   ```

The Browser Javascript Object Hierarchy

The Javascript Object Hierarchy exposed in a browser looks like this:



Note For JavaScript programs running on the Notes Client some of the object relationships illustrated above do not make sense. For example, there is only one form attached to a document at any one time. Furthermore, not all objects are available on the Notes client.

Where JavaScript Gives You Access in Domino

With JavaScript, you can use the three events *onFocus*, *onBlur*, and *onClick* to perform field validations, as well as for buttons.

It also gives you a common language between the Notes client and a browser that enables applications to be coded only once for both environments.

Mapping Domino Objects to the Document Object Model

This section explains how Domino objects are mapped to the Document Object Model as it is used in Web browsers.

The following list shows for each Domino object:

- Its type in the Document Object Model
- The HTML that is generated by Domino
- A JavaScript example using the object

Text Field

DOM type	Text
HTML	<INPUT TYPE="text" NAME=... ...>
Example	document.forms[0].textField.value = "default";

Date/Time Field

DOM type	Text
HTML	<INPUT TYPE="text" NAME=... ...>
Example	document.forms[0].dateField.value = "1999/01/01";

Number Field

DOM type	Text
HTML	<INPUT TYPE="text" NAME=... ...>
Example	document.forms[0].counter.value = parseInt(document.forms[0]. counter.value) + 1);

Rich Text Field

DOM type	TextArea or Applet
HTML	<TEXTAREA NAME=... ...> or <APPLET NAME=... ...>
Example	If the rich text field is shipped to a browser as HTML, or is used within the Notes Client, the field is treated as a TextArea. The value of a TextArea is a Unicode string. To clear out a field named "richText": document.forms[0].richText.value = ""; If the rich text field is shipped to a browser as an applet, the <APPLET> tag is used. To set a field named "richText" to be "Please don't!": document.applets.richText.setText("text/html", "Please <B>don't!<B>");

Authors/Names/Readers Field

DOM type	Text
HTML	<INPUT TYPE="text" NAME=... ...>
Example	In Notes, extra "helper" buttons will appear to aid in inputting a value. These helper buttons aren't accessible to JavaScript.

Password

DOM type	Password
HTML	<INPUT TYPE="password" NAME=... ...>
Example	var passwd = document.forms[0].passwordField.value;

Formula Field

DOM type	Text
HTML	<INPUT TYPE="text" NAME=... ...>
Example	document.forms[0].formulaField.value = "Select @All"

Dialog List, Listbox, Combobox Field

DOM type	Select
HTML	<SELECT><OPTION ...> ... </SELECT>

Example	In Notes, extra "helper" buttons will appear to aid in inputting a value. These helper buttons aren't accessible to JavaScript. To find out which keyword has been selected from a (single valued) keyword list associated with a field named "Grade":
	var list = document.forms[0].Grade;
	alert(list.options[list.selectedIndex].text);

Checkbox Field

DOM type	Checkbox
HTML	<INPUT TYPE="checkbox" NAME=... ...>

Example	To show each choice selected in a field named "checkboxField":
	var boxes = document.forms[0].checkboxField;
	for (var i = 0; i < boxes.length; i++) {
	  var box = boxes[i];
	  if (box.checked) alert(box.value + " is checked");
	}

Radio Button Field

DOM type	Radio
HTML	<INPUT TYPE="radio" NAME=... ...>

Example	To display which radio button from the field "radioField" is checked:
	var result = "--no result--";
	var buttons = document.forms[0].radioField;
	for (var i = 0; i < buttons.length; i++) {
	  var button = buttons[i];
	  if (button.checked) {
	  result = button.value;
	  break;
	  }
	}
	alert(result);

Action

DOM type	Link
HTML	<A NAME=... onClick=... ...>
Example	If an action bar is emitted as an applet to a Web browser, the actions are inaccessible to JavaScript. To programmatically fire the first action on a page: document.links[0].click();

Computed Text

DOM type	Span
HTML	<SPAN ID=...> ... </SPAN>
Example	Accessed as a "Span" in a browser but is inaccessible within Notes. To change the value of computed text field named "cText": document.all.cText.innerHTML = "new value";

Link Hotspot

DOM type	Link
HTML	<A HREF=... NAME=...>
Example	To change where a link named "launch" will link to, if clicked: document.links.launch.href = "http://www.lotus.com";

Text Pop-up Hotspot

DOM type	not accessible
HTML	inline text
Example	n/a

Button Hotspot

DOM type	Button
HTML	<INPUT TYPE="button" NAME=... ...>
Example	To change the label of a button named "pushMe": document.forms[0].pushMe.value = "Go ahead. Push me!";

Formula Pop-up Hotspot

DOM type	not accessible
HTML	inline text
Example	n/a

Action Hotspot

DOM type	Link
HTML	<A HREF=... NAME=... onClick=...>
Example	To programmatically perform the action (as determined by the onClick handler) behind the first hotspot on a page: document.links[0].click();

Picture, Image Resource

DOM type	Image
HTML	<IMG NAME=... ...>
Example	To change the image associated with a picture named "effects": document.images.effects.src = "/db.nsf/RollOver?OpenImageResource";

Picture, Image Resource Hotspot

DOM type	Area
HTML	<AREA NAME=... ...>
Example	To programmatically leap to the link associated with a hotspot named "hotSpot": document.images.hotSpot.click();

Java Applet

DOM type	Applet
HTML	<APPLET NAME=... ...>
Example	To call a public method named "setDate" of an applet named "DatePicker": document.applets.DatePicker.setDate(...supply parameters...);

Form	
DOM type	Window and Form
HTML	<HTML> ... <BODY onLoad=... onUnload=...> <FORM NAME=... ... onSubmit=... onReset=...> ... </FORM> </BODY> </HTML>
Example	Domino's concept of a form maps onto both a Window and a Form in the DOM world. The onSubmit and onReset handlers are accessed through the Form DOM object. All other handlers are hung on the BODY tag in HTML and accessed via the Window DOM object. To programmatically submit a form from a button named "submitButton", add the following code to submitButton's onClick handler: this.form.submit();

Note In a browser, when a document is rendered in read mode, all fields are generated as inline text and are inaccessible. In Notes, fields in read mode are accessed exactly as they would be in edit mode: document.forms[0].fieldName. Though you are free to change their values, these new values aren't stored in the back-end document.
Furthermore, in a browser computed fields are generated as inline text and are generally inaccessible.

Examples of Adding JavaScript to Forms

This section lists some examples of JavaScript.

Using a JavaScript Library

If you have a large JavaScript library of functions, it might be better to reference a single document that stores the library, rather than copying it into each form or design element that requires the functions. In order to do this, place the following HTML on a form:

```
<SCRIPT LANGUAGE="JavaScript"
       SRC="/dbName/JSview/Module1/$file/module1.js">
```

Note In this example we have assumed that we have a view (*JSView*) that collects the document with attachments that are JavaScript files. We access a document using a key value (*Module1*), and so we reach its attached file (*module1*).js. The Domino server will recognize the .js files only if the HTTPD.cnf file contains the following:

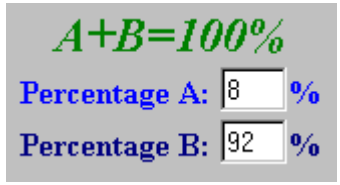
```
AddType .js application/x-javascript binary 1.0 #JavaScript
```

Caution This only works for a Web browser.

The next section contains some examples of using JavaScript in forms and fields. Each of the following examples shows some effects that you can implement using JavaScript.

Example 1: Auto-Refresh, Field Validation, and Help Fields

The form in the figure below allows a user to insert a percentage value in one of the two fields, and the other field is then computed as a complementary percentage. For example, if a user inserts 70 in A, he or she will see 30 appearing in B immediately after changing the focus. If a number greater than 100 is entered, a JavaScript alert message is displayed. Also, when the user puts the mouse inside a field, a help message is displayed on the bottom bar of the Web browser.



Note All validation operations are performed locally, without calling any server tasks.

To create this form, do the following:

1. Create two Numeric Editable fields, *PercentA* and *PercentB*.
2. In the *onBlur* event of *PercentA*, type:

```
if(this.form.PercentA.value<=100)
{
this.form.PercentB.value=100 - this.form.PercentA.value;
}
else
{
  alert('Invalid Percentage A!');
};
window.status='';  "
```

3. In the *onFocus* event of *PercentA* enter:

```
window.status=
  'Insert Percentage A and look to Percentage B'
```

   onBlur is an event triggered each time the focus is moved from a field. Here, it is used to trigger the validation and auto-refresh "procedure" (if()...else block) and to set the message bar to blank (window.status= ""). Fields accessed in these procedures are those on the Web form, so you cannot access hidden or computed Domino fields using JavaScript, because these are not fields on the Web page.

The *onFocus* event occurs when a user enters an input field, and here it is used to set the status bar message in order to use it as a help field.

4. In the *onBlur* event of PercentB write:

```
if(this.form.PercentB.value<=100)
{
this.form.PercentA.value=100 - this.form.PercentB.value;
}
else
{
  alert('Invalid Percentage B!');
};
window.status='';
```

5. In the *onFocus* event of PercentB write:

```
window.status=
  'Insert Percentage B and look to Percentage A'
```

   This JavaScript code mirrors that in step 2, and it doesn't need further explanation.

An alternative to *onBlur* is to use *onChange* as a triggering event for the validation. This event occurs only when the value of a field is different from the previous value.

Note There are many solutions for the problem solved by this example. Another possible solution is to store a unique JavaScript function for the validation of the entire form in the header of the Web page, instead of doing the validation inside each field, and then to call it using the following syntax, for an event such as *onChange* or *onBlur*:

```
onBlur="validateForm()"
```

In the same way, you can call a procedure defined in the header of the form that refreshes fields on that form.

If you prefer to use the validation and translation formulas of the Domino form, you can use JavaScript to force the execution of these formulas every time a value is changed in a field. Do the following:

- Put the following in the *onChange* event of a triggering field: **this.form.submit()**
- Put this formula in the **$$Return** field in order to return immediately to the form after "refreshing":

```
"[/"+@Subset(@DbName;-1) + "/$defaultView/"+
@Text(@DocumentUniqueID)+"?EditDocument]"
```

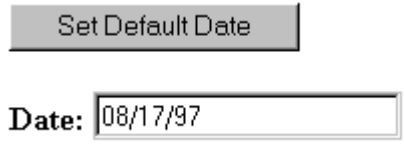
In this way, you can also have hide-when formulas working every time the value changes in that field.

Keep in mind that this is only a simulation of a field refresh, since the document is actually saved.

To avoid this, you can use a button to perform the formula: *@Command([ViewRefreshFields]).* In this way, the form will refresh without the document being saved, but the user must click the button to see the results.

Example 2: Setting Field Values

The field in the figure below can be reset to today's date, by clicking the button. Though this example is very simple, it illustrates a function which could be very useful — having a button that resets all the fields on a form to their original values. Again, this action is performed without calling the server.



To create this sample, do the following:

1. Create a button on the form and add the following to the *onClick* event:

   ```
   this.form.Date.value = this.form.DefaultDate.value;
   ```

   The action behind the button assigns the value of the "DefaultDate" field to the "Date" field.
2. Create an Editable Date field and call it *"Date"*.
3. Create an Editable Date field and call it *"DefaultDate"*, and put in its default formula: *@Today*.
4. In *HTML Attributes* of "DefaultDate" write:

   ```
   "TYPE=\"Hidden\""
   ```

Caution This field will not appear on the Web but it is defined on the Web form, so it does exist on the browser. If you use the Domino hide-when formulas instead of TYPE="Hidden", the field would not be sent to the browser, so the sample would not work.

Example 3: Using JavaScript with Keyword Fields

In this example the input field is completed automatically when the user selects a value from a list. Since the selected field is separated by the keyword list, the user may also introduce a value that is not on the list.

To create this sample, do the following:

1. Create *Choices* as a keyword field. In the *onChange* event insert:

```
form.Country.value =
form.Choices.options[form.Choices.selectedIndex].text;
```

   onChange triggers the assignment of the selected value to the field named Country. SelectedIndex is the number of the selected item of the list.

2. Create *Country* as an Editable Text field.

Example 4: Changing an Image on Mouse-Over or Mouse-Out

The image displayed on the screen can be changed when the mouse floats in and out of it. The following example alternates two different logos, depending on the position of the mouse pointer.

Note This only works in a browser as Domino does not support the onMouseOver or onMouseOut events on the Notes client.

To create this sample, do the following:

1. Create a form and put the following JavaScript in the "HTML Head Content" object of the form:

```
<SCRIPT LANGUAGE="JavaScript">
logoIBM=new Image(100,40);
logoIBM.src =
"/ChilesDirect/ChilePepperSite.nsf/Banners/IBM/
$file/IBM.gif";
logoLotus=new Image(100,40);
logoLotus.src=
"/ChilesDirect/ChilePepperSite.nsf/Banners/Domino/$file/
DominoSquareLogo.gif";
function showLogo(logoName)
{
     logo = eval(logoName+".src");
     document.images["Banner"].src=logo
}
</SCRIPT>
```

Caution Remember to replace double quotes ("") with backslash double quotes (\"") before using it in the field formula.

The JavaScript will be inside the header of the Web page, so that all objects of this JavaScript are allocated before other elements on the Web page.

- **logoIBM** and **logoLotus** are the definitions of two images.
- **logoIBM.src** and **logoLotus.src** store the URL of the images. You will need to replace their values with the URL of your sample images.
- **showLogo()** is a function that replaces the content of the "Banner" image with that of one of the two defined above; the selection of the image depends on the value of **logoName**, which is a parameter (a string) passed to the function.

2. Add the following HTML code to the form, using pass-thru HTML style:

```
<a href=http://www.ibm.com
   onMouseOver="showLogo('logoIBM'); return true;"
   onMouseOut="showLogo('logoLotus')">
<IMG NAME="Banner"
SRC="/ChilePepperSite.nsf/Banners/Entrevision/$file/
Entrevision.gif">
</a>
```

Explanation:

- **onMouseOver** is the event that triggers the function call showLogo('logoIBM').
- **onMouseOut** is the event that triggers the function call showLogo('logoLotus').
- **"Banner"** is the name of the image that is replaced each time the mouse enters or exits its area.

Example 5: Updating Frames Using JavaScript

This short sample can be used when you need to change the content of two frames at the same time.

1. Insert this code in the HTML Head Content object of the form:

```
"<SCRIPT Language=JavaSript>
function changeFramesContent(URL1,URL2)
{
    top.Frame1.location=URL1;
    top.Frame2.location=URL2;
}
</SCRIPT>"
```

We have named the two frames Frame1 and Frame2.

2. Create a new button to update the frames, and in the *onClick* event add the following:

```
onClick="changeFramesContent(URL1,URL2);
```

Cookies and Domino

This section will cover the following topics:

- What is a cookie?
- Cookie syntax
- Cookie security
- An example

What is a Cookie?

A cookie is data which a Web server can store on the client's machine using the Web browser. In a later session this data can be retrieved by the server again. This mechanism can be used to remember the browser specific information across several pages, for example, the items you have collected in your shopping cart during your visit in an electronic shopping mall.

A cookie is also useful when you need to simulate a session between a browser and a client; you can store some status variables on the client so that, on the next connection, the server can read them and re-establish the 'session' starting from the last status.

As an example, you can store user preferences about the layout of some Web pages in a cookie variable, so that the Web pages are customized for each user every time they connect to the site.

Cookie Syntax

Cookies are simply text strings of the following format:

```
"CookieName=Value; expires=Date; path=Path;
            domain=DomainName; secure;"
```

where:

- **CookieName=Value** is the name and value of the variable sent to the browser.
- **expires=*Date*** contains a string, for example: "01-Apr-1998 08:00:00 GMT". If omitted, the cookie is deleted at the end of the browser session.
- **path=*Path*** and **domain=*DomainName***; if the URL doesn't match the path and domain, the cookie is not sent to or retrieved from the browser.
- **secure** This is optional and determines whether or not the cookie must be sent exclusively on a secure connection.

Cookie Security

The following list summarizes the security features for cookies:

- A cookie cannot be an executable file, so it cannot alter the file system.
- A cookie variable can be read and altered only by the server that has created it.
- An expired cookie cannot be used.
- A cookie can also be sent and retrieved on a secure connection.

Since cookies are stored on the user's machine, everyone who has access to that machine can alter the cookies information. Therefore, it is better to use them to store non-sensitive information only.

Note The way that you use cookies should be similar to how a Notes developer would use *Environment Variables.*

An alternative to using cookies is to use Profile Documents. These special documents can store users' information and, since they are Domino objects, the level of security is much higher than that of cookies. Profile documents are centrally located on the server. If you are designing an application for a large number of users take this overhead into consideration. For many intranet applications, profile documents provide a good method for keeping track of client information.

Example

The following example comes from the Domino 1.5 User's Guide.

The program uses JavaScript and cookies to load a page into the browser and save a cookie called "**Cookie_Man**" in the user's cookie file. When Domino loads the page, it includes the number of times that the site has been visited before.

To create this sample, prepare a new form, and then do the following:

1. Write the following JavaScript in the HTML Head Content object of the form, but remember to use a "\" before the double quotes.

```
"<script>
cookieName = "Cookie_Man";
function doCookie()
{
   if(document.cookie)
   {
     index = document.cookie.indexOf(cookieName);
   } else
   {
     index = -1;
   }
    if (index == -1)
```

```
    {
     document.cookie= cookieName+"=1;
      expires=Tuesday, 01-Apr-1998 08:00:00 GMT";
    } else
    {
     countbegin =
        (document.cookie.indexOf("=", index) + 1);
     countend =
        document.cookie.indexOf(";", index);
   if (countend == -1)
   {
     countend = document.cookie.length;
   }
    count = eval(document.cookie.substring(countbegin,
                   countend)) + 1;
    document.cookie=cookieName+"="+count+";
    expires=Tuesday, 01-Apr-1998 08:00:00 GMT";
  }
}

function getTimes()
{
  if(document.cookie)
  {
    index = document.cookie.indexOf(cookieName);
    if (index != -1)
    {
       countbegin =
       (document.cookie.indexOf("=", index) + 1);
       countend = document.cookie.indexOf(";", index);
     if (countend == -1)
     {
        countend = document.cookie.length;
     }
     count =
      document.cookie.substring(countbegin, countend);
     if (count == 1)
     {
       return ("<b>"+count+"</b> time before.</b>");
     } else
    {
       return ("<b>"+count+"</b> times before.</b>");
     }
  }
}
   return ("<b>0</b> times before.</b></b>");
}
</script>"
```

Explanation:

- **cookieName** is the name of the cookie; its value is "Cookie_Man".
- **doCookie()** updates the cookie, thus incrementing its value. The cookie string has a format such as "***cookieName=Value; expirationParameters***", so some string operations are done to extract the value, increment it and then rebuild the cookie.
- **getTimes()** reads the cookie value, and returns a message string with the cookie value.

2. Write in the *onLoad* event of the form:

```
doCookie()
```

The procedure *doCookie()* is called every time the form is opened.

3. Write the following script on the form, using Pass-Thru HTML style:

```
<script>
document.write("You have visited this site "+ getTimes());
</script>
```

This JavaScript write calls the *getTimes()* function and writes a message containing the number of times the user has opened this form.

LiveConnect — JavaScript Access to the Domino Classes

The LiveConnect technology allows JavaScript to initiate applet communication within browsers and is also implemented in the Notes 5.0 client.

LiveConnect is a proprietary Netscape Technology (since 3.0) and is also implemented within Internet Explorer. It also allows applet - script (IE - partial support), script - applet (IE - partial support) and applet - applet communication, but this is not fully supported in the two browser environments.

Accessing an Applet From JavaScript

Consider the following applet:

```
<APPLET CODE="Hello.class" NAME="Hi" WIDTH=150 HEIGHT=25>
</APPLET>
```

JavaScript can access an applet via the applet name or applets array in the JavaScript document object:

- document.applets.Hi
- document.applets[0] or document.applets["Hi"]

On accessing the applet, JavaScript can also access the Java public methods/properties of the applet:

- document.applets.Hi.methodname
- document.applets.Hi.variable

Obvious advantages include repainting applets with new data at runtime, without roundtrips to the server via submits.

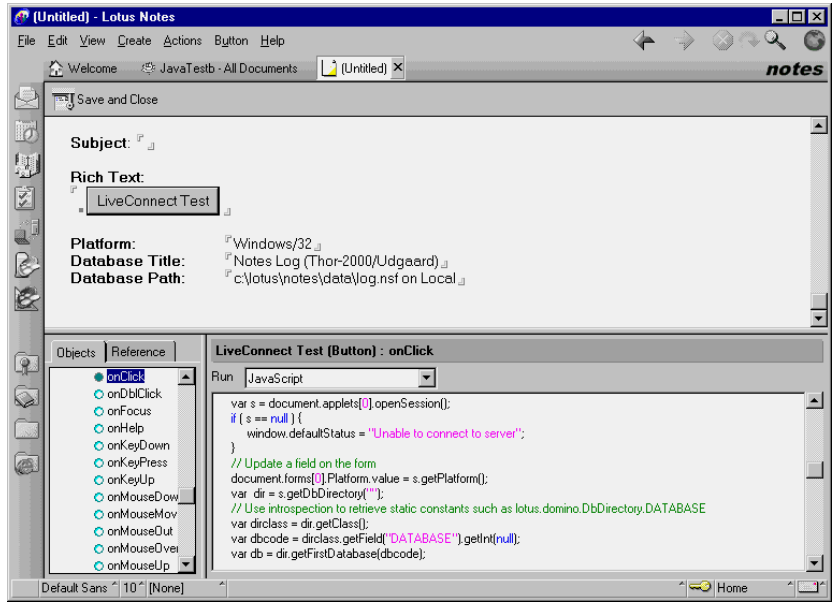
Accessing Java/CORBA Applets via LiveConnect

Consider the current Notes Client Programmability model. The Domino Object Model (DOM) is accessed by LotusScript in the event handlers. On the Web, the scripting language is JavaScript, which has no interface to the Domino Object Model (DOM). Java has an interface to the DOM, but the API's are remote. This is where you can utilize a CORBA applet.

A CORBA-enabled applet can access a remote Domino Session object and, in combination with LiveConnect, make this property available to JavaScript via a public property or method. An HTML page can now have a persistent session with the Domino server via JavaScript and utilize the DOM, without the need to submit the page to the server for each transaction.

Example

The following example illustrates how LiveConnect can be used in the Notes client. It has been taken from the Lotus Domino Toolkit for Java/CORBA 2.0.



The Rich Text field on the document contains a 1 pixel square embedded applet. The applet code is minimal as you can see below:

```
import lotus.domino.*;
public class PinpointApplet extends lotus.domino.AppletBase {
      public void notesAppletInit()
      {
            setLayout(null);
            setSize(1,1);
      }
}
```

The important thing here is that the applet extends the AppletBase class. This class implements the method **AppletBase.openSession()** which we will use to get a Domino session object.

The Rich Text field on the document in the figure also has a button that, when clicked, locates the applet, passes the openSession() method to it and then accesses the Domino Object Model through JavaScript. You can see all the JavaScript code for that button.

```
// Sample JavaScript code accessing back-end data via Java
// Retrieves the first or second database in the DBDirectory
{
      // Once session is obtained, we can utilize any class
       // in the Domino Object Model. Can also
      // use document.applets[0].openSession(user, pwd);
      var s = document.applets[0].openSession();
      if ( s == null ) {
      window.defaultStatus = "Unable to connect to server";
      }
      // Update a field on the form
      document.forms[0].Platform.value = s.getPlatform();
      var  dir = s.getDbDirectory("");
      // Use introspection to retrieve static constants
       // such as lotus.domino.DbDirectory.DATABASE
      var dirclass = dir.getClass();
```

```
      var dbcode = dirclass.getField("DATABASE").getInt(null);
      var db = dir.getFirstDatabase(dbcode);
      // For fun, switch between first and second DB each time
       // button is clicked
      if ( document.forms[0].DatabaseTitle.value ==
db.getTitle() ) {
            db = dir.getNextDatabase();
      }
      db.open();
      document.forms[0].DatabaseTitle.value = db.getTitle();
      var server = db.getServer();
      if ( server == "" ) server = "Local";
      document.forms[0].DatabasePath.value = db.getFilePath() +
" on " + server
}
```

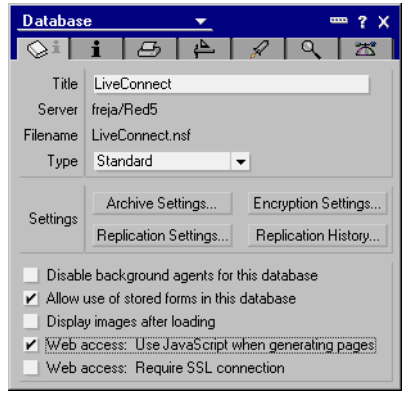
Once JavaScript has retrieved the session object reference, it can utilize the full Domino Object Model in the JavaScript code.

In the code above, the JavaScript code uses the Session object to access the following data, which is placed into the respective fields on the form.

- Platform of the Domino server
- Title of the first or second database on the Domino Server
- File path of the first or second database on the Domino Server

From a performance perspective, the applet initially takes a small amount of time to load from the server. Once it is loaded and initialized, however, access to the remote session object is fast.

For the button to implement the JavaScript, the Use JavaScript When Generating Pages option must be selected.



The applet and the button with the JavaScript code can be placed in the form design, so that every document created automatically contains them. The applet can be in a hidden paragraph, as users do not need to see it.

It is advisable to place complex code into a public method within the applet to correctly handle Java Exception conditions.

This method could also be used for hiding JavaScript implementation code as Java inside an applet.

If the form had many applets, the one session reference could be shared among the applets via the InfoBus technology.

External Tools

The API for Domino and Notes

The C and C++ API for Domino and Notes allows you to write a program that processes data in a Domino database, or moves data in and out of Domino. The API accesses the Domino database layer, much as the Domino Object Model itself accesses it. You can also use the API to access the server software, the Tools menu in the workstation software, and the File Types list in the File Export dialog box.

Note Because Domino R5.0 supports a CORBA/IIOP architecture, you are also able to run API programs through the Web. In this case, the client uses the server API's. For more information about CORBA/IIOP architecture, read Chapter 11: Advanced Domino Programming.

You can write an API program to do the following:

- Extract external data, reformat it, and store it in the Domino database.

  For example, you can retrieve information from SQL records.

- Extract Domino data, reformat it, and store it in an external application.

  For example, you can retrieve Notes workflow status data into a word processor or executive information management (EIS) system.

- Add commands to the File - Tools menu.

  For example, when a user chooses your new command, Domino can launch your program and pass user context information to it, such as which view is active, whether the user is editing a document, and which field contains the cursor. Your program can compute new values and enter them into Domino fields.

- Implement server add-in tasks.

  For example, you can implement a task that takes conditional actions beyond Notes background macro capabilities. A server add-in task functions as a daemon. It has no user interface and runs in the background like other server tasks.

- Create a custom file export format.

  For example, when a user selects your new file type in the Notes File Export dialog box, Domino launches your program and exports data to it.

  For more information about the API products refer to the documentation found at:

  `http://notes.net/notesua.nsf/Product?OpenViewSummary`

Summary

In this chapter we have covered some of the basic methods for programming the Domino Object Model using LotusScript and JavaScript. In the next chapter we will cover some of the more advanced methods, such as Java, CORBA and the LSX toolkit.

Chapter 11
Advanced Domino Programming

This chapter will discuss some of the more advanced methods of coding applications for Domino. We will cover Java, CORBA/IIOP, OLE automation, and writing your own LotusScript Extensions using the LSX toolkit.

Java

As the Web evolves, Java becomes a more important and more commonly used programming language. Domino offers you the option to write your applications in Java. For example, you can write your Domino agents in Java.

Domino supports Java programs written in Java 1.1.x.

About Java Domino Classes

Java Notes classes are created by modifying some of the LotusScript Extension (LSX) architecture to include a Java "adapter" to compose the new Java Domino classes. The Java Domino classes have similar functions to some of the LotusScript Domino back-end objects. You can use these classes from any Java program — within the Notes Designer environment or outside of it — as long as Notes Release 5 is installed on the machine.

Internally, Java Notes classes execute the same C++ code as the LotusScript Domino back-end objects, only the language syntax is different.

A Java program is generally made up of a number of files. You must designate one as the Base Class, which is the starting point for the Java program. For efficiency, typically for improving applet download speeds, you can bundle all of the class files and additional resources (for example GIF files) into a single compressed Java Archive file. The imported Java files can be of the following types:

- Class - *.class
- Archive - *.jar

For example, when you write a Java agent program, the class you write must extend the class *AgentBase*. The code you want to execute when the agent runs is in the *NotesMain()* method.

To work with Java in Domino you need the Lotus Domino Toolkit for Java/CORBA 2.0. You can download the toolkit from this Web site

```
http://www.lotus-developer.com
```

Java Coding Conventions

There are some conventions you should follow to write a Java program. These are as follows:

Classes

The names of the Java classes are similar to the LotusScript classes except that they begin with the "lotus.domino"-prefix. The table below shows how some of the Java Domino classes correspond to LotusScript objects:

Java Class	*LotusScript Object*
lotus.domino.Session	NotesSession
lotus.domino.DbDirectory	NotesDbDirectory
lotus.domino.Database	NotesDatabase
lotus.domino.View	NotesView
lotus.domino.Document	NotesDocument
lotus.domino.Item	NotesItem
lotus.domino.RichTextItem	NotesRichTextItem

Note The lotus.domino package has the same content as the Release 4.6 lotus.notes package plus new classes, methods, and other enhancements. The Release 4.6 lotus.notes package continues to be supported for backwards compatibility only. It does not contain the new classes, methods, and other enhancements.

Methods

Method names are written with the first character being lower case, for example getFirstDocument. Of course, there are some exceptions such as FTSearch.

Note By convention you should start your own classes with the first character as uppercase.

Caution Java is case sensitive; the wrong case causes an error.

Properties

To access properties in Java you also have to use methods. In Java, properties are implemented through methods, known as accessors, which use the following naming conventions:

- The name of a method used to get the value of a non-boolean property is the name of the property prefixed with "get".
- The name of a method used to set the value of a property is the name of the property prefixed with "set".

- The name of a method used to get the value of a boolean property is the name of the property prefixed with “is”.

Parameters and Return Values
Parameter and return values differ from LotusScript as needed to match the different data types in Java. For example, in LotusScript boolean values are represented by Variants. In Java they are of type boolean.

Object Containment Hierarchy
In Java you cannot create lotus.domino objects using the “new” modifier. All lotus.domino objects must be created with lotus.domino methods emanating from the root Session object.

Agents, Applets, Applications, and Servlets

Java programs can take one of several forms, each with its own characteristics. The differences between these forms can be summarized:

Java agents complement the familiar LotusScript agents and, to a large degree, they can be used interchangeably when dealing with back-end operations. Some reasons for choosing Java over LotusScript are: existing programmer knowledge, multi-threading, a more fully featured language, extensibility through (non-visual) beans, etc.

Applets allow a Notes developer to create a richer GUI environment for the end user. Applets will be dynamically downloaded from the server and executed on the client’s machine and will work with either Web browsers or Notes clients. The functions of applets can vary widely, from simple news tickers to complex database front ends. Java applets are subject to the Java Sandbox security model, which prevents unauthorized applets from accessing sensitive machine resources and from performing certain operations. By default applets will not have access to the Notes back-end classes. If this is required, then CORBA is needed (see the CORBA section later in this chapter).

Java applications differ from applets in that they are not dynamically loaded from the server, they are similar to traditional executables in this respect. However, Java applications typically run outside the Java “Sandbox” security model and can thus access machine and network resources denied to an applet. A Java application can be loosely regarded as analogous to a stand-alone application which accesses the Notes object model, for example a C or Visual Basic program. By default applets and applications will not have access to the Notes back-end classes, if this is required then CORBA is needed (see the CORBA section later in this chapter).

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. 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 a 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). Again a loose analogy can be drawn to the ability in Domino to invoke an agent directly from a HTTP request (myagent?openagent&param1=value1).

The model for Java agents differs from Java applets in a number of ways:

- Java agents are written explicitly for Domino. Applets are often designed be served up by any Web servers.
- Java agents behave in the same way as LotusScript agents but Java applets behave like Java applets in any Web-authoring environment.
- Java agents only run within a Domino-supplied Java runtime environment whilst Java applets run in both Domino-supplied Java runtimes and browser-supplied runtimes.
- Java agents are structured in the same way as Java applications (not as applets). They run within a Domino-supplied context as opposed to applets whose context is provided in part by the browser and in part by the codebase parameter specified as part of the applet tag. For agents, CodeBase and DocBase are not meaningful ways of getting ahold of additional classes. Instead, as with other Java applications, classes, and resources are located within Jar files and the class path.
- Java agents can access Domino databases directly using the Java Domino classes. Applets can only access Domino objects within Notes using URLs. Note that nothing precludes a Java agent from using URLs to access Domino objects in Notes.
- Agents do not have a UI (and consequently do not use resources as much as applets). Java Agents run in a relaxed security environment like Java applications do. You can wrap an application or agent in a SecurityLoader, typically this would be used in a tightly controlled secure environment when running a semi-trusted application. This feature is being built into JDK1.2 but can be achieved in 1.1x.

Adding CORBA to the Picture

One of the major enhancements in the Domino R5.0 embrace of Internet standards is the support for CORBA.

Common Object Request Broker Architecture (CORBA) is an open standard defined by the Object Management Group (OMG). CORBA serves as middleware for a distributed computing environment whereby clients can invoke methods on remote APIs residing on other computers. CORBA uses Internet Inter-ORB Protocol (IIOP) for communication over a TCP/IP network.

CORBA/IIOP support enables Domino developers to create applets that can be downloaded to the client and can be remotely invoked in Domino services, for example, to initiate a workflow process. In addition, CORBA/IIOP enables information to be processed efficiently over networks within an open standards-based framework and to distribute work effectively between clients and servers, ultimately lowering the cost of ownership.

Benefits of Using CORBA

Some advantages to using CORBA are:

- You can use Domino Object Model (DOM) back-end classes to support CORBA.
- The client does not have to deal with issues such as networking or security.
- CORBA allows many different clients to use the same objects (not copies of the objects). The latest version of the object is always used.
- Client applications can be in different languages from the Server Objects.
- Java ORBs and Stubs can be downloaded to the client at runtime, which means:
  - Users don't have to install the application on the client before running it.
  - Clients are always working on the most current version of the application.
  - Network computers are supported as clients as the application is removed when the computer is turned off.

For a more detailed look at the CORBA internals look in the appendix.

How and When to Use CORBA

CORBA support can be easily added to Java applets and applications to extend their reach into the Domino back end. In order to utilize CORBA you must make some small changes to your server and Java programs.

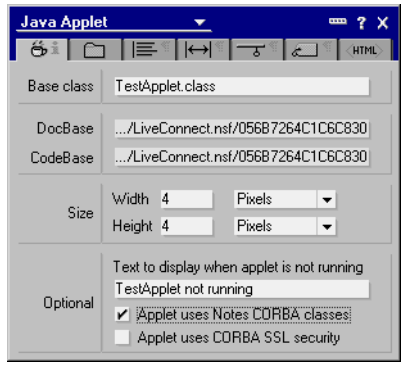
A Java program using CORBA has the following requirements:

Server

- The server tasks HTTP and DIIOP must be running. Ensure that the notes.ini file contains the following line:

```
ServerTasks=<any other tasks>,http,diiop
```

To enable an applet for CORBA, import your applet into a form and select the appropriate properties from the applet InfoBox.



For performance reasons, when a CORBA enabled applet is loading in the Notes client all the calls are transparently made to the Notes DLLs rather than the Java classes.

Compiling and Running a Java Program

The new package, lotus.domino, which comes with Domino R5.0 supports local and remote calls to the Notes object interface. This package contains the same classes and methods as the lotus.notes package shipped with Domino R4.6 plus new classes, new methods, and some other enhancements.

Note The Domino R4.6 lotus.notes package is supported for backward compatibility only.

A Java program using the Domino classes has the following requirements:

Server

- The server tasks HTTP and DIIOP must be running. Ensure that the notes.ini file contains the following line:

```
ServerTasks=<any other tasks>,http,diiop
```

Designer

- Ensure that the NOTES.INI file contains the following line:

```
ALLOW_NOTES_PACKAGE_APPLETS=1
```

- Include NCSO.jar and Notes.jar in your CLASSPATH environment, for example:

```
set
CLASSPATH=<other>;<domino>\java\NCSO.jar;<domino>\Notes.jar
```

  Notes.jar contains the high-level lotus.domino package, the lotus.domino.local package for local calls, and the old lotus.notes package. NCSO.jar contains the high-level lotus.domino package and the lotus.domino.corba package for remote calls. Strictly, you do not need NCSO.jar if you are not compiling remote calls and you do not need Notes.jar if you are not compiling local calls or old calls.

Your class code must import the high-level lotus.domino package:

- import lotus.domino.*

Runtime Requirements

- A machine running a Java application that makes local Notes calls must contain Domino R5.0 (Client, Designer, or Server) and must include Notes.jar in the CLASSPATH.
- A machine running a Java application that makes remote Notes calls need not contain Domino R5.0, but must contain NCSO.jar and must include NCSO.jar in the CLASSPATH.
- A machine running a Domino R5.0 agent that makes Notes (Java) calls must include Notes.jar in the CLASSPATH.

  Note A machine running an applet that makes Notes calls needs no Domino software or CLASSPATH assignments.
- The server must be running when remote calls are made.

Remote Calls to lotus.domino Package

In order for a Java application for remote runtime access of lotus.domino you must create a Session with the NotesFactory method createSession(String IOR, String user, String pwd). NotesFactory is new with R5.0 and the lotus.domino package.

The IOR (initial object reference) parameter is required to access a Domino server remotely. It is a string contained in the file ior.txt in the notes directory of the Domino server. The NotesFactory method getIOR(String host) returns the IOR for a given host.

The second and third parameters must be a user name and Internet password in the Domino directory on the server being accessed. If empty strings are specified, anonymous access must be permitted by the server.

The application must not use the NotesThread method. NotesThread is for local access only.

This example demonstrates an application using remote calls:

```
import lotus.domino.*; // replaces old lotus.notes package
public class platform3 implements Runnable
{
  String host=null, IOR=null, user="", pwd="";
  public static void main(String argv[])
    {
      if(argv.length<1)
      {
        System.out.println("Supply Notes server name");
        return;
      }
      platform3 t = new platform3(argv);
      Thread nt = new Thread((Runnable)t);
      nt.start();
    }

  public platform3(String argv[])
  {
    host = argv[0];
    if(argv.length >= 2) user = argv[1];
    if(argv.length >= 3) pwd = argv[2];
  }

  public void run()
    {
    try
      {
        IOR = NotesFactory.getIOR(host);
        Session s = NotesFactory.createSession(IOR,user,pwd);
        String p = s.getPlatform();
        System.out.println("Platform = " + p);
      }
    catch (Exception e)
      {
        e.printStackTrace();
      }
    }
}
```

Applet Calls to lotus.domino Package

An applet intended for run-time access of lotus.domino extends AppletBase and puts its functional code in the methods notesAppletInit(), notesAppletStart(), and notesAppletStop(). AppletBase is new with Domino R5.0 and the lotus.domino package. You do not have to distinguish between local and remote access. AppletBase will make local calls if the applet is running on a machine with Domino installed and remote calls otherwise.

Domino will automatically supply the IOR.

Here is an example of an applet:

```
import lotus.domino.*;
public class platformApplet extends AppletBase
{
  java.awt.TextArea ta;
  public void notesAppletInit()
  {
    setLayout(null);
    setSize(100,100);
    ta = new java.awt.TextArea();
    ta.setBounds(0,0,98,98);
    add(ta);
    ta.setEditable(false);
    setVisible(true);
  }
  public void notesAppletStart()
  {
    Session s;
    try
    {
      // Can also do openSession(user, pwd)
      s = this.openSession();
      if (s == null) { //not able to make the connection, warn user
          ta.append("Unable to create a session with the server");
        return;
      }
      String p = s.getPlatform();
      ta.append("Platform = " + p);
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
 //     this.closeSession(s);
    }
  }
}
```

Setting Security Options for Java Applets

You can now set security options for applets to prevent unauthorized access to your Notes file system or to Notes Java classes. You create an execution control list that identifies what people and groups you trust with access to your Notes system. When an applet runs on your workstation, Notes checks for execution rights of the person or group that signed the applet. If an applet is signed by a person or group without the correct authorization, Notes alerts you to the illegal operation. You can abort the operation and not run the applet, trust the signer of the applet one time, or automatically add the signer to the execution control list.

Note that this security model only applies to applets running on the Notes client. Applications running on a Web browser must follow the security model set by the browser.

To set applet security:

1. Choose File - Preferences - User Preferences.
2. Click the Security options button on the Basics page.
3. Click Java applet security to display the security panel.

   Enter a person or group name and assign access rights to the file system and/or Notes Java classes. The Add button lets you enter a name or choose one from a Public Address Book.
4. Click OK and close the dialog box when you have completed your entries.

Note The implementation of this applet security system removes the restriction on using Notes classes in Java applets.

Using the NotesThread Class

A stand-alone program must use the lotus.domino.NotesThread class, which extends Java.lang.Thread. You can either extend NotesThread or implement the Run-able interface. If you extend NotesThread, the entry point to the functional code must be public void runNotes(). If you implement run-able, the entry point must be public void run().

- A Domino or Domino agent program must extend the lotus.notes.AgentBase class, which extends lotus.domino.NotesThread. The class that contains the agent code must be public. The entry point to the functional code must be public void NotesMain().
- The lotus.domoino.Session class is the root of the Notes back-end object containment hierarchy. For stand-alone programs, use one of the NotesFactory.createSession methods to create a Session object. For agents, use the AgentBase method getSession().

- System.exit must not be used to terminate a program using the NotesThread class (and by extension the AgentBase class). In an agent, System.exit throws SecurityException. In a stand-alone program, System.exit may cause corruption problems.
- For foreground agents, System.out and System.err output goes to the Java debug console. For locally scheduled agents, System.out and System.err output goes to the Domino log.

Creating a Java Agent

Example 1: Java Agent

This example shows an agent that runs on newly created and modified documents since the agent was last run. The program works on the unprocessed documents, prints the form name of each document, and marks each document as processed. The first time the agent runs, the agent returns all of the documents in the database. Thereafter, the agent returns those documents that updateProcessedDoc has not touched.

1. Create an agent:
   - Name the agent.
   - Select When should this agent run = Manually from Actions Menu.
   - Which documents should it act on = All documents in database.
   - Select Java as your source code and write the agent code.

```
import lotus.domino.*;
import java.util.*;

public class myagent extends AgentBase
{
  public void NotesMain()
    {
    try
      {
        Session s = getSession();
        AgentContext ac = s.getAgentContext();
        DocumentCollection dc =
ac.getUnprocessedDocuments();
        Document doc;
        int size = dc.getCount();
        System.out.println("Count = " + size);
        doc = dc.getFirstDocument();
        while (doc != null)
          {
          System.out.println
             (" *** " + doc.getItemValue("form"));
          ac.updateProcessedDoc(doc);
```

```
            doc = dc.getNextDocument(doc);
          }
        }
      catch (Exception e)
        {
          e.printStackTrace();
        }
      }
}
```

- Save it.

Example 2: Using Java Notes Classes

This sample Java program is from Lotus Technology Learning Center. The Java code is commented to help you understand how the Java Notes class is implemented.

This program creates an instance of NotesThread, a class which extends the Java Thread class. It allows Notes to properly initialize and terminate per thread in a convenient way for the programmer.

This sample program does the follow things:

1. Creates a new Notes session.
2. Opens a database (in this case, the local Address Book).
3. Accesses the People view.
4. Searches the People view for the entered name.
5. Accesses the document that matches the search criteria.
6. Pulls the Spouse field out of the document.
7. Prints the Spouse field in a message output.

To Run This Sample:

1. Add a person John Smith and his spouse Mary Smith into the local Address Book. John Smith will be used as a parameter to the command to run the Java program.
2. Write the following code into a Java program(.java), set your PATH and CLASSPATH, for example, as follows:

   ```
   PATH = c:\jdk1.1.3\bin;c:\notes\;
   CLASSPATH = c:\jdk1.1.3\lib\classes.zip;c:\notes\notes.jar;
   ```

3. Compile the Java program.

   Note We used Java JDK Version 1.1.3 from SUN. You can download it from www.javasoft.com.

4. Type the command:

   ```
   javac  myjavafile.java
   ```

The output is a file named abe.class.

5. Run this class file at a DOS command prompt:

```
C:\jdk1.1.3\bin> java abe.class  John Smith
```

The output will be like this:

```
Creating Notes session...
User name = CN = John     Smith OU=CAM  O= Lotus
Spouse of John is Mary Smith
Date Created : 08/15/97 16:00:00 PM EDT
```

The sample program is listed below for your information:

```
/* Copyright 1997, Iris Associates, Inc.
Sample Java program, for illustrative purposes only.
*/
import lotus.domino.*;
import java.lang.*;
import java.util.*;

class abe implements Runnable
{
public String g_name;

// if you run the class from the command line...
public static void main(String argv[])
throws Exception
{
// print out a message, then exit, no args provided
if (argv == null || argv.length == 0)
  System.out.println("Usage: java abe <user name>");

else
  {
  // create new instance of abe
  abe t = new abe();


  // store name to look up in the instance
  t.g_name = argv[0];

  // make sure the Notes lsx is loaded
  NotesThread.load(true);

  // create a thread instance for running abe, start it
  NotesThread nt = new NotesThread((Runnable)t);

  // start the thread, call our runNotes()
  nt.start();
```

```
  }
}

// this would get called if we ran it from java.lang.Thread
// instead
public void run()
{
runNotes();
}

public void runNotes()
  {
  int i;
  try
    {
  System.out.println("Creating Notes session...");
  Session s = NotesFactory.createSession();

  // show off, print the current user's name
  System.out.println("User name = " + s.getUserName());

  // get db instance for the name and address db
  Database db = s.getDatabase("","names.nsf");

  // find the "People" view
  View view = db.getView("People");

  // search for the name provided
  view.FTSearch(g_name);

  // for now, ignore multiple matches
  Document doc = view.getFirstDocument();

  // look up contents of the "spouse" field
  String name = doc.getItemValueString("Spouse");
  System.out.println("Spouse of " + g_name + " is " + name);

  // also print out the date the document was created
  System.out.println("Date created: " + doc.getCreated());
   }
  catch (Exception e)
    {
    e.printStackTrace();
    }
  }
}
```

CORBA/IIOP

Domino R5.0 uses an architecture called Common Object Request Broker Architecture (CORBA). This is an open standard defined by the Object Management Group (OMG). CORBA serves as middleware for a distributed computing environment whereby remote clients can invoke methods on remote APIs residing on other computers. CORBA uses Internet Inter-ORB Protocol (IIOP) for communication over a TCP/IP network.

CORBA/IIOP support enables Domino developers to create applets that can be downloaded to the client and can be remotely invoked in Domino services, for example, to initiate a workflow process. In addition, CORBA/IIOP enables information to be processed efficiently over networks within an open standards-based framework and to distribute work effectively between clients and servers, ultimately lowering the cost of ownership.

Benefits of Using CORBA

Some advantages to using CORBA are:

- You can use Domino Object Model (DOM) back-end classes to support CORBA.
- The client does not have to deal with issues such as networking or security.
- CORBA allows many different clients to use the same objects (not copies of the objects). The latest version of the object is always used.
- Client applications can be in different languages from the Server Objects.
- Java ORBs and Stubs can be downloaded to the client at runtime, which means:
  - Users don't have to install the application on the client before running it.
  - Clients are always working on the most current version of the application.
  - Network computers are supported as clients as the application is removed when the computer is turned off.

CORBA Architecture

In order to understand how the CORBA/IIOP architecture works, we will first take a closer look at CORBA.

The heart of CORBA is the Object Request Broker (ORB). This is the object bus, i.e. it's like a PC bus but instead of sending bits/bytes/instructions it's sending objects and is potentially distributed. The job of the ORB is to act as a middleman, allowing objects to make requests of each other. Although the ORB operates in a client/server environment, objects that the ORB works with can function as either clients or servers, depending on the circumstances. If an object is receiving and processing a request, then it is acting as a server. If the object is making a request, then it is acting as a client.

The ORB handles these requests regardless of programming language, operating system, or platform. The mechanism that allows ORBs to handle requests transparently is the Interface Definition Language (IDL), which is used to declare the boundaries and interfaces of an object. Much like an independent arbitrator, the IDL is neutral and independent of the object and the ORB, yet it binds providers of distributed object services to their clients.

CORBA IDL uses inheritance to encapsulate objects and so it is able to reuse code very easily. Furthermore, the beauty of IDL is that you can concisely define APIs, yet still have the freedom to define the IDL methods in any programming language that provides CORBA bindings, for example, COBOL, C, C++, Smalltalk, and Java.

To make IDL truly independent, CORBA uses an Interface Repository (IR), the purpose of which is to store the method signatures of objects so that the signatures can be dynamically retrieved and updated at runtime. In this way, all objects in the enterprise system can learn about other objects' interfaces, the methods the interfaces support, and the parameters the interfaces require.

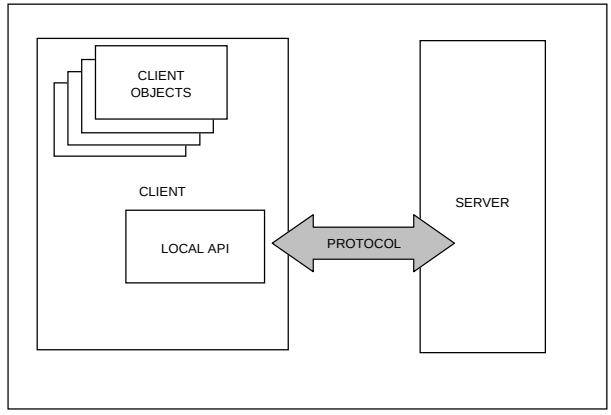
When you bring together the ORB, the IDL, and the Interface Repository, you have a basic model of CORBA. The model doesn't include all the pieces of the architecture, but it gives you an idea of how heterogeneous objects interact using CORBA.

Internet Inter-ORB Protocol (IIOP)

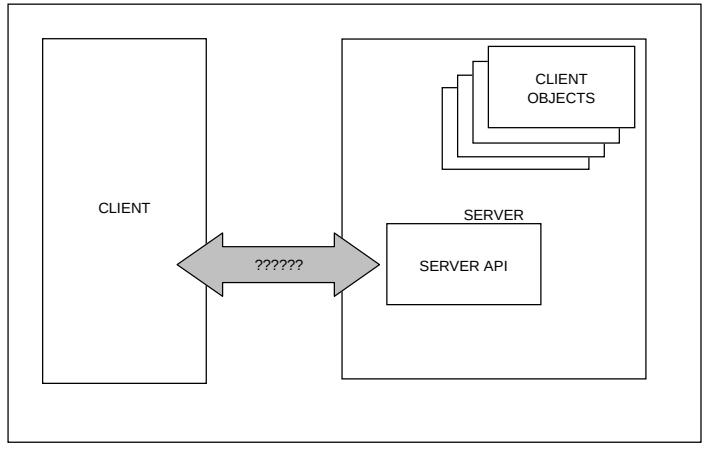
Internet Inter ORB Protocol is a protocol also developed by Object Management Group (OMG), which enables CORBA solutions to communicate over the World Wide Web (WWW). It enables browsers and servers to exchange data (integers, arrays, and complex objects) unlike HTTP, which only supports text transmissions.

CORBA and Domino

In earlier releases, the Domino client and server could communicate together using APIs, but meant that all APIs and objects which were needed had to reside on the client's disk. This was sometimes difficult, for example, where many clients needed a particular application, and it also required additional disk space on the client.



In this environment, what happens if a non-Domino client wishes to utilize these APIs and there are no local Domino APIs present? The Domino server has server APIs to implement the DOM (Domino Object Model), but the real issue is how can the user access these remote APIs from a browser?

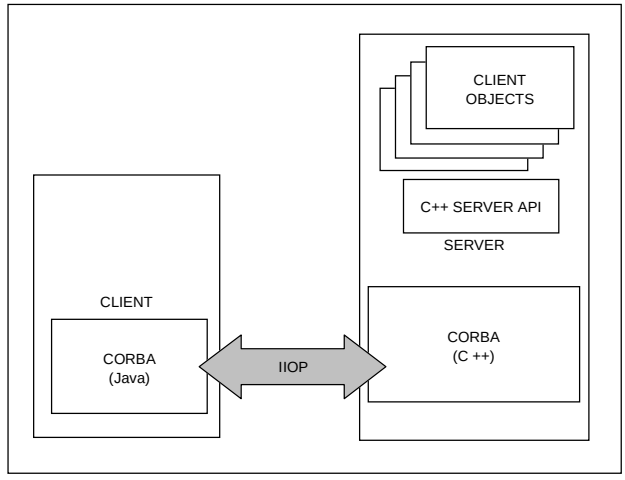


In earlier Domino versions specific challenges included the following:

- Within a Web browser, applets had limited access to the database on the server via Java API calls to remote Domino information after loading. Also, the DOM could not be directly accessed by browser clients but only indirectly via Web agents, triggered on document load or save.
- Within a Notes client, applets could not use the Java Notes classes as they made native calls to local DLLs and this could compromise Java applet security. Also, these applets could not retrieve Domino information after initial loading. Applet programmability options were not seamless to mixed client audiences.
- Standalone Java applications required an installed Notes client to use the 4.6 Java classes as they made native calls to locally installed Domino DLLs (the Java classes were a wrapper around the C++ core).

CORBA/IIOP support in Domino R5.0 solves these problems. As mentioned earlier, CORBA serves as middleware and facilitates the design and implementation of distributed systems by providing a transport trough for distributed objects to locate and exchange data with each other and provide language, operating system, hardware platform, and networking interoperability.

The following figure shows how a browser can use the Domino database to use CORBA:



This allows Java programs on remote clients such as applets in browsers and standalone Java applications to access the Domino Object Model (DOM) on the Domino server. From an implementation standpoint, a remote client instantiates and references DOM objects as if they were resident on the client, but in fact the client is communicating with objects on the server.

Thanks to CORBA/IIOP Domino is able to:

- Use applets that can create a permanent DOM session with the Domino server and access the back-end objects as if they were in a Notes client. Also, the browser programmability mode can now more closely resemble the Notes client programmability model. You can now associate Java or JavaScript with the W3C (World Wide Web Consortium) events and access data either on or off the current document without refreshing or submitting the current page. For an example of how to access the top level Notes session object via JavaScript see Chapter 10.
- Switch the transport dynamically depending on context; if the applet is in a browser, CORBA is used. If it's in a Notes Client then the Notes API is used because it's more efficient. Security options in the User Preferences can control which applets, signed by whom, can access the file system or Notes Java classes. The same CORBA applet will use remote calls to the server from a browser. This is seamless to the programmer as long as the applet is designed specifically for CORBA. The most compelling example of this technology is the ability to place a custom applet on a form and have that applet access the DOM in both the Notes client and a browser.
- Use standalone Java applications which have access to the Domino server directly via CORBA without a client being installed locally.

Note CORBA is often only associated with Java, but the real strength of CORBA is its support of heterogeneous programming environments. In the real world many legacy systems and newly developed applications are written in different languages and they need integration in order to work together. Language inter operability allows objects running in heterogeneous languages to make invocations to each other. A CORBA interface can be mapped into a number of popular programming languages on the client or server, for example, C, C++, Smalltalk, Ada, COBOL, and Java already have CORBA interfaces.

Coding the CORBA Applet

This section gives you an example of how to use CORBA within a Domino application.

An applet intended for CORBA run-time access of Domino needs to be designed in a specific way.

Step 1

Import the lotus.domino package into your applet source file. This package is contained in the NCSO.jar or cab file and is new to Domino R5.0. It contains the new DOM classes and CORBA Client stub classes.

```
import lotus.domino.*;
```

Step 2

Your applet class must extend the AppletBase class.

This class already implements the standard applet methods init( ), start( ), end( ), stop( ) as final methods, therefore you cannot create (or override) these in your applet. Instead these final methods call the following methods respectively notesAppletInit( ), notesAppletStart( ), notesAppletEnd( ) and notesAppletStop( ). These methods are also contained in the AppletBase class and are not final. If you wish to implement the functionality that you would normally in the traditional applet methods, you must override the respective notesAppletXXXX( ) method in your applet class. You do not have to distinguish between local and remote access. AppletBase will make local calls if the applet is running through the Notes client and remote CORBA calls if it is running through a browser.

```
import lotus.domino.*;
   public class myApplet extends AppletBase
  {
        public void notesAppletInit()
        {
                // Your Applet Init Code
         }

        public void notesAppletStart()
        {
                // Your Applet Start Code
        }

         public void notesAppletEnd()
        {
                // Your Applet End Code
         }

        public void notesAppletStop()
        {
               // Your Applet Stop Code
         }
  }
```

Step 3

To access a Session object within your applet you use the getSession( ) inherited instance method of the AppletBase class. As mentioned previously, the getSession( ) method call will also instantiate and initialize the Client Side ORB as well as request a remote Session object reference from the Domino server. Each applet on a HTML page invoking the getSession method will instantiate another Client ORB. This may not be desirable, methods for inter-applet communication such as the InfoBus technology may

be used to alleviate this situation. Another issue is where to invoke your getSession( ) method. This should be in either the notesAppletInit( ) or event handlers for the applet. You don't want to be calling this code every time an applet is restarted.

```
import lotus.domino.*;

    public class myApplet extends AppletBase
    {
        private Session s = null;
        public void notesAppletInit()
        {
            try
            {
                    s = this.getSession();
            }
            catch (NotesException e)
            {
                    e.printStackTrace();
            }
        }
     }
```

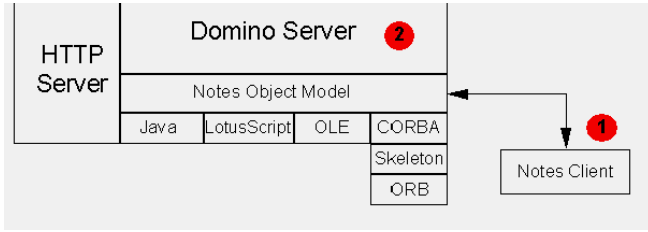
Step 4

Compile your class. If you are using the JDK or a third-party IDE such as Visual Age for Java or Visual Cafe, then make sure that the NCSO.jar file is in the CLASSPATH for these IDEs. Your applet is ready to be implemented in the Domino environment.

Implementing CORBA Applets Within Domino

Once the applet is developed, it must be included in the Domino environment.

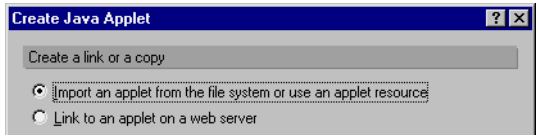
Steps 1 and 2 explain what to set up in Domino to include your applet and make your Domino server CORBA aware.



Step 1

You can create an applet in the rich text field of a Domino Document or Domino Designer Form or Page. Either way, you get the option of importing

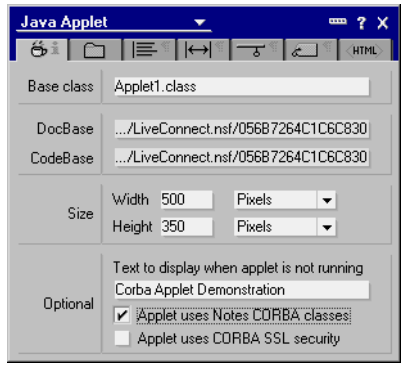
or linking to an applet from the file system.



If you use the import option, you must import all applet related classes into Domino including the NCSO.JAR file which contains the CORBA Java Client Stubs as well as CORBA Client ORB classes. This file is located in the Domino\html directory.

If you choose to link to an applet on a Web server, the NCSO jar and cab files are in the HTML directory on the Domino server.

After creating the applet, you modify the applet parameters via the Properties box. If you wish the applet to be CORBA enabled on a HTML page served by Domino, you must select Applet uses Notes CORBA classes as follows.



When this option is selected, Domino automatically adds applet parameters to the applet HTML tag before serving the document to the Browser.

```
<APPLET WIDTH="500" HEIGHT="350"
CODEBASE="/LiveConnect.nsf/98024b2..... /$FILE"
CODE="Applet1.class" ALT="Corba Applet Demonstration"
ARCHIVE="NCSO.jar">
<PARAM NAME="NOI_IOR"
VALUE="IOR:010000002e00000049444c3a6c6f7475732
..........000000">
<PARAM NAME="NOI_COOKIE_URL"
VALUE="/LiveConnect.nsf?GetOrbCookie">
</APPLET>
```

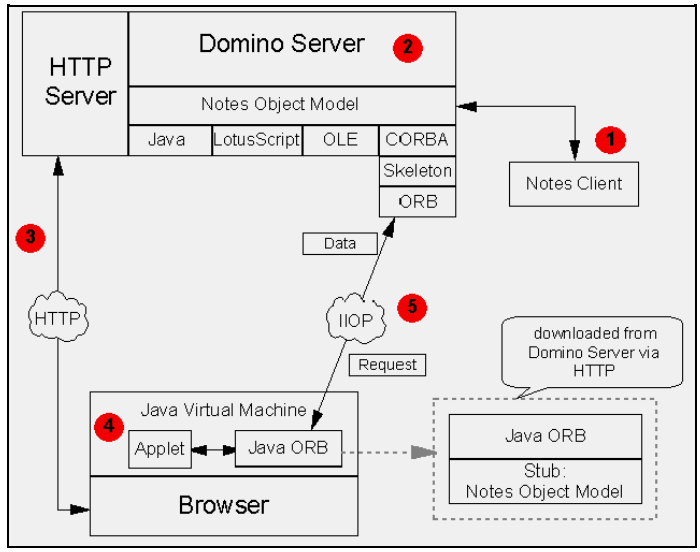
- The NOI_IOR parameter is the object reference to the Notes object server, our DIIOP addin. This is the object on which you call CreateSession.
- If the CORBA option is not selected, the object reference will not be added, the CORBA applet code thinks that it is running in the context of the Notes Client and tries to make native calls to the local Domino DLLs which causes an error.
- The NOI_COOKIE_URL parameter ensures single user login, i.e. the CORBA applet is not challenged for a Username/Password again. The cookie comes from the server when the client logs on.
- If the applet is running within the Notes Client, this check box option is ignored and the two extra parameters are not added and the applet will access the local Domino APIs. This may seem a security risk but the Security Options under the User Preferences menu option allows the User to define what applet security to enforce on whom.

The Object Reference String that is added to the applet Parameter list is generated on the Domino Server when starting the DIIOP task for the first time and created in the DIIOP_IOR.TXT file in the Domino HTML directory. If this file is somehow deleted, the file is regenerated again when the DIIOP task is re-started.

Applets at Runtime

The following figure shows the architecture of the CORBA/IIOP interface for Java applets.

Steps 2, 3, and 4 describe what happens when a Web browser interacts with a Domino server that includes a CORBA enabled applet.



Step 2

Domino serves a HTML page to the Web browser which includes the reference to embedded CORBA enabled Java applets. If the CORBA option in the applet properties is checked, the Domino server will automatically include the 2 extra parameters needed by CORBA, discussed above. The Web browser then retrieves the Java applet from the Domino HTTP server. This includes the applet classes and the NCSO.jar file which contains the Client ORB classes and the DOM Client Java Stubs.

Step 3

The applet is loaded into memory and the first invocation of the getSession( ) method instantiates and initializes the Client ORB. If more than one CORBA applet is loaded into the HTML page, each getSession( ) invocation will instantiate another Client ORB. Technologies such as the InfoBus, which allow applets to share information, could be used here to share the one Session object reference.

Step 4

Requests for instantiation of remote Server objects or method calls on existing remote objects are passed through the Client ORB back to the listening Server ORB via IIOP. Remember, all DOM implementation is on the server, so only data can be returned. The session between the applet and the remote server object will persist until either side decides to disconnect. If the server goes down during a request, objects will be discarded and the request must be repeated.

Domino and OLE Automation

OLE (Object Linking Embedding) is a feature of Windows and is also supported on the Macintosh. The OLE object model is used by developers to expose the objects of one product to another. OLE Automation is an OLE service for integrating applications. Two key elements of OLE Automation are the OLE Automation Server and OLE Automation Controller or Client. OLE Automation Servers expose applications' functionality as objects to other applications. These objects have properties (data) and methods (functions). OLE Automation Controllers can control objects in OLE Automation Servers through their properties and methods. Simply put, OLE Automation is the process in which an OLE Automation Controller sends instructions to an OLE Automation Server. You can call upon an OLE Automation Server object's code to perform a variety of tasks that you do not want to, or cannot, perform in your own code.

Domino can act both as an OLE Automation Server providing functionality to other applications and as an OLE Automation Controller where a Domino application integrates with functionality offered by an external OLE Automation Server application (for example a spreadsheet).

First we will cover how you create solutions using Domino as an OLE Automation Server and then we will describe how the Notes client can act as an OLE Automation Controller.

Accessing the Domino Object Model Using OLE Automation

To get access to the Domino Object Model through OLE Automation the Notes client must be installed on the workstation where the application code will execute.

As described in the introduction, OLE Automation exposes the full Domino Object Model to the OLE Client Application. This gives your OLE Client application the ability to implement a wide range of function where Domino handles such underlying complexity as networking, authentication, access control, and so on.

Some typical examples of Domino functionality utilized through OLE Automation are:

- Sending mail
- Storing information in Domino databases
- Integrating an external application with a Domino application

Client applications that utilize Domino as an OLE Automation Server are often spreadsheets and word processors in productivity suites like Lotus SmartSuite® and Microsoft Office, but they can also be specialized programs developed in C++ or Visual Basic.

We will first discuss in general how the OLE client connects to Domino objects and then we will walk through an example, look at some code snippets, and finally talk about runtime errors and how to debug them in Visual Basic

- Sending information from Excel using Domino
- More examples
- Runtime errors and debugging

Connecting to Domino Using OLE Automation

You can create a reference to Domino objects, like NotesDatabase, NotesACL, NotesLog, and NotesDocument, in your code. There are two "entry points" that you can use

- NotesSession

  Use NotesSession if you don't need to use elements from the Notes Client user interface — for example, for storing data directly in a Domino document or sending mail where your application supplies all necessary information

- NotesUIWorkspace

  Use NotesUIWorkspace if you want to integrate your application with a Domino application running in the Notes client — for example, to take advantage of data validation already found on a form in a Domino database or to have the Domino mail application handle addressing

Refer to the description of the Domino Object Model to see how you can get a reference to any object in the Domino Object Model (like NotesDatabase, NotesDBDirectory, NotesACL, NotesItem, and NotesUIDocument) from one of these two top level objects.

If you are using Visual Basic either as a standalone development tool or as the macro language in an application, when you declare a variable that references a Domino object you can manipulate the object using the object.property and object.method syntax in Visual Basic. This allows you to get and set values for Domino Objects and execute their methods in a simple way.

Note Remember to clear any variables that reference Domino objects once you are done using them so they can be released from memory. To clear a variable in Visual Basic set it to Nothing.

Sending Information From Excel Using Domino

Domino R5.0 supports using mail from Microsoft Office applications without any programming. However, if you don't want to send a whole spreadsheet, but just a small memo with a few essential numbers from the spreadsheet, you must write a small macro.

We will walk you through an example of how to create a button in an Excel spreadsheet that triggers a macro. This macro connects to Domino and creates a memo containing the content of a cell and then sends the memo off to a recipient group named Supervisors.

The way to do this may vary depending on the version of Excel you are using. If in doubt refer to the documentation in Excel on how to create a button that triggers a macro.

1. Start Excel and create a new blank spreadsheet.
2. Make sure the Forms toolbar is visible (View - Toolbars - Forms).
3. Click the Button in the Forms toolbar (if in doubt rest the mouse over the choice until the hover help appears — it should say "Button").
4. On the spreadsheet, click and hold the mouse button down while you drag the mouse to define the rectangle for the button. Release the mouse button when the size is right
5. Excel shows a dialog where you can assign a macro to run when your newly created button is clicked. You haven't created the macro yet so you do it now. Click the New button.
6. This opens a Visual Basic module in which you can write the macro. In the window where there are two lines like this

   ```
   Sub Button1_Click()
   End Sub
   ```

   Enter the following code:

   ```
   Sub Button1_Click()
   Dim session As Object
   Dim db As Object
   Dim doc As Object
   Set session = CreateObject("Notes.Notessession")
   'set db to database not yet named
   Set db = session.getdatabase("", "")
   'Now, set database to default mail database
   Call db.openmail
   'Get the value of cell D1 in this sheet
   Mycell = Sheet1.Cells(1, 4)
   ' create a mail document in Domino
   Set doc = db.createdocument
   ```

```
msg = "Mail has been sent: " & Date & "  " & Time & Chr(10)
& _        & "The value in Cell D1 [ " & Mycell & _
         " ] is in the body of the message"
Call doc.replaceitemvalue("SendTo", "Supervisors")
Call doc.replaceitemvalue("Subject", "Excel message")
Call doc.replaceitemvalue("Body", msg)'send the message
Call doc.Send(False)
MsgBox doc.getitemvalue("Body")(0)
MsgBox session.UserName
Set session = Nothing  ' close connection to free memory
End Sub
```

7. Close the Visual Basic Editor window.
8. On the spreadsheet, type some number in cell D1 (e.g. 1-2-3).
9. Click the button you just programmed. The macro does the following:
   - Connects to Domino (NotesSession)
   - Opens the mail database of the current user (that is you)
   - Creates a new document in the mail database
   - Reads the value of cell D4 and constructs a message containing that value
   - Writes the message together with recipient and subject to fields in the new mail document and sends it off to the people in the group Supervisors
   - Then the macro reads the value of the Body field from the mail document it just created and displays that value in a message in Excel
   - The macro also gets the name of the current user from Domino and displays that to the user
   - Finally the macro destroys the connection to Domino to free up memory

Here are a few things to note about the code:

Domino Objects Are Declared as Objects in Visual Basic

```
Dim session As Object
Dim db As Object
Dim doc As Object
```

To access the Domino objects they must be declared as OLE Objects in Visual Basic, while they are declared as their native classes in LotusScript (e.g., Dim db As NotesDatabase). If copying LotusScript code to a Visual Basic environment you must change these declarations.

As the Domino objects simply are declared Object in Visual Basic we cannot use the New method of the Domino classes as we can in LotusScript. Instead, we have to derive our Domino object from the NotesSession or NotesUIWorkspace object. If you have the following code in LotusScript,

```
Dim db As New NotesDatabase(""; "test.nsf")
```

and want to use it in Visual Basic you must change it like this:

```
Dim db As Object
......
Set db = session.GetDatabase("";"test.nsf")
```

Use CreateObject to Connect to the Top Level Domino Object

To access all the back-end classes in the Domino Object Model we must use this code:

```
Set session = CreateObject("Notes.NotesSession")
```

This creates a reference to the NotesSession object that is created externally. From the session object we can get to any back-end class.

Access and Create other Domino Objects Using the Domino Object Hierarchy

Here you see how to get to the NotesDatabase and NotesDocument classes through the session object:

```
Set db = session.getdatabase("", "")
............
Call db.openmail
............
Set doc = db.createdocument
............
Call doc.replaceitemvalue("Subject", "Excel message")
............
Call doc.Send(False)
```

Using the session object we can get a database instance and set it to be the current user's mail database by invoking the Openmail method.

Using the database object you can create a new document using the CreateDocument method.

Finally when you have the document object you can create or replace fields in the document with the ReplaceItemValue method and you can mail the document with the Send method.

More Examples

Here are other examples illustrating different ways to access Domino objects from Visual Basic

Example 1

This example accesses the current document in the Notes client:

```
Dim workspace As Object
Dim UIdoc as Object
set workspace = CreateObject ("Notes.NotesUIWorkspace")
set UIdoc = workspace.CurrentDocument()
```

To access the current document (NotesUIDocument object) in Visual Basic, you first need to get a reference from NotesUIWorkspace. Then you use the CurrentDocument property of the NotesUIWorkspace class to get the reference to the NotesUIDocument class. You declare workspace as an object which references NotesUIWorkspace. You declare UIdoc as an object which references NotesUIDocument.

```
Dim workspace As Object
Dim UIdoc As Object
```

Create an externally creatable object.

```
set workspace = CreateObject ("Notes.NotesUIWorkspace")
```

Use the externally creatable object to access a lower-level object doc.

```
set UIdoc = workspace.CurrentDocument()
```

Example 2

This example accesses the view called "By Author" in the Domino database test.nsf.

```
Dim session As Object
Dim db As Object
Dim view As Object
Set session = CreateObject ("Notes.NotesSession")
Set db = session.GetDatabase("","test.nsf")
Set view = db.GetView ( "By Author")
```

To access the NotesView class, you get the reference to NotesView from the NotesDatabase class. Since NotesDatabase is not an externally creatable object, you will get its reference from a higher-level class of the hierarchy, which is NotesSession. You get the reference to NotesDatabase using the GetDatabase method of the NotesSession class. Then you get the reference to NotesView using the GetView method of NotesDatabase.

We declare session as an object which references NotesSession.

```
Dim session As Object
```

We declare db as an object which references NotesDatabase.

```
Dim db As Object
```

We declare view as an object which references NotesView.

```
Dim view As Object
```

We get the reference to NotesSession by using the CreateObject function of Visual Basic, and store it in session.

```
Set session = CreateObject ("Notes.NotesSession")
```

We get the reference to NotesDatabase using the GetDatabase method of NotesSession, and store it in db.

```
Set db = session.GetDatabase("","test.nsf")
```

We get the reference to NotesView and store it in view.

```
Set view = db.GetView ( "By Author")
```

Example 3

This example shows how to access NotesItem in the current document in the Notes client:

```
Dim workspace As Object
Dim UIdoc As Object
Dim doc As Object
Dim item As Object
Set workspace = CreateObject ("Notes.NotesUIWorkspace")
Set UIdoc = workspace.CurrentDocument
Set doc = UIdoc.Document
Set item = doc.GetFirstItem ("Subject")
```

The item you are accessing is an item in the current document. So you need to get the document reference from NotesUIDocument. This is not an externally creatable object. Therefore, we first get the reference from NotesUIWorkspace to get the reference to NotesUIDocument and then to the other lower-level classes.

We declare workspace, UIdoc, and doc as objects.

```
Dim workspace As Object
Dim UIdoc As Object
Dim doc As Object
```

We declare item as an object which references NotesItem.

```
Dim item As Object
```

We get the reference from NotesUIWorkspace and NotesUIDocument.

```
Set workspace = CreateObject ("Notes.NotesUIWorkspace")
Set UIdoc = workspace.CurrentDocument
```

We get a reference to NotesDocument using the Document property of NotesUIDocument, and store it in doc.

```
Set doc = UIdoc.Document
```

We get reference to NotesItem using the GetFirstItem method of NotesDocument to get the first Item named "Subject."

```
Set item = doc.GetFirstItem ("Subject")
```

Note When using Notes classes through OLE automation in Visual Basic, for some methods, you must use type numbers in arguments rather than type constants which are used in LotusScript. However, you can always get the type number using "Messagebox Cstr(TypeConstant)" in LotusScript. For example, in LotusScript you use the type constant READERS to add a readers field item to a document:

```
Dim NewANames As New NotesItem(doc,"DocReaders",array,READERS)
```

To add a readers field through OLE Automation first get the type number in LotusScript:

```
Messagebox Cstr(READERS)
```

When you execute the code a messagebox with the number 1075 appears. Thus in your OLE Automation code you must enter:

```
Dim NewANames As New NotesItem(doc,"DocReaders",array, 1075)
```

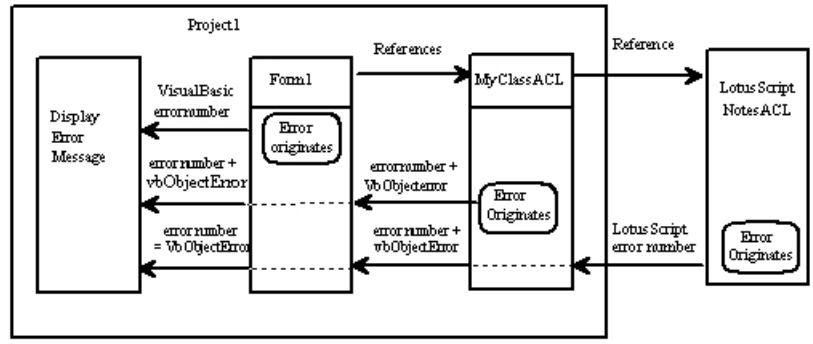
Runtime Errors and Debugging

Run-time errors occur (and are detected by Visual Basic) when a statement attempts an operation that is impossible to carry out.

When you use Domino object classes in Visual Basic the error-handling code is especially important because code from the Domino Object Model is used remotely from within your Visual Basic application. Where possible, you should include code to handle errors that Domino objects may generate. When there is a run-time error generated in a Domino object, Domino handles the errors it can handle, and regenerates the errors which it cannot handle. Visual Basic will automatically remap untapped errors, and the error-handling code in your application is needed to handle the errors.

In applications that use Domino objects and derived classes based on Domino objects, it becomes more difficult to determine where an error occurs.

The following figure shows a Visual Basic application that consists of a form module, which references a class module, which in turn references a Domino ACL object.



Regenerating Errors Between Forms, Classes, and OLE Automation Domino Objects

Domino handles the part of the error arising in the Domino object. If Domino cannot handle a particular error arising in the NotesACL, but regenerates it instead, Visual Basic will pass the error to the referencing object, MyClassACL. Visual Basic automatically remaps untapped errors arising in objects outside of Visual Basic as error code 440.

The MyClassACL object can either handle the error (preferable), or regenerate it. The OLE interface specifies that any object regenerating an error that arises in a referenced object should not simply propagate the error (pass it as error code 440), but should instead remap the error number to something meaningful. When you indicate the error condition, if your handler can determine what the error is, it should map it to an undefined error number. Add the new number to the intrinsic Visual Basic constant vbObject Error to notify other handlers that this error was raised by your object (MyClassACL).

Whenever possible, a class module (MyClassACL module in our case) should try to handle every error that arises in the object. It references errors that are not handled by that object. However, there are some errors that it cannot handle because it cannot anticipate them. There are also cases where it is more appropriate for the referencing object to handle the error, rather than the referenced object.

When an error occurs in the form module, Visual Basic raises one of the predefined Visual Basic error numbers as described in the online Help.

When you regenerate an error, leave the error object's other properties unchanged. If the raised error is not trapped, the Source and Description properties can be displayed to help the user take corrective action.

A class (MyClassALC) module might include the error handler to accommodate any error it might trap, regenerating those it is unable to resolve.

Debugging a Visual Basic Application That Uses OLE Automation

When you are debugging an application that has a reference to an OLE automation object or a class defined in a class module, you may find it confusing to determine which object generates an error. To make this easier, you can select the Break in Class module option on the Advanced tab of the Options dialog box (available from the Tools menu). With this option selected, an error arising in a form or standard module will invoke an error handler, if one is available. An error in a class module or an object in another application that is running in another instance of Visual Basic will cause that class to enter the debugger's break mode, allowing you to analyze the error. An error arising in a compiled object will not display the debug window in break mode; rather, such errors will be handled by the object's error handler, or trapped by the referencing module.

Accessing Other Applications From Notes Using OLE Automation

Just as other applications can utilize Domino functionality through OLE Automation so can the Notes client utilize functionality in applications that are OLE Automation Servers. An OLE Server application can provide services that would require extensive effort to develop in Domino, such as the cell engine from Excel or 1-2-3 or the sophisticated text handling capabilities of Word or Word Pro.

You can embed the OLE Application and its data in a document and let Domino serve as the container, providing replication, security, and access control as well as summarizing the data in views and making it possible to search for information using the Domino Full text indexing capabilities. Domino provides the very best container for these objects, as it frees users from having to learn the hierarchical file system (all Domino documents, even those with objects in them, can have a word-based title). In addition, the Domino development capabilities and integrated messaging services make using OLE applications as part of a workflow application fast and easy.

You can also use an OLE application that is external to Domino without storing the application or its data in Domino, for example using Automation your program may transfer some data from Domino to a spreadsheet to create a chart that is printed using the print engine in the spreadsheet.

Classes of OLE Objects

Before giving examples of how to control OLE Automation servers we will discuss how an OLE application often exposes several classes and how those classes exhibit different behavior.

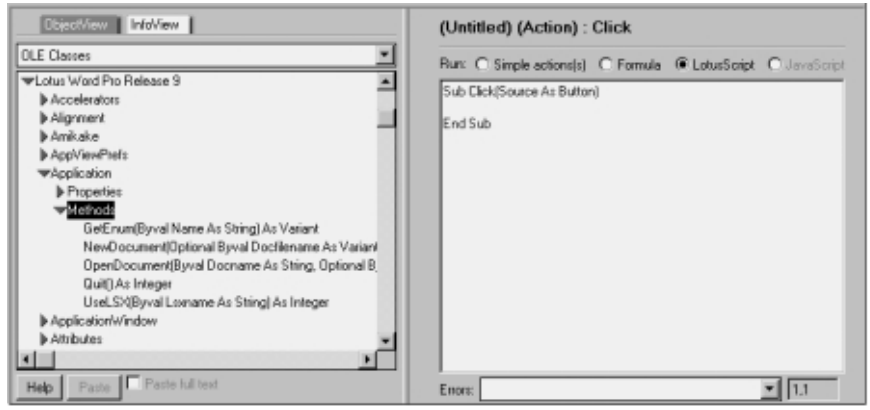
When an OLE Application is installed, the names of all the object classes it exposes for OLE Controllers to utilize are written to the Windows Registry. For example, Lotus Word Pro registers the following classes during installation:

- Word Pro.Application
- Word Pro.Document
- Word Pro.SmartMaster

Some of the classes define objects that contain other objects defined by other classes in the application. A Word Pro.Application object can contain a number of Word Pro.Document objects. They define different behavior. A Word Pro.Document object has methods like .Print, .Save, and .Close; but if you want to create a new document, you must access the Word Pro.Application object and its .NewDocument method. Using the objects' properties you can get to their parent objects or the objects they contain. From a Word Pro.Document object, you can get the handle of the Word Pro.Application object through the .Application property of the document. From a Word Pro.Application object, you can get a handle to the active document using the .Active Document property, and so on. Some OLE objects are also aggregations of several OLE classes. This means that you can get properties and execute methods defined in different classes on the same object. Refer to the documentation for specific applications for further information on this.

Not all OLE classes in an application define objects that can be embedded in a Domino document. While you can embed a Word Pro.Document object, you cannot embed a Word Pro.Application object. However, you can access the properties and methods of a Word Pro.Application object *from* an embedded Word Pro.Document object. It is important that you understand what class of object you are working with and how you can move to its parent or one of its siblings through the object properties.

When you are working with LotusScript in Domino Designer, you can see properties and methods of all registered OLE Classes by going to the reference tab in the InfoList and selecting OLE Classes in the listbox.



For more details, refer to the documentation that comes with the application you want to Automate.

LotusScript Functions and Methods to Use for OLE Applications

When working OLE applications in LotusScript you have several options to choose from. First we will briefly mention the functions and methods you can use. Then read on for some actual examples of OLE Automation.

CreateObject Function

Creates an OLE Automation object of the specified class.

```
CreateObject ( className )
```

You get the className (like "Word Pro.Application") from the application documentation.

Note OLE Automation objects created with the CreateObject *function* are external to Domino. They cannot be embedded in documents. You must use the CreateObject *method* of the NotesUIDocument class to embed objects.

GetObject Function

Opens an OLE Automation object contained in an application file, or returns the currently active OLE Automation object of the specified class.

```
GetObject ( pathName [ , className ] )
```

Specify either a pathName or a className. If you specify a className you will get a handle to the current object of the specified class. Using a pathName as parameter is similar to double-clicking the file name in the

Explorer, with the added value that you get a handle to the object. As with the CreateObject function, you cannot use this function for embedding objects in Domino documents.

CreateObject Method in NotesUIDocument Class

In a document in Edit mode, embeds an OLE object in the current rich text field.

```
notesUIDocument.CreateObject([name [, className [, pathName]]])
```

The name parameter is optional. If you name the embedded object you can later get to the object using that name. You must either specify a className or refer to an existing file with the pathName parameter. Refer to the OLE application documentation for which className to use, or see "How to Determine the Class Name of an Object to be Embedded" later in this chapter.

GetObject Method in NotesUIDocument Class

Given a name, returns a handle to the OLE object of that name.

```
notesUIDocument.GetObject( name )
```

You can use this to get to an OLE object embedded manually or by the CreateObject method in NotesUIDocument. Use the name specified when creating the embedded object. The name of manually embedded object is the name displayed in the listbox in the Create Object dialog. You can see the name of an embedded object by selecting it in the Domino document. Then an extra menu appears in the Notes client. The name depends on the type of embedded object (Worksheet, Document, Chart etc.). Click the menu and then select Object Properties from the menu. This opens the properties box for the object. On the first tab you can see the object name.

Note The Create Object and GetObject functions take different parameters than the methods with the same names.

EmbedObject Method in NotesRichTextItem

Depending on the parameters, it creates an attachment, an embedded object, or a link in a rich text field.

For embedding objects, this method works in the same way as the CreateObject method in NotesUIDocument, except that this method returns an instance of NotesEmbeddedObject that has methods for further manipulation of the embedded object.

GetEmbeddedObject Method in NotesRichTextItem

Given the name of a file attachment, embedded object, or object link in a rich text item, returns the corresponding NotesEmbeddedObject.

Activate Method in NotesEmbeddedObject

Causes an embedded object or object link to be loaded by its OLE application and returns a handle to the OLE object:

```
notesEmbeddedObject.Activate( show )
```

The show parameter is a boolean. True makes the OLE Application visible, while it is loaded invisible when the False parameter is used.

Using the Activate methods on an NotesEmbeddedObject combines getting a handle to the OLE object and executing the Activate method of the OLE object.

Note Even though Activate is listed as a method of the NotesEmbeddedObject class, it really is a method that is supported by the OLE class for the embedded object. You do not have to get an instance of NotesEmbeddedObject to be able to execute the Activate method. Any function or method that returns a handle to an OLE object can be used. Once you have the handle, you can use the Activate method against it.

DoVerb Method in NotesEmbeddedObject

Given the name of a verb, executes the verb in an embedded object.

```
notesEmbeddedObject.DoVerb( verb )
```

Embedded objects may allow you to issue verbs (like &Open and &Edit) against them. You can see which verbs an embedded object supports by right-clicking it to get its menu.

DoVerb does not return a handle to the embedded object, so if you want to use DoVerb for OLE Automation you must combine its use with the GetObject function.

Embedding OLE Objects

You can embed OLE objects in Domino documents using LotusScript or you can do it manually. OLE objects are stored in rich text fields.

Embedding an OLE Object Manually

To embed an OLE object manually:

1. Make sure the document is in edit mode and the cursor is in the rich text field where you want to place the object.
2. Select Create - Object. The Create Object dialog will be displayed.
3. You have options to embed either an ActiveX control, a new document for an installed OLE application, or an existing file created with an OLE application.

4. When you have made your selection and clicked OK the object is embedded and launched ready for you to work on it.

Depending on how the application supports OLE, it is either launched in-place or out-of-place. An application that is launched in-place stays in the window of the Notes client, and the menu bar is changed to include the menu options of the embedded application. If the application has to be launched out-of-place, or the user specifically has chosen to do so, it will open in its own application window.

Embedding an OLE Object Using LotusScript

If you want to program the embedding of an OLE Object in a document you must use LotusScript. Embedding of OLE objects cannot be done in formula language or JavaScript. In LotusScript you use the CreateObject method of the NotesUIDocument class (described above).

The current document must be open in edit mode and the cursor must be in an editable rich text field.

With no parameters, this method displays the Create Object dialog box, which allows the user to select the object to create.

You can use the handle returned to send commands to the application using OLE. If you want to do so you must make sure to assign the returned handle to a global variable, else the OLE application will close when the variable with the handle comes out of scope (as will happen if you assign it to a local variable, for example in a button click event).

If you want to embed an existing file as an object, you must use the GetObject statement to give LotusScript a handle on the object.

How to Determine the Class Name of an Object to be Embedded

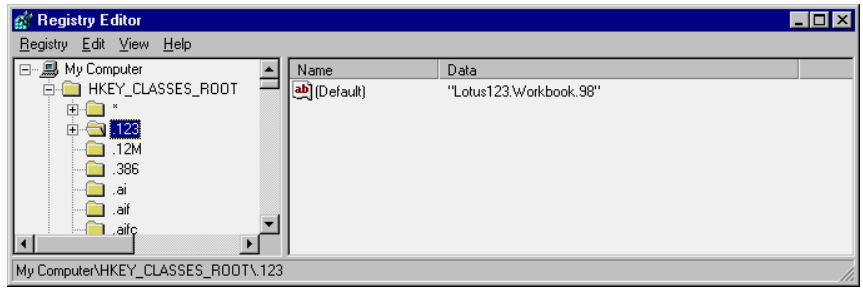
When selecting to embed an OLE object manually in the dialog listbox you will see the names of those classes that represent embeddable objects, but you only can see the “display name” of the class. You cannot see the ClassName in the format that is required for embedding a new object using LotusScript

To determine the name of an object class either refer to the documentation that came with the application or use the Windows Registry Editor.

You can use the Windows Registry Editor to determine the ClassName of an application like this:

1. Launch the Windows Registry Editor by typing “regedit” at a command line prompt.
2. Once the registry is open, expand HKEY_CLASSES_ROOT. A menu of keys associated with each registered application’s file extension displays.

3. Look for the appropriate key name — normally the file extension associated with the applications data files — for example the key for a Lotus 1-2-3 Workbook is ".123". Select that key to display the folder contents.
4. When the folder opens, the (Default) Name and Data display in the right window pane. The Data value displayed for the (Default) is the class name you use when embedding objects.



In this example you can see that the ClassName for a Lotus 1-2-3 Workbook object from the Lotus SmartSuite Millennium Edition is "Lotus123.Workbook.98."

Note An entry for an application object in the HKEY_CLASSES_ROOT section of the Registry does not confirm that an application supports OLE Automation. You must check the application's documentation for OLE Automation support.

To create a new blank Word document in the Body field of the current document in the Notes client, write a script like this:

```
Dim ws as New NotesUIWorkspace
Dim uidoc as NotesUIDocument
Set uidoc = ws.CurrentDocument
Call uidoc.GotoField( "Body" )
Call uidoc.CreateObject("Meeting Minutes", "Word.Document.8")
```

This will embed a new Word document in the rich text field named Body and launch it with Word.

Adding a Table to an Embedded Word Document

In the example above you did not get a handle to our OLE object because you just wanted to embed and launch the Word document. In the next example you can see how to use LotusScript to continue to work with the document after it has been created. The sample script embeds a Word document, creates a table in the document, and writes text in row 1, cell 2 of the table.

```
Dim ws As New NotesUIWorkspace
Dim uidoc As NotesUIDocument
Dim handle As Variant 'handle to the word document
Dim t As Variant               'OLE holder for the table
Dim r As Variant               'OLE holder for the range
Set uidoc = ws.CurrentDocument 'Get a handle to current doc
uidoc.EditMode = True 'Make sure document is in edit mode
uidoc.GotoField("Body")        'Goto the richtext field
Set handle = uidoc.CreateObject("","Word.Document.8")
Set r = handle.Range(0, 0)     'Create a range that points
                               'to the top left corner
Set t = handle.Tables.add(r,2,2)      'create a 2 by 2 table
                                      'ref'd by the range
Set r = handle.Range(1,2)      'Move into the table
Call r.InsertAfter( "Entering cell text using Automation")
```

This example can easily be expanded to get the text to write in the Word table from fields in documents in the Domino database, do some more formatting of the table, and finally print it using the print engine in Word.

Getting Data From an OLE Object Using the Domino Front-End Classes

You just saw how you can send data from Domino documents to an OLE Object. In the next example, you will see how the data can go the other way. You have an embedded Excel spreadsheet and you want to transfer the value in spreadsheet cell (1,1) to the field Num in our current Domino document:

```
Dim ws As New NotesUIWorkspace
Dim uidoc As NotesUIDocument
Dim handle As Variant
Dim CellValue as Variant
Set uidoc = ws.CurrentDocument
Set handle = uidoc.GetObject( "Microsoft Excel Worksheet")
CellValue =  handle.Activesheet.cells(1,1).value
Call UIDoc.FieldSetText( "Num",  Str(CellValue))
```

Note When the GetObject method is used, the object name is taken as the argument. Objects will have names by default when they are created. To check the name of an object, you give it focus. When the object is highlighted, it will appear as a menu choice. To see the object name, select Object - Properties.

Example — Running a Macro In Excel Using OLE Automation

This code shows how to run a macro in an embedded Excel spreadsheet.

```
Dim ws As New NotesUIWorkspace
Dim uidoc As NotesUIDocument
Dim doc As NotesDocument
Dim v As Variant
Dim eo As NotesEmbeddedObject   'May have to be global
Dim handle As Variant
Set uidoc = ws.CurrentDocument
Set doc = uidoc.Document
Set v = doc.GetFirstItem("Body")
'This tries to get a handle to the richtext field
If Not (v Is Nothing) Then
   'This is error checking if the is no richtext object
   Set eo = v.embeddedobjects(0)
   'Get a handle on the embedded object if there is one
   If Not (eo Is Nothing) Then
               Set handle = eo.Activate(True)
              handle.Application.Run("CopySummary")
         Else
              Msgbox "There is no embedded object"
        End If
        Exit Sub
     End If
     Msgbox "There is no richtext item"
End Sub
```

This code runs the macro "CopySummary" in the embedded Excel 97 spreadsheet. Often a macro is dependent on whether the spreadsheet is visible — so remember to check whether you should activate it with the visibility parameter set to true.

Note If you place this code in a button and want to continue working with the activated spreadsheet after the code has executed, you must place the declaration of the object variable in the globals section. This line should be placed in globals:

```
Dim eo As NotesEmbeddedObject
```

If you declare the object within the scope of the button, the object variable will be destroyed when the script has run, and the spreadsheet will close.

Using OLE Automation Without Embedding

If you want to utilize functions in an OLE Server Application without embedding it in a document, you use one of these LotusScript functions:

- CreateObject
- GetObject

They allow the OLE Automation Controller to access the objects of the OLE Automation Client. The function, GetObject, is used to open a specific existing file in the application. Both functions return an OLE Automation object reference. The Set statement is required to assign this object reference to a Variant variable type.

Caution CreateObject and GetObject are available as *functions* in LotusScript, as well as well as *methods* of the class NotesUIDocument. They don't take the same arguments. Use the methods to work with OLE objects embedded in a Domino document and the functions to control OLE applications outside of the Notes client.

Examples of Automating Lotus SmartSuite Applications

Lotus SmartSuite applications use the following object names to expose their objects for OLE Automation:

- Approach.Application
- Freelance.Application
- Word Pro.Application
- Lotus123.Workbook

Below you will see examples of how to automate SmartSuite applications external to Domino.

All the sample code uses the CreateObject function to associate variables and object references returned. The function, GetObject, could be substituted for CreateObject to open an existing file. For example:

```
Set xl = CreateObject("Lotus123.Workbook")  'Loads 123
Set xl = GetObject("c:\lotus\123\work\123\sheet.123", _
"Lotus123.Workbook")  'Loads 123 and opens the file sheet.123
```

The SmartSuite applications Visible property is set to false by default. This is usually the way an OLE Automation application would work. However, for these examples the Visible property has been set to true. This is a good practice for designing and debugging OLE Automation applications.

1-2-3 Example

This code example instantiates a 1-2-3 Object and sets the object to a Variant variable. The application's Visible property is set to true. A new 1-2-3 workbook is created and set. A range is set. The contents of the first cell on the sheet is set to a string. The user is prompted to close the application with a messagebox. The application is then closed.

```
Dim  w123 As Variant
Dim doc As Variant
Dim r As Variant
Set w123 = CreateObject("Lotus123.Workbook")
w123.Application.Visible = True
Set doc = w123.Application.NewDocument
Set r = doc.Ranges("A:A1..A:B10")
r.Cell(0, 0, 0).Contents = "Hello 123 from OLE Application"
MessageBox "Close 123"
w123.Application.Quit
```

Word Pro Example

This code example instantiates a Word Pro Object and sets the object to a Variant variable. The application's Visible property is set to true. A new Word Pro document is created and set. A text string is inserted into the document. The user is prompted to close the application with a messagebox. The application is then closed.

```
Dim wp As Variant
Set wp = CreateObject("Word Pro.Application")
wp.Visible = True
wp.NewDocument ("WPTest.lwp")
wp.Text.INSERTTEXT ("Hello Word Pro from OLE Application")
MessageBox "Close Wordpro"
wp.Application.QUIT
```

Approach Example

This code example instantiates an Approach Object and sets the object to a Variant variable. The application's Visible property is set to true. A new Approach document is created (OpenDocument creates a new APR file if passed a DBF file for the first parameter) and set. The active view name is set to a string and will appear on the active tab. The user is prompted to close the application with a messagebox. The application is then closed.

```
Dim ap As Variant
Dim doc As Variant
Set ap = CreateObject("Approach.Application")
ap.Visible = True
Set doc = ap.OpenDocument("videos.dbf", ap.Path & "demo")
ap.ActiveView.Name = "Hello Approach from OLE Application"
MessageBox "Close Approach"
ap.Application.Quit
```

Freelance Example

This code example instantiates a Freelance object and sets the object to a Variant variable. The application's Visible property is set to true. A new Freelance document is created and set. The title textblock on the active page is set to a string which causes the textblock to remain in Edit mode. The textblock LeaveEditMode method is called. The user is prompted to close the application with a messagebox. The application is then closed.

```
Dim fl As Variant
Dim doc As Variant
Set fl = CreateObject("Freelance.Application")
fl.Visible = True
Set doc = fl.NewDocument
doc.ActivePage.Title.TextBlock.Text = "Hello FLW From OLE"
doc.ActivePage.Title.TextBlock.LeaveEditMode
MessageBox "Close Freelance"
fl.Application.QUIT
```

Note The help documentation, *Developing SmartSuite Applications Using LotusScript*, that ships with SmartSuite explains OLE Automation in more detail. This help documentation must be selected during the install of SmartSuite as, by default, it is not installed. If you did not install it during your initial install of SmartSuite, you can install it using the Custom Install method.

Troubleshooting

During your work with OLE Automation you may encounter error messages. This section describes some error messages and what you can do to troubleshoot the problem further.

"Cannot Create Automation Object" or "Automation Object Error" Using CreateObject Function

When you run a script that contains the CreateObject function, one of the following errors occurs:

- Cannot Create Automation Object
- Automation Object Error

Use the following steps to troubleshoot the error:

1. Ensure that the application supports OLE Automation. To do this, check the documentation for the application.
2. Make sure you have ample disk space available. (It is recommended that you have at least 10MB of free disk space.)

3. Using RegEdit, check whether you are using the proper syntax to create the object. See "How to Determine the Class Name of an Object to be Embedded" earlier in this chapter.

   Note An entry for an application object in the HKEY_CLASSES_ROOT section of the Registry does not confirm that an application supports OLE Automation. You must check the application's documentation for OLE Automation support.

4. After confirming that an application does indeed support OLE automation, try the CreateObject function for another application (such as Visual Basic). If the same error is generated, there is likely a problem with the Registry. You could then take the following steps:

   - Reinstall the application.
   - If reinstalling does not eliminate the error, you may choose to back-up your Registry file, delete all references to the application and then reinstall the application. For directions on backing up your Registry, refer to the Microsoft Windows 9x/NT documentation and the Microsoft Web site.

"Instance Member Does Not Exist"

This message appears if the OLE object that you have got a handle to does not support the method or property that you are trying to work with.

You may have misspelled a property or a method. The error can also occur if the OLE Automation Server application has been upgraded and the developer has changed its underlying OLE object model. For example, when MS Office 95 was upgraded to MS Office 97 the behavior of Excel changed. For the sample program below the following error would appear after the upgrade:

```
"Instance member CELLS does not exist"
```

The original program was like this:

```
Dim rtitem as NotesRichTextItem
Dim object as NotesEmbeddedObject
Dim handle as Variant
'...set value of rtitem...
Set object  = rtitem.EmbedObject( EMBED_OBJECT, _
"Microsoft Excel Worksheet", "", "Report" )
Set handle = object.Activate ( False )
handle.Cells( 1,1 ).Value = 100   'THIS LINE CAUSES THE ERROR
handle.Parent.Save
```

The object model had been changed. Formerly, you would get an object that supported the Cells property when you activated the embedded object. In the upgraded version this was no longer true. The script had to be changed as follows:

```
Dim rtitem as NotesRichTextItem
Dim object as NotesEmbeddedObject
Dim handle as Variant
'...set value of rtitem...
Set object  = rtitem.EmbedObject( EMBED_OBJECT, _
"Microsoft Excel Worksheet", "", "Report" )
Set handle = object.Activate ( False )
handle.Application.Cells( 1,1 ).Value = 100        'CHANGED*
handle.Parent.Save
```

"The Object's Application or Class is Not Available"

This message may appear when trying to open an embedded object for editing. This may be caused by having different versions of the OLE Automation server application installed on different workstations.

For example, an embedded object has been edited or created with a MS Office 97 application. If you try to open it with a MS Office 95 application you will get the message "The object's application or class is not available."

Things to Check When You Are Stuck

There are many things that can influence how well an OLE Automation application works. If you have a problem in your application and are having trouble resolving the problem, remember to check the following as well:

1. Does the problem exist on more than one machine? Focus on troubleshooting one machine only. Configuration issues (registry problems, memory, crashing, and so on) can be addressed only on one machine at a time.
2. Can you create a new object in Notes using the same OLE server application? If so, then the OLE server application is registered correctly. Attempt to embed the object in question into a new Notes document. Try to narrow down whether the problem is form-specific, document-specific, database-specific, Notes server-specific, or object-specific.
3. Can you create a new object in another OLE client?
4. Is the Notes document in Edit mode before you launch the object?

   The Notes document must be in Edit mode before launching the OLE object; otherwise the following error will occur: "Changes will not be saved in a read-only document".

Considerations Before Using OLE Embedding/Automation

As we have already described, OLE Automation enables sophisticated functions to be added to an application without having to write everything from scratch.

However, it is important to understand that there are Pros and Cons to the use of OLE Automation and the way that you use it in your application.

Some points to consider are described in the following section.

Can I Control What Version of an OLE Application the User Has Installed?

This is especially important if application data is to be stored as embedded objects in Domino documents. If all users are not upgraded at the same time and the underlying object model in the OLE application has changed, this could mean that some users (with the previous version of the OLE Application) cannot get to the data in the embedded objects. Also make sure that the class names you use for creating new embedded objects are not hard coded into the application, but are being fetched from profile documents or similar sources.

The Size Impact of Embedded Objects

A Domino document with an embedded object will be considerably larger than a similar document that contains the corresponding data in its own fields. Normally this will not impact performance very much, but if you have users on lines with low bandwidth this can affect the response time when opening documents, even just for reading. If you are comfortable with the inner workings of OLE Automation in your server application, you can just store the raw data in the Domino document. Then launch the OLE application at the user's workstation and build the object on-the-fly as well as fetch the processed data back to the Domino document using OLE Automation. Also, consider how much disk space needs to be reserved for your application.

Time to Launch External Application

If the OLE Automation Server application isn't already in the workstation's memory it can take considerable time to launch it. Make sure that the users don't have to wait for the launching application if it only gives them 'nice-to-have' functionality. Most users prefer to miss a few advanced functions if they can get their daily work done in a fast and efficient manner. Also, be careful with the OLE AutoLaunch options for forms. Let the users control when they want to launch an application whenever possible.

The LSX Toolkit

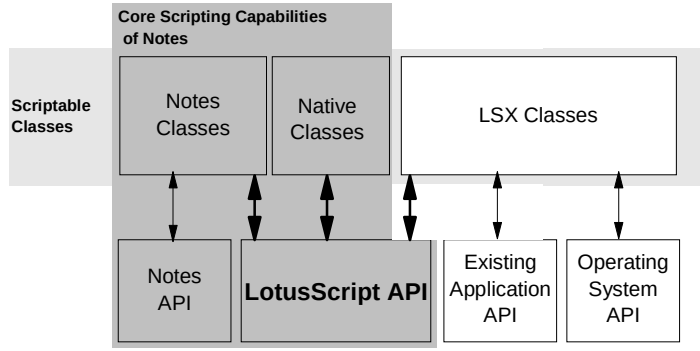
What Is an LSX?

An LSX (Lotus Software eXtension) is a dynamic library of objects (or classes) written in the C++ programming language. You can use these objects from languages and language interfaces like Java, CORBA, OLE, and LotusScript. You can treat LSX objects just like any other Domino objects. For example, you can create new objects from their classes, invoke methods, and get their properties.

The source programming language of LSXs is C++, which enables you to use APIs of some other applications. After an LSX is loaded by Domino, the LSX registers its C++ class definitions as corresponding LotusScript classes. This means an LSX extends the functionality of LotusScript running in Domino, because it enables any Domino application to connect to resources and functionality of external applications.

Note You may write parts of your LSX code in other programming languages and develop only the interface in C++.

The following figure shows the extended LotusScript capabilities introduced by LSXs:



Using an LSX

Several LSXs are available for access to relational database management systems, as well as for transaction system integration. If none of the existing LSXs fits your needs, you have the option of developing your own LSX. You are supplied with a LotusScript Extensions Toolkit that facilitates the mapping of your C++ code to LotusScript classes.

You can use all the classes of an LSX in your LotusScript event scripts by putting the USELSX statement in the "Options" event of the "Globals" definition section of a script.

Note The USELSX statement offers you two options. You can pass the LSX name as a library filename including the full path (USELSX "C:\MYLSX\SAMPLE.DLL"), or you can use the name that is associated with that LSX in the LSX class registry. For example, if the LSX is registered as MYSAMPLE=C:\MYLSX\SAMPLE.DLL, the statement looks like USELSX "*MYSAMPLE". You may prefer the latter because it is much more location independent.

As soon as you leave the "Globals" section, Domino loads and registers the LSX. All new classes are now available for your event scripts. LotusScript performs a type-check of all references to the new classes in your script against the registered class definitions. Furthermore, when you select the Reference tab in the Programmer's Pane in Domino Designer, and select Domino:classes in the combo box, the registered LSX classes are displayed, including their properties and methods.

If you declare an LSX twice, using the USELSX statement, LotusScript will use the LSX library that is already loaded.

Finally, it should be noted that an LSX is not very tightly coupled with Domino. It only interacts with the LotusScript interpreter embedded in Domino to provide the functionality. For that reason, you may use an LSX in any Lotus product that supports a LotusScript interpreter of Release 3 or higher.

Using the LSX Toolkit

This section describes the LSX Toolkit. For more details about its architecture refer to the section Architecure and Advanced Design later in this chapter.

Overview

The LSX Toolkit is a software development environment that enables you to implement new LSXs in the C++ programming language.

When Do You Need the LSX Toolkit?

There are certain situations that may lead you to develop your own LSXs:

- Need to access external applications.

  For example, an LSX may define classes to access a specific DBMS, a document management system, and so on.

- Need to access part of an operating system.

  For example, an LSX may define specialized classes to gain access to system resources such as the window system or communication facilities.

- Need to implement an algorithm where efficiency or code-size requirements make it undesirable to implement in LotusScript.

  In this case, the LSX is not being used to script an extra application, but as an alternative to using LotusScript itself. The LSX classes are an alternative (or a supplement) to LotusScript native classes.

- In cases where you wish to preserve Domino as a single user environment while employing multiple applications.

Software Prerequisites

In order to develop and test an LSX with reasonable efficiency using the Toolkit, the following software is required on your workstation:

- One of the following supported development platforms:

  Windows 95/98, Windows NT (on both the Intel and DEC Alpha architectures), OS/2 Warp, HP-UX, Sun SPARC Solaris, Solaris Intel Edition, AIX, OS/400, or OS/390.

- A standard C++ development environment for the development platform. This includes a C++ compiler, a C++ debugger, the platform's linker, and a Make utility.

  Note For more detailed information about hardware and software requirements, refer to the *LSX Toolkit Guide* (the online documentation included in the Toolkit).

- The installed LSX Toolkit.

- The application to be scripted.

What the LSX Toolkit Contains

The LSX Toolkit includes:

- Header files

  These files contain definitions you need to build your LSX.

- C++ base class source code

  Usually, this code will be used unmodified in each LSX.

- Session class template code

  This is intended to be used in your LSX sources.

- Source code for LSX examples

  There are working examples. Most of these compile and run on most of the Tookit-supported platforms.

- LotusScript source files

  These are header files that define the LSX API for the LSX builder.

- The LSX Wizard

- Build tools and testing tools

Each platform subdirectory contains the executables and libraries for a particular development platform. The same applies to the OBJS subdirectories contained in each of the SRC subdirectories. They store the compiled object modules of the sources.

Note If you are developing for a single platform, only one of those platform directories is of interest. On the other hand, if you install the Toolkit on a network file system accessible from multiple development platforms, this directory structure serves as a basis for cross-platform development, since you can produce different library formats from the same source code.

Utilities for Building and Testing

The C++ build tools comprise makefiles for all supported development platforms as well as front end DO_IT batch-file build utilities.

Certain special LSX build tools are furnished as executable files in the platform-specific subdirectories of \LSX\BIN:

- LSXTEST, a GUI test frame for writing, running, and debugging LotusScript scripts, including running LSX modules.
- LSXRUN, a command-line test frame for testing LotusScript scripts, including running LSX modules. It does not depend on any graphical user interface.
- LSXREG, for registering your LSX with the platform's class registry.

Creating an LSX

This section describes how to create a new LSX using the LSX Wizard.

The LSX Wizard consists of a Domino database and an LSX. The Wizard compiles information about the LSX you want to create and generates a near-complete set of buildable files. This allows you to develop an LSX very quickly.

The LSX Wizard has been completely redesigned in the LSX Toolkit Release 3.0. Here is some of the new features and enhancements:

- **Code regeneration**

  The Wizard will preserve existing user code if you modify, add or delete elements of an existing LSX and then click "Generate LSX Files" again.

- **New user interface**

  The new interface is much more intuitive and allows the editing of all elements specified in the Wizard.

- **Long file name support**

  The names provided for projects and classes will be used to name files generated by the Wizard. Users will no longer be restricted to the old 8.3 file names.

- **Generated method stubs look like regular C++**

  It is no longer necessary to understand the LotusScript internal data structures or architecture. All dispatching and argument parsing code is moved out of your way and into a separate file that should not need any modification. The generated method stubs handle arguments and return values in the standard C++ way.

- **Improved argument support**

  Allows for specification of argument names for methods that will appear both in your code and in the browser, allows for specification of optional arguments, and specifies that arguments be const when passing by reference.

- **Support for additional datatypes**

  It is no longer necessary to declare your arguments or return value as Variant in order to use dates and arrays. Dates now show up in the drop-down list with all the other datatypes. For arrays, there is a checkbox to indicate that the item is an array of the selected type. Generated code for array types now uses new array classes, so you don't need to have an in-depth understanding of the LotusScript data structures. The Boolean data type is also supported.

- **New classes to simplify access to the LotusScript datatypes**

  New classes have been added to support strings, dates, arrays, and Domino Objects (Notes back-end classes). The Wizard generates code that uses these new classes instead of the raw LotusScript data structures.

- **Option to not expose the "new" method**

  This feature is useful for the creation of abstract classes and for classes that should only exist in the context of a container class. (An object of this kind is created by a call to a "Create" method on the container class.)

- **Generated code for constants**

  No more need to manually expose constants in LotusScript. Specify them in the Wizard instead and it will do all the work for you.

- **Generated Microsoft Visual C++ project files**

  Instead of building your LSX from the command line, on Win32 platforms, you can insert the generated project file (.dsp) into your Microsoft Visual C++ workspace.

The LSX Development Process

The development process for an LSX consists of four steps. The LSX Wizard gives support for the first two steps:

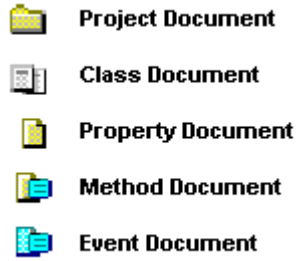
1. Create a set documents in the LSX Wizard database that represents object model of your LSX.
2. Generate LSX files.
   The Wizard LSX does this when you invoke the action "Generate LSX files" in the Wizard user interface.
3. Edit the generated source files.
   You may use any text editor for this task.
4. Compile and test the LSX.
   You can use the generated MAKE files or the provided DO_IT batch file.

Using the LSX Wizard

The LSX Wizard Database is a user interface designed for specifying the object model of your LSX Project. An LSX Project is a collection of documents which include one project document, at least one class document, and several property, method and event documents.

These documents are represented in a hierarchical Notes view. A project can contain several classes. Each class can contain several property, method and event documents. Since the property, method and event documents are at the same view level, a user can distinguish the type of document by the

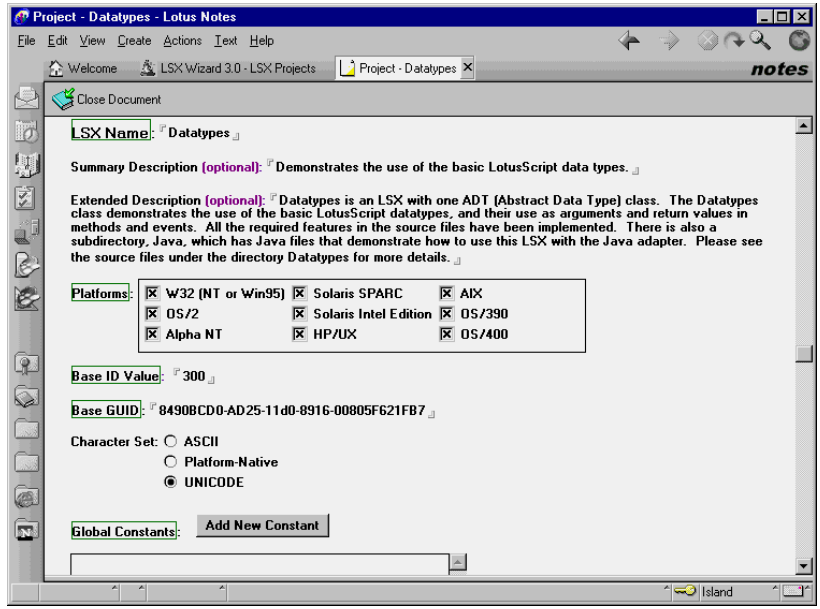
signature displayed in the view. Another distinguishing factor is the unique icon that represents each type of document in the view.



In this section we will review some documents from the Datatypes sample LSX. The Datatypes sample is a good resource for an LSX developer, because it contains a myriad of design features.

Project Document

To create a new project in the view, click the Project action button. This opens the Project form:



1. Enter the name of your LSX.

   The name you specify here has three roles. It becomes:

   - The Project name in the LSX Projects view

- The source-file directory name (e.g., lsx\src\Datatypes) in the file system on the workstation hard disk
- The filename of the DLL or other shared library that is the built LSX.

2. Provide a description (optional).
3. Specify the target platform of your LSX.
4. Enter a Base ID value and a Base GUID. The GUID must be unique for every class. LotusScript uses it as an identifier for the class.

   Note On a Windows platform, the Wizard uses the Windows API to generate the GUID. If a GUID-generating utility is not available, the Base GUID field will be blank. In this case you have to fill in a GUID that you have previously obtained.
5. Select the character set type. This field specifies how to represent strings passed into and out of the LSX. In most cases you will keep the default ASCII.
6. Define global constants (optional). To define constants, click the Add New Constant button, and enter the name and the constant's value; or enter/paste them into the global constants listbox, like this:

```
LS_RED      25
```

Mapping the Project Document to the C++ Generated Code

As mentioned above, the name provided in the project document will be the name given to the filename of the DLL or other shared library that is the built LSX; and a directory used for the generated project files will be created with this name.

Significance of the Project Document in LotusScript

Since the project name is the name given to the LSX, it will also be the name passed to the USELSX statement. The USELSX statement is used to load an LSX.

For example:

```
USELSX "c:\lsx\bin\w32\Datatypes.dll"
```

Class Document

To define a class, select a project in the view and click the Class action button. This opens the Class form:

1. Enter the class name.
2. Specify the containing class. The default value is Default, which means that the class is contained by the Session class (which is unexposed).
3. Click the radio button:
   - Base if you are creating a base class
   - Derived if you are creating a derived class. If you select this option, a box pops up where you select the class to derive from.
4. Select the appropriate attributes:
   - Hide from IDE browser

     Do not show in object browser
   - Expanded

     Gives dynamic property binding
   - Collection

     Specifies that the class you are creating will hold a collection of objects (a container class).
   - Unexposed “new” method

     Forces the user to create objects of this class from a create method in another class.

The New Method

When the Class document is saved, a Method document called "New" which corresponds to the class will be created. The "New" method is directly related to the class, so we will review both documents together.

Mapping the Class Document to the C++ Generated Code

The name you provide in the class document will be used to generate a C++ class. If you derive your class from another class, then the C++ class will be derived from this class. Otherwise, your class will be derived from LSXBase. A set of files (<*classname*>.cpp, i<*classname*>.cpp, <*classname*>.hpp, i<*classname*>.tab) will be generated for each class.

The new method will be used to determine the signature of the class constructor. You may change the signature of the new method by adding arguments. However, the new method must be a function that returns the data type of the class.

From Datatypes.hpp:

```
Class Datatypes : public LSXBase
//{{LSX_AUTHOR_CODE_Additional_Base_Classes
//}}
{
      ...
}
```

From Datatypes.cpp:

```
Datatypes:: Datatypes( LSPTR(LSXLsiSession) pContainer,
LSSSHORT theIntArg, LSSLONG theLongArg, LSFLOAT4 theSingleArg,
LSXString theStringArg)
   : LSXBase((LSPLTSTR)"Datatypes",
pContainer->LSXGetInstance(), CDATATYPES_DATATYPES_ID,
pContainer),
   //{{LSX_AUTHOR_CODE_Additional_Base_Class_Init1
   //}}
   M_ABoolean(LSXFalse),
   m_ACurrency( ),
   m_ACurrencyArray(),
   m_ADate(LsiInst),
   m_ADateArray(),
   m_ALong(theLongArg),
   m_ALongArray(),
   m_AnObject(LSNULL),
   m_AnObjectArray(),
   m_AShort(theIntArg),
   m_ASingle(theSingleArg),
   m_AString(theStringArg),
   m_AStringArray(),
   m_AVariant(LSNULL)
   //{{LSX_AUTHOR_CODE_Internal_Member_Init1
   //}}
{
   //{{LSX_AUTHOR_CODE_Constructor1
      m_ACurrency.Hi = 0;
      m_ACurrency.Lo = 0;
   //}}

   //You must do reference counting whenever you maintain a
   //reference to an object that LS knows about.  Otherwise
   //it will get deleted when LS is done with it, even if you
   //aren't.
   if (m_AnObject)
      m_AnObject->AddRef();
}
```

Significance of the Class Document in LotusScript

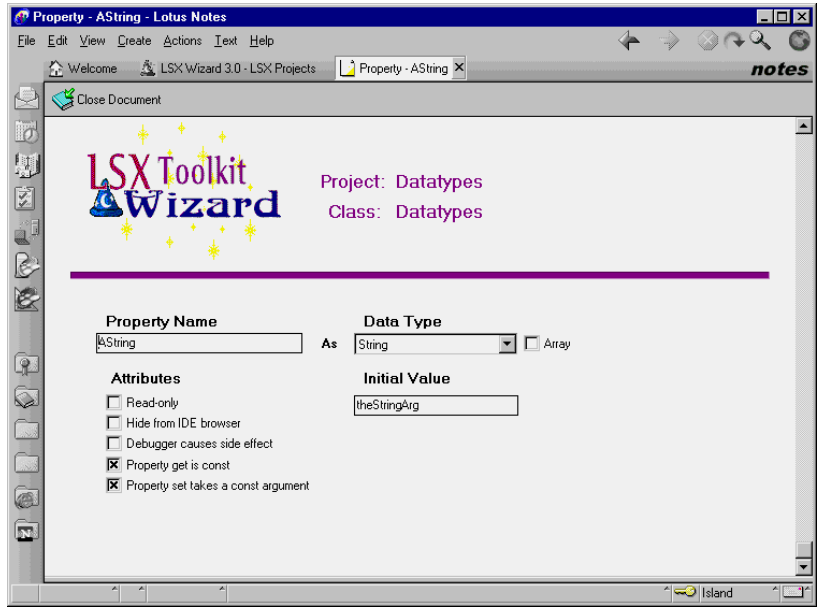
The name provided in the class document will be used as the name of your class in LotusScript. If you add arguments to the new method associated with the class, then these arguments will be required when you instantiate an object in LotusScript.

From Datatypes.lss:

```
Dim X as Datatypes
Set X = new Datatypes(1, 2, 1.1, "Hello LSX Datatypes")
```

Defining Properties for a Class

To create a Property document, select a class and click Property in the button bar. This displays the Property form:



1. Enter a property name.
2. Select a data type from the drop down list. Except for Boolean and Variant the value may also be an array.
3. Provide an initial value (optional).
4. Specify attributes.

Mapping the Property Document to the C++ Generated Code

The name you provide in the Property document will be used to generate up to two C++ inline member functions (a Get<*property name*> and a Set<*property name*>). If your property is read only, then only a get function will be generated. Otherwise a get and set method will exist for each property.

From Datatypes.hpp:

```
inline const LSXString& GetAString() const
 {
    //{{LSX_AUTHOR_CODE_PropertyGet_AString
    //}}

    return m_AString;
```

```
}
inline void SetAString(const LSXString& AString)
{
   //{{LSX_AUTHOR_CODE_PropertySet_AString
   //}}
   m_AString = AString;
}
```

Significance of the Property Document in LotusScript

In LotusScript, just the property name will be used to reference the property. The context of the property will constitute whether a get or set is being called. If your property is read only, then any attempt to assign a value to the property will generate a compile time error.

From Datatypes.lss:

Set

```
X.AString = "Goodbye LSX Datatypes"
```

Get

```
Print "New value in Property AString is: "& X.AString
```

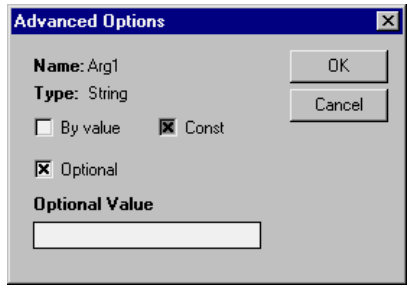
Defining Methods for a Class

To create a Method document, select a class and click Method in the button bar. This displays the Method form:

1. Enter a method name.
2. Select Sub or Function as the method type.

   Note A Function is a method that has a return value. A Sub is a method with no return value.
3. If you are defining a Function, specify the data type of the return value.
4. Specify arguments for the method by typing an argument name and selecting a data type. You may rename, reorder and change advanced options for the argument at any time during the design process. To change the option, just click the command button next to the argument to display the appropriate options based on the data type.

   Note Not all options are available for each data type.



Mapping the Method Document to the C++ Generated Code

The name you provide in the method document will be used to generate a C++ member function. You may change the signature of the new method by adding arguments. Optional arguments must be listed last.

From Datatypes.cpp:

```
LSXString Datatypes::StringDemoFunction(LSXString Arg1,
LSXString& Arg2)
{
   //{{LSX_AUTHOR_CODE_Method_StringDemoFunction

      //This method demonstrates passing string arguments
      //by value and by reference and returning a string value.
      //This first argument is by value, the second argument
      //is by reference.

      //Append exclamation points to the 2nd argument
      Arg2 = Arg2 + LIT_STR("!!!");

   return Arg1 + Arg2;

   //}}
}
```

Significance of the Method Document in LotusScript

In LotusScript, the method name will be used to call the method. You must provide any arguments that are not specified as optional.

```
Dim retString as string
Dim aString as string
aString = "Second String"
print "aString = " & aString
retString = X.StringDemoFunction("First String", aString)
print "aString should now equal 'SecondString!!!' : " & aString
print "retString should now equal 'First StringSecond String' :
" & retString
```

Defining an Event for a Class

This task is similar to the definition of a method. However, an event has no return value.

Generating the LSX Files

When you are finished entering your object model into the Wizard, the Wizard will generate the LSX source files for you. Click the button Generate LSX Files in the button bar. This will generate the complete set of files.

Note Since the Wizard generates all files each time you click the button, it may take a few seconds to create and write the files.

The generated files are written to the directory lsx\src\<*lsxname*>, where <*lsxname*> is the name as specified at the project level.

For example, in a Windows installation the following files will be produced:

- four files for each class defined in the LSX
  - <*classname*>.hpp
  - <*classname*>.cpp
  - i<*classname*>.cpp
  - i<*classname*>.tab
- lsxsess.hpp and lsxsess.cpp for the LSX's session class
- project-level files lsxapplx.h, lsxapplx.cpp, and guidfile.h
- MAKE files, .def files, and batch files for building the LSX
- library inclusion files

Editing the Generated Files

After the Wizard has generated the source file for a particular project, you will edit some of these files to add your implementation to the generated stubs. Usually, you will iterate through this cycle:

1. Re-specifying the LSX in the Wizard's Domino database
2. Generating LSX source files
3. Editing the generated LSX source files

You will do this, for example, when you have to add a new class to your LSX. When you are finished with your changes to the LSX's definitions in the Wizard's user interface and you run the process of generating LSX files again, the contents of the directory lsx\src\<*lsxname*> are completely replaced. To prevent losing your changes, the Wizard uses a special mechanism called tagging. Some of the generated files contain tags to mark the locations where you can post-edit. Any changes between a begin tag and the corresponding end tag are protected. The Wizard carries them over when writing the next version of the file.

The tags look like this:

```
//{{ LSX_AUTHOR_CODE_tagname
//}}
```

Caution To ensure that your edits are preserved, place your edits only between tags. Edits outside tagged locations are not protected and will be overwritten.

Note Files whose filename begins with the letter **i** are designed to need no editing and will therefore never contain tags.

Compiling the LSX

The Wizard generates a MAKE file for you. Check the file LSX_MAKE.<*system*> in the lsx\src directory if the switches match your environment.

Using LSX Data Types

The LSX interface headers define macros for common C++ data types to compensate for platform differences. You are encouraged to use them instead of the intrinsic C++ language types, because they are widely used throughout the Toolkit source code, and you should be familiar with them.

Data Type Macro	*Meaning*
LSVALUE	Any of the following data type macros except the unsigned versions; actually a union of all of them plus a member Type that tells you the type of the value. (See the section "Using Data Type Descriptions," below).
LSSBYTE	Same as signed char.
LSUBYTE	Same as unsigned char.
LSSSHORT	Same as INTEGER in LotusScript and signed short int in C++.
LSUSHORT	Same as unsigned short int.
LSSINT	Same as signed int.
LSUINT	Same as unsigned int.
LSSLONG	Same as LONG in LotusScript and signed long in C++.
LSULONG	Same as unsigned long.
LSFLOAT4	Same as float.
LSFLOAT8	Same as double.
LSBOOL	Allowed values: LSTRUE, LSFALSE.
LSPVOID	Pointer to void (null pointer is LSNULL).
LSPBYTE	Pointer to (unsigned) character array.
LSPTR (x)	Pointer to x.
LSSTATUS	Same as LSSSHORT; used for return status values.

Some more type macros are defined for string data types:

Data Type Macro	*Meaning*
LSCHAR	Same as char.
LSPLTCHAR	Same as LSCHAR.
LSCLICHAR	Same as LSCHAR.
LSUNICHAR	Same as LSUSHORT.
LSSTRING	Unicode string; pointer to LSUNICHAR.
LSCLISTR	ANSI strng; pointer to LSCLICHAR.
LSPLTSTR	Platform-native string; pointer to LSPLTCHAR.
LSUNISTR	Unicode string; pointer to LSUNICHAR.

Finally, the remaining data structures defined in LotusScript are represented by the following C++ type definitions:

C++ Data Type Definition	*Meaning*
LSsDate	Variant of data type 7 (as returned by the LotusScript function "Date"); same as LSFLOAT8. For an explanation, refer to the *LotusScript Language Guide*.

struct LSsCurrency {
  unsigned long Lo;
  long Hi;
}
	Currency data type in LotusScript.

PLSVALUE	A variant; same as LSPTR (LSVALUE).

Using Data Type Descriptions

Argument passing to LSX class methods is different from the way C++ passes arguments. Instead, an LSX class method receives a packed array structure as a single argument containing the actual parameters of the method. Each of the array members stands for one parameter, and includes information about the data type and the value.

So, the data type information itself is coded using certain symbolic integer constants. The basic ones are as follows:

LotusScript Data Type Code	*Related Data Type*
LSVT_EMPTY	EMPTY value for variants.
LSVT_NULL	NULL value for variants.
LSVT_SHORT	LSSSHORT (Integer data type in LotusScript).
LSVT_LONG	LSSLONG (Long data type in LotusScript).
LSVT_SINGLE	LSFLOAT4
LSVT_DOUBLE	LSFLOAT8
LSVT_CURRENCY	LSsCurrency
LSVT_DATE	LSsDate
LSVT_STRING	Depends on the specified character translation. Either a LSSTRING for UNICODE, or LSPBYTE for LMBCS, or LSPLTSTR for ANSI.
LSVT_BOOLEAN	boolean
LSVT_VARIANT	PLSVALUE
LSVT_UNISTR	LSsValueUniStr; same as LSPTR (LSUSHORT).

Furthermore, there are constants for list and array type descriptions. Refer to the file INC\LSIVAL.H and the *LSX Toolkit Guide* for details.

Using LotusScript System Services

The LotusScript client API offers you several system services. You are encouraged to use them rather than directly accessing the operating system API. This helps you to write LSXs which can be ported to other platforms more easily.

The following paragraphs give you an overview of some of the services. For a detailed description, refer to the files contained in the directory INC\SYS. For example, the file management service is declared in the file LSSFMGR.H; the actual service is enclosed in SERVICE_DECL_BEGIN(FILEMGR) and SERVICE_DECL_END. It consists of a set of functions you may call in any of your class methods.

For all but the memory management system service, you have to prepare your class to use the service.

Preparatory Steps

1. First, extend your LSX class slightly by adding a new private member:

```
class NewClass : public LSXBase
{
      //...
   private:
      //...
      PLSSFILEMGR   pFM;  // system services file manager
      //...
};
```

   The member type is taken from the file INC\SYS\LSSFMGR.H.

2. Then, extend the class constructor to initialize it:

```
NewClass:: NewClass (LSPTR(LSXSession) s,
                     PLSADTMSGCREATE args)
   : //... initialization of base and class members
{
   LShINSTANCE   hLSInstance; //  LS instance handle for
                              //  this class object

   // get the instance
   hLSInstance=s->LSXGetInstance();

   // get the file manager service handle
   this->pFM = hLSInstance->Services->pFMGR;
   //...
}
```

You will find a complete list of available system service handles in the file INC\LSSRVMGR.H. They are defined as members of the structure LSSsAnchor.

3. Now your class methods can access the service:

```
void NewClass:: NewMethod (PLSADTMSGMETHOD args)
{
   assert (this->pFM);   // check that the file service
                         // is available

   //...
   this->pFM->"any FM service function"
}
```

Memory Management Service

The API provides function calls for allocating and releasing heap memory. When the LSX uses this service, the LotusScript instance gains complete control over the dynamically allocated memory.

In order to make use of the C++ ability to redefine the new operator, the file SRC\COMMON\LSXCOMM.CPP defines a special version of the operator new, operator new [], and a corresponding delete operator that perform calls to the API's memory service functions. This means that you don't have to know the API memory functions; you will use them implicitly by using these C++ operators.

In contrast to the ordinary new operator, the versions defined for LSXs get a so-called placement argument, a handle to the LotusScript client API. The following code fragment shows you how to use them:

```
void NewClass:: NewMethod(PLSADTMSGMETHOD args)
{
    LSPTR (LSSSHORT) *aNewInt;
    LSPLTSTR  *aNewString

    // create a new integer (always pass this->LsiInst)
    aNewInt = new (this->LsiInst) LSSSHORT;
    // create a new string of length 42 (incl. trailing 0)
    aNewString = new (this->LsiInst) LSPLTCHAR [42];
    //...
    // do something
    //...
    delete aNewInt;
    delete aNewString;
}
```

File Management Service

The LotusScript client API offers you a rich set of file management functions including:

- Functions for file access: Open, PathOpen, Close
- Functions to work on file contents: Read, Write, Seek
- Functions for file attributes: GetAttr, SetAttr, DateTime, FileSize
- Directory functions: ChDir, CurDir, MkDir, RmDir, DirFirst, DirNext

The following code gives you an example how to use these service API functions:

```
void NewClass:: NewMethod(PLSADTMSGMETHOD args)
{
   // Purpose: Open a file, append a text string, and close it

   // set the open flags: lock and access type
   LSUSHORT LSOpenMode = (LASFM_SHARE_EXCLUSIVE |
                          LASFM_ACCESS_WRITE);
   lfile hFile; // a file handle

   // try to open the file; it fails when it doesn't exist
   hFile = pFM->Open ("\\lsx\\newmthd.txt", LSOpenMode);

   if (hFile < 0)  // it has to be created
      hFile = pFM->Open ("\\lsx\\newmthd.txt",
                         LSOpenMode | LASFM_ACCESS_CREATE);

   // seek to the end of the file
   if (pFM->Seek (hFile, 0, LASFM_SEEK_EOF) == LASFM_ERROR)
   {
      // perform error handling!
   }

   LSUSHORT charsWritten;

   // now write the text string...
   charsWritten = pFM->Write (hFile,
                              "A text string", 13);
   // ...and append a newline
   charsWritten += pFM->Write (hFile,
                              LASFM_EOL, LASFM_EOL_LEN);
   // check that all characters are written
   if (charsWritten != 13 + LASFM_EOL_LEN)
   {
      // perform error handling!
   }

   // close the file
   pFM->Close (hFile);
}
```

Interprocess Services

This service is defined in the file INC\LSSIPC.H. It offers some functions you may already be familiar with (from script programming in Domino):

- SendKeys
- SendKeysCancel
- Shell
- AppActivate

Platform Services

The platform service definition gives you access to some system values and functions. For example:

- GetDate/SetDate for handling the system date
- GetTime/SetTime for handling the system time
- Environment to retrieve system environment variables
- MsgBox to display texts in a message box

The following example demonstrates the usage of the MsgBox function. Of course, the class must be prepared to access the platform service anchor:

```
void NewClass:: NewMethod(PLSADTMSGMETHOD args)
{
   // Purpose: Display a text in a message box.

   // The box shall consist of an information icon
   // and Yes and No buttons
   LSUSHORT button;

   button = pPLAT->MsgBox ("Do you want to see more?",
                           4 + 64,  // same as MessageBox !
                           "Please decide");
   switch (button)
   {
   case 6: // YES
      //...
   case 7: // NO    (same return codes as for MessageBox)
      //...
   }
}
```

Testing an LSX

To test your new LSX during development, you can either write event scripts in a Domino database that use the LSX classes, or you can use a test tool shipped with the LSX Toolkit. Because LSX testing isn't concerned with Domino functionality, it is much more convenient to use the Toolkit test tools.

The LSXTEST Tool

In general, LSXTEST presents an integrated development environment for writing, compiling, executing, and debugging LotusScript programs.

In particular, it helps you to write and debug LSX test scripts that contain a USELSX statement to load the LSX.

For example, LSXTEST allows you to:

- Open, edit, and save LotusScript .LSS files.
- Compile scripts and save the compiled LotusScript modules.
- Load compiled modules.
- Set breakpoints to interrupt script execution.
- View the values of variables and the stack frame for the current breakpoint during execution.

Many of the features are also available as command line options passed to LSXTEST, so you can automate most of the test steps.

Example

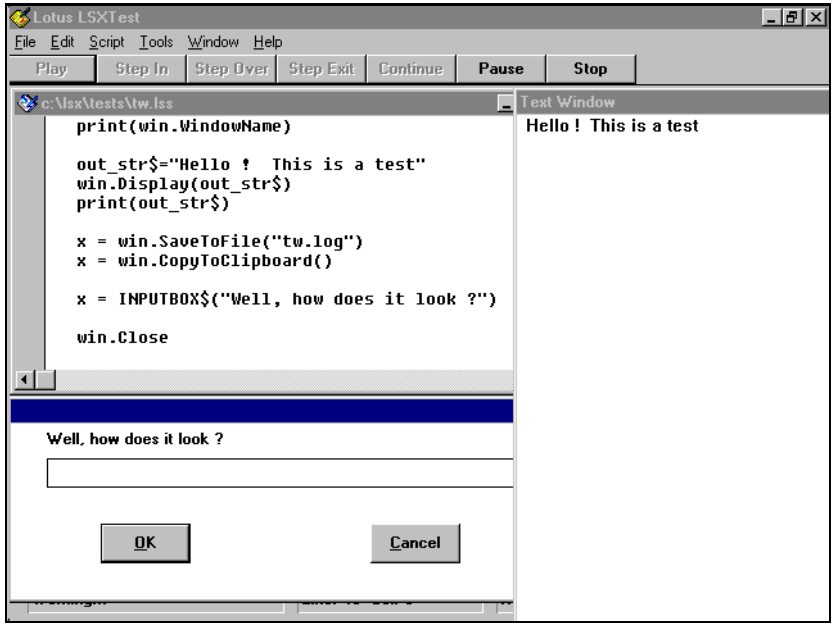
The Toolkit contains test scripts for all sample LSXs. To load and run one of them in LSXTEST:

1. Start an LSX development session with a command prompt window.
2. Start LSXTEST. Actually, the command name is different for each platform. For example, in OS/2, it's LSXTESTO.
3. Choose File - Open. The file dialog box is displayed.
4. Select one of the test scripts in the directory TESTS, for example TW.LSS. The script is displayed in a new window.

   Note This sequence of steps is also accomplished by invoking LSXTEST with the script file name.
5. Click the Play button to run the script. It uses the TextWindow sample LSX to display a new window.

   Note Remember to first compile the LSX in the directory SRC\TextWindow.

The following figure shows the result:



The LSXRUN Tool

LSXRUN provides a minimal runtime environment for testing scripts. It is invoked by a command line, and does not require, or depend on, any graphic user interface.

LSXRUN runs LotusScript source files, and outputs a report of its activities to the screen and an optional log file. For further details, refer to the *LSX Toolkit Guide*.

Deploying an LSX

The LSX Runtime Environment

The runtime environment of an LSX consists of the following:

- Lotus Domino R5.0. Actually, it can be any Lotus product that supports the LotusScript interpreter (Release 3.0 or higher).
- The LSX itself.
- The scripted application: the application for which the LSX provides an object model.

You must know the location where the LSX will eventually execute, because the LSX and the scripted application must be available at the location where they are used. If you define an agent that runs on a server, and it uses the LSX, the location of the LSX is the Domino server. There are other types of agents, for example "Manually from Actions Menu," that will run on the Domino workstation. If these agents use an LSX, its location is the workstation.

LSX Installation

To distribute your LSX, you may want to write an installation program that copies the LSX (and probably the scripted application) to the desired directory, and performs some necessary initialization tasks such as setting up environment variables.

The common installation program is a batch command file. As your LSX is intended to run on multiple platforms (ideally on all Domino platforms), the installation program should consider the operating system used for installation. This allows the installation to behave differently for different platforms. Look at the Toolkit installation program for examples of how to differentiate between operating systems.

LSX Registration

The registration of the LSX classes influences the method used for accessing them from within LotusScript. Either the script loads the LSX by passing a complete path to the UseLSX statement, or it just references a name in the LSX class registry.

If you choose the first option, your installation program must copy the LSX library to the location referenced by the scripts. In fact, it is very difficult to find pathnames for libraries that consider multiple operating systems, the different directory structures and drive names, and file name restrictions.

Therefore, the class registry is the recommended place to store a complete path of the LSX library, together with a symbolic name, the key, that you then use in the UseLSX statements in your scripts. The key uniquely identifies your LSX on the system. Using this option, LotusScript will look up the path in the LSX class registry.

You cannot redistribute Lotus' registration program LSXREG to call it from the installation program. Instead, you have to write your own registration procedure for your classes.

In Windows 95/98 and Windows NT, the information is stored in the folder HKEY_LOCAL_MACHINE\SOFTWARE\Lotus\Components\LotusScriptExtensions\2.0. In all other platforms, the file NOTES.INI is used as the class registry.

Apart from the inherent language features of LotusScript, scripts always require an embedding application context like Domino that provides them with "physical" objects to work on. Therefore, the LotusScript instance responsible for compiling and executing scripts contains an open interface to enable connecting to an embedding application. This separation of functionality into embedding and embedded components, and a well-defined interface between them, forms the basis of the LSX integration.

The Extensible LotusScript Architecture

On startup, Domino creates a LotusScript instance for all further script processing.

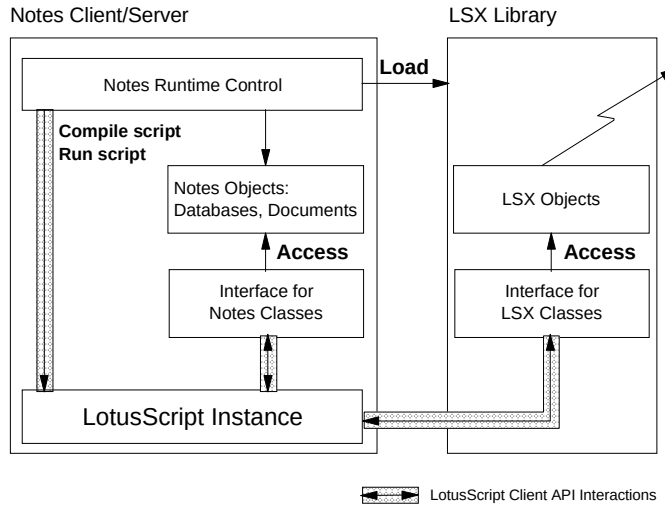
This instance provides certain services via a LotusScript client API which is accessible by an API identifier, referred to as the LotusScript instance handle. Domino gets the handle as a result of the creation.

Domino as the embedding application controls the operation of LotusScript through this API. Domino presents LotusScript source or compiled code to the LotusScript instance via this API, and LotusScript compiles and executes.

Crucial to the LotusScript architecture is the fact that the LotusScript client API contains services to register new classes. Domino uses these services to register its LotusScript Domino classes.

The implementation of each class is included in the Notes code space: part of the registration function serves to specify the entry points that the LotusScript instance can call to execute the scripted behavior. This means that Domino supplies the LotusScript instance with callback functions that implement class constructors, methods, and property access.

The same technique applies to the integration of an LSX module that is compiled and linked as a dynamic library. First, Domino loads the library and calls a well-known function entry point in the library with the handle of the LotusScript client API. This LSX function uses that handle to register the LSX classes, including the methods and properties that make up the class definitions. Since the callback functions that implement the registered functions, and the methods of the classes, are also in the LSX library, the LotusScript instance knows how to execute the external implementation.



The following sections describe the interactions between the LSX and the LotusScript instance embedded in Domino.

LSX Integration

After loading the LSX, Domino obtains the address of the LSX message procedure. All further communication between Domino and the LSX happens via this message procedure, which implements a standard set of messages. The two most important messages are INITIALIZE and TERMINATE for LSX enrollment.

Having retrieved the message procedure address in the LSX, the first message Domino sends to the LSX is INITIALIZE, passing the LotusScript instance handle as a parameter.

As mentioned previously, it is possible that an LSX is simultaneously used by more than one Lotus application. For example, if the LSX is a shared library loaded on a multi-user platform, such as a UNIX platform, its code may be shared between multiple Domino workstations, each of them embedding a LotusScript instance. In that case, the LSX is loaded only once, but it receives multiple INITIALIZE messages, each indicating the start of a session with a new LotusScript instance. The LSX is responsible for maintaining all these sessions, and for performing a proper cleanup of each session when a TERMINATE message for that session arrives.

LSX Initialization

In the initialization phase, an LSX must register its classes with the LotusScript instance, using the passed handle.

Registering a class means supplying LotusScript with a complete class definition that will enable processing any runtime operations on the class,

including creating and destroying class instances (objects). The class definition information includes the class name; class ID; version number; parent class ID; tables to define the properties, methods, and events of the class; and other miscellaneous information.

Since the implementation of each function or class is in the LSX code space, LotusScript must call back to the LSX at runtime to create and manipulate instances of that class. So, part of registering a class is providing a callback function for the LSX to use at runtime when LotusScript calls back with a request to carry out operations on objects that the running script has specified. For a given class, this function is known as the class control procedure. It must handle the object manipulation messages sent to it by LotusScript, such as the CREATE message to create an object.

Once the INITIALIZE call returns, the LSX is idle, except when it receives a message that it must respond to.

Object Creation

When an executing script requests a new instance of an LSX class, the LotusScript instance calls the registered class control procedure for that class, sending it the CREATE message. After the new object is created, it is added to a particular list containing all objects that were created in the current session.

An object presents itself to the LotusScript instance via an object control interface. LotusScript uses this interface for all further interactions with a new object. It defines a standard set of messages for object method invocation and property access.

Object Deletion

Deletions are handled in a similar manner. The LotusScript instance sends a DELETE message together with an object ID to the appropriate class control procedure, which has to delete the object and update the session object list.

Runtime Manipulations on Objects

The object control interface receives messages for method invocation, setting and getting properties, and several other operations. The interface must map the message parameters onto the corresponding LSX class methods and attributes to effect the intended object behavior.

Event Notifications

The LSX class method implementation may raise events to signal special conditions to the executing script. As provided for in the LotusScript language, the script can catch them with installed event handlers. Likewise, LSX methods can cause errors to be raised which are then handled in the executing script. The LotusScript client API includes appropriate functions for that purpose.

Part of the definition for any LSX class that is registered with LotusScript are the events raised (if there are any), and under what conditions they are raised.

LSX Termination

Just before destroying a LotusScript instance in which the LSX is loaded, Domino sends the TERMINATE message to the LSX message procedure. The LSX is responsible for cleaning up any of its objects that belong to that instance.

The LSX must guarantee that LSX class objects of other LotusScript instances remain valid: it must clean up only the objects of the current session's object list.

Understanding the C++ LSX Class Framework

The LSX Toolkit supplies you with a set of C++ classes and functions that are to be used in an LSX on top of the LotusScript client interface. This code provides higher level services for the LSX, including class registration utilities and the infrastructure for handling LotusScript callbacks.

When you develop a new LSX, the Toolkit code forms a framework in the sense that you can reuse its functionality by deriving your LSX classes from Toolkit classes, and by extending the implementation of certain global Toolkit functions. Therefore, you need to know where the supplied classes and functions are located, and how they interact with each other.

Important LSX Source Files

The directory INC contains C header files for the LotusScript client API. In the file INC\LSILSX.H, the C++ struct LSsLsxInstance defines the raw interface through which all communication between the LSX and the LotusScript instance occurs.

The directory SRC\COMMON contains the following files:

- LSXBASE.HPP
- LSXBASE.CPP
- LSXCOMM.HPP
- LSXCOMM.CPP

They have a central role in any LSX built using the LSX Toolkit.

Note You can use the code in this directory without any modification.

The SRC\COMMON\LSXBASE.[C or H]PP files constitute an isolation layer within the LSX. LSXBASE.HPP defines the LSXBase base class, an abstract C++ class that every class in your LSX should inherit from. LSXBASE.CPP implements the base class. This LSXBase class serves mainly as an interface class; it comprises the object control interface (by which LotusScript accesses

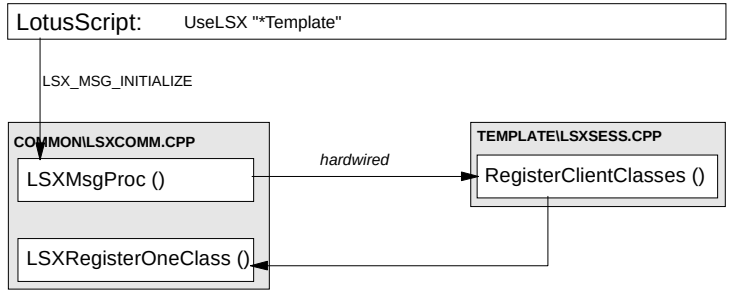
the LSX class objects), and provides your classes with an easy callback mechanism. Moreover, it performs some of the object protocol messages automatically, and it contains a linked list implementation for maintaining a hierarchy of LSX objects.

The files SRC\COMMON\LSXCOMM.[C or H]PP constitute most of the interface between the LSX and Domino and its embedded LotusScript instance. They provide essential services for the LSX, such as an implementation for the LSX message procedure, a generalized class control procedure, and registration utility functions. The files LSXSESS.[C or H]PP define and implement the class LSXLsiSession. As described previously, it is possible that an LSX is connected to multiple LotusScript instances at a time. For each connection, a single LSXLsiSession object maintains information about the objects created in it, to ensure a proper session cleanup. Moreover, the file LSXSESS.CPP contains the (LSX specific) class registration code.

Flow of Control Within the Framework

We will now consider how the Toolkit framework implements the interactions with the LotusScript instance. The main LSX tasks, such as initialization, object creation, and object manipulation, are described from an implementation point of view. You will find the LSX specific code sections in the SRC\TEMPLATE files that you have to modify for your LSX.

After being loaded by Domino, an LSX registers its LSX message procedure LSXMsgProc, located in SRC\COMMON\LSXCOMM.CPP. Then the LotusScript instance calls that function with an INITIALIZE message parameter:

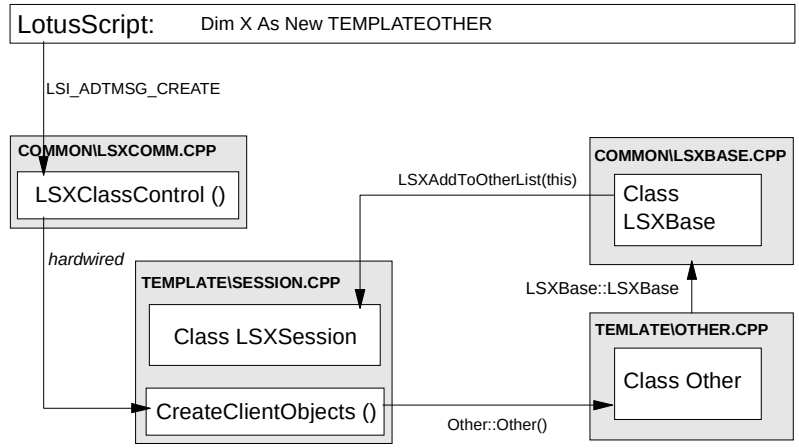


In order to register all LSX classes, the message procedure calls the function RegisterClientClasses. This LSX specific function knows the LSX classes, namely LSXLsiSession and Other, retrieves the type information from the static tables in the .TAB files, and registers each of them with a separate call to the function LSXRegisterOneClass. Eventually, this utility function uses the LotusScript client API to perform the registration.

Note To register your LSX classes, you have to modify the function RegisterClientClasses.

Part of a class registration is to provide a class control procedure which the LotusScript instance uses to execute class operations. The Toolkit includes a generic function LSXClassControl that can be used for all LSX classes. The function LSXRegisterOneClass registers it as the related callback function.

This callback function is used when LotusScript encounters a New statement for an LSX class in an executing script.



The LotusScript instance calls LSXClassControl with the message parameter LSI_ADTMSG_CREATE and the ID of the LSX class. Because class object creation requires LSX specific knowledge, it simply passes the call to the function CreateClientObjects.

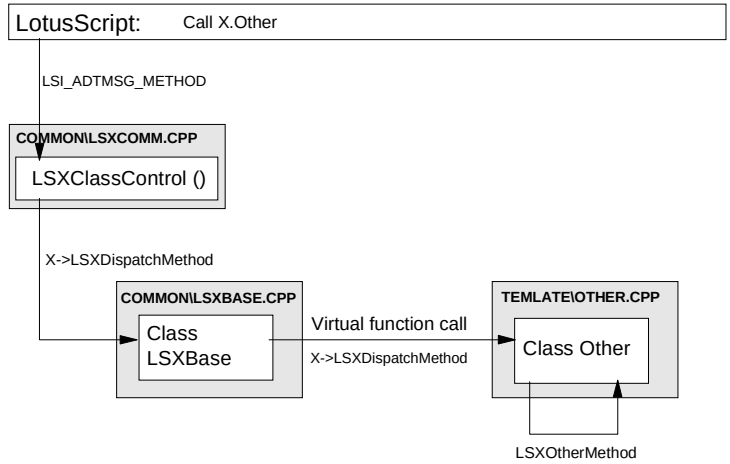
This function decides, based on the given class ID, what kind of object is to be created, and calls the C++ new operator for this LSX class. As usual with C++, the constructor of that class first calls the constructor of the base class (which is always LSXBase). The base class constructor now registers itself with the session object of the session in which it is created. Furthermore, it saves the given LotusScript handle as an object attribute, so that the LSX class object can use the LotusScript API later on. Eventually, the body of the LSX class constructor can implement application specific object initialization as needed.

When the new LSX class object is created, the function CreateClientObjects returns (to the calling class control procedure) a handle to the object control interface.

The LotusScript instance now uses this handle to manipulate the object. In fact, the class LSXBase comprises the object control interface, because it is derived from it. Again, LotusScript accesses this interface by a callback function, and the Toolkit design strategy is to use the same control procedure for objects as for classes. This means that the LotusScript instance

finally calls the function LSXClassControl to perform object manipulations, passing the message together with a class ID, an object handle, and a set of parameters.

Let us now consider an example: a method invocation on a given object.



Receiving an LSI_ADTMSG_METHOD message, the LSXClassControl function first converts the given object handle into an LSXBase object. Then it calls the method LSXDispatchMethod on that object, which is defined as a pure virtual function in LSXBase. Therefore, it actually calls the function defined in the derived class Other. Now, this function determines which method invocation is requested (by looking at the method ID), calls it, and passes the return value back to the caller.

As you can see, method invocation on objects is a very straightforward implementation. It takes advantage of the class LSXBase, which is designed as an interface class. Other runtime manipulations occur in the same manner. For example, a script statement to access an Other object property is sent as a LSI_ADTMSG_PROP_GET message to the function LSXClassControl which calls the method LSXGetProp on the LSXBase object. Again, the function is declared as virtual, and the object is actually of class Other, so that the function LSXGetProp in the Other class is called. Finally, this function is provided with the property ID, and can take the appropriate action.

Other interactions between the LotusScript instance and the LSX, such as object deletion and LSX termination, are done almost automatically. The implementation of the LSXLsiSession class ensures a proper cleanup per session, and calls the destructor of any other LSX class objects as needed.

LSX Design Decisions

The following section contains some general design considerations for all LSX implementations.

LSX Class Design

You need to consider what kinds of data structures to use to represent the object attributes in your class model.

Besides scalar types such as INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING, and VARIANT, LotusScript supports arrays and lists. Beyond these, you can use any of the classes you define in the LSX. All of these data structures are available for declaring and using as data members of your LSX classes. The same is true for the parameters and return value types of the class methods.

Note This implies that you cannot directly interface to the Domino product classes such as NotesDocument. You have to break them down to the types that LotusScript supports.

A further way to structure an LSX class is to define and register it as a collection class. A collection class is a container of items that can be accessed directly via indexing or via the LotusScript ForAll language iteration construct. The allowable language constructs are to access the values, the properties, and the methods of an individual item, or of every item in the collection.

Object Control Interface

LotusScript follows the conventions of a COM (Common Object Model) interface in accessing client objects. It is the object control interface, a C++ structure named ILsiADTControl.

Either the ILsiADTControl structure may be contained as a member in the definition of each class, or the base class LSXBase may inherit from it. The usual design strategy, and the default in the Toolkit examples, is inheritance.

In future releases, it is planned to include an OLE adapter in the Toolkit enabling you to expose objects in any of your LSX classes to OLE automation. This technique will require the ILsiADTControl to be inherited. For now, we recommend using the default, letting the LSXBase class inherit from ILsiADTControl.

In the Toolkit, the high-level flag variable EMBED_ADT governs the choice. It is referenced when building object files for the example LSXs, using makefiles in the Toolkit. By default, the EMBED_ADT flag is undefined, so that the example LSXs compile with ILsiADTControl inherited.

Character Sets

Another design decision is what character set to use to represent LSX-maintained strings that must be passed to the LotusScript instance.

The LotusScript internal representation is UNICODE. However, an LSX or an embedding application can specify any of four string communication representations to LotusScript:

- The platform-native character set (currently ANSI)
- UNICODE
- LMBCS
- ASCII

This means that a string will be presented to LotusScript in that representation. LotusScript is responsible for converting the string to UNICODE as needed for its own purposes.

Caution LotusScript will translate string message parameters passed to the LSX into the representation specified during the class registration. The current implementation of the utility function LSXRegisterOneClass specifies the platform-native character set, which is sufficient for many, but not all, applications.

Note An LSX cannot specify that all strings passed between it and LotusScript use one of the four representations. The LSX has to specify which representation to use for each string individually.

Portability Issues

You need to decide early on whether your LSX is to be written for one platform or several. Single-platform design allows you to write C++ source code to take advantage of specific compiler features and system services. However, the resulting source code may not be portable.

In the LSX Toolkit, the provided C++ framework code is platform-independent concerning compiler features and system services. A platform-specific header file is selected and included in those files. The selection criterion differentiates the platforms: 32-bit Windows, OS/2, and UNIX.

The LotusScript instance offers you standard systems services: memory management, file management, national language string support, interprocess communication, dynamic library system, and others. Your LSX implementation should access these system services only through the provided LotusScript interface.

Caution The Toolkit overrides the default C++ new operators to use LotusScript memory management services.

Graphical User Interface

You can write the LSX to present its own user interface, since it is a separately loaded library. However, any such interface that you may implement is independent of Domino, and you cannot build interactions between it and Domino. In particular, an LSX running on a server cannot invoke it.

Globally Unique IDs for LSX Classes

The client object interface is standardized as an OLE2/COM-style interface. This ensures that client objects are accessed consistently across LotusScript applications.

So, for each class in your LSX, you have to assign a globally unique ID (GUID) to identify your class to LotusScript. LotusScript will not allow an LSX to register a class that has the same GUID as an already-registered class.

A GUID is a 16-byte identifier. In Windows, a GUID is the same as a Windows GUID used for OLE objects. Some compilers on Windows platforms include a tool to create GUIDs. For detailed information, refer to the *LSX Toolkit Guide*.

Accessing LSX Class Method Arguments

LSX class method arguments are packed in a single array which is passed to your method implementation. So, any of your methods will look like this:

```
void NewClass:: NewMethod(PLSADTMSGMETHOD args)
```

The type PLSADTMSGMETHOD, defined in INC\LSILSX.H, is a pointer to the following structure:

```
struct LSFAR LSADTMSGMETHOD
{
   PLSVALUE        pArg;      // Array of Arguments.
   LSUSHORT        nArg;      // Number of Arguments.
   LSADTMETHODID   idMeth;    // Method ID.
   LSADTCLASSID    idClass;   // class ID for method.
};
typedef LSPTR (LSADTMSGMETHOD)    PLSADTMSGMETHOD;
```

The idClass and idMeth members inform you about the class ID and the method ID, respectively. The nArg member tells you the size of the pArg which is actually an array. So, pArg has members from index 0 to nArg - 1. The size depends on whether the method returns a value or not. If it does not, nArg equals the N_**NEWMETHOD_NEWCLASS**_ARGS -1, where N_**NEWMETHOD_NEWCLASS**_ARGS +1 is the number of method arguments you specified in the **NewClass**.TAB file. If the method does return a value, nArg is equal to this constant.

So, basically you access the message parameters by accessing the pArg member. It is an array whose members can be of any type, as defined by the type LSVALUE. But LSVALUE provides you with information about the type of value, and therefore you should always access a method parameter in the following way:

```
// this method doesn't return a value, so it's first
// parameter is at index 0
void NewClass:: NewMethod(PLSADTMSGMETHOD args)
{
   PLSVALUE   pVal = LSNULL;
   // check that the number of passed parameters equals
   // the expected number (means: the declared number of
   // arguments). The constant N_NEWMETHOD_NEWCLASS_ARGS
   // should be defined in NewClass.HPP
   assert (args->nArg == N_NEWMETHOD_NEWCLASS_ARGS - 1);
   //...
   // access the n-th parameter which should be of type
   // INTEGER, for example.
   pVal = &args->pArg[n - 1];
   assert(pVal->Type == LSVT_SHORT);

   // now, it is safe to access the value of the parameter
   LSSHORT nthParm = pVal->vShort;
}
```

Note To apply this method to a class constructor, simply consider it as a method with no return value. So, the first argument starts at index 0.

In this example, the method parameter is accessed by the expression pVal->vShort. In general, to access any of the types whose descriptions are listed in the previous table, the following definition of the data type LSVALUE applies:

```
struct LSsValue
{
   union
   {
      LSSSHORT      vShort;        // LSVT_SHORT
      LSSLONG       vLong;         // LSVT_LONG
      LSFLOAT4      vSingle;       // LSVT_SINGLE
      LSFLOAT8      vDouble;       // LSVT_DOUBLE
      //...
      LSsCurrency   vCurrency;     // LSVT_CURRENCY
      LSsDate       vDate;         // LSVT_DATE
      LSSTRING      vString;       // LSVT_STRING
      //...
      LSsValueBool  vBool;         // LSVT_BOOLEAN
      PLSVALUE      vVar;          // LSVT_VARIANT
```

```
      //...
      LSsValueUniStr  vUniStr;     // LSVT_UNISTR
      // —- Convenience Values for callbacks
      //     when translation is specified
      LSPBYTE       vLmbcs;        // LSVT_STRING
                                   // (translated)
      LSPLTSTR      vChars;        // LSVT_STRING
                                   // (translated)
      LSPBYTE       vBytes;        // LSVT_STRING
                                   // (translated)
      //...
   }; // end of union
   LSVALTYPE   Type;   // Value Type
   //...
};
```

There are also members for the types which are lists and arrays. Refer to the file INC\LSIVAL.H and the *LSX Toolkit Guide* for details.

LSX Error Values

The data type LSSTATUS is frequently used as a return type, to return either LSX_OK or any of the error constants defined in the file INC\LSIERR.H.

Accessing LSX Class Property Arguments

For each class with properties exported to LotusScript, you will set up two class methods **NewClass**::LSXGetProp and **NewClass**::LSXSetProp. For the latter, the question arises how to access the new value that the property should get.

The declaration is as follows:

```
LSSTATUS NewClass:: LSXSetProp(PLSADTINSTDESC pInstDesc,
                               PLSADTMSGPROP  param)
```

The first argument, pInstDesc, is a pointer to a structure describing the called object in terms of object control interface, related class ID, and current LotusScript instance.

The second argument is more important since it names the property to be changed, and the new value. It points to the following structure:

```
struct LSADTMSGPROP
{
   PLSVALUE         valProp;   // Property Value.
   LSADTPROPID      idProp;    // Property Id.
   LSADTCLASSID     idClass;   // Class ID for this property.
};
```

The member idProp stores the ID of the property to be changed, as you registered it in the .TAB file. The valProp member contains the new value of the property. You use this structure as follows:

```
LSSTATUS NewClass:: LSXSetProp(PLSADTINSTDESC pInstDesc,
                               PLSADTMSGPROP param)
{
   LSSTATUS   stat = LSX_OK;
   PLSVALUE   pVal = param->valProp;
   LSSSHORT   len;

   switch (param->idProp)
   {
      case CTEMPLATE_NEWCLASSPROP_PROPERTY:
         // access property (assuming it's a
         // LotusScript integer)
        len = pVal->vShort;
        //...
     default:
        assert (LSFALSE);
     }
   return stat;
}
```

Summary

This chapter documented and showed examples of some of the more advanced methods of coding applications for Domino. We discussed Java, CORBA/IIOP, OLE automation, and using the LSX Toolkit.

Chapter 12
Development Dos and Don'ts

This chapter contains some of the lessons that have been learned by the team that wrote this book, both in developing their own Domino applications and in using the Domino R5.0 code. The chapter does not document the definitive way to approach a Domino development project, but it does provide some useful hints and tips that may help you in developing your own applications.

Before You Write a Single Line of Code

Lesson 1 - Getting a Business Sponsor

How can a Domino development project fail? There can be any number of causes, but one of the most important ones is that the project does not have the full commitment and support of a project sponsor. For a Domino project to succeed it must be seen as being a high priority project within the company and the best way to achieve this is to gain sponsorship from as senior a person in the company as possible.

Lesson 2 - Communication

Once you have a project sponsor you need to ensure that they communicate the project status on a regular basis to the employees who will be affected by the application. Domino projects sometimes involve some change for the people that use the application and, traditionally, people resist change for a number of reasons, one of which is uncertainty. Regular communication can help to reduce the level of uncertainly and so make the introduction of the new application easier.

Lesson 3 - Ensure That There is a Real Business Need

It may seem a strange thing to say, but it is vital to ensure that the application you are creating has a real business need. Ask yourself and your customer, "Will people *need* to use this application?" If the answer is not a strong YES, then the application will typically become one of the Domino applications that sits on a server and is used heavily in the first month or two as people see what the application does, but that usage trails off to a minimal amount as time goes on.

Lesson 4 - Understanding the Deliverables

Before you start your project, understand fully with your customer what it is that you are expected to deliver, and what your customer is expected to deliver to you in the way of support and access to business resources and people.

Lesson 5 - Planning Your Application

Domino is categorized into the Rapid Application Development (RAD) tool set. A RAD tool enables you to build and deploy a functional application in a very short period of time. While this cuts development time dramatically, it also means that Domino developers tend to dive straight into creating an application without going through any planning or design work beforehand. While this is sometimes an acceptable approach, for example, if you are creating a very simple database containing one or two forms, anything larger must definitely be planned and designed thoroughly beforehand.

Lesson 6 - Even Domino Has Limitations

While Domino is one of the most feature-rich development environments available, it does have some limitations. The trick is in knowing the balance of when to use Domino to perform a task, and when to use a specialized tool. For example, the Domino storage capability is constantly increasing and improving, but you would not store 10 million records in a Domino database and expect it to perform as a relational database. The Domino database architecture is designed as a flexible object store that can store many different types of data, and it is not necessarily the best tool for large quantities of "plain" data.

Lesson 7 - Project Scope Creep

Make sure that you do not keep adding small functional improvements to your application that detract from the major development focus. Domino makes it easy to quickly add a field or a new view, and because of this ability, development projects can sometimes run longer than intended. It is important to stay with the original development plan and deliver what was originally intended. Once the application is ready for deployment, freeze the design in a template and start working on a Release 1.1 that contains all the additional features that were requested.

Creating Your Application

Lesson 1 - Use Professional Graphics

We have all tried it. Grab a paint package and create some buttons, backgrounds and banners for our Domino application. The truth is that unless you are very artistic your application will suffer from the use of poorly-designed graphics. Using a professional graphics designer to create images will improve the users experience of your application dramatically and will give it a professional touch.

Lesson 2 - Design the Outlook as Thoroughly as Possible

When you are creating an application try to describe the outlook of an application as thoroughly as possible, so that you don't spend a lot of time moving image locations and changing the graphics in your forms, views, or pages.

Lesson 3 - Try to Standardize on a Web Browser

When creating a Domino application that will be accessed by Web browsers, try to enforce a standard browser software package and release. While it is encouraging that the developers of Web browsers continue to push the boundaries of their products, there are inconsistencies between the different products and even between different releases of the same product. Although you have to develop to the lowest common denominator when designing an application for the Internet, it is sometimes possible to standardize on a single product and release for an intranet or extranet application where you are more in control of what software is used to access your servers.

Lesson 4 - Comment Your Code

A very easy thing to do when you are writing your applications but so much harder to do a month later! Always comment your code so that anyone that has to support your application at a later stage can easily understand and modify it.

Lesson 5 - Try to Avoid Hard Coding

When you are developing an application try to avoid hard coding values in your code or formulas. For example, use the @Subset(@DbName; -1) instead of writing the database name. This allows the user to change the database name without causing the code to stop working. Another trick is to create a simple form with two fields, one as a unique value to display in a view and the other to contain the variable value. Create a hidden sorted view to display the unique value and the variable value, and use a formula, such as **`myvar:=@dblookup("; "; "MyView"; MyUniqueValue; 2`**) in the field formula. In this way you can just update the value in the document rather than changing the code.

Lesson 6 - Use the Appropriate Design Elements and Events

Domino offers many events that you can use in your code. Split your code into small routines that you can use in these events. Make a clear design of your application, including a list of global variables and switches.

Lesson 7 - Provide Meaningful Error Messages

If the user input does not pass your field verification routines provide the user with helpful information as to what the application is expecting. For example, the error message "You have entered a wrong number" gives no help if you are checking for a number in the range: 1 to 100.

Consider using a Profile document to control whether or not errors are written to the NotesLog. The log can be sent to the developers for investigation. This helps you to find errors that occur in production. Also, write meaningful debug statements. For example, "The specified file could not be opened" contains no information. It would be better to write "File <*filename*> could not be opened."

Lesson 8 - Document Your Application

We all hate documenting an application design but it is an absolute necessity. Start with a general overview of the application and then describe the function for each of the forms, views, subforms, pages, framesets and so on. A database design document does not mean that you just create a document from the database design synopsis, which is useful but does not explain the sequence of events that occur in order to achieve a task.

Lesson 9 - Be Aware of Performance Options

Make sure that you agree with the application owner on how scalable the application must be, and how users will access it. Even though your application may start out just being used at one centrally-located department, it may spread out company-wide or further. Remember when you add those professional graphics from Lesson 1 that some users may access the application with the bandwidth you get from a 28.8 Kbps modem. A graphic that loads in a few seconds when sitting on a high speed intranet may take much longer to load over a slower line.

Here are some tips on working with views, full text indexes, and agents:

Views

1. Limit the complexity of views. Don't do calculations in the views. If you need to do any calculations, do them in the form and save them with the document. There is a trade-off between the extra disk space it will use versus the extra time it will take to open the view.
2. Keep keyword and category values as short as possible. This will speed up indexing.
3. Limit the categorization and sorting of your views. If you can have the information simply sorted, rather than categorized, you will save LOTS of time. A view with more than one categorized column is especially expensive.
4. Use Dynamic View Sort where possible to reduce the number of views in the database.
5. Use @Now and @Today Functions in views with extreme caution. These time-related functions invalidate the views causing them to be rebuilt every time the functions are executed.
6. Set Default Categories as either All Collapsed or All Expanded. This will allow the Notes system to process requests faster.
7. Avoid frequent indexing. Indexing is CPU intensive and should be kept to a minimum. That is not to say that you should avoid indexing when it is really needed. Consider using the "Auto, at most every *n* hours" index refresh option.
8. Use the Cache option with @DbLookup and @DbColumn Functions and select Minimize use of these functions. This will store the results of the query in a memory cache and avoid expensive lookups. Use temporary variables to store query results.
9. If you use @DbLookup or @DbColumn, look up a column number, not a field name.

Full Text Indexes

1. Try to delay indexing of databases which require Full text indexes. Full text indexing is very expensive in terms of CPU resources and may slow down server response time.
2. Minimize the use of access control lists with databases containing Full text indexes. When the index is rebuilt all ACLs must be validated.
3. Use the Case Insensitive option and/or Whole Word for Full text indexing. This will significantly reduce the cost of updating the Full text index.
4. Use views to perform structured searches from applications rather than Full text indexes. View searches are quicker, less resource intensive, and safer than Full text searches.

Agents

1. 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, the more time it will take to execute and the higher the load on the server.
2. For large data sets, use views with selection formulas rather than coding them into the agent itself. When updating a view the server only has to update for documents that have changed. Coding the selection formula into an agent or making the agent check every document in the database each time it runs can be very time and resource expensive.

Handing Over Your Application to Production

Lesson 1 - Perform Quality Assurance of the Application

Have another experienced Domino developer review your application to make sure that it is developed according to standards, and is maintainable. In addition to having an application that is "perfect," it must also adhere to the rules of deployment defined by the support organization responsible for the production environment. Some issues to consider are:

- Does the application require use of Middleware or API programs?
- Are unrestricted agents allowed to run on the production servers?
- Are there any time limits to how long an agent can run?
- Does the application have non-standard backup and recovery requirements?

Applications that do not adhere to the standard rules of deployment for an organization may have to be placed on servers dedicated to such "non-conforming" applications.

Lesson 2 - Supply an Installation Test Verification Case

The System Administrator who installs the application on the production system cannot be expected to have an intimate knowledge of the application. Create a few test cases that can be used to verify that the application has been installed and set up correctly. For example, if you do lookups in other databases and use a set up document to specify the external database, have the System Administrator perform an action that triggers the external lookup.

Lesson 3 - Document the Application Requirements

As well as creating development documentation for your application, you should also consider what information is required by the users of the application and the people that have to support it. Depending on the complexity of the application and the skills of the users you may consider supplying the following information in addition to the Design Documentation.

Help Desk Instructions

Specify the level of support expected by the Help Desk for this application. Give an overview of the application, including a list of all prerequisite systems/products or refer to the design documentation. List contact persons (application developer, application owner and so on). List known issues and workarounds. If you supply any training information, refer to that as well.

Training Instructions

The detail required in the training documentation will depend on the complexity of the application and the skill level of the target users. It can be a simple statement of the skills level required by the users, such as which standard courses they should have attended. If more is needed and the application is rolled out to most users at one time, the training instructions could be a reference to an application workshop. For a deployment that is spread over a longer period of time, a specially written "Getting Started" guide may be required

When Your Application is Deployed in Production

Lesson 1 - Define a Maintenance Server

As soon your application goes into production, the ACL has to be maintained by the database Manager. If your database is replicated to other servers, define one server where all the administrative work is performed. For example, changing the ACL on different servers before replication is performed will end up with one change getting lost. Provide all involved servers with the appropriate access rights (be aware of Reader fields), otherwise information and design changes will not replicate properly, causing inconsistency in your application.

Lesson 2 - Get Feedback From Your Users

Even though you, as the application developer, may have a very good understanding of the business process your application supports, most often people with a different backgrounds and skill sets will be the daily users of the application. Make sure that you collect feedback from these daily users so that you can incorporate their experience in updates and new applications.

There are several ways to do this. Go out to the users in their normal environment, see how they use the application and listen to their comments. Look at the Help Desk statistics to determine how much support is required for the application and in which areas. You could also set up a discussion database for feedback that the users can reach from within the application. However, do this only if the discussion database will be monitored by you, fellow developers, or super users, so that users will get replies.

Summary

This chapter has documented some of the lessons learned when developing applications with Lotus Notes and Domino. The information given is by no means a definitive list, or a list of the only considerations to take into account in your projects, but we hope that they provide you with a better insight into the wider issues that an application developer needs to consider.

Chapter 13
Introducing DECS and Database Connectivity

Domino Enterprise Connectivity Services (DECS) is a Domino server task that allows application developers to link their Lotus Domino databases to relational databases and access data from them in real time. DECS works by capturing certain Lotus Domino database events on the server, such as opening a form and triggering a predefined action on the relational database.

Installing and Running DECS

To load DECS on the Domino server, simply type the following at the server console:

```
LOAD DECS
```

The server will respond with the message, "Connection Server Started" along with the current date and time.

To shut down the DECS server, type the following at the server console:

```
TELL DECS QUIT
```

The server console responds with the message, "Connection Server Shutdown Complete" together with the current date and time.

To see if DECS is currently running, type **`SHOW TASKS`** at the server console and look for the "Connection Server" task in the listing.

Supported Data Sources

DECS can provide real-time connectivity to the following data sources:

Source	*Prerequisites*
IBM DB2	• DB2 Connect Personal Edition • DB2 Enterprise Edition **or** • DB2 Client Application Enabler (CAE) 2.1.2 or later • In addition, to connect to DB2 on an AS/400 or mainframe, DB2 Connect or the DB2 Access client must be installed.
EDA/SQL	• EDA/Client software for the host operating system. The EDA/Client version must be Release 3.2 or later and must be 32-bit on Windows NT and OS/2 • An EDA server on the platform where the EDA supported database resides • Connectivity to the EDA server
Open Database Connectivity (ODBC)	• The ODBC driver appropriate to the operating system • The driver must be 32-bit on NT and OS/2 • The ODBC driver must be thread-safe • The ODBC Administrator must be present • There must be correctly defined ODBC data sources in the ODBC Administrator
Oracle	• With an OS/2-based server: Oracle SQL*Net Version 2 • With a Windows NT based server: Oracle SQL*Net Version 1 or 2 • In either case the SQL*Net must be the same version as the SQL*Net installed on the Oracle data server. A network connection must exist between the server machine and the Oracle data server machine via SQL*Net • Native Oracle connectivity support requires Oracle Version 7.2 or later • OS/2 works only with Oracle 7.3 • Oracle Version 7.3 and HP-UX: You must obtain the Oracle fix for bug #441647. This patch applies to Oracle's libclntsh.sl • Oracle 8: NotesPump™ and DECS link with Oracle 7.3 libraries, which use SQL*Net for the communications layer. If you are using Oracle 8 with DECS or NotesPump, you must install Oracle SQl*Net. You may use the SQLNET Easy Config to configure SQl*Net.

continued

Source	*Prerequisites*
Sybase	• With an OS/2-based or Windows NT-based: System10 Netlib • A network connection must exist between the Domino server and the Sybase SQL server via Netlib
Zmerge Text	• No additional requirements other than access to the text file

Setting Up Connectivity to DB2

This section provides information about the software required to connect to DB2. This information is provided to help you get started. You should refer to the documentation for the specific software you are using for complete instructions.

IBM is currently shipping IBM DB2 UDB Version 5. The specific product to use will depend on your environment.

Here is a brief listing of connectivity software available with DB2 Version 5:

- **DB2 Workgroup Edition**: Includes Client Pack CD for client connectivity, but does not include support for MVS/ESA, OS/390, OS/400, VM, and VSE.
- **DB2 Enterprise Edition**: Includes all the functionality of DB2 WorkGroup Edition, plus support for host connectivity providing users with access to DB2 databases residing on host systems including MVS/ESA, OS/390, OS/400, VM, and VSE.
- **DB2 Client Application Enabler**: Enables a client workstation to access the DB2 server. Refer to DB2 documentation for supported platforms.
- **DB2 Universal Database Personal Edition**: Formerly known as DB2 Single Server. Enables you to create and use local databases, and to access remote DB2 databases. Available for OS/2, Windows 95, and Windows NT.
- **DB2 Connect Enterprise Edition**: Formerly known as DDCS Multi-user gateway. Provides access from clients on the network to DB2 databases that reside on hosts such as MVS/ESA, OS/390, OS/400, VM, and VSE.
- **DB2 Connect Personal Edition**: Formerly known as DDCS Single-User. Provides access from a single workstation to DB2 databases residing on hosts such as MVS/ESA, OS/390, OS/400, VM and VSE. This product is only available for OS/2, Windows 95, and Windows NT.

Refer to the documentation provided with IBM DB2 Universal Database (the manual entitled *Road Map to DB2 Programming*, Appendix A, "About DB2 Universal Database").

Testing Connections With LCTEST

Before getting started with DECS there is a useful tool called LCTEST that helps you determine whether your server is correctly configured to access the relevant database.

Tip Establishing that your server has been correctly configured at this stage will potentially save you hours of anguish later.

Before running LCTEST, you must have the appropriate software installed on the Domino host for each data source you want to test. The remaining chapters of this manual provide information about the software required for each of the supported data sources.

Running LCTEST

Follow the steps below to run LCTEST.

1. Locate the LCTEST.EXE program specific to your operating system platform in the Domino program directory. The LCTEST program has the following names for each of the associated operating system platforms:
   - NLCTEST.EXE: for Windows 95 or Windows NT (Win32)
   - ILCTEST.EXE: for OS/2
   - ALCTEST.EXE: for Windows NT/Alpha
2. Double-click the program name to launch it or type the program name at the system prompt. The LCTEST screen appears, as shown below:

3. Enter the number of the test you want to run and press **ENTER**. Depending on the type of data source you are testing, you are prompted to enter additional information to log in to the specified data source.

DB2 Connectivity Test

You should test for connectivity to the DB2 servers. To test for connectivity:

1. Run the version of the test program LCTEST, located in the Domino program directory appropriate to your operating system.
2. Select DB2 from the program menu.
3. When the program prompts for a DB2 Database, UserName, and Password, enter valid connection information. The database must be cataloged in the DB2 database directory. Refer to your DB2 Client documentation for further information on configuring a connection to a database.
4. After entering the DB2 database, user name, and password information, the program will attempt to connect to the DB2 server. A message will appear, telling you if the test was successful or not.
5. You can retry a connection by entering Y at the Try Again? [N] prompt. This enables you to re-enter all of the required information, in case a mistake was made in spelling or you entered the wrong database, user name, or password the first time around.

Configuring DECS

There are two methods for configuring DECS to access a supported external data source:

1. Use the Connection Server Administrator database to create the Connection and Activity documents.
2. Use the LotusScript Extension for Lotus Domino Connectors (LSX LC).

Using the Connection Server Administrator

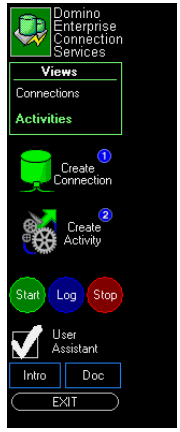
The most simple method of creating a link to a database is by using the wizards provided. These wizards guide you through a step-by-step process to connect your Lotus Notes application to an external data source.

Before you can use an external data source, you need to create two documents in the Connection Server Administrator database: a Connection document and an Activity document. The Connection document describes what kind of data source DECS is connecting to and the Activity document describes what to do with the connection once it has been made. We will explain more about how to configure these two documents later in this section.

The DECS Administrator Navigator

The DECS Administrator navigator allows you to create new Connection and Activity documents in the database and lets you control the DECS process running on the server.

The DECS Administrator navigator is shown below:



The Views Section: Selects a view of defined connections for data, or a view of RealTime activities.

The Green Database Icon: Creates a new data connection. This launches a form for defining the connectivity necessary to access an external data source.

The Cogs Icon: Creates a new RealTime activity. When the User Assistant is active, this launches a wizard that prompts you through the process of defining a RealTime connection between the Domino application and the external data source. When the User Assistant is turned off, this displays a blank RealTime Activity document which you can edit.

The Green Start Icon: Begins execution of the currently selected RealTime activity. This has no effect if the current selection is already executing. This is disabled when in the Connections view.

The Blue Log Icon: Displays the status of the currently selected RealTime activity. If the current selection is running, this will display the current status and any errors that have occurred. If the current selection is not running, this will display the results of the most recent execution. This is disabled in the Connections view.

The Red Stop Icon: Ends execution of the currently selected RealTime activity. This has no effect if the current selection is not running. This is disabled in the Connections view.

The Tick or Check: Toggles the User Assistant. When turned on, this enables the RealTime activity wizard and provides additional help. This is useful for first time and infrequent users. The New Activity button runs a wizard to guide you through creating the RealTime Activity document and provides information to assist in the creation and editing of the document.

The Intro Icon: displays the "Help About" document.

The Doc Icon: Displays the online documentation.

The Exit Icon: Closes the Connection Server Administrator.

Creating a Simple RealTime DB2 Connection

Let's now take a look at how we can create a real time link from a simple Domino database into DB2.

In the following example, we are going to create a Connection document to the sample database that ships with DB2, create a simple Domino database to display the information, and create an Activity document to retrieve the information from DB2 and display it in the database.

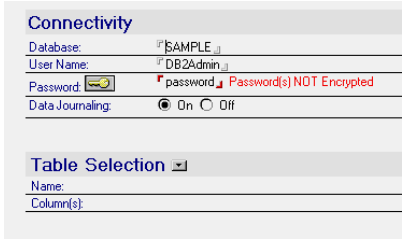
Note The following example assumes that you have already installed and configured DB2, the SAMPLE database and have a connection to the DB2 server. It also assumes that you have installed DECS and have it running on the Domino server.

1. If you have not already done so, start the Notes client.
2. Open the Connection Server Administrator database on the server. The connection navigator is displayed (if it is not, click the Connections hotspot in the navigator).

   Note You cannot open the database locally on your server.
3. To create a new link to the datasource, click the green database icon in the navigator. The "Other" dialog box is displayed.

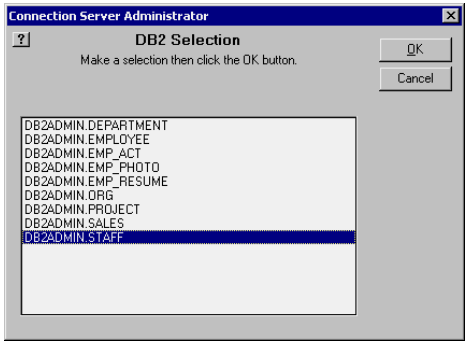
4. Select Connection to DB2 and click OK. The Connection to DB2 document is displayed.

5. Enter the information as shown in the following figure:



Note The User Name and Password may be different on your system.

6. Click the small arrow next to the text Table Selection and wait a few seconds while the DECS server connects to the database and retrieves a list of available tables. The DB2 Selection dialog box is displayed with a list of available tables the user ID has access to.

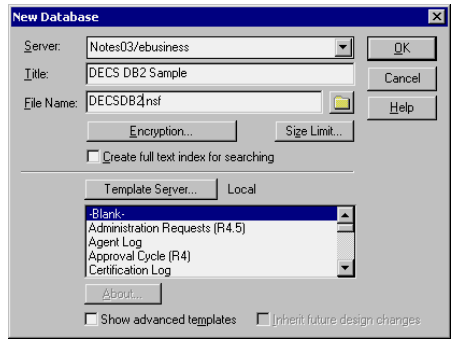


7. Select the DB2ADMIN.STAFF table and click the OK button. There is a short delay while DECS accesses the selected table and retrieves a list of the available fields and their data types — this will be useful later when creating the Domino application.
8. Click the Save and Close action button. The document is saved and you are returned back to the Connection Server Administrator navigator.

Creating the Lotus Notes Database

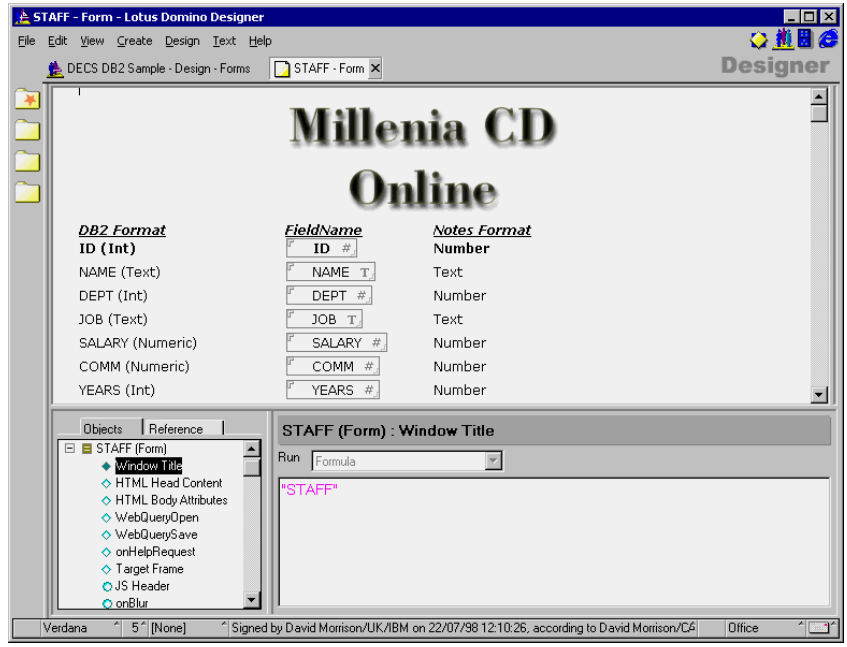
The next stage is to create a database in Domino that contains the fields you want to retrieve and display from the external data source. To keep it simple, we are going to create a database with a single form that contains all the fields from the DB2 table and a single view.

1. Choose File - Database - New from the menu. The New Database dialog box is displayed. Complete the dialog box with the settings shown in the following figure and click the OK button to create the database.



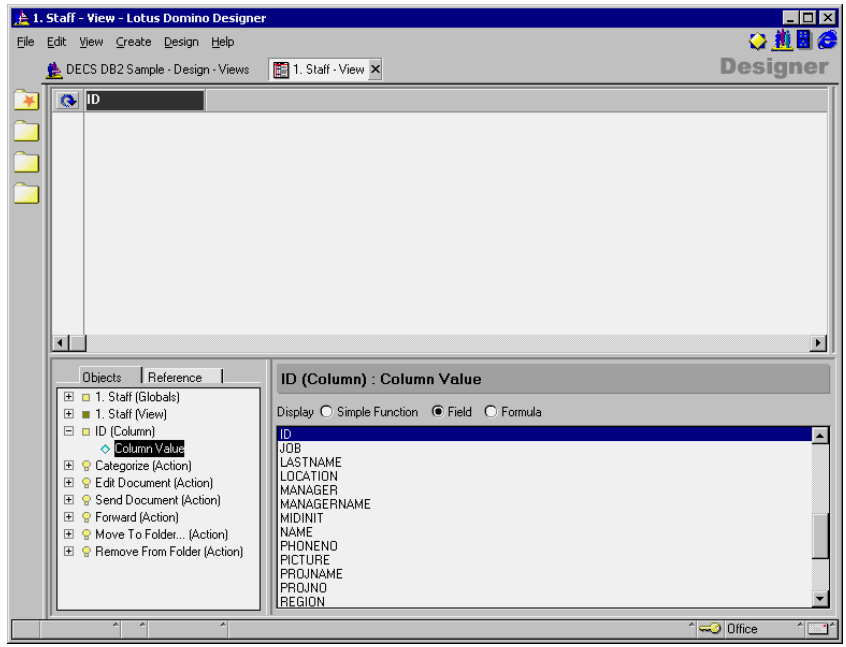
2. When the new database has been created, choose Create - Design - Form from the menu bar. The default blank new form is displayed.

3. By using the information from the DECS DB2 Connection document we created earlier, we can create a form that contains all the fields from the DB2 table. In the following diagram the left-hand column has been copied and pasted from the DECS Connection document, the center column contains the Notes fields, and the third column lists the Notes data format.



Note The order of the fields pasted in from the DECS Connection document has changed.

4. Save the new form by choosing File - Save from the menu bar, name the form STAFF and then choose File - Close to close the form.
5. Now let's create a simple view that contains just the ID field in a column. Select Views from the Design navigator and open the (*untitled) view.
6. Double-click the first column to bring up the InfoBox and enter ID in the Title field, click the Sorting tab, select Ascending, and close the InfoBox.
7. Select Field from the Design pane, and select the ID field.



8. Save the view as "1. Staff" and close the view.

Creating the Activity Document

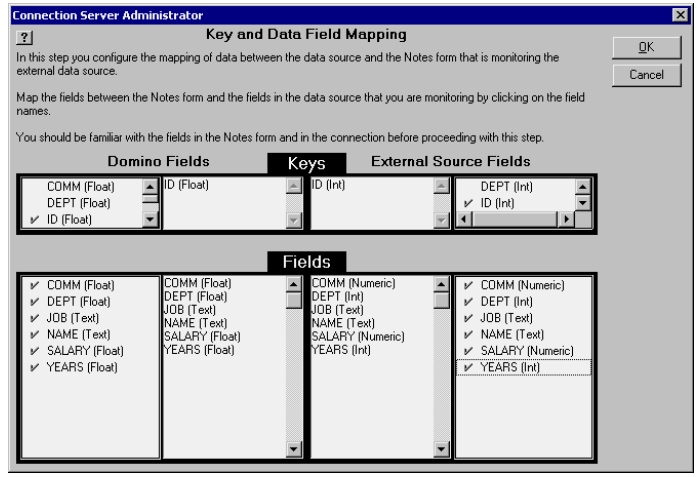
The next step is to create an Activity document for the new database and the Connection document you created earlier.

1. Open the DECS Connection Server Administrator database.
2. Click the small icon in the navigator with the cogs and a small number two. This will start the Configuration Wizard by creating a new RealTime Activity document. A pop-up dialog box with some Help is displayed. After reading the Help, click the OK button to close the dialog box.
3. The next screen that is displayed is the Select Domino Database dialog box which contains a listing of all the databases on the server. Select the DECSDB2.nsf database you created earlier and click the OK button. The Notes Selection dialog box is displayed.

4. Select the STAFF form from the listing and click the OK button.
5. DECS then needs to know which data source you want to connect your Domino database to, so select DB2 SAMPLE (using "DB2Admin") — DB2Admin.STAFF from the listing and click OK.
6. The next dialog box displayed is the Key and Data Field Mapping dialog box. This dialog box is split into two sections: the first asks you which is the unique key that will link the external data source to the Notes application, and the second asks you which fields should be transferred from the external data source and mapped to the Domino database fields.

   Important The field that you select for the key field from the external data source must be unique. It is this field that DECS uses to link the Notes application and the external data source.

   In our example the unique key is the ID field, so select this in the Domino Fields and also in the External Source Fields list boxes.
7. Select all the fields in the bottom Fields section on both sides and click the OK button.



8. The RealTime Event Selection dialog box is displayed. This dialog box lets you select which events DECS should monitor on the Domino server. The options are Create, Open, Delete, and Update. Select Open from the list and click the OK button.
9. A message appears confirming that the Activity document now has all the required information to run. After reading this, click the OK button to close the dialog box.
10. Click the Save and Close button on the Action Bar to save and close the document.
11. If the Activities view is not displayed, click Activities in the navigator.

Populating the Lotus Notes Database With Key Data

Before we can access any information from the external data source, we must populate the Domino database with values from the key column in the DB2 database. Luckily, there is a very simple way to do this courtesy of DECS.

1. From the DECS Connection Server Administrator database, display the Activities view. If it is not displayed, click Activities in the navigator.
2. Select the activity you created in the previous section.
3. Choose Actions - Tools - Initialize Keys from the menu bar. A warning message tells you that you are just about to create a new Notes document in the Domino database for every key in the external data source. Read this message and click the Yes button to start the process.
4. After a short delay while the DECS server creates the Notes documents, a message will be displayed telling you how many documents were created. This number will equal the number of unique keys in the external data source. Click the OK button to close the message box.
5. Open the DECS DB2 Sample database and you will see that a number of documents have been created.

Note Only the ID field from the external data source has been copied into the Notes documents.

Running the DECS Activity

Before you can use the DECS activity, you must start it from the DECS navigator.

1. From the Activities view, select the activity you created earlier between the DB2 STAFF table and our DECS DB2 Sample database.
2. Click the green Start button in the DECS navigator. Notice that the status on the Activity document changes to Starting. After a few seconds, refresh the view by pressing **F9** until the status changes to Active.
3. Open the DECS DB2 Sample database and open any of the documents displayed in the view. Any fields that contain data in the external data source will now be displayed in the Notes form.

A Brief Outline of What is Happening

When a RealTime activity has been enabled, DECS opens a link to the data source and monitors the specified Domino database for certain events, such as an open event on a particular form.

When a recognized event is captured, DECS reads the Key field value from the document and generates a query into the external data source to retrieve the required field values.

DECS then adds these field values into the Notes document and displays it to the user.

Accessing More Than One Table

More often than not, you will probably need to access data that is stored in multiple tables rather than a single table, and DECS allows you to do this in a number of ways.

1. Create a view of your data in the external database. A view is effectively a table that is created in memory from an SQL SELECT query and, to most applications, appears as a real table.

   Note Don't get confused between a Notes view and a database view; the statement above refers to creating a view in the external datasource, not in Notes.

2. Create multiple connections in the DECS Connection Server Administration database and use the Monitor Order field to control the execution of each connection.

3. Create a stored procedure in the database that accepts the required Key fields to execute an SQL query as an input parameter. It should return the fields you require to populate your form.

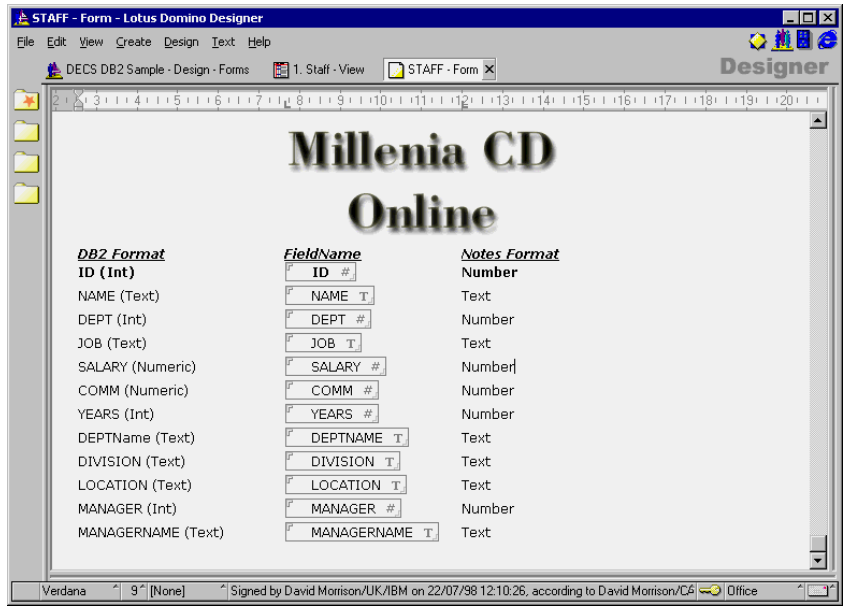
The method that you choose depends on the individual application you are creating. For example, if you have a large number of tables to join together, creating a view is probably not a good option because there is a significant overhead on the external data source database server to generate the view before the data can be returned.

The simplest method to access data from multiple tables is to create multiple Connection documents to a data source and specify a different value in the Monitor Order fields of each.

Monitor Order

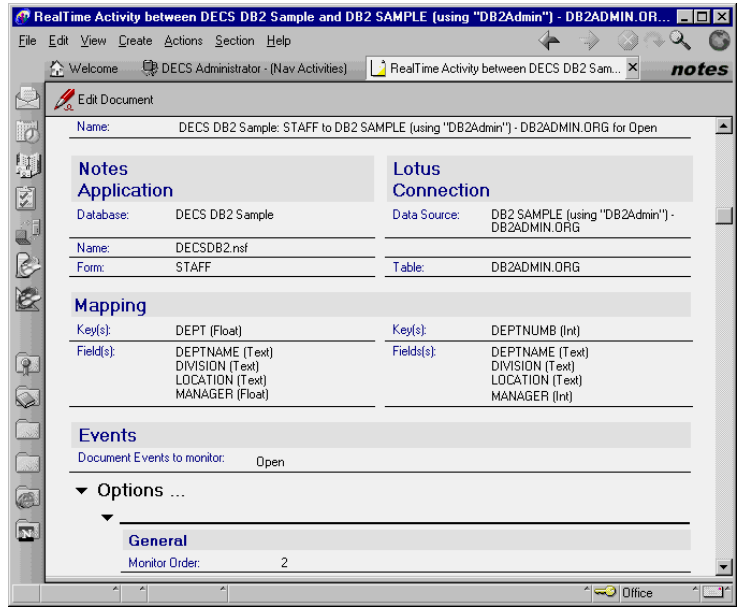
In the following example we will add the employee department name, division and location from the ORG table of the SAMPLES DB/2 database.

1. Open the STAFF form created in the previous example and add the following fields to the bottom of the form:



2. To retrieve the DEPTNAME, DIVISION and LOCATION fields from the DB/2 table we will use the DEPT field as the index field. Because this field is not stored in the Domino database, but is retrieved from the DB/2 database using the Activity document we set up earlier, we must configure the new Activity document to trigger after the first activity has completed.
3. Create a new Connection document as before, but this time use the ORG table from the DB/2 SAMPLE database.
4. Create a new RealTime Activity document and select the ORG Connection document and the Domino database where your STAFF form is located. Select DEPT and DEPTNUMBER as the key field and DEPTNAME, DIVISION, LOCATION, MANAGER as the fields to retrieve from DB/2.

5. Expand the Options - General section and type the value “2” into the Monitor Order field. The following is a screen capture of the new Activity document settings:



Note The Monitor Order field value is set to “2” at the bottom of the screen.

6. Save and close the document by clicking the Save and Close action button at the top of the screen.
7. Start the activity by clicking the Start button in the navigator.
8. After the activity has started, open the DECS DB2 sample database and open one of the documents. You will see, at the bottom of the screen, that the DEPTNAME, DIVISION, LOCATION and MANAGER fields have been completed with the employee’s information.

LotusScript Extension for Domino Connectors

A new LotusScript Extension (LSX), the LotusScript Extension for Lotus Domino Connectors (LSX LC) is delivered with DECS. This LSX allows you to access the functionality of DECS from within the LotusScript environment and code agents, action buttons, form events and so on, to perform actions on an external data source.

The classes within the new LSX are:

- **LC_Session** to handle available connectors and errors.
- **LC_Connection** to handle the connection to the data source.
- **LC_FieldList** to handle arrays of row data from the data source.
- **LC_Field** to handle individual data fields from the data source.
- **LC_Currency, LC_Datetime, LC_Numeric, LC_Stream** to handle individual data types.

The LotusScript Extension for Lotus Domino Connectors is similar to the NotesPump LSX, so if you are familiar with the syntax of the NotesPump LSX it will not take you long to understand the new LSX LC. There are some excellent example scripts and documentation provided with the LSX LC that you should look at to become familiar with the functionality of this LSX. Below is an example to get you started.

Reading Data From an External Source into a Notes Form

The following example displays employee information in a Notes document using the SAMPLE database from DB/2 UDB 5. It shows you how to populate a dynamic keyword list from the external database and then, using this value as a key field, retrieve additional information on the employee from multiple tables in the external data source.

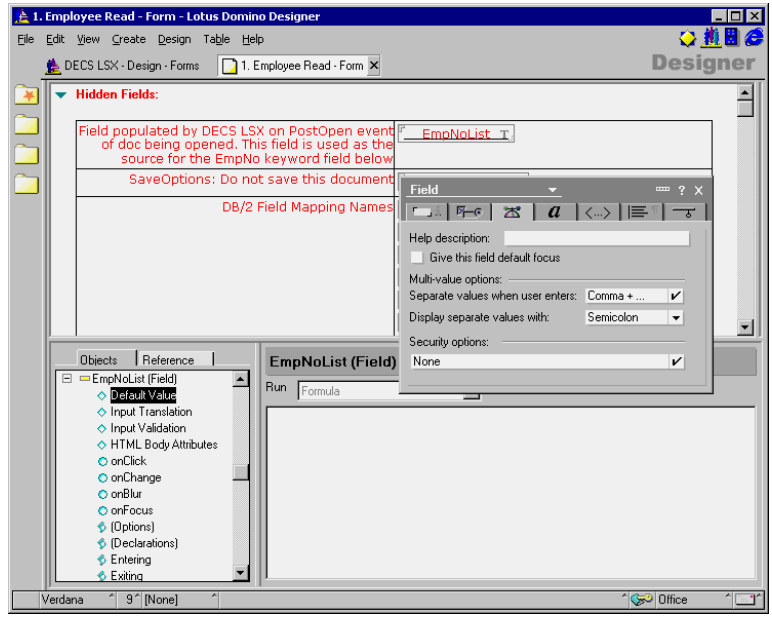
Note This code assumes that you have installed the necessary code on your workstation to connect to DB2. To use this code in production you would need to install the DB/2 code on each Notes workstation that is to connect to the data source. You could also install the DB/2 code on the Domino server and run an agent on the Domino server, or create a Domino Web application that executes the LSX LC code on the Domino server.

Step 1. Retrieving the Key Values from DB/2

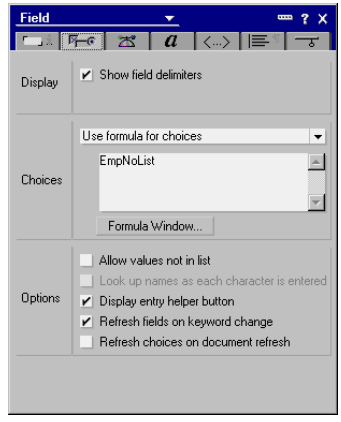
The first step is to populate the keyword field in the Notes document with the unique keys from the DB/2 database. We have done this by executing LotusScript code in the PostOpen event of the form. This code logs in to the DB/2 SAMPLE database and runs a simple query on the EMPLOYEE table to retrieve a list of all the values in the EMPNO and LASTNAME field. The

retrieved information is added to a keyword field on the Notes document and the document is refreshed.

1. Create a new form in your database and add a hidden text field called EmpNoList. Select Allow Multi-Values on the Basics tab of the InfoBox for the field. On the Advanced Options tab of the InfoBox, select Commas under Multi-value options.



2. Create a new field called EmpNo and make this field an editable computed keyword field with the formula EmpNoList (the hidden text field we have just created).



3. For the form Options event, add the following code:

```
USELSX "NLSXLC"
```

4. For the forms PostOpen event, add the following code:

```
Sub Postopen(Source As Notesuidocument)
     Dim LC_S As New LCSession
     Dim LC_Conn As New LCConnection("db2")
     Dim LC_FldLst As New LCFieldList(1)
     Dim LC_Field As New LCField(LCTYPE_TEXT, 1)
     Dim Count As Long
     Dim SelectStatement As String
     Dim workspace As New NotesUIWorkspace
     Dim uidoc As NotesUIDocument

     Set uidoc = workspace.CurrentDocument
     On Error Goto ErrorHandler
     LC_Conn.Userid = "db2admin"
     LC_Conn.Password = "password"
     LC_Conn.Database = "SAMPLE"

     LC_Conn.Disconnect
     LC_S.ClearStatus
     LC_Conn.Connect
     debug_a = LC_S.GetStatusText(LC_S.GetStatus)
    ' Perform the select from the EMPLOYEE table
     ' and for display convienience order it by the
     ' lastname field
   SelectStatement = "SELECT * FROM EMPLOYEE ORDER BY
LASTNAME"
     count = LC_Conn.Execute(SelectStatement, LC_FldLst)
     If count <> 0 Then
          count = LC_Conn.Fetch(LC_FldLst, 1, 1)
          Set LC_Field1 = LC_FldLst.GetField(1)
          Set LC_Field2 = LC_FldLst.GetField(4)
          IDs = ""
          LastNames = ""
          While (count > 0) And LC_S.Status = LC_Success
            Ids = Ids + LC_Field2(0) + "|" + LC_Field1(0) + ","
            count = LC_Conn.Fetch(LC_FldLst, 1, 1)
          Wend
          uidoc.FieldSetText "EmpNoList", IDs
         uidoc.refresh
     End If
     End
' Handle any errors we encounter and display a message
ErrorHandler:
     Dim msg As String
     Dim ErrorTxt As String
```

```
     Dim msgcode As Long
     Dim Status As Integer
     If LC_S.status <> LCSuccess Then
          status = LC_S.GetStatus(errortxt, msgcode, msg)
     Else
          errortext = "Error: " & Err() & ": " & Error()
     End If
     Messagebox errortxt, MB_IConInformation + MB_OK, "LS
Error"
     End
End Sub
```

We will now look at this code and explain each section.

5. The first part of the code sets up the connection to the data source and executes and SQL query.

   It sets up the connection using DB2Admin as the Userid to access the SAMPLE DB/2 database, password as the password and SAMPLE as the database to connect to:

```
LC_Conn.Userid = "db2admin"
LC_Conn.Password = "password"
LC_Conn.Database = "SAMPLE"
```

   Now, to make sure that any previous sessions that may have aborted are cleaned up, we make sure that the connection is disconnected and that the connection status is reset to normal:

```
LC_Conn.Disconnect
LC_S.ClearStatus
```

   Now, issue the connect command. You can use the debug_a variable when debugging to check the status of the connection attempt:

```
LC_Conn.Connect
debug_a = LC_S.GetStatusText(LC_S.GetStatus)
```

   Create an SQL select statement to retrieve the required data from the data source. Here we are selecting all columns from the EMPLOYEE table and ordering it by the employee last names. This is so that when we display them in the Notes document, they are displayed in a logical order:

```
SelectStatement = "SELECT * FROM EMPLOYEE ORDER BY
LASTNAME"
```

   Now execute the SQL statement on the connection and return the values in to the LC_FldLst variable. Set a variable to the number of rows

returned. In some data sources this will be -1, meaning that the number of rows is undetermined, this is not an error:

```
count = LC_Conn.Execute(SelectStatement, LC_FldLst)
```

6. Step through each row that is returned from the SQL Select statement and add it to the text field on the Notes form.

   If the value returned from the SQL statement is not zero, that is, there was no error, fetch a field list record from the data source:

```
If count <> 0 Then
  count = LC_Conn.Fetch(LC_FldLst, 1, 1)
```

   Now, set two LC_Field variables to the values stored in columns one and four. In our EMPLOYEE table this is the EMPNO field and the LASTNAME field:

```
Set LC_Field1 = LC_FldLst.GetField(1)
Set LC_Field2 = LC_FldLst.GetField(4)
```

   Set two variables to a blank text string:

```
IDs = ""
LastNames = ""
```

   While the variable count is greater than zero (there are still rows to retrieve from the data source) and the Lotus Connectors session status is OK, get each record from the data source:

```
While (count > 0) And LC_S.Status = LC_Success
```

   Set the value of the variable IDs to the value stored in the LASTNAME field and add concatenate this together with the EMPNO field. Because we are building a list we add the comma to the end of each value pair as the delimiter.

   Note In Notes you can set up aliases for keyword fields so that they display one value, but store another. In this example we are going to use this feature to display a keyword list field of Lastnames to the user and use the alias (which is the corresponding EMPNO) to query the data source. In an alias list, you separate values with the vertical bar:

```
IDs = IDs + LC_Field2(0)  +  "|" + LC_Field1(0) + ","
```

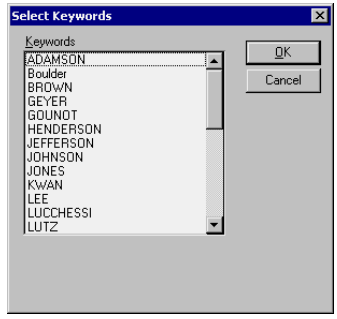
   Finally, we set the value of our text field to the value stored in the variable IDs:

```
uidoc.FieldSetText "EmpNoList", IDs
```

7. An example of the values stored in the IDs field is shown below:

   ADAMSON|000150, Boulder|000010, BROWN|000200, GEYER|000050, GOUNOT|000340, HENDERSON|000090, JEFFERSON|000230, JOHNSON|000260, JONES|000210, KWAN|000030, LEE|000330, LUCCHESSI|000110, LUTZ|000220, MARINO|000240, MEHTA|000320, NICHOLLS|000140, O'CONNELL|000120

8. If you save and test this form, you should be able to see a list of employee names from the DB/2 EMPLOYEE table displayed in the Keywords list field.



   Note The aliases for LastNames are hidden. Aliases are only used by the program.

Step 2. Querying the Tables Using the Selected Key Value

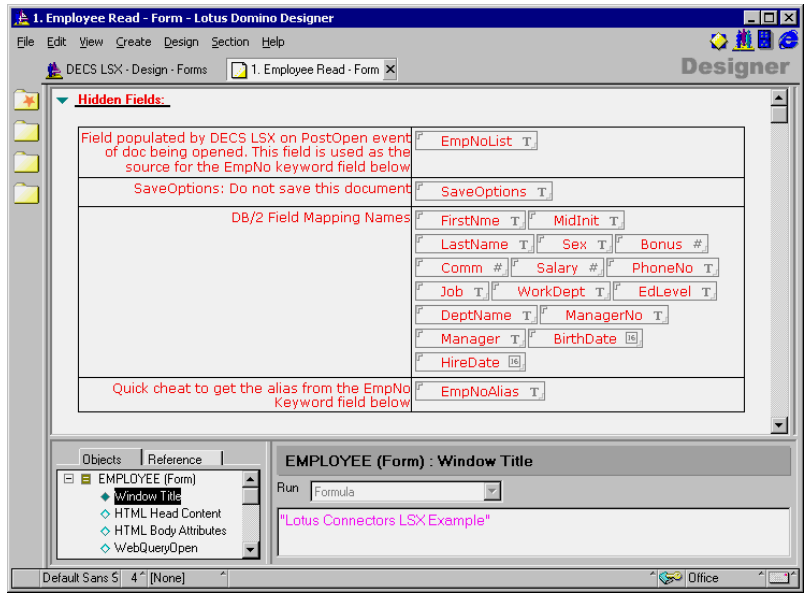
The next step in the example is to create the link between the selected keyword value and the database, so that we can populate the Notes document with information about the employee.

1. Open the form again in Edit mode and add the following editable hidden fields. We want these to be hidden because this is a Read-only document.

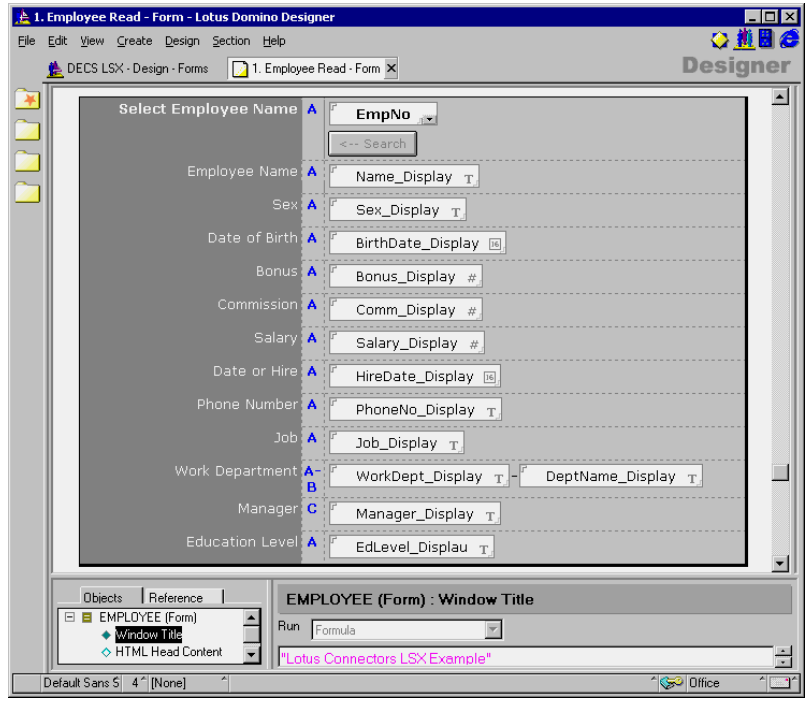
   FirstNme, text
   MidInit, text
   LastName, text
   Sex, text
   Bonus, number
   Comm, number
   Salary, number
   PhoneNo, text
   Job, text
   WorkDept, text
   EdLevel, text
   DeptName, text

DeptName, text
ManagerNo, text
BirthDate, date
HireDate, date

2. Add a hidden text field called SaveOptions and set the default value to zero (0). SaveOptions is a special field in Notes that does not allow the user to save the document when set to zero, and always saves the document when set to one (1).
3. Add a new hidden computed field called EmpNoAlias with a formula of EmpNo. This is a quick cheat field that we will use to get the alias from the selected LastName in the keyword field EmpNo.
4. Below is a list of all the hidden fields in the form:



5. Now add corresponding Computed for Display Only fields to the form and assign the hidden fields as the default values. The following figure shows the fields on the form:



Note For the Name_Display field, the fields FIRSTNME, MIDINIT and LASTNAME have been added together.

6. Create a button on the form by choosing Create - Hotspot - Button from the menu bar. This is the button that we will click to populate the rest of the fields on the form based on the value selected in the keyword field EmpNo. For the Click event for the button enter the following code:

```
Sub Click(Source As Button)
     Dim LC_S As New LCSession
     Dim LC_Conn As New LCConnection("db2")
     Dim LC_FldLst As New LCFieldList(1)
     Dim LC_FldLst2 As New LCFieldList(1)
     Dim LC_Field As New LCField(LCTYPE_TEXT, 1)
     Dim Count As Long
     Dim SelectStatement As String
     Dim workspace As New NotesUIWorkspace
     Dim uidoc As NotesUIDocument
     Set uidoc = workspace.CurrentDocument
     On Error Goto ErrorHandler
```

```
LC_Conn.Userid = "db2admin"
LC_Conn.Password = "password"
LC_Conn.Database = "SAMPLE"
LC_Conn.Connect
Debug_A = LC_S.GetStatusText(LC_S.GetStatus)
EmpNo = uidoc.FieldGetText("EmpNoAlias")
SelectStatement = "SELECT * FROM EMPLOYEE " _
  "WHERE EMPNO = '" & EmpNo & "'"

count = LC_Conn.Execute(SelectStatement, LC_FldLst)

If count <> 0 Then
     count = LC_Conn.Fetch(LC_FldLst, 1, 1)
     uidoc.FieldSetText "FirstNme",
       LC_FldLst.FIRSTNME(0)
     uidoc.FieldSetText "LastName",
       LC_FldLst.LASTNAME(0)
     uidoc.FieldSetText "MidInit",
        LC_FldLst.MIDINIT(0)
     uidoc.FieldSetText "Sex", LC_FldLst.SEX(0)
     Set dt_TempDate = New NotesDateTime(
        LC_FldLst.BIRTHDATE(0))
     uidoc.FieldSetText "BirthDate",
        dt_TempDate.DateOnly
     uidoc.FieldSetText "Bonus",
        LC_FldLst.BONUS(0)
     uidoc.FieldSetText "Comm", LC_FldLst.COMM(0)
     uidoc.FieldSetText "Salary",
       LC_FldLst.SALARY(0)
     Set dt_TempDate = New NotesDateTime(
       LC_FldLst.HIREDATE(0))
     uidoc.FieldSetText "HireDate",
       dt_TempDate.DateOnly
     uidoc.FieldSetText "PhoneNo",
       LC_FldLst.PHONENO(0)
     uidoc.FieldSetText "Job", LC_FldLst.JOB(0)
     uidoc.FieldSetText "WorkDept",
       LC_FldLst.WORKDEPT(0)
     uidoc.FieldSetText "EdLevel",
       LC_FldLst.EDLEVEL(0)
    If LC_FldLst.WORKDEPT(0) <> "" Then
          SelectStatement = "SELECT * FROM DEPARTMENT
            WHERE DEPTNO = '" &
            LC_FldLst.WORKDEPT(0) & "'"
          count = LC_Conn.Execute(SelectStatement,
            LC_FldLst2)
          If count <> 0 Then
               count = LC_Conn.Fetch(LC_FldLst2,1,1)
               uidoc.FieldSetText "DeptName",
```

```
                    LC_FldLst2.DEPTNAME(0)
                   uidoc.FieldSetText "ManagerNo",
                    LC_FldLst2.MGRNO(0)
                  SelectStatement = "SELECT * FROM
                   EMPLOYEE WHERE EMPNO = '" &
                   LC_FldLst2.MGRNO(0) & "'"
                   Set LC_FldLst = New LCFieldList(1)
                   count = LC_Conn.Execute(
                     SelectStatement, LC_FldLst)
                   If count <> 0 Then
                        count = LC_Conn.Fetch(LC_FldLst,
                          1, 1)
                        uidoc.FieldSetText "Manager",
                          LC_FldLst.LASTNAME(0)
                   End If
              End If
         End If
         uidoc.refresh
    End If
End
ErrorHandler:
    Dim msg As String
    Dim ErrorTxt As String
    Dim msgcode As Long
    Dim Status As Integer
   If LC_S.status <> LCSuccess Then
         status = LC_S.GetStatus(errortxt, msgcode, msg)
    Else
         errortext = "Error: " & Err() & ": " & Error()
    End If
    Messagebox errortxt, MB_IConInformation + MB_OK, "LS
       Error"
    End
End Sub
```

7. This script initially follows the same course as the PostOpen form event. It sets up a connection to the DB/2 database and then connects to it. The next step is to send an SQL query to the database to select all rows from the EMPLOYEE table where the employee number, EMPNO, is equal to the one selected in the keyword field on the document.

   First, we retrieve the value stored in the EmpNoAlias hidden field. This field only stores the alias for the selected item, not the actual text of the last name:

```
EmpNo = uidoc.FieldGetText("EmpNoAlias")
SelectStatement = "SELECT * FROM EMPLOYEE WHERE EMPNO = '" &
EmpNo & "'"
```

8. Next, check to see if the value of count is zero (0), an error, and if it is not, fetch the first row from the result set:

```
If count <> 0 Then
count = LC_Conn.Fetch(LC_FldLst, 1, 1)
```

9. Start retrieving rows from the result set and display them on the form:

```
uidoc.FieldSetText "FirstNme",LC_FldLst.FIRSTNME(0)
uidoc.FieldSetText "LastName",LC_FldLst.LASTNAME(0)
```

   For date values you will need to assign them to a NotesDateTime value and use the relevant property for the type of date field you have set up. Here we simply use the DateOnly value:

```
Set dt_TempDate = New NotesDateTime(
    LC_FldLst.BIRTHDATE(0))
uidoc.FieldSetText "BirthDate",dt_TempDate.DateOnly
```

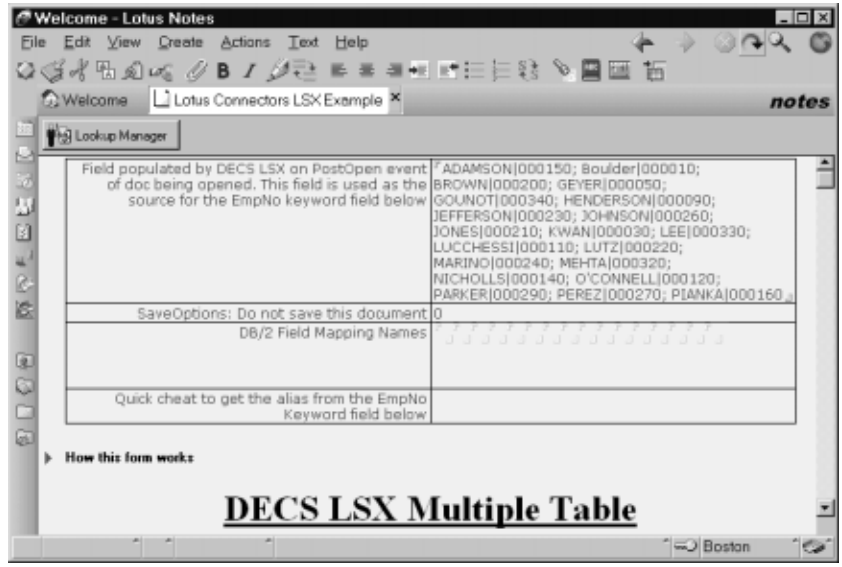
10. To retrieve the department name, we need to query the DEPARTMENT table using the value retrieved from the WORKDEPT field in the EMPLOYEE table. The following code first determines whether there is a value in the WORKDEPT field and if so generates a new SQL query:

```
If LC_FldLst.WORKDEPT(0) <> "" Then
SelectStatement = "SELECT * FROM DEPARTMENT
_WHERE DEPTNO = '" & LC_FldLst.WORKDEPT(0) & "'"
```

11. Finally, the manager's name is retrieved from the EMPLOYEE table using the value retrieved for the MGRNO field from the DEPARTMENT table:

```
SelectStatement = "SELECT * FROM EMPLOYEE
_WHERE EMPNO = '" & LC_FldLst2.MGRNO(0) & "'"
```

12. The figure below shows the fully populated Notes document:



In summary, this example has populated a keyword field from a DB/2 database with the last names of employees and used the employee number as a keyword alias to retrieve data from different tables and display them in the Notes document.

Note This example shows you how to create multiple select statements on multiple tables. In a production environment it is likely that you will construct a single SQL statement joining the tables together, for example:

```
SELECT A.EMPNO, A.LASTNAME, A.WORKDEPT, B.DEPTNAME
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.WORKDEPT=B.DEPTNO
```

You will then have all the fields you require in a single result set.

Updating Data From Notes to an External Source

Following on from the example above which read data from a table, the next logical step is to be able to update those values in the external data source from information entered into the Notes document.

The following example shows you how to perform a simple update into the sample DB/2 EMPLOYEE table which contains a unique primary index, the EMPNO field.

1. Open the form you created earlier for the READ example above and create a new button called Update. To create the button choose Create - Hotspot - Button from the menu
2. For the Click event for the button add the following LotusScript code.

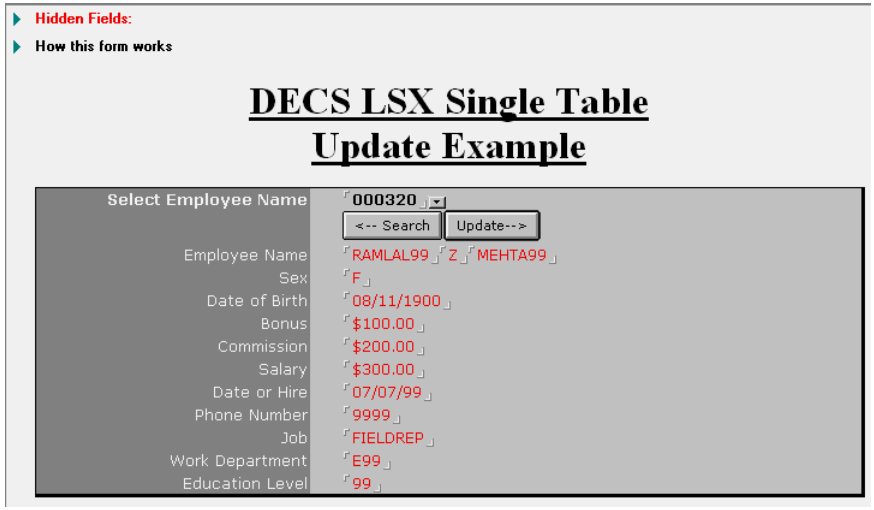
```
Sub Click(Source As Button)
     Dim workspace As New NotesUIWorkspace
     Dim uidoc As NotesUIDocument
     Dim srcCon As New LCConnection("db2")
     Dim fldLst As New LCFieldList(1, _
             LCFIELDF_TRUNC_DATA+LCFIELDF_TRUNC_PREC)
     Dim FirstNmeFld, MidInitFld As New LCField(LCTYPE_TEXT, 1)
     Dim LastNameFld, SexFld As New LCField(LCTYPE_TEXT, 1)
    Dim BirthdateFld, Bonusfld As New LCField(LCTYPE_TEXT, 1)
    Dim CommFld, SalaryFld As New LCField(LCTYPE_TEXT, 1)
     Dim HireDateFld, PhoneNoFld As New LCField(LCTYPE_TEXT, 1)
     Dim JobFld, WorkDeptFld,As New LCField(LCTYPE_TEXT, 1)
     Dim EdLevelFld As New LCField(LCTYPE_TEXT, 1)
     Dim EmpNoKeyField As New LCField(LCTYPE_TEXT, 1)
     Dim EmpNo As String
     Set uidoc = workspace.CurrentDocument
     ' Set up our connection to the data source
     SrcCon.Database = "SAMPLE"
     SrcCon.UserID = "DB2Admin"
     SrcCon.Password = "surfer"
     SrcCon.Metadata = "EMPLOYEE"
     SrcCon.Connect
     ' Get the value stored in the EmpNo field on the
     ' Notes document
     EmpNo = uidoc.FieldGetText("EmpNo")
     ' Issue a SQL Select statement to retrieve the row that
     ' we are to update
     SelectStatement = "SELECT * FROM EMPLOYEE
                      _WHERE EMPNO = '"
                       & EmpNo & "'"
     ' Retrieve the row into our FldLst variable
     count = SrcCon.Execute(SelectStatement, FldLst)
     count = SrcCon.Fetch(FldLst, 1, 1)
     ' Make sure that there was no error,
     ' for example, we could not retrieve a row.
     If count <> 0 Then
          ' These next lines assign our LCFIELD variables to
          ' their counterparts in the LCFIELDLIST array
          Set EmpNoKeyField = fldlst.EMPNO
          Set FirstNmeFld = fldlst.FIRSTNME
          Set MidInitFld = fldlst.MIDINIT
          Set LastNameFld = fldlst.LASTNAME
          Set SexFld = fldlst.SEX
```

```
            Set BirthdateFld = fldlst.BIRTHDATE
            Set Bonusfld = fldlst.BONUS
            Set CommFld = fldlst.COMM
            Set SalaryFld = fldlst.SALARY
            Set HireDateFld = fldlst.HIREDATE
            Set PhoneNoFld = fldlst.PHONENO
            Set JobFld = fldlst.JOB
            Set WorkDeptFld = fldlst.WORKDEPT
            Set EdLevelFld = fldlst.EDLEVEL

            ' Copy the values in the Notes document to the
            ' LCFIELD variables
            FirstNmeFld.text = uidoc.FieldGetText("FirstNme")
            MidInitFld.text = uidoc.FieldGetText("MidInit")
            LastNameFld.text = uidoc.FieldGetText("LastName")
            SexFld.text = uidoc.FieldGetText("Sex")
            BirthDateFld.text = uidoc.FieldGetText("BirthDate")
            Bonusfld.text = uidoc.FieldGetText("Bonus")
            CommFld.text = uidoc.FieldGetText("Comm")
            SalaryFld.text = uidoc.FieldGetText("Salary")
            HireDateFld.text = uidoc.FieldGetText("HireDate")
            PhoneNoFld.text = uidoc.FieldGetText("PhoneNo")
            JobFld.text = uidoc.FieldGetText("Job")
            WorkDeptFld.text = uidoc.FieldGetText("WorkDept")
            EdLevelFld.text = uidoc.FieldGetText("EdLevel")

            ' This next line is the important one! It tells the
            ' update which field to use as the key in the data
            ' source, in our case this is the employee number.
            ' If you don't include a key on an update with a
            ' table that does not contain a unique primary key
            ' you will overwrite all records in the database
            ' with this new row.
            EmpNoKeyField.Flags = EmpNoKeyField.Flags Or _
                                  LCFIELDF_KEY
            ' Issue the update command
            count = SrcCon.Update (fldLst, 1, 1)
            ' Count tells us how may records were updated,
            ' hopefully just the one.
            Print count & " record(s) updated"
        End If
        ' Disconnect from the data source.
        SrcCon.Disconnect
End Sub
```

3. Below is a figure of how the screen looks with the new buttons.



This script is similar to the one we used to get data from the external data source. We set up a connection to the data source in the same way and executed an SQL query against the source. The major lines of code to look at are at the bottom:

This first line sets a flag, LCFIELDF_KEY, on the EmpNoKeyField variable to tell the connection to use this as a key field:

```
EmpNoKeyField.Flags = EmpNoKeyField.Flags Or LCFIELDF_KEY
```

The next line issues the update command on the database and returns the number of rows in the external data source that were modified:

```
count = SrcCon.Update (fldLst, 1, 1)
```

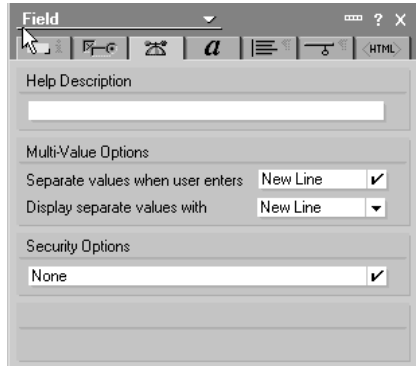
Multi-Value Data

Starting with Domino 4.65 and 5.0.1, the DECS has expanded capabilities around the usage of multi-value fields. Also known as the Collapse/Expand MetaConnector in LEI (the Lotus Enterprise Integrator), this functionality has been built directly into DECS activity forms. This new functionality allows activities to group multiple records with a common key value into a single Notes document. By using multi-value fields, these records can appear as table data as displayed in the following figure. All records with the same grouping key field value(s) are combined (collapsed) until a different key value is encountered.

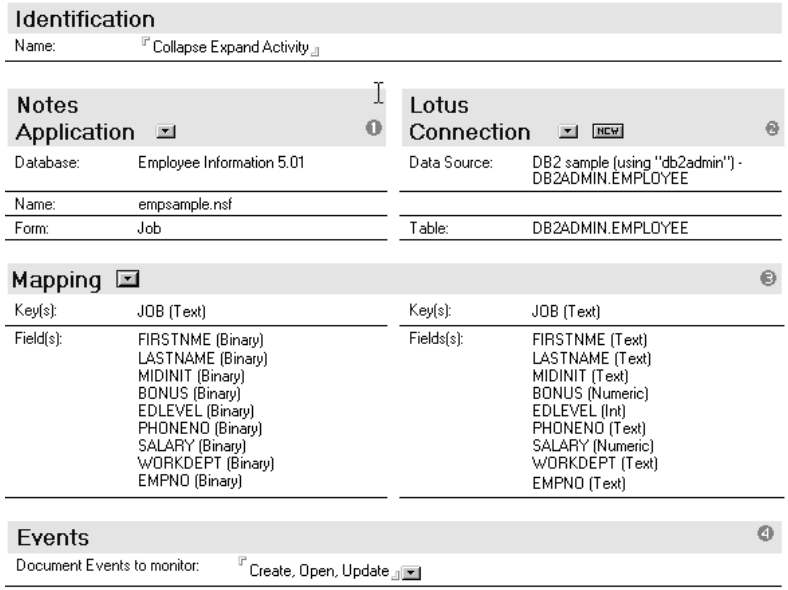
JOB: CLERK

LASTNAME:	FIRSTNAME	EmpNo	Dept	Phone #	Salary	BONUS:
JEFFERSON	JAMES	000230	D21	2094	$22,180	$400
JOHNSON	SYBIL	000260	D21	8953	$17,250	$300
MARINO	SALVATORE	000240	D21	3780	$28,760	$600
O'CONNELL	SEAN	000120	A00	2167	$29,250	$600
PEREZ	MARIA	000270	D21	9001	$27,380	$500
SMITH	DANIEL	000250	D21	0961	$19,180	$400

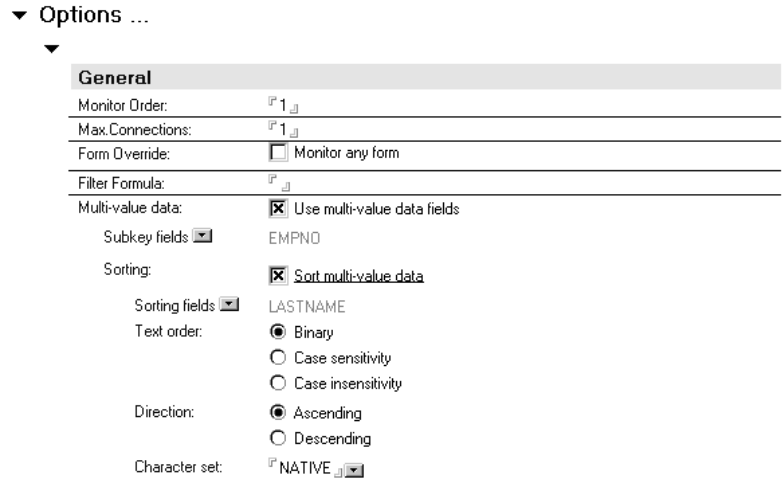
This table was formed by creating a two-row table on a Notes form and adding fields enabled for multi-values, with separator options set to "new line."



In all DECS activities, a key must be defined as one or more fields. In the preceding table we used the field “JOB” as the key field. Our Collapse/Expand activity looks like any other DECS activity, with the exception that our “key” value field is not a unique key.



To enable a DECS activity to work with multi-value data, check the "Multi-value data" check box under the Options section of the DECS activity. This will expand the section displaying additional selections. An important option for changes to be written back to a datasource is the “Subkey fields” option. This option allows the selection of a unique key to identify this individual record in our back-end data source.



For example, if six employees are collapsed into one document with the job title as the key, and data regarding one employee is changed, then the employee number is required to ensure that the correct record is updated. In this example, the employee number is the sub key and provides a unique identification across the individual external records for this employee when it is expanded back to the external data source.

Also note that we have numerous sorting options as well. These options allow us sort our data contained in our "collapsed records." This functionality is also performed in LEI by use of a separate Meta-Connector.

One last note, after your activity is created in the DECS administrator, we still have one more step. Select the new activity, then choose "Actions" from our top menu bar, and select the "Inialize Key" option. This action will create stub documents based off our key field in the activity. These documents will serve as containers for our data after the activity is enabled.

Summary

In this chapter we discussed how to create Connection and Activity documents in the DECS navigator to enable real-time transfer of data from an external data source into Notes. We also discussed how to use the Lotus Connectors LSX to access an external data source via LotusScript.

We also introduced additional functionality that allows multiple back-end records to be "collapsed" into individual Domino documents.

Chapter 14
Using Other Database Connectivity Tools

This chapter covers various methods available to connect Domino databases to and from external applications. The tools covered are NotesSQL, an ODBC driver that allows you to query Domino databases via SQL, the Domino JDBC driver, a Java SQL driver that allows you to query Domino databases through the Java language, and the LS:DO, an ODBC-compliant LSX for Domino applications.

NotesSQL

NotesSQL is the Lotus Notes ODBC driver for Windows, which enables ODBC DBMSs and data query tools to access, query, and report on Domino-based information. ODBC is a de facto standard interface for accessing information normally stored within an RDBMS. NotesSQL is a SQL API to Domino, with full level I ODBC 2.0 compliance and many level II extensions.

The current release of NotesSQL is 2.05; it provides the following new features over previous releases:

- On Windows 95 and Windows NT (Alpha or Intel), new thread-handling design allows interaction with thread-sharing Internet applications and reporting tools. It also fixes known problems related to using NotesSQL with Microsoft Internet Information Server (IIS), Cognos Impromptu, and Intersolv SequeLink products.
- Additional testing on 32-bit Windows platforms to ensure interoperability with leading Internet application software. Specifically tested were: Lotus BeanMachine Release 1.1, IBM Visual Age for Java (Enterprise) Version 1.0, Microsoft Active Server Pages and Internet Database Connector (IDC) on IIS 4.0, Microsoft Visual InterDev 1.0, and NetObjects Fusion Version 2.02.
- The restriction that forced an outer join clause to be the first item in the FROM clause of a SELECT statement has now been relaxed. The outer join clause can now be anywhere in the FROM clause.
- NotesSQL now properly evaluates a CR/LF in a text field in a WHERE clause of a SELECT statement.
- File DSN support.

Note NotesSQL 2.05 should be used with Domino R5.0.

What is ODBC?

ODBC is a call-level interface to databases which has been defined by Microsoft. In contrast to embedded SQL and PL/SQL, programs using ODBC are meant to be database-independent. This is achieved by a driver manager which passes the calls from the application program to the database. The ODBC library contains function calls that allow an application to connect to a DBMS, execute SQL statements, and retrieve results. Furthermore, it defines standard error codes and standard data types.

While embedded SQL is efficient and portable across different hardware and operating systems, the source code must be recompiled for each new environment. Additionally the program is tightly coupled with the database it was compiled for. There is no need for implementation-specific transformations on the source code of ODBC programs (such as pre-processing).

ODBC is more flexible than embedded SQL in the sense that the same programs (object code) can be used to access different DBMSs and can ignore underlying data communications protocols between it and a DBMS.

ODBC Conformance Level of NotesSQL

There are three conformance levels specified in the ODBC API:

- Core Level
- Extension Level 1
- Extension Level 2.

These are just general guidelines, and not all the drivers available today support all three levels. Moreover, even if an ODBC driver supports a conformance level, many ODBC drivers conform to both the Core Level or Extension Level 1 which is a superset of the Core Level.

NotesSQL supports all three levels of conformance except for some APIs of Extension Level 2. This is described in the Limitations section later in this chapter.

Core Level

Core Level is a minimum function set of the ODBC specifications. It mainly supports:

- Allocation and deallocation of the environment
- Connection and disconnection of a database
- SQL preparation and execution
- Fetch data
- Transaction control

  Note This is not supported in NotesSQL. Domino does not have a transaction mechanism.

Extension Level 1
Extension Level 1 extends the Core Level function set. It mainly supports:

- Retrieving the table schema
- Connecting to a database interactively
- Getting and putting data of a result set

Extension Level 2
Extension Level 2 contains more sophisticated functions. It mainly supports:

- Primary key and foreign key

  Note This is not supported in Notes, and thus, not in NotesSQL.

The following three features are not supported in NotesSQL:

- Table and column privilege control
- Stored procedure
- Cursor control

SQL Grammar Conformance Level of NotesSQL

The following three levels are available:

Minimum SQL

Minimum SQL only supports the character data type and simple operations.

DDL	*DCL*	*DML*	*Expressions*	*Data Types*
CREATE TABLE DROP TABLE		SELECT INSERT UPDATE Searched DELETE Searched	Numeric Operations (+, -, *, /, <, >, <=, >=, =, <>)	CHAR VARCHAR LONGVARCHAR

Core SQL

Core SQL supports DCL, and many operations and data types.

DDL	*DCL*	*DML*	*Expressions*	*Data Types*
ALTER TABLE CREATE INDEX CREATE VIEW DROP INDEX DROP VIEW	GRANT REVOKE	SELECT full syntax	Subselect Aggregation (SUM, MIN, MAX, AVG, COUNT)	DECIMAL NUMERIC SMALLINT INTEGER REAL FLOAT DOUBLE

Extended SQL

Extended SQL supports advanced operations, such as cursor related operations and outer join.

DDL	*DCL*	*DML*	*Expressions*	*Data Types*
		UPDATE Positioned DELETE Positioned Outer Join Cursor Control SELECT FOR UPDATE	Scalar functions	TINYINT, BIGINT, BINARY, VARBINARY, LONG, BIT, DATE, TIME, TIMESTAMP

Note NotesSQL supports Minimum SQL and some of the other SQL grammar.

Checking the Conformance Level

If you want to check the conformance levels of your ODBC driver, such as NotesSQL, the SQLGetInfo ODBC function can help you do this. The SQLGetInfo function is included in Extension Level 1 as an ODBC API conformance level.

NotesSQL supports the SQLGetInfo function.

Constant and Function Declarations in Visual Basic

In most programming environments, such as Visual Basic, you need to describe function declare statements in the ODBC DLL to use the NotesSQL functions.

In ODBC programs, SQLAllocEnv, SQLAllocConnect, SQLConnect, and SQLAllocStmt are essential. If required, you can also use SQLDriverConnect and SQLError.

```
'** Constant Declarations for SQLGetInfo
'** to check ODBC API and ODBC SQL Conformance Level
Public Const SQL_ODBC_API_CONFORMANCE As Long = 9
Public Const SQL_ODBC_SQL_CONFORMANCE As Long = 15
'** For ODBC API Conformance Level
Public Const SQL_OAC_NONE As Long = 0
Public Const SQL_OAC_LEVEL1 As Long = 1
Public Const SQL_OAC_LEVEL2 As Long = 2
'** For ODBC SQL Conformance Level
Public Const SQL_OSC_MINIMUM As Long = 0
Public Const SQL_OSC_CORE As Long = 1
Public Const SQL_OSC_EXTENDED As Long = 2
'** Options for SQLDriverConnect
Public Const SQL_DRIVER_NOPROMPT As Long = 0
```

```
Public Const SQL_DRIVER_COMPLETE As Long = 1
Public Const SQL_DRIVER_PROMPT As Long = 2
Public Const SQL_DRIVER_COMPLETE_REQUIRED As Long = 3
'** Return Code
Public Const SQL_ERROR As Long = -1
Public Const SQL_INVALID_HANDLE As Long = -2
Public Const SQL_NO_DATA_FOUND As Long = 100
Public Const SQL_SUCCESS As Long = 0
Public Const SQL_SUCCESS_WITH_INFO As Long = 1
'** ODBC Functions to issue SQLGetInfo
'** To get an Environment Handle
Declare Function SQLAllocEnv Lib "odbc32.dll" (phenv&) As
Integer
'** To get a Connection Handle
Declare Function SQLAllocConnect Lib "odbc32.dll" (ByVal henv&,
phdbc&) As Integer
'** To establish a connection
Declare Function SQLConnect Lib "odbc32.dll" (ByVal hdbc&,
ByVal szDSN$, ByVal cbDSN%, ByVal szUID$, ByVal cbUID%, ByVal
szAuthStr$, ByVal cbAuthStr%) As Integer
'** To establish a connection with a dialog box
Declare Function SQLDriverConnect Lib "odbc32.dll" (ByVal
hdbc&, ByVal hWnd As Long, ByVal szCSIn$, ByVal cbCSIn%, ByVal
szCSOut$, ByVal cbCSMax%, cbCSOut%, ByVal fDrvrComp%) As
Integer
'** To get a Statement Handle
Declare Function SQLAllocStmt Lib "odbc32.dll" (ByVal hdbc&,
phstmt&) As Integer
'** To get information on an ODBC driver
Declare Function SQLGetInfo Lib "odbc32.dll" (ByVal hdbc&,
ByVal fInfoType%, ByRef rgbInfoValue As Any, ByVal cbInfoMax%,
cbInfoOut%) As Integer
'** To get error information
Declare Function SQLError Lib "odbc32.dll" (ByVal henv&, ByVal
hdbc&, ByVal hstmt&, ByVal szSqlState$, pfNativeError&, ByVal
szErrorMsg$, ByVal cbErrorMsgMax%, pcbErrorMsg%) As Integer
```

How to Issue SQLGetInfo in Visual Basic

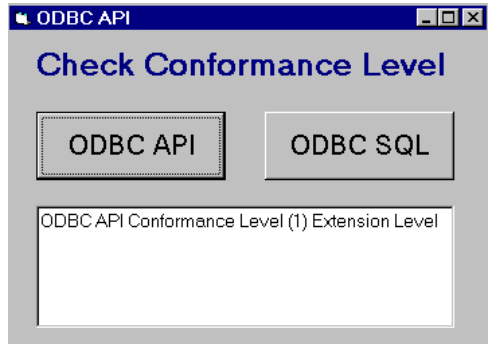
The following program illustrates getting information on the ODBC API Conformance level. If you don't need a dialog box to specify a data source name, you can replace SQLDriverConnect with SQLConnect.

```
Private Sub Command1_Click()
Dim ret As Integer
Dim msg As String
Dim rInfo As Long
Dim rSize As Integer
Dim connect As String * 255
Dim connectLen As Integer
```

```
Dim henv As Long, hdbc As Long, hstmt As Long
'** To get an Environment Handle
ret = SQLAllocEnv(henv)
If ret = SQL_ERROR Then
  Call ErrorMSG(henv, hdbc, hstmt)
  Exit Sub
End If
'** To get a Connection Handle
ret = SQLAllocConnect(henv, hdbc)
If ret = SQL_ERROR Then
  Call ErrorMSG(henv, hdbc, hstmt)
  Exit Sub
End If
'** To establish a Connection
ret = SQLDriverConnect(hdbc, Me.hWnd, "", 0, connect,
Len(connect), connectLen, SQL_DRIVER_PROMPT)
If ret = SQL_ERROR Then
  Call ErrorMSG(henv, hdbc, hstmt)
  Exit Sub
End If
'** To get a Statement Handle
ret = SQLAllocStmt(hdbc, hstmt)
If ret = SQL_ERROR Then
  Call ErrorMSG(henv, hdbc, hstmt)
  Exit Sub
End If
'** To get ODBC API Conformance Level
ret = SQLGetInfo(hdbc, SQL_ODBC_API_CONFORMANCE, rInfo, 300,
rSize)
If ret = SQL_ERROR Then
  Call ErrorMSG(henv, hdbc, hstmt)
  Exit Sub
End If
Select Case rInfo
Case SQL_OAC_NONE:   msg = "Core Level"
Case SQL_OAC_LEVEL1: msg = "Extension Level 1"
Case SQL_OAC_LEVEL2: msg = "Extension Level 2"
End Select
Text1.Text = "ODBC API Conformance Level (" & rInfo & ") " &
msg
End Sub
```

How It Works

The following figure shows how the above Visual Basic program works:



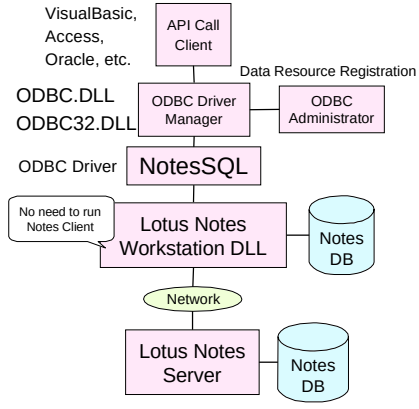
When you click the ODBC API button, the ODBC API Conformance Level is displayed in a text box. The SQL Data Sources dialog box is displayed for you to specify a data source name, since SQLDriversConnect is issued without any data source names to establish a connection in the above program.

Technical Advantages

NotesSQL allows the developer to issue SQL statements against Domino databases. This is a significant advantage to developers who wish to use Domino data in their applications. In essence, NotesSQL is really an SQL API to Domino, with full level I ODBC 2.0 compliance and level II extensions.

Structure

The way a target database is accessed from an ODBC client, such as a Domino database from Visual Basic, is the same as for any other ODBC driver. The following figure shows how to reach a Domino database from an application which uses NotesSQL:



When to Use NotesSQL

If you want to access a Domino database from an RDBMS or from another application development tool, such as Oracle or Visual Basic, use NotesSQL. NotesSQL is designed for query and reporting tools and other ODBC-compliant DBMSs and tools to access Domino data. For example, users often need reports that incorporate data from both Domino and a DBMS. A sales force automation application can use Domino to capture information from field sales, such as customer feedback, contact management and sales forecasts, while customer orders are often managed by DBMSs.

NotesSQL allows an external DBMS or query tool to perform table joins or combine the data from both sources in the same report. The date of last contact from the Domino sales force application could be combined with the latest customer order date via NotesSQL to produce a report on the length of the sales cycle at a customer site or across a customer set.

Query tools leveraging NotesSQL provide structured data analysis of sales forecast information stored in Domino. Similarly, that same information collected in Domino can be pulled into DB2, for central storage and distribution.

If you already have an ODBC-compliant program to access an RDBMS, you may also be able to access a Domino database without any modification of your program. In this case, the only thing you have to worry about is the difference between an RDBMS and a Domino database, such as data types, the conformance level of SQL, and so on, as a Domino database is not a relational database.

Hardware and Software Requirements

To use Domino data through ODBC, you must have:

- NotesSQL, the Domino ODBC driver.
- Lotus Notes Designer for Domino Release 4.6 or Lotus Notes Workstation release 4.x.
- An ODBC Driver Manager version 2.0 or later.
- One of the following:
  - Microsoft Windows 3.1 or later, or
  - Microsoft Windows 95, or
  - Microsoft Windows NT 3.5 or later, or
  - IBM OS/2 Warp or later

Lotus Notes Designer for Domino or Lotus Notes Workstation must be installed. A full user license is required; desktop and mail IDs are not supported. Domino database files can reside on a server. You do not need to have local copies of these files, but you must have at least reader access to them through Notes.

Installing NotesSQL

All 32-bit Windows platforms (Windows 95 or NT for Intel or Alpha)

1. Run SETUP.EXE and follow the prompts that appear.

   Note During installation, a dialog box asks you if you want to install ODBC 3.0 components or cancel the installation. Click OK to continue.

2. At the end of the installation process, the ODBC Data Source Administrator dialog box appears so you can add Lotus Notes Data Sources. If you are upgrading from a previous 2.x version of NotesSQL, note that you do not have to make any changes to your Data Sources. All your previously added Data Sources will work with this release.
3. For ease of access, move this documentation database to your Notes data directory.

16-bit Windows

1. Run SETUP.EXE, which loads Microsoft ODBC Setup, and follow the prompts.
2. When the Install Drivers dialog box appears, select the NotesSQL driver and click OK. (You do not need to modify any information in the Advanced dialog box.)
3. At the end of the installation process, the ODBC Data Source Administrator dialog box appears so you can add Lotus Notes Data Sources. If you are upgrading from a previous 2.x version of NotesSQL, note that you do not have to make any changes to your Data Sources. All your previously added Data Sources will work with this release.
4. For ease of access, move this documentation database to your Notes data directory.

OS/2

1. At the command prompt, set the ODBC_PATH environment variable to the directory where you want to install the driver. For example:

   ```
   set ODBC_PATH=C:\ODBC
   ```

   (You do not need to do this if you have already installed the Visigenic ODBC Driver Manager.)

2. At the command prompt, run SETUP.EXE.

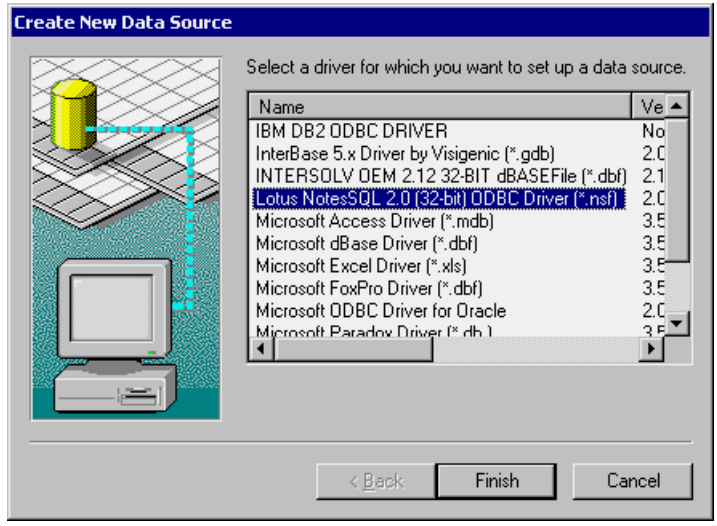
3. When prompted for the installation directory, use the default directory (the directory specified by the ODBC_PATH environment variable).
4. When the Install Drivers dialog box appears, select the NotesSQL driver and click OK. You do not need to modify any information in the Advanced dialog box.
5. At the end of the installation process, the ODBC Data Source Administrator dialog box appears so you can add Lotus Notes Data Sources. If you are upgrading from a previous 2.x version of NotesSQL, note that you do not have to make any changes to your Data Sources. All your previously added Data Sources will work with this release.
6. For ease of access, move this documentation database to your Notes data directory.
7. Reboot the system.

Adding a Lotus Notes Data Source

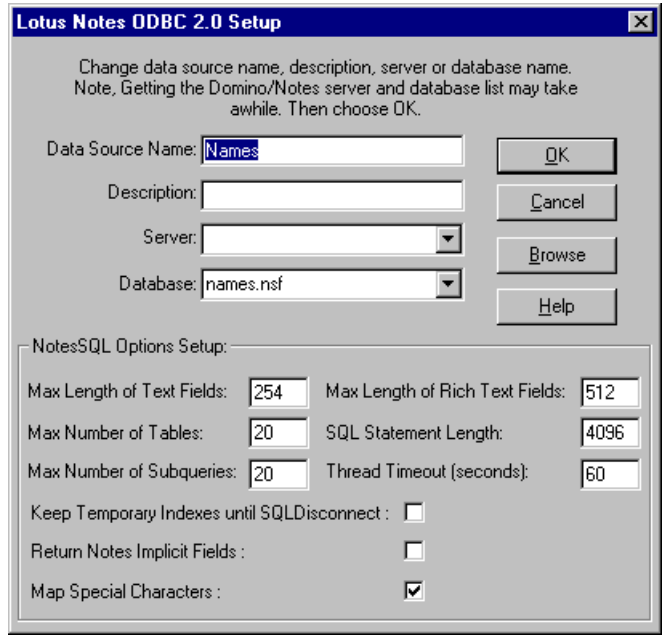
Before you can connect to a Domino database with NotesSQL, you must add a data source for it. You can change or delete a data source at any time. You can use as many data sources as you like with a particular driver, provided no two data sources have the same name.

To add a Lotus Notes data source:

1. Start the ODBC Administrator.
2. Choose Add.
3. In the Add Data Source dialog box, select Lotus Notes from the Installed ODBC Drivers list box.



4. Choose OK. This brings up the NotesSQL Configuration dialog box.



Configuring a Lotus Notes Data Source

This section lists the parameters of a Lotus Notes Data Source with a description of each. There are many parameters, but most have reasonable defaults. We suggest you try using the defaults first. The three dialog box fields that must be filled in are marked with an asterisk.

Fields that ask for numeric values check your input. If you specify a value above the allowed maximum, the maximum value is substituted. If you specify a value below the allowed minimum, the minimum value is substituted. If you specify an invalid value, such as a letter, the default value is substituted.

Data Source Name

Enter a name that identifies the data source in the Data Source Name text box. For example, add the name “Employee” to identify the ODBC connection to an employee database.

Description

Enter a description of the data source in the Description text box. For example, add the description “Hire date, salary history, and current review of all employees” to describe the contents of the employee database.

Server
Enter the name of the Notes server that contains the Domino database you want to open in the Server text box. Leave the text box blank if the Domino database is on a local disk.

Database
Enter the path and name of the .NSF file you want to open in the Database text box. For example:

- C:\PERSONNEL\EMPLOYEE.NSF for a file on a local disk, or
- PERSONNEL\EMPLOYEE.NSF for a database on a Lotus Notes server.

Whether the data is local or on a server, the path is relative to the Notes data directory on that machine. If the Notes data directory is D:\NOTES\DATA\, PERSONNEL\EMPLOYEE.NSF points to D:\NOTES\DATA\PERSONNEL\EMPLOYEE.NSF.

Max Length of Text Fields
This parameter specifies the maximum number of bytes NotesSQL allows in a string. This limits the number of characters returned from a Notes text field and the length of a string to be inserted into a Notes text field.

- The maximum value allowed is 15,360. The minimum value allowed is 2.
- The default value is 254. If the database will be used exclusively or primarily with Lotus 1-2-3, we recommend a value of 511. This is the maximum length of a string in a cell in Lotus 1-2-3.

Max Number of Tables
This parameter specifies the maximum number of tables in a single query.

- The maximum value allowed is 100. The minimum value allowed is 1.
- The default value is 20.

Max Number of Subqueries
This parameter specifies the maximum number of nested subqueries in a SQL statement.

- The maximum value allowed is 100. The minimum value allowed is 1.
- The default value is 20.

Keep Temporary Indexes until SQLDisconnect
This option controls the saving of temporary indexes. Select the option to save temporary indexes until SQLDisconnect. Leave the option unselected to delete indexes at the end of each SELECT result.

- The option is unselected by default.

Return Notes Implicit Fields
This option controls whether certain Notes fields are provided through SQLStatistics and SQLSpecialColumns. Turn it off for maximum interoperability. Turn it on for backward compatibility with NotesSQL 2.0 or earlier.

- The option is unselected by default.

Map Special Characters
This option controls whether certain characters are mapped to the underscore character (_). For details, see the section "Mapping Notes Names to SQL Names" later in this chapter.

- The option is selected by default.

Max Length of Rich Text Fields
This parameter is the same as Max Length of Text Fields, but for rich text fields.

- The maximum value allowed is 15,360. The minimum value allowed is 2.
- The default value is 512. If the database will be used exclusively or primarily with Lotus 1-2-3, we recommend a value of 511. This is the maximum length of a string in a cell in Lotus 1-2-3.

Note Notes databases have long text fields called rich text fields. The Body field of a Notes document is often a rich text field. If you want to retrieve all the data in such a field, be sure that the maximum string length you specify is large enough.

SQL Statement Length
This parameter specifies the maximum length of an SQL statement passed to SQLPrepare or SQLExecDirect.

- The maximum value allowed is 32,767 (32K). The minimum value allowed is 2.
- The default value is 4,096 (4K).

Thread Timeout (available on 32-bit Windows platforms only).
- All calls are serialized so that they proceed one after the other. This parameter specifies the number of seconds one thread should wait for another thread to finish.
- The maximum value allowed is 99,999. The minimum value allowed is 0. A value of 0 means wait forever.
- The default value is 60.

Connecting to a Data Source

You can use NotesSQL with any application that supports ODBC drivers for external data access. To connect to the Domino database, you select the data source you defined using the Setup dialog box.

If your Notes user ID is password protected, Notes prompts for a password when you first connect to a remote database using NotesSQL. This information is not database specific and is retained for the length of the session, so you will not need to enter it more than once if you connect to more than one database.

Notes never prompts for a user ID. Your user ID is part of your workstation installation.

Connection Strings

The following keywords are supported for the SQLDriverConnect call:

Keyword	*Description*
DSN	The name of the data source.
Database	The name of the Domino database, with a path if necessary.
Server	The name of the Notes server where the database is located. If the database is on the local workstation, leave the field blank.

For example, to connect to the Personnel data source in the directory PERSONNEL on server HR_1, use the following connection string:

```
DSN=Personnel; Database=Personnel\employee.nsf; Server=HR_1
```

The following table describes how SQL components map to Notes components:

SQL Components	*Notes Components*	*Comments*
Table	Form or View or Universal Relation	SQL Tables map to either Notes forms or Notes views. However, a Domino database contains only one real table, referred to as the Universal Relation. This table has the same name as the database.
Column	Form Field or View Column	When creating a NotesSQL Table or View, it is best to avoid the use of column names that are ODBC or SQL reserved words or that contain characters other than letters, numbers, or underscores.
Index	View	A SQL Index maps to a Notes view in which all sorted columns refer directly to fields in a single form, and which selects documents from only that form.
View	View	A SQL View maps to a Notes view that selects documents from one form, in which all columns are calculable from the form.

Note When a Domino database has both a form and a view with the same name, NotesSQL cannot distinguish between them in SQL statements and the view cannot be accessed.

Mapping Notes Names to SQL Names

Notes is more flexible about names than SQL. When naming a form or view, Notes allows many special characters and sequences of characters that are not part of the standard SQL syntax.

It is not uncommon to number views so that they will appear in a certain order in the Create menu, for example:

1. Products
2. Suppliers

If developers want hierarchical view names, they use the backslash:

Products\By Name

Products\By Age

These common Notes naming conventions are not allowed by standard SQL syntax, which does not allow the use of periods, spaces and forward slashes in unquoted identifier names.

Mapping

Because of this difference in syntax support, NotesSQL optionally maps certain characters to the underscore character (_). This mapping is configurable for each data source. The default is to map the names. We recommend using the default unless it causes a form or view to be inaccessible. This can happen when two similar names (for example, “Employees$” and “Employees_”) are mapped to the same name.

If you choose to map names, NotesSQL maps forms, views and their aliases, form fields, and view columns. The characters mapped are:

~ ` ! @ # $ % ^ & * ( ) - + = { } [ ] \ : ; " ' < > , . / ? and the space character.

The Universal Relation

NotesSQL recognizes Notes forms and views as tables. In addition to forms and views, every Domino database contains a table that has the same name as the database. This table is called the Universal Relation.

The Universal Relation contains all fields defined in all forms in the Domino database. The Universal Relation is the only true table in a Domino database. As a result, SQL tables created by NotesSQL behave more like SQL views than traditional relational database tables.

For example, with NotesSQL, you can create a Notes form with the CREATE TABLE statement. However, the DROP TABLE statement deletes the Notes form but does not delete any data from the database. Using DROP TABLE with NotesSQL is like deleting a SQL view, in that the data remains in the database. You can view the data through other views that use the same field names, or by referencing the Universal Relation table.

Continuing this example, if you create a new table with the same name as a previously deleted table, and use some of the field names from the deleted table, you could find data in the table before you insert any data. This is because the table is actually a view of existing data in the Universal Relation table. Documents (records) in the Domino database contain the name of the form used to create them. NotesSQL uses the form name stored in the document to identify the document when selecting from a form.

Using the Universal Relation

- You must have read/write access to a database to use its Universal Relation. Without read/write access, you cannot perform any operation (including SELECTs) on the Universal Relation.
- The same field name can be used in more than one form with different data types in a Domino database. Therefore, you must use explicit field names in a SELECT clause that references the Universal Relation. In particular, you cannot use SELECT *. You can only perform text operations on fields in the Universal Relation because the data type for all fields is character.
- The Universal Relation can participate in a self-join but cannot participate in any other kind of join.
- NotesSQL supports the SELECT statement on the Universal Relation. NotesSQL does not support this on a read-only .NSF file.
- NotesSQL supports the CREATE VIEW statement on the Universal Relation.
- NotesSQL does not support INSERT, DELETE, UPDATE, DROP TABLE, or DROP VIEW statements on the Universal Relation.
- NotesSQL uses the type "UNIVERSAL" to differentiate the Universal Relation in the result set of SQLTables.

Using SQL Tables from Derived Forms and Views

NotesSQL recognizes Notes forms and views as tables. In addition, the driver recognizes the Universal Relation as a table. However, Notes forms and views have very different properties that affect the performance of data access and display with NotesSQL.

Views in Notes databases list documents in a specific order. Avoid selecting from a table based on a Notes view and then specifying a different sort order. When you specify a different sort order on an existing view, Notes creates a temporary table on your workstation and re-sorts the documents. Creating a large temporary table and sorting the documents in that table will take a long time.

Note Temporary table creation requires you to have read/write access to the database. If you have read-only access, you cannot perform an operation such as SELECT with ORDER BY or GROUP BY unless there is an existing view you can use to support the operation.

To display information from a view in a different sort order, use a table based on a Notes form and create an index on the form using the order you want, or create a new view either in Notes or through ODBC.

Tables derived from Notes forms are not necessarily indexed for fast access. When selecting data from a table based on a Notes form, NotesSQL looks for a view that acts as an index on that table. If such a view is present, access to the table will be fast. If no such view is present, access to the table may be very slow. In particular, if the table is small in relation to the database, it will take a long time for the driver to locate all the records in the table. This is because Notes must search the entire database and check every record to see if it belongs to the table.

Example: Using SQL Tables Derived From Notes Forms and Views

The Notes Personal Address Book (NAMES.NSF) is a good database to use as an example to compare the use of forms or views in a database. The Personal Address Book database includes:

- A form called Person
- A view called People with a sort key on LastName

The following statement is the most efficient way to find people in the Personal Address Book sorted by LastName:

```
SELECT LastName
FROM People
ORDER BY LastName
```

People is a Notes view. This query is efficient because NotesSQL can use the index already associated with the People view that lists LastName in the right order. Now assume you want to list people sorted by their mailing addresses. You could use the following statement:

```
SELECT LastName, Mail_Address
FROM People
ORDER BY Mail_Address
```

Since the People view is not sorted on Mail_Address, NotesSQL uses the People index, generates a temporary table, and creates a temporary index on Mail_Address. This results in slower performance.

A more efficient way to achieve the same result is to issue the following statement:

```
SELECT LastName, Mail_Address
FROM Person
ORDER BY Mail_Address
```

Person is a Notes form. If there is no index on Mail_Address, NotesSQL generates a temporary index on Mail_Address but does not need to generate a temporary table. This statement is faster than the previous statement, which used ORDER BY on a view-based table. This statement can be executed even faster if the user creates an index in Notes or by using the CREATE INDEX statement in NotesSQL.

Column, Index, Table, and View Names

Naming Rules

A column, index, or view name can be up to 32 characters long. A table name can be up to 64 characters long.

Index, view, and table names can consist of letters, digits, underscores (_), dollar signs ($), and spaces. If an index, view, or table name contains a space or is the same as a SQL reserved word, it must be enclosed in double quotes.

Column names can consist of letters, digits, underscores (_), and dollar signs ($). Column names cannot contain spaces or conflict with a SQL reserved word.

Creating Columns, Indexes, Tables, and Views

When creating an index or view, remember that index names and view names must be unique within a database. Don't use the name of an existing index or view.

When creating a table, remember that table names must be unique within a database. Don't use the name of an existing table or view.

When creating or altering a table, remember that column names must be unique within a table. Don't use the name of another column in the table.

Name Visibility

If a Domino database contains both a view and a form with the same name, NotesSQL will see the form but not the view.

Using Notes Views as Indexes

NotesSQL will use an existing Notes view as an index if it meets the following criteria:

- The view selection formula in the Notes view design is either SELECT Form = "name" or SELECT @All.
- SELECT Form = "name" is faster than SELECT @All, unless all documents in the database were created using the same form. Note that views created with SELECT @All are not described as indexes through SQLStatistics. They are used by NotesSQL to improve performance.
- Each sorted column in the view is defined as a simple reference to a field in that form (no formulas).
- At least one column in the view is sorted.

An easy way to see this information at a glance in Notes is to select File - Database - Design Synopsis. In the dialog box, choose Views, then click OK. Notes generates a detailed synopsis document that includes all the above information and more.

Unique Indexes

In Notes there is a view property labeled "Unique keys in index (for ODBC access)." When this setting is checked, NotesSQL describes the sorted columns of the view as the keys of a unique index. This happens through the ODBC call SQLStatistics. Having these indexes allows several third-party applications to issue updates against NotesSQL.

Use some care with this feature. Checking the box doesn't make the view a unique index in the relational database sense. Specifically, it doesn't prevent duplicate records. It only puts the first document with a particular sorted value into the view. Therefore, if you are using this feature, updates may affect more than one record and ORDER BYs may generate small result sets.

Here are some guidelines to follow to avoid these problems:

- Only use the check box for one view that selects from each form that will be updated.
- Choose a view with sorted columns that are each one field (not a formula or expression).
- Choose a view with sorted columns that together uniquely identify a record.

View Column Definitions

If a column in a view does not refer directly to a field, NotesSQL creates a name for that column in the result set. The name is the "$" character followed by a number (for example, "$2").

If a view column definition or a Computed for Display field in a form contains one of the @functions listed below, NotesSQL returns no results for that column. These columns are not reported by SQLColumns. These columns display data when viewed from Notes, but this data is not available through a SQL query.

- @All
- @DeleteDocument
- @DeleteFields
- @DocChildren
- @DocLevel
- @DocNumber
- @DocParentNumber
- @DocSiblings
- @Error
- @IsCategory
- @IsExpandable
- @Unavailable

Note Expressions passed in SQL statements must be either valid SQL expressions or valid Notes expressions. Otherwise, the results are unpredictable.

Data Types

ODBC SQL to Notes data type mapping:

ODBC SQL Data Type	*Lotus Notes Data Type*
SQL_CHAR	Text
SQL_VARCHAR	Text
SQL_LONGVARCHAR	Text
SQL_DECIMAL	Number, Fixed format
SQL_NUMERIC	Number, Fixed format

continued

ODBC SQL Data Type	*Lotus Notes Data Type*
SQL_SMALLINT	Number, Fixed format
SQL_INTEGER	Number, General format
SQL_REAL	Number, General format
SQL_FLOAT	Number, General format
SQL_DOUBLE	Number, General format
SQL_DATE	Time
SQL_TIME	Time
SQL_TIMESTAMP	Time

In addition to the SQL data types, Notes supports two additional data types:

1. Multi-Valued Fields (List Fields)

NotesSQL supports multiple values in fields. NotesSQL contains information about which fields can have multiple values and the underlying data type for these values. NotesSQL does not support multi-valued (list) fields in DDL statements (CREATE TABLE, ALTER TABLE, CREATE INDEX, CREATE VIEW).

Use the following notation in DML statements (SELECT, INSERT, UPDATE searched, UPDATE positioned, DELETE searched, DELETE positioned) to specify a list of values:

```
'string;string;string'
```

See below for more information about the correct separator character to use.

Text Fields

NotesSQL returns all the data in multi-valued fields composed of text as a single string. Data items are separated by the display separator character (the character defined in Notes as the one to use to separate multiple values when displaying them). For example:

```
'a;b;c'
```

Note An extra semicolon can appear in the result if the data was entered in Notes, any character other than a semicolon was used as the display separator character, and the list contains an item that includes a semicolon.

NotesSQL accepts a list of strings for insertion in a multi-valued field of type Text. You must separate the strings with the display separator character defined for that field. If a comma is the display separator character for a field, you must specify a string, for example:

```
'a,b,c'
```

This value creates a list in the Notes document if the field allows multiple values. If the field does not allow multiple values, the value appears in Notes as a single string. Data retrieved through NotesSQL produces the same result in either case.

Numeric and Date Fields
NotesSQL returns only the first value in the list when the multi-valued fields are numeric or date fields.

Currently, NotesSQL accepts only a single numeric or date value for insertion in a multi-valued field of type Numeric or Date.

2. Rich Text Fields

NotesSQL returns only the text part(s) of a Notes rich text field. NotesSQL cannot create a rich text field.

The driver supports only the LIKE operator in a WHERE clause for testing rich text fields. The driver does not support any other tests on rich text fields.

Additional restrictions on rich text fields:

- If a rich text field in Notes has embedded bitmaps or large attachments, text following the bitmap may not be returned by NotesSQL.
- If a rich text field contains only blank lines, NotesSQL will not return data from that field.
- String constants in a LIKE statement are restricted to a maximum of 254 characters, including any embedded carriage returns.
- You cannot insert a string longer than 15,360 bytes into a rich text field.

Note SQLGetTypeInfo returns ODBC SQL data types. All conversions in Appendix D of the Microsoft ODBC SDK Programmer's Reference are supported for the ODBC SQL data types listed above.

Note SQL_TIMESTAMP fields do not support fractions of a second.

Note SQL_VARCHAR or SQL_LONGVARCHAR data types cannot be used to create rich text fields. They create text fields.

Notes to SQL Data Type Mapping

Lotus Notes Data Type	*ODBC SQL Data Type*
Number, Fixed format	
Number, Percent format	SQL_DECIMAL
Number, General format	
Number, Scientific format	SQL_FLOAT

continued

Lotus Notes Data Type	*ODBC SQL Data Type*
Time	Depending on format, this can be SQL_TIME, SQL_DATE, or SQL_TIMESTAMP
Text	SQL_VARCHAR
Keyword	SQL_VARCHAR
Multi-value list	SQL_VARCHAR
Rich text field	Text portion only, as SQL_LONGVARCHAR
Section	Not supported

Note Numeric values greater than 10 to the 99th power are not supported. Some products that use ODBC drivers have limitations on the exponent and precision size of real numbers. Please check the limits of the product you are using for additional limit information.

Note NotesSQL does not check the keyword list for Keyword fields. It is possible to insert values not on the keyword list when the "allow values not in list" box is not checked.

Note NotesSQL does not support Notes Section security. Users of NotesSQL can read and modify fields anywhere on a form.

Note You cannot insert a string longer than 15,360 bytes into a rich text field.

Summary of Supported ODBC SQL Grammar

The following ODBC SQL grammar is supported. For full details on how to use each statement, refer to the NotesSQL Reference database.

Supported Statements

- ALTER TABLE
- CREATE INDEX
- CREATE TABLE
- CREATE VIEW
- DELETE Searched
- DELETE Positioned
- DROP INDEX
- DROP TABLE
- DROP VIEW
- INSERT

- SELECT
  - FOR UPDATE
  - FROM
  - GROUP BY
  - HAVING
  - ORDER BY
- UNION
- UPDATE Searched
- UPDATE Positioned
  - WHERE

Supported Expressions, Functions, and Operators

Numeric Operators

NotesSQL supports the following numeric operators in expressions:

Operator	*Meaning*
+	Addition
-	Subtraction
*	Multiplication
/	Division

Predicate Operators

Operator	*Meaning*
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal
=	Equal
<>	Not Equal
BETWEEN	Specifies a range of values between a lower and upper boundary.
IN	Specifies a member of a set of specified values or a member of a subquery.
LIKE	Use for matching a pattern. Wildcard characters in LIKE predicate: Use an underscore (_) to represent a single character. Use a percent symbol (%) to represent any number of characters. Use a backslash (\ ) as the escape character.

continued

Operator	*Meaning*
NOT	Use the NOT operator with another operator to specify a search condition that is false. For example: NOT IN, NOT LIKE, or NOT BETWEEN.
ANY	Use to compare a value to each value returned by a subquery. Can be prefaced by =, <>, >, >=, <, <=, or =. =ANY is equivalent to IN. <>ANY is equivalent to NOT IN.
SOME	SOME is an alternate keyword for ANY.
ALL	Use to compare a value to each value returned by a subquery. Can be prefaced by =, <>, >, >=, <, or <=.
EXISTS	"True" if a subquery returned at least one record.

Column Functions

Column functions can be part of a SELECT clause. A column function takes an entire column of data as its argument and produces a single data item that summarizes the column. For example, the AVG column function takes a column of data and computes its average.

The argument to a column function can be a field name or an expression. NotesSQL supports the following Column functions:

Function	*Description*
AVG	Returns the average of the values in a numeric field or expression. For example, AVG(SALES) returns the average of all values in the "SALES" column.
COUNT	Returns the number of values in any field expression.
MAX	Returns the highest value in any field expression. For example, MAX(SALES) returns the highest SALES field value.
MIN	Returns the lowest value in any field expression. For example, MIN(SALES) returns the lowest SALES field value.
SUM	Returns the total of all values in a numeric field expression. For example, SUM(SALES) returns the sum of all SALES field values.

Exceptions to ODBC SQL Grammar

NotesSQL supports most SQL statements and clauses in the ODBC Minimum and Core grammar. The following table describes exceptions:

Grammar	*Exception*
NULLs	There is no explicit NULL in Notes. NotesSQL uses NULLs to provide access to the Notes function @IsAvailable. See "NULLs" below for details.
ORDER BY clause	NotesSQL supports ordering by expressions that aren't in the project list. This is not standard SQL but many applications use it.
ALTER TABLE	The following keywords are not supported: NULL NOT NULL
CREATE INDEX	The UNIQUE keyword is not supported
CREATE TABLE	The following keywords are not supported: NULL NOT NULL UNIQUE PRIMARY KEY REFERENCES No table constraint definition
PARAMETER RESTRICTIONS	Parameters are supported in INSERT, DELETE, and SELECT SQL statements. They cannot be used with CREATE TABLE, CREATE VIEW, and other statements that manipulate the structure of tables and views (DDL). Arrays of parameters are not supported.
RESTRICT and CASCADE	Dependencies are only recognized if they were originally created using the driver. DROP TABLE CASCADE will only remove dependent views created using the driver. DROP TABLE RESTRICT will only prevent the removal of a table if a dependent view was created using the driver.
GRANT and REVOKE	Not supported. All access control is handled implicitly by Notes.

NULLS

NotesSQL uses NULLs to provide access to the Notes function @IsAvailable. For example, suppose you are comparing a field to NULL, as in:

```
SELECT * FROM Employee WHERE Name IS NULL
```

NotesSQL checks @IsAvailable("Name"). If @IsAvailable("Name") is TRUE, the WHERE clause is FALSE. Otherwise, the WHERE clause is TRUE.

Continuing the parallel, updating or inserting NULLs through NotesSQL removes fields from documents. So, for example, the following will create a document with no Name field:

```
INSERT INTO Employee (EmpNo, Name) VALUES (10, NULL)
```

Likewise, the following will remove the Name field from a document:

```
UPDATE Employee SET Name=NULL WHERE EmpNo=10
```

Known Limitations or Problems With NotesSQL

- A text field length of more than 254 is not recommended. It is available so that you can read long Notes text fields. Notes text fields are mapped to the ODBC data type VARCHAR. The ODBC specification for the maximum string length of a VARCHAR field is 254. Some applications won't work with longer fields. You may experience problems even if the data is shorter than 254 characters.
- Scalar functions are supported if there is underlying Notes support. Consequently, these scalar functions are not supported:
  - String: ASCII, INSERT, LOCATE
  - Numeric: TRUNCATE
- NotesSQL was developed to the ODBC 2.0 specification. ODBC 3.0 Driver Manager vendors have tried to be backward-compatible with ODBC 2.0 drivers, but several features are not available.
  - Several APIs were added in 3.0. None of these is supported.
- The results in "Driver Capabilities: SQLGetInfo Return Values" were generated with ODBC 2.0.
- There may be problems using categorized views. They don't map to the relational data model.
- You can't use a form or view with a double quote (") in the name.
- View selection formulas marked as "easy" (as opposed to "formula") in the radio button on the view design may be hiding some detail. Look in the design synopsis to see the full view selection formula.
- NotesSQL does not support multi-threading on OS/2, 16-bit Windows, or the Macintosh due to limitations in the underlying thread support.

Example: Accessing Notes From Visual Basic

Visual Basic has a number of ways of accessing Lotus Notes using the ODBC feature:

- **Remote Data Control** (RDC)

  This is a visual control to deal with remote data access. Basically, no programming is needed to access Notes. This feature can provide read and write access to a database, but it is often used only to retrieve database data.

- **Remote Data Object**

  Some methods and properties are available in the Remote Data Object to access an ODBC database.

- **ODBC API Call**

  Environment handles, Connection handles and statement handles are retrieved in both the Remote Data Control and the Remote Data Object. They can be combined with each other.

Program Structure

The following sections provide some detailed information on how our sample application accesses a Domino database from Visual Basic.

Creating a Data Source List

When the form module of our example is loaded, a data source list is created by the ListDataSources subroutine and listed in a listbox as follows:

```
'** Form_Load is executed, when a Form is loaded.
Private Sub Form_Load()
'** To create a data source list using ODBC API
Call ListDataSources
End Sub
'** ListDataSources can make a data source list
'** and display it in a listbox
Sub ListDataSources()
Dim ret As Integer
Dim dataSource As String * 32
Dim dsDesc As String * 2048
Dim dsLen As Integer, dsDescLen As Integer
Dim henv As Long, hdbc As Long, hstmt As Long
'** To get an Environment Handle
ret = SQLAllocEnv(henv)
If ret = SQL_ERROR Then
  Call ErrorMSG(henv, hdbc, hstmt)
  Exit Sub
End If
'** To get Data Source List
'** Fetch the First Record
```

```
ret = SQLDataSources(henv, SQL_FETCH_FIRST, dataSource, 31,
dsLen, dsDesc, 2047, dsDescLen)
Do
  If ret = SQL_ERROR Then
    Call ErrorMSG(henv, hdbc, hstmt)
    Exit Sub
  ElseIf ret = SQL_NO_DATA_FOUND Then
'** When the end of records, Exit Do-sLoop
    Exit Do
  End If
  list1.AddItem dataSource
'** Fetch a Next Record
  ret = SQLDataSources(henv, SQL_FETCH_NEXT, dataSource, 31,
dsLen, dsDesc, 2047, dsDescLen)
Loop
End Sub
```

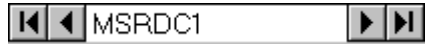
Creating a Table List

To create a table list according to the data source name you specified in the data source list, click the Get Table List button. It is easy to get information about table names in a database using one of the RDO properties (rdoTables) as follows.

```
'** This subroutine is executed, when a button is clicked
Private Sub Command2_Click(
Dim tbNum As Integer
Dim tables As Variant
MSRDC1.SQL = ""
MSRDC1.DataSourceName = text2.Text
'** To connect to a data source
MSRDC1.Refresh
List2.Clear
'** To get table names in a data source and set them to a
listbox
For Each table In MSRDC1.Connection.rdoTables
  List2.AddItem tables.Name
Next
End Sub
```

Issuing a SQL Statement and Getting a Result Set

Before a SQL statement is executed, you need to create the following Remote Data Control anywhere in your Visual Basic form. It should be invisible using a property. To execute a SQL statement with this control, specify a data source name in the DataSourceName property and a SQL statement in the SQL property.



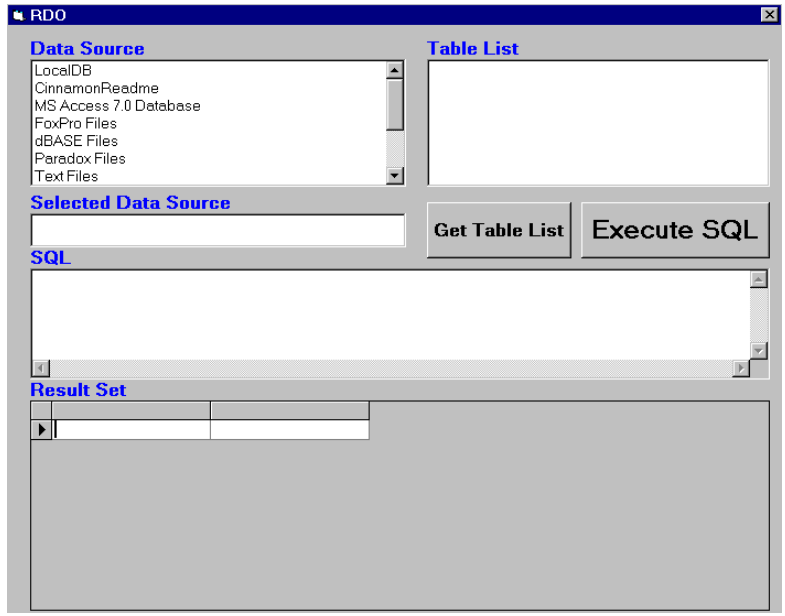
```
'** This subroutine is executed, when a button is clicked
Private Sub Command1_Click()
'** To Ignore Run Time Error
On Error Resume Next
MSRDC1.DataSourceName = text2.Text
MSRDC1.SQL = Text1.Text
'** To issue a query
MSRDC1.Refresh
'** Cancel To Ignore Run Time Error
On Error GoTo 0
End Sub
```

Note Our sample program does not provide for error handling, as we just intended to show some behaviors of NotesSQL. When you input an incorrect SQL statement, nothing will happen. If you want to create a fully developed error handling routine, remove the On Error statements in the above program.

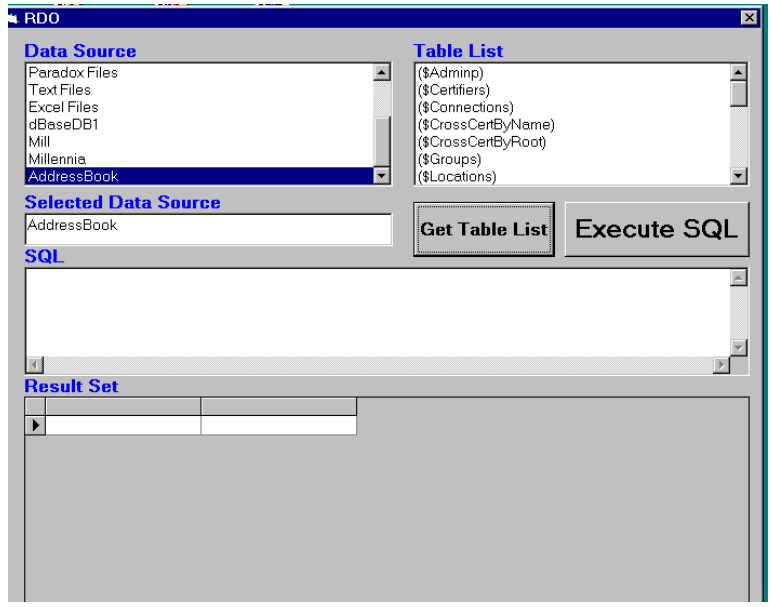
How It Works

This is how it works:

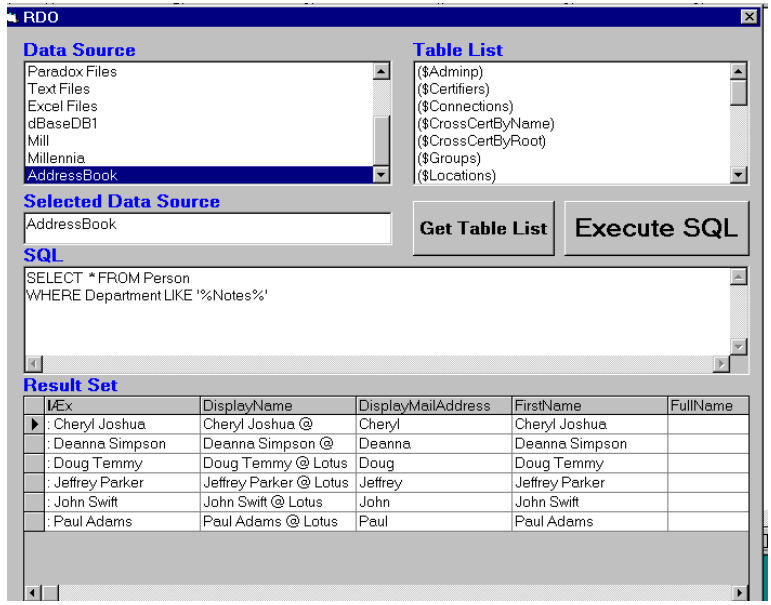
1. After the program is launched, the following data source list is displayed. The data source list is only created at launch time.



2. Choose a data source from the list and click the Get Table List button. This displays a table list on the right-hand side of the dialog box. The data source name which you chose is also displayed in the Selected Data Source field. The table list is only used to refer to table names in a database and has no effect on the following SQL query.



3. To get a SQL result set, specify a SQL statement in the SQL text box and click the Execute SQL button. The following figure shows a result set of a query to an Address Book in Lotus Notes using the NotesSQL ODBC driver:



Example: Using NotesSQL With Microsoft Active Server Pages (ASP)

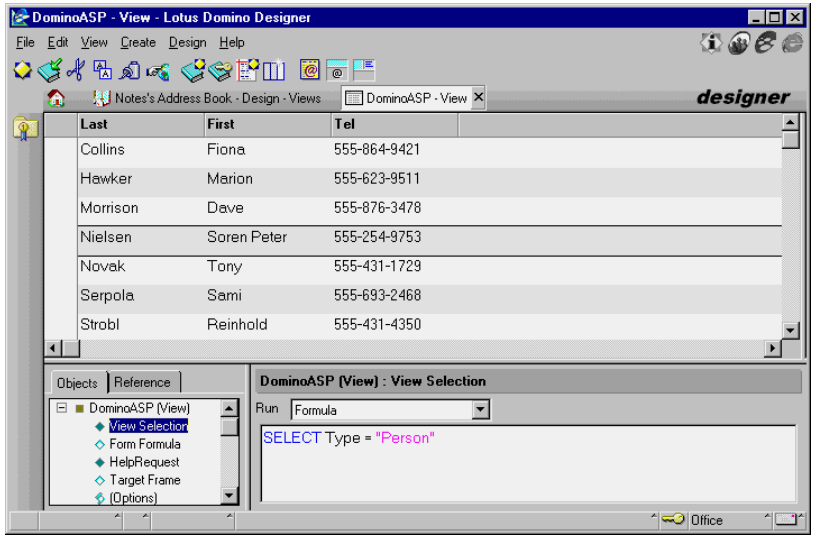
In the following section we will set up Microsoft Internet Information Server to access Domino databases using the NotesSQL driver and FrontPage 98.

Note Before you start you must make sure that your IIS server is configured to run ASP pages and has the Microsoft FrontPage extensions installed on the server. You must also have at least the Lotus Notes client software installed.

1. In Domino Designer create a new view in the Address book and name it DominoASP.
2. Create three columns in the view containing the fields LastName, FirstName and OfficePhoneNumber.
3. Make the LastName column sorted.
4. Name the columns; Last, First, Tel.
5. Set the selection formula to

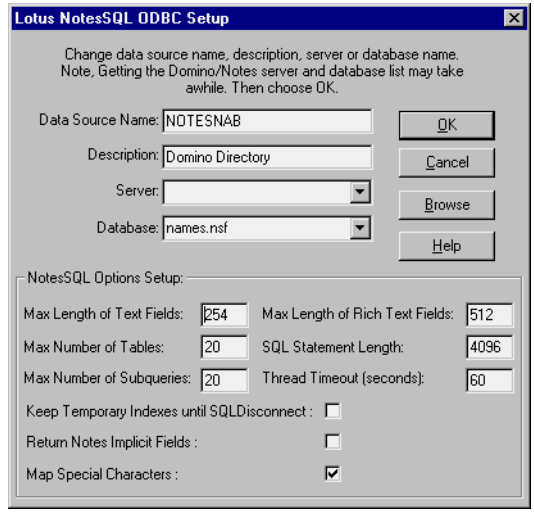
   ```
   SELECT TYPE = "Person"
   ```

6. Your view should now look like this:

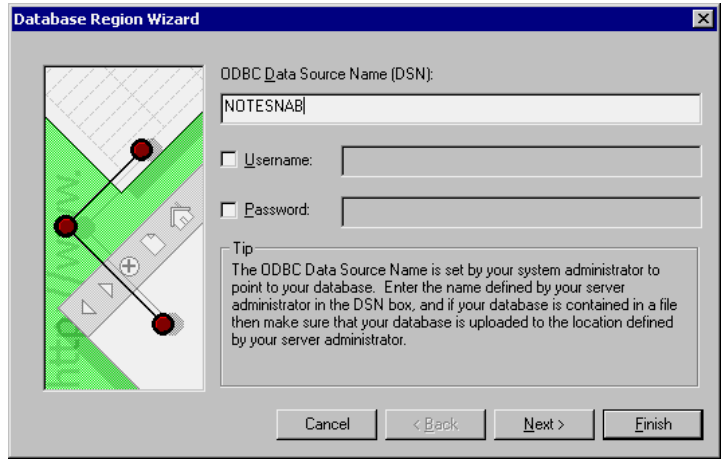


7. Save the view.
8. Open the Windows Control Panel and double click the ODBC Administrator icon.
9. Select the System DSN tab and click Add.
10. Select the Lotus Notes 2.0 (32-bit) ODBC Driver (*.nsf) from the list of available drivers and click Finish.

11. Complete the setup panel as follows: Data Source Name = NOTESNAB, Server=*your server name*, Database=names.nsf. The dialog box should look like the screen below.

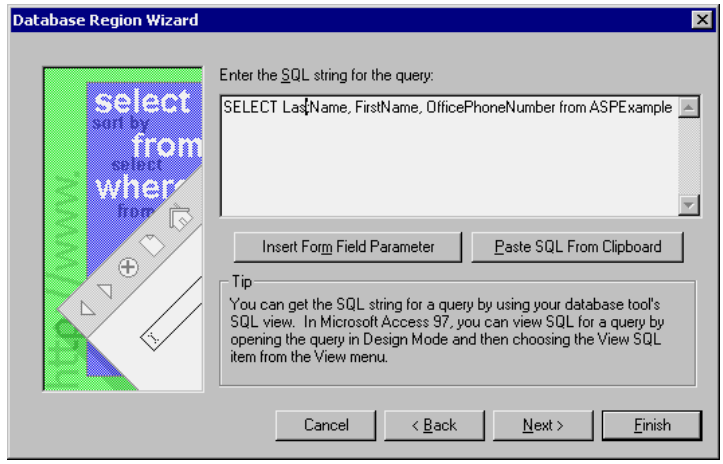


12. Click OK to close the setup dialog box and OK again to close the ODBC Administrator dialog box.
13. Start FrontPage 98 and create and open a new page in the FrontPage Editor.
14. Choose Insert Database - Database - Region Wizard.… from the menu.
15. On the first page of the wizard, type **NOTESNAB** as the ODBC data source and click Next.



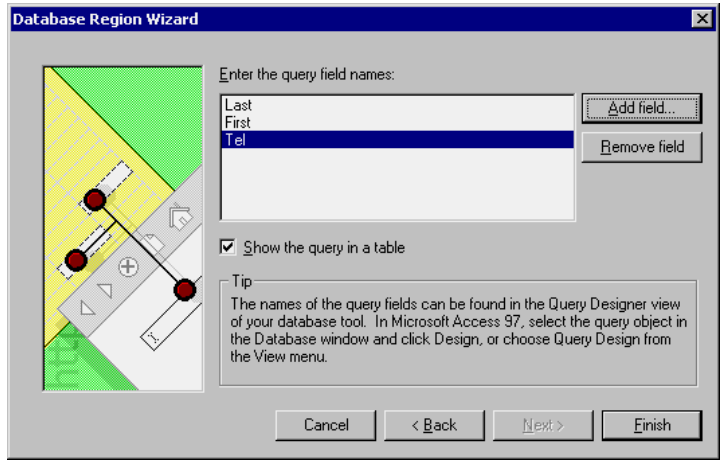
16. In the next dialog box enter the following SQL statement:

```
SELECT LastName, FirstName, OfficePhoneNumber from
ASPExample
```



17. Click the Next button.

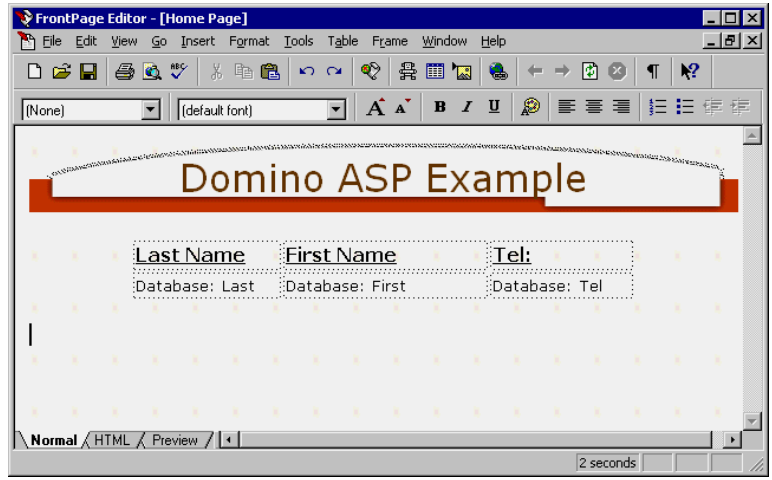
18. In the next dialog box enter the three column names from the view by clicking Add and typing **`Last, First, Tel`** respectively.



19. Make sure that the Show the Query in a table check box is ticked and click the Finish button.

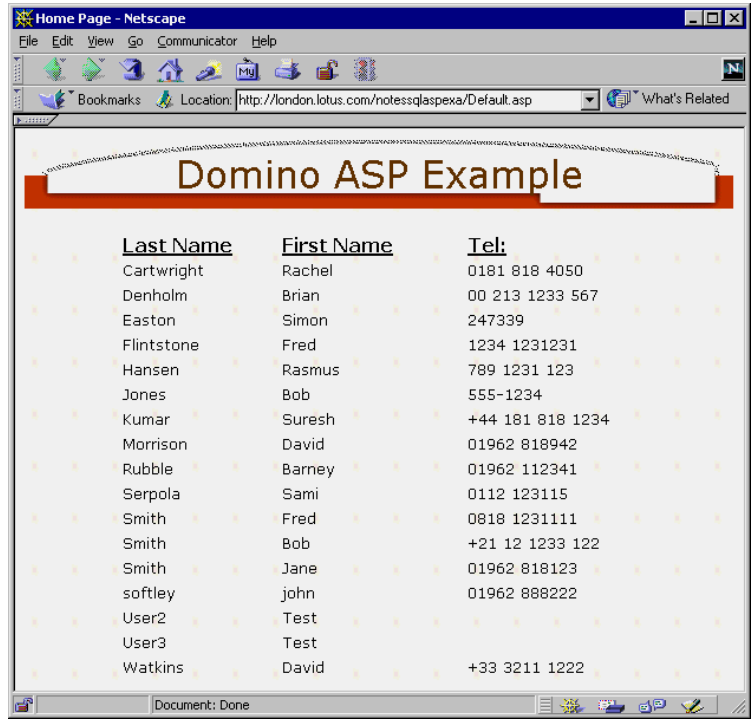
20. You will receive a warning message telling you that you must save this file as an ASP file rather than an HTML file. Click OK to remove the message.

21. Your FrontPage screen will now look like this.



22. Choose File - Save As and type **`Default.ASP`** as the filename. Click OK to save the file.

23. Choose File - Preview in Browser from the menu. Select your preferred browser and click OK. You will see a screen like the one below.



Domino Driver for JDBC

The Lotus Domino Driver for JDBC is a Type 2 JDBC driver that allows you to access Domino databases from Java.

What is JDBC?

JDBC is an object interface that allows Java applications and applets to retrieve and manipulate data in database management systems using SQL. The interface allows a single application to connect to many different types of databases through a standard protocol. JDBC handles details for such tasks as connecting to a database, fetching query results, committing or rolling back transactions, and converting SQL types to and from Java program variables. JDBC is implemented as a driver manager with multiple drivers. Each driver links the application to a specific type of database.

JDBC was first introduced in the Java Development Kit (JDK) 1.1 from Sun Microsystems. The JDBC classes and interfaces are part of the java.sql package. The major components of JDBC are the JDBC driver manager and the underlying drivers. JDBC uses the driver manager to handle finding and loading a driver. A JDBC data source consists of the data the user application wants to access and its associated parameters. Each JDBC driver processes JDBC method invocations, sends SQL statements to a specific data source, and returns results to the application.

JDBC drivers generally fit into one of four types:

1. The JDBC-ODBC bridge provides JDBC access via ODBC drivers. NotesSQL (the Domino/Notes ODBC driver) may be used with the JDBC-ODBC bridge.
2. A native-API, partly-Java driver converts JDBC calls into calls on the client API for the DBMS in question. This style of driver requires that some binary code be loaded on the client machine. Domino Driver for JDBC is a Type 2 driver.
3. A net-protocol, all-Java driver translates JDBC calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect its all-Java clients to many different databases. This is the most flexible Java alternative.
4. A native-protocol, all-Java driver converts JDBC calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Internet access.

Note From a functionality and SQL syntax view point, the JDBC driver for Domino is the same as NotesSQL.

Domino Driver for JDBC Data Types

Domino Driver for JDBC Types Mapped to Java Object Types

This table shows the mapping from Domino Driver for JDBC types to Java object types that is used by the getObject/setObject methods of Domino Driver for JDBC.

The JDBC types BIT, TINYINT, BIGINT, BINARY, VARBINARY and LONGVARBINARY are not supported by Domino Driver for JDBC.

Lotus Domino Driver for JDBC Type	*Java Object Type*
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
SMALLINT	Integer
INTEGER	Integer
REAL	Float
FLOAT	Float
FLOAT	Double
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Java Object Types Mapped to Domino Driver for JDBC Types

The Java Object Types Boolean, Long, and byte[ ] are not supported by Domino Driver for JDBC.

String will always map to VARCHAR.

Java Object Type	*Lotus Domino Driver for JDBC Types*
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
Integer	INTEGER
Float	REAL
Double	DOUBLE
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Conversions by setObject between Java Object Types and Target Domino Driver for JDBC Types

An "X" means that the given Java object type may be converted to the given Domino Driver for JDBC type.

Note Some conversions may fail at runtime if the value presented is invalid.

	SMALLINT	INTEGER	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	CHAR	VARCHAR	LONGVARCHAR	DATE	TIME	TIMESTAMP
String	X	X	X	X	X	X	X	X	X	X	X	X	X
java.sql.BigDecimal	X	X	X	X	X	X	X	X	X	X			
Integer	X	X	X	X	X	X	X	X	X	X			
Float	X	X	X	X	X	X	X	X	X	X			
Double	X	X	X	X	X	X	X	X	X	X			
java.sql.Date								X	X	X	X		X
java.sql.Time								X	X	X		X	
java.sql.Timestamp								X	X	X	X	X	X

Use of getXXX Methods to Retrieve Domino Driver for JDBC Datatypes

An "O" means that the method can retrieve the Domino Driver for JDBC type. An "X" means that the method is recommended for that Domino Driver for JDBC type.

The methods getByte, getLong, getBoolean, getBytes, and getBinaryStream are not supported.

	SMALLINT	INTEGER	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	CHAR	VARCHAR	LONGVARCHAR	DATE	TIME	TIMESTAMP
getShort	X	O	O	O	O	O	O	O	O	O			
getInt	O	X	O	O	O	O	O	O	O	O			
getFloat	O	O	X	O	O	O	O	O	O	O			
getDouble	O	O	O	X	X	O	O	O	O	O			
getBigDecimal	O	O	O	O	O	X	X	O	O	O			
getString	O	O	O	O	O	O	O	X	X	O	O	O	O
getDate								O	O	O	X		O
getTime								O	O	O		X	O
getTimestamp								O	O	O	O	O	X
getAsciiStream								O	O	X			
getUnicodeStream								O	O	X			
getObject	O	O	O	O	O	O	O	O	O	O	O	O	O

Creating a Connection

To create a connection in your Java application to a Domino database using the Domino JDBC driver, you need to perform a number of steps. The following example describes how to connect to your Local Address Book, Names.nsf, and print a list of all the people stored within it.

Note We used IBM VisualAge for Java 1.1 to create this example.

1. Before you can use JDBC within Java, you must import the java.sql package which is included with Java 1.1.

   ```
   import java.sql.*;
   ```

2. You then need to import the Domino JDBC driver package so you can access the Domino specific functions.

   ```
   import lotus.jdbc.domino.*;
   ```

3. Next you create an instance of the driver and register it with the JDBC driver manager using the Class.forName function. If the function produces a ClassNotFoundException error you need to catch it and act accordingly.

```
try {
  Class.forName("lotus.jdbc.domino.DominoDriver");}
catch (ClassNotFoundException e) {
  System.out.println("ClassNotFoundException: " +
                     e.getMessage()); }
```

4. You now need to specify where the database that the JDBC driver is to connect to is located. This is done using the JBDC URL. A JDBC URL for the Domino driver is in the format:

```
jdbc:domino/<filename>[/<server>][;<keyword>=<attribute>[;
<keyword>=<attribute>]...]
```

where *jdbc:domino* is the protocol and subprotocol, *filename* is the fully qualified path and filename to the Domino database, *server* is the name of the server where the database resides, and *keyword* is one or more optional keywords. For a list of all the available keywords and corresponding attributes, see the JDBC driver documentation.

In our example we are going to connect to the names.nsf database on the Red5 server. The URL would look like this:

```
connStr = "jdbc:domino/names.nsf/red5.lotus.com";
```

If we needed to connect to the database JdbcDemo.nsf on the local machine in a subdirectory from the Domino data directory of Samples, the URL would look like this:

```
"jdbc:domino/samples\\jdbcdemo.nsf";
```

Note We use the reversed double backslash characters to designate the file path, and no server name.

5. You can now use the URL to establish a connection to the database using the getConnection method of the JDBC driver manager. The second and third parameters of the getConnection method are for userid and password, which are not used by the Domino JDBC driver, since they are retrieved from the current Notes userid file. In this next line of code you connect to the database and set a connection object, con:

```
con = DriverManager.getConnection(connStr,"","");
```

6. Next you need to tell the connection what you would like to do with the database. This is done with SQL commands. In this example you will retrieve the firstname and lastname fields from the person document so the query looks like this:

```
String sql = "SELECT FirstName, LastName FROM Person";
```

7. Having created an SQL query, you need to create a JDBC statement object to use with it. A statement object sends SQL statements to a database.

```
stmt = con.createStatement();
```

8. You can now execute the SQL query on the database using the executeQuery method of the JDBC statement object you created in the previous step. There are two methods to use when executing SQL commands on a database:

   - execute Query, which returns a result set, and
   - executeUpdate, which is used with INSERT, UPDATE and DELETE commands and returns the number of rows in the database that were affected.

   In this example, you are simply querying the database so you will use the executeQuery method:

```
ResultSet rs = stmt.executeQuery(sql);
```

   A result set gives you access to the data retrieved from the table in the database. The table rows are retrieved in sequence and the result set object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The "next" method moves the cursor to the next row.

9. To access more detailed information about the information within the result set, you can create a ResultSetMetaData object. The ResultSetMetaData object contains information such as column names, column types, and number of columns. To access these properties, use the following code:

```
ResultSetMetaData rsmd = rs.getMetaData();
```

10. Once you have access to information within the result set, such as column names, you can use these to format the output of the data by adding column headings. The following code loops through each of the column headings and prints them to the console window. It also adds an additional line containing dashes to separate the headings from the data that will follow:

```
// Find number of columns in the result set
int colCount = rsmd.getColumnCount();
System.out.println(colCount);

// Array to hold max display size per column
int[] len = new int[colCount+1];

// Print column Labels as header
for (int i=1; i<= colCount; i++)
{
  // Get column label.
  String label = rsmd.getColumnLabel(i);
  // Store the maximum of display size or label length
```

```
  if (label.length() > 10)
        len[i] = label.length();
  else
        len[i] = 10;
  // Print label
  System.out.print(label);
  // Pad with blanks
  fill(" ",len[i]-label.length());
  // Column seperator
  System.out.print("  ");
}
// New line
System.out.println();
for (int i=1; i<= colCount; i++)\
{
  fill("-", len[i]);
  System.out.print("  ");
}
// New line
System.out.println();
```

The two functions used in the above code, fill() and printCol() are shown below:

```
  static void fill(String s, int times)
  {
       if (times <= 0)
             return;
      for (int i=0; i<times; i++)
      {
          System.out.print(s);
      }
  }
  static void printCol(int len, String s)
  {
       System.out.print(s);
       fill(" ",len-s.length());
       System.out.print("  ");
  }
```

11. You can now output the information within the result set to the console. You will use the getString method of the result set object to retrieve the columns and PrintCol to output them.

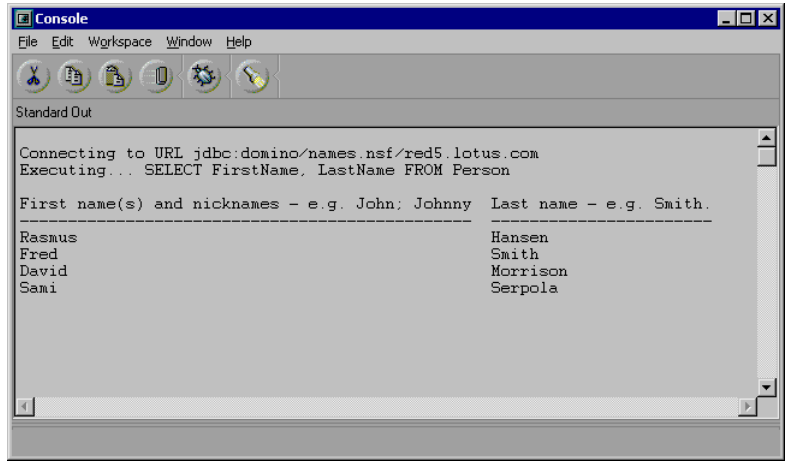
```
while (rs.next())
{
  // Get all columns as String.
  String  Fname = rs.getString(1);
  printCol(len[1], Fname);|
  String  Lname = rs.getString(2);
```

```
        printCol(len[2], Lname);
        System.out.println();
    }
```

12. Finally, you close both the statement object and the connection object using the close methods of each.

```
stmt.close();
con.close();
```

13. The following screen shot shows the result of running the code:



For your convenience, the entire Java code is included below:

```
import java.util.*;
import java.sql.*;
import lotus.jdbc.domino.*;

public class Domino_Sample1
{
      public static void main(String[] args)
      {
            Connection         con;
            Statement          stmt;
            ResultSet          rs;
            ResultSetMetaData  rsmd = null;
            String sql =
            "SELECT FirstName, LastName FROM Person";
            String connStr =
            "jdbc:domino/names.nsf/red5.lotus.com";

            try {
                  try {

Class.forName("lotus.jdbc.domino.DominoDriver"); }
```

```
                    catch (ClassNotFoundException e) {

System.out.println("ClassNotFoundExecption: " +
e.getMessage());
                    }

                    // GET CONNECTION
                    System.out.println();
                    System.out.println("Connecting to URL " +
                    connStr);

                    con =
                    DriverManager.getConnection(connStr,"","");
                    SQLWarning warning = con.getWarnings();

                    // Create Statement
                    stmt = con.createStatement();

                    // Execute statement
                    rs = stmt.executeQuery(sql);
                    System.out.println("Executing... " + sql);
                    System.out.println();

                    // Get Result set metadata
                    rsmd = rs.getMetaData();

                    // Find number of columns in the result set
                    int colCount = rsmd.getColumnCount();

                    // Array to hold max display size per column
                    int[] len = new int[colCount+1];

                    // Print column Labels as header
                    for (int i=1; i<= colCount; i++)
                    {
                          // Get column label.
                          String label = rsmd.getColumnLabel(i);
                          // Store the maximum of display size or
                          label length
                          if (label.length() > 10)
                            len[i] = label.length();
                          else
                            len[i] = 10;
                          // Print label
                          System.out.print(label);
                          // Pad with blanks
                          fill(" ",len[i]-label.length());
                          // Column seperator
                          System.out.print("  ");
                    }
                    // New line
                    System.out.println();
```

```
                    for (int i=1; i<= colCount; i++)
                    {
                         fill("-", len[i]);
                         System.out.print("  ");
                    }
                    // New line
                    System.out.println();
                    // Fetch all rows in the result set
                    while (rs.next())
                    {
                         // Get all columns as String.

                         String  Fname = rs.getString(1);
                         printCol(len[1], Fname);

                         String  Lname = rs.getString(2);
                         printCol(len[2], Lname);

                         System.out.println();
                    }

                    // Close the statement
                    stmt.close();

                    // Close the connection
                    con.close();

               } catch (Exception e) {
                    System.out.println(e.getMessage());
               }
     }
       // Print a column and pad it with blanks + separator
       static void printCol(int len, String s)
       {
            System.out.print(s);
            fill(" ",len-s.length());
            System.out.print("  ");
       }
       // Print a string N times
       static void fill(String s, int times)
       {
            if (times <= 0)
                 return;
           for (int i=0; i<times; i++)
           {
               System.out.print(s);
           }
       }
}
```

Using IBM VisualAge for Java Version 1.0

The VisualAge for Java Data Access Builder lets you rapidly develop data access programs using the JDBC API. In much the same way that the ODBC standard defines an API that enables applications to access any ODBC-compliant database, the JDBC standard defines a common base on which other data access tools can be built. If you want to leverage existing ODBC databases using Java, Sun's JDBC-ODBC bridge driver translates JDBC calls into ODBC calls.

The Data Access tool (DAX) of VAJava generates JDBC code using a wizard which interfaces only with ODBC drivers. The end result is an applet that loads the JDBC-ODBC bridge and connects to a database using the JDBC-ODBC URL syntax specifying the ODBC data source.

Currently, the only way to load other JDBC drivers is to modify the property values of the Driver and the URL before you generate the applet code. To use VAJava and Domino Driver for JDBC, you must:

- Have the NotesSQL ODBC driver.
- Before generating the JDBC code, modify the property values of the driver and the URL.

For more information, see the VAJava documentation.

Using IBM WebSphere Application Server

IBM WebSphere Application Server provides a runtime environment for Java servlets with connectors to common database formats. It runs on HTTP servers including Netscape Enterprise Server, Netscape FastTrack Server, Microsoft Internet Information Server, Apache Server, and the Lotus Domino Go Webserver.

Servlets enable request/response services on an HTTP server. When a client sends a request to a Web server, the servlet can construct the response that the server sends back to the client. A servlet can be loaded automatically when the Web server is started, or the first time a client requests its services. After loading, a servlet keeps running, waiting for more client requests.

To Configure and Run Domino Driver for JDBC Using IBM WebSphere

1. Set up Domino Driver for JDBC support:
   - Log in to IBM WebSphere Application Server 1.0 ServletExpress Manager.
   - Add the full path of Domino Driver for JDBC (for example, c:\notes\jdbcsql\lib\JdbcDomino.jar) to the Java ClassPath edit field on the "Basic" tab (Manage - Setup - Basic).

2. Write your own servlet to load and run Domino Driver for JDBC. For example:

```
import lotus.jdbc.domino.*;
...
public class Sample1 extends HttpServlet
{
   public void doGet (HttpServletRequest req,
HttpServletResponse res)
   throws ServletException, IOException\
   {
      Connection     con;
      PrintWriter    out;
      String connStr = "jdbc:domino/JdbcDemo.nsf";
      res.setContentType("text/html");
      out = res.getWriter();
      out.println("<html>");
      Out.println("<head><title>Sample1
                  Servlet</title></head>");
      out.println("<body>");
      try {
         try {

Class.forName("lotus.jdbc.domino.DominoDriver"); }
         catch (ClassNotFoundException e) {
             out.println("ClassNotFoundExecption: " +
                         e.getMessage());
             throw new Exception ("Servlet Failed.");
         }
         // GET CONNECTION
         out.print("<p>");
         out.print("Connecting to URL " + connStr);
         con = DriverManager.getConnection(connStr,"","");
         /*
         * We now have a valid connection to our Domino
         * database and can query it as usual.
         *
         Put your code here !!
         *
         */
         // Close the connection
         con.close();
         out.println("</body></html>");
         } catch (Exception e) {
            out.println(e.getMessage());
            out.println("</body></html>");
         }
   }
}
```

LotusScript Data Objects and ODBC

This section describes the LS:DO (LotusScript:Data Object) and @DBCommand, @DBLookup, @DBColumn functions that can be used to access data resources from a Notes application.

The above features are based on Open Database Connectivity (ODBC) technology.

When you are developing your Notes application, you need to decide what your data access needs are and which products best meet those needs. Each product has different functionality as well as performance and programmability. When you have completed this section, you should know when to use these methods, and how to use them.

Data Resource Access

When you develop a Notes application, you often need to implement data integration between Notes and other data resources such as RDBMS, spreadsheet data, and ASCII delimited text files. In enterprise Notes application development, this becomes even more important as you integrate legacy database resources in your design.

Database Access Facilities

The following tools enable Notes applications to connect to data resources through ODBC or native database access:

- **LS:DO (LotusScript:Data Object)**

  This is a LotusScript Extension (LSX) which provides additional LotusScript classes for accessing other data resources via ODBC.

- **@DBCommand, @DBLookup, @DBColumn using ODBC**

  These are @functions for ODBC data access. The functions @DBLookup and @DBColumn are frequently used to access Notes databases as well as ODBC-compliant databases.

The choice of tool to use in a particular situation depends on the functionality and performance of the particular tool. The following table summarizes the characteristic differences between the tools covered in this section:

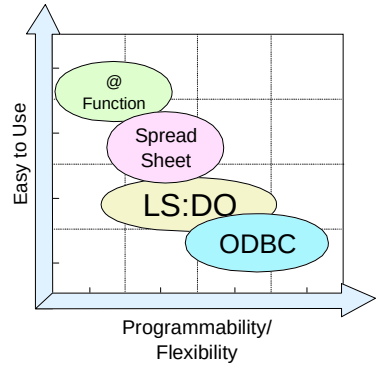
Characteristics	*LS:DO*	*@DBLookup @DBColumn*	*@DBCommand*
Based on ODBC	X	X	X
Available in LotusScript	X	*1	*1
Has a Class	X		
Read Only		X	X
64KB Data Limit		X	X

*1 Technically, LotusScript can perform @functions under the Evaluate function.

Furthermore, we must consider some ease-of-use versus programming functionality and flexibility. For example, @functions are useful to retrieve small amounts of data on the fly without complex sequences, but they are limited in the number of ways to access data.

LS:DO can be used in more complex situations with much more flexibility from a programming perspective, for example, result-set handling to read and update records queried by SQL. LS:DO is also easy to use if you are familiar with LotusScript. The figure below, may help in deciding which tool to use in a particular situation:

Relationship among Data Access Features



What is ODBC?

The Open Database Connectivity (ODBC) standard is a set of functions developed by Microsoft to access Relational Database Management Systems (RDBMS) like Oracle, DB/2, Informix and others. There are two software components required to use ODBC:

- **ODBC Driver Manage**r is a set of APIs in the ODBC dynamic link library. Those APIs are called by client programs like LS:DO, NotesSQL, and so on, in order to access an RDBMS via ODBC.
- **RDBMS ODBC driver** is the driver for specific RDBMSs like NotesSQL, DB2, Oracle etc. The ODBC driver allows you to issue any SQL statements in Data Definition Language (DDL), Data Control Language (DCL), and Data Manipulation Language (DML) using SQLExecute or SQLExecDirect with the ODBC API. In addition, other ODBC Drivers enable you to get information about columns attributes, index, privileges of column, drivers, foreign keys of tables, and other RDBMS entities.

Using ODBC Connections

You can use ODBC with:

- APIs in your C, C++, Basic, LotusScript, or any other programming language programs.

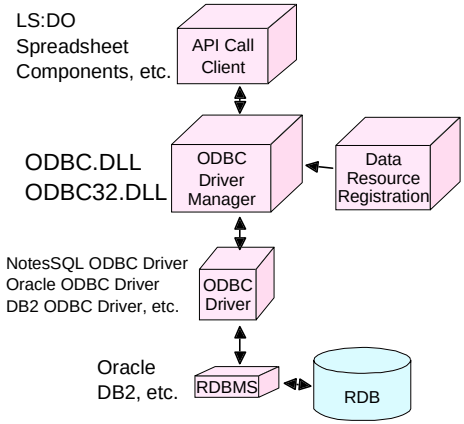
  Note The programming language you use must support calls to a Dynamic Link Library (DLL), because all of the ODBC functions are in the ODBC DLL.

- Compliant high-level tools such as LS:DO, Lotus Spreadsheet Component in LotusScript, and Data Access Object in Visual Basic.

ODBC Access Flow

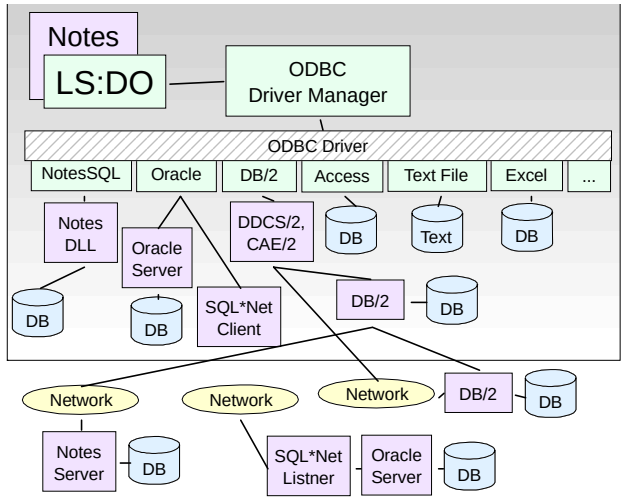
The process by which a program accesses a database through ODBC is shown below:

1. The program makes a call to the ODBC API.
2. The ODBC driver manager parses the requested command.
3. The ODBC driver manager decides which ODBC driver is required according to database resource information registered in advance through the operating system.
4. The requested command is passed to the specific ODBC driver for the database being accessed.
5. The ODBC driver composes a series of commands for the particular RDBMS and sends them to the RDBMS.
6. The results, if available, are sent to the calling routine.



There are many ODBC drivers. Usually they are provided by RDBMS vendors, but others come from independent software vendors like InterSolv or Visigenic. The following figure conceptually shows LS:DO making connection paths to RDBMSs as an example. It also shows other ODBC

drivers that are capable of accessing ASCII delimited text files, spreadsheets, and other types of data resources besides RDBMS:



There are two types of ODBC driver managers in the Windows environment: the 16-bit ODBC driver manager and the 32-bit ODBC driver manager. You must ensure that the one you use matches the application environment you are in. For example, when you use the 32-bit Windows version of Lotus Notes, you need a 32-bit ODBC driver manager and a 32-bit RDBMS driver.

LotusScript:DataObject (LS:DO)

What Is LS:DO?

The LotusScript:Data Object (LS:DO) provides full Read and Write access to external ODBC data sources using the complete control and flexibility of the LotusScript structured programming language.

The LS:DO consists of a set of three classes:

- ODBCConnection
- ODBCQuery
- ODBCResultSet

These classes come complete with a powerful set of properties and methods and full SQL capabilities. Yet at the same time, the LS:DO is easy to learn and use because its design is consistent with LotusScript BASIC syntax and the LotusScript Notes classes.

Concepts

The LS:DO is available on both the Notes client and the Domino server. LS:DO is excellent for real-time data access from any LotusScript event in Notes, such as clicking a button, exiting a field, or opening a document. LS:DO real-time data access is the best choice for:

- Optimizing data entry
- On-the-fly lookups
- Immediate updates
- Input validation
- Avoiding duplicate entries
- Mobile user queries and updates

Optimizing Data Entry

Many designers use Notes as the data entry point for an application, which can synchronize that data with a RDBMS or use the RDBMS for long-term data storage and archiving. The LotusScript Data Object can provide the following functionality on the fly:

- **On-the-fly look up**
  Once a user enters a customer name and exits the field with the **TAB** key or a mouse click, LotusScript code can immediately perform an SQL query to one or several external back ends, retrieve the customer record matching that name, and fill in the remaining fields in the form, such as address, city, phone, and contact name.
- **Immediate updates**
  LotusScript gives you the flexibility to update the information in the RDBMS the moment the user saves a new document in Notes, or in batches at scheduled intervals. When another document is created in Notes, you can be sure that document will access the most current information in the relational DBMS.
- **Input validation**
  Is the right salesperson assigned to that customer in the Notes form? Is the regular salesperson for that region currently overloaded with assignments, indicating that a backup person should be assigned to the task? The LotusScript Data Object can retrieve that information from the DBMS that indicates these conditions, and the LotusScript fully structured programming constructs enable you to evaluate that data and act accordingly.
- **Avoiding duplicate entries**
  Once a user enters a customer name, the LS:DO can query the back end for variations on that customer name, for example, to ensure that the same customer is not entered with an "Inc." as opposed to a "Co." in the DBMS.

Mobile User Queries and Updates

One of the most exciting results of the intersection between RDBMSs and Notes is that mobile Notes users can take their access to the RDBMS with them on the road. For example, sales representatives on the road often find themselves with last-minute opportunities to visit customers in different cities. If the customer's information is contained in the mainframe RDBMS, the salesperson is forced to call someone in the office, and ask them to look up the information, which cannot happen from a hotel room outside business hours.

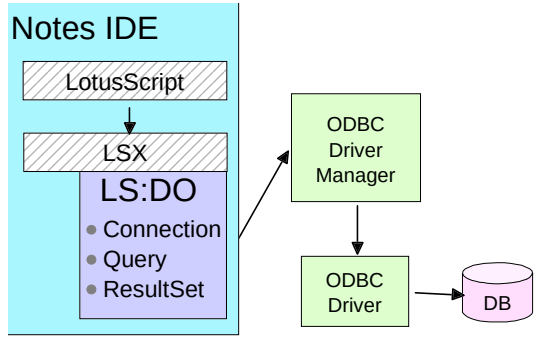
The LotusScript Data Object has the ability to run on Domino servers as well as clients, and coupled with Notes native replication capabilities, solves the problem. With an integrated Notes/RDBMS application, a user can do the following:

1. Compose a query request within an application on their mobile Notes client, such as "What are the customer contacts and activity in this city?"
2. Replicate the query to the Domino server, where a waiting LS:DO agent sees the new document, authenticates and performs the query, stores the results in that document, and saves it.
3. Replicate the query results back to the user's laptop in moments, even during the same dial-up connection if they choose, for analysis and review.

Architecture

In addition to allowing users to issue SQL statements to RDBMSs, the LS:DO also offers data manipulation capabilities. The LS:DO supports and manages result sets as well as provides an interface for directly using SQL when appropriate. The result set management takes the form of caching result sets, supporting navigation through the result set, and managing individual row updates regardless of the underlying driver's cursor or ODBC conformance capabilities.

The following diagram is a schematic representation of the components in the LS:DO framework that allow a Notes application to access a database:



When to Use LS:DO

The LS:DO is best suited to handle the following situations:

- **LotusScript programming environment.** If you develop an application using the LotusScript environment, you can easily utilize ODBC access through the LS:DO classes.
- **Low-volume data transfer.** LS:DO is more suited for low-volume access to data resources. From a performance perspective LS:DO is not well suited to moving large volumes of data.
- **Easy data access.** When your application needs to both read and to update data in an RDBMS, LS:DO is an easier way than the ODBC API or the @DBCommand because of the classes allowing you to work with result sets.
- **Real-time direct access.** LS:DO is integrated directly into a Notes application and so on.

Differences Between LS:DO and ODBC

LS:DO is a high-level abstraction of the ODBC feature, that enables you to design more complicated operations toward RDBMS, but requires more detailed knowledge about the ODBC architecture and ODBC APIs. Let's look at three aspects of both methods:

- Programming Environment
- Functionality
- Performance

Programming Environment

Calling ODBC APIs requires passing many arguments. You have to be careful with the different argument types. A wrong argument type may cause unexpected severe errors, and could make your system unstable.

The LS:DO is more intuitive and at a higher level of abstraction than the ODBC API. Also, the LotusScript development environment checks syntax on the fly.

LS:DO is available only in LotusScript and some development environments that are compliant with OLE clients, such as Visual Basic. LS:DO is one of the LotusScript Class Libraries (LSXs). This enables you to benefit from the object-oriented and event-driven programming environment provided by the Notes Integrated Development Environment.

Functionality

Through LS:DO classes, you can update data in a result set, which is then automatically reflected to the original table. It is much easier to update data using LS:DO methods than using an SQL statement.

ODBC functions are calls from C or C++ programs to the Dynamic Link Library. There are three conformance levels:

- Core Level
- Extension Level 1
- Extension Level 2

There are more than 50 functions depending on the version number of the ODBC driver manager and the ODBC driver.

The following table conceptually shows which method in LS:DO calls which ODBC APIs. Each LS:DO method corresponds to a combination of some ODBC APIs:

Functions	*LS:DO Method*	*ODBC API*
List all data sources registered	Connection. **ListDataSources**	SQLAllocEnv SQLDataSources SQLFreeEnv
Establish a connection to DB	Connection.**ConnectTo**	SQLAllocConnect SQLBrowseConnect SQLFreeConnect SQLAllocConnect SQLConnect
List all tables in a database	Connection.**ListTables**	SQLAllocStmt SQLTables SQLFetch SQLGetData
Execute an SQL	ResultSet.**Execute**	SQLSetStmtOption SQLExecDirect
Fetch data from a result set	ResultSet.**GetValue**	SQLNumResultCols SQLColAttributes SQLFetch SQLGetData

Performance

ODBC provides better performance than LS:DO in some cases because the C++ program can directly access the ODBC driver manager, whereas LS:DO has some overhead due to the language architecture. Although LS:DO can make it easy to retrieve and update records in a result set, ODBC API calls

are more powerful because of more complex and efficient record data handling using fetch and retrieve of records, parameterized SQL, and cursor features.

Software Requirements

The software requirements are:

- **ODBC driver manager 2.0 or later.** You need to install the appropriate ODBC driver manager as required by your operating system and by the applications that use the ODBC features. The 32 bit ODBC driver manager comes with Visual Basic 4.0, Lotus SmartSuite 97, Office95, Visual C++ 4.0, and others. The 16 bit ODBC driver manager comes with Windows 3.1 and Windows 95.
- **ODBC drivers for specific RDBMSs.** For example, if you create a program to access the Oracle DB server, you must install an Oracle ODBC driver which corresponds to the ODBC driver manager type already installed. There are many drivers provided by many software companies for RDBMSs and other data resources as well.

  Some of them are listed in the following table:

	Microsoft	*Intersolv*	*Lotus*
Notes			X
1-2-3			X
Access	X		
DB2		X	
dBASE	X		X*1
Excel	X		
FoxPro	X		
Informix		X	
Ingres		X	
SQLServer	X		
Oracle	X	X	X*1
Paradox	X		
Sybase		X	
Text File	X	X	X*1

*1 This driver is bundled in Lotus SmartSuite.

Registering ODBC Data Sources

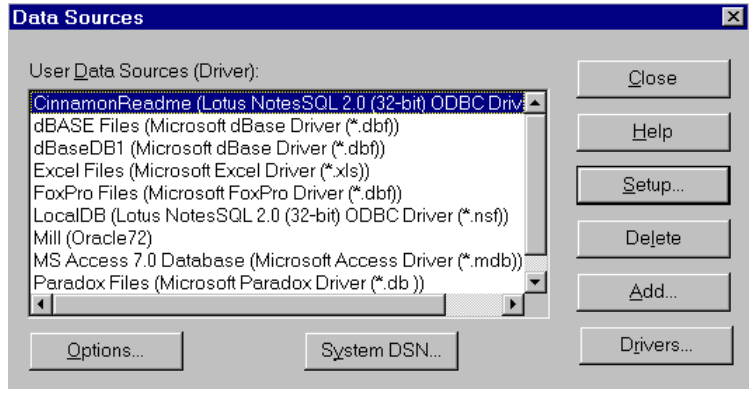
To register ODBC data sources, follow these steps. Our example is based on an Oracle DB Server connection for Windows 95. The basic operations are practically the same on the other platforms.

1. Double-click the ODBC driver manager icon in the Control Panel.

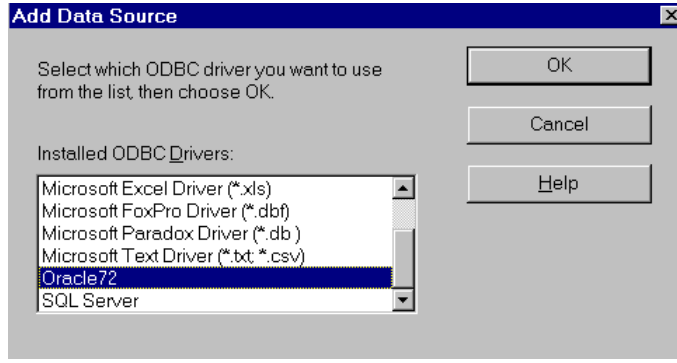
32-bit
Administrator

Note The above icon image and icon title depend on the driver manager you installed on your system.

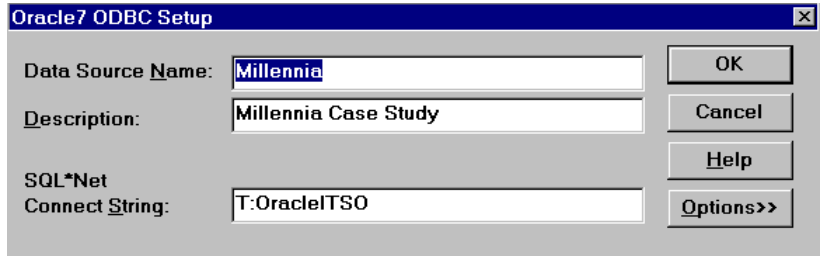
2. The Data Sources dialog box displays. You can see all data sources previously defined. To add a new data source, click the Add… button.



3. The Add Data Source dialog box displays. You can see all the ODBC drivers installed on your PC. Select the ODBC driver for your application and click OK. In our example, we selected the Oracle 72 ODBC driver:



4. If you select the Oracle ODBC driver, you'll need to fill in the appropriate information as follows. This dialog box will vary depending on the driver you choose.



- The Data Source Name you enter will be the one you specify in your programs, such as the ConnectTo method in LS:DO, whenever you connect to the Oracle Database.
- The Description field is just an explanation for this data source.
- The SQL*Net Connect String is important to establish the connection. In our case, the "T" means that the TCP/IP protocol is used during communication. The rest of the string "OracleITSO" is a TCP/IP hostname for the Oracle DB server on Windows NT. Optionally, you can specify the Oracle instance ID here. For a full description and different connect string examples, click the Help button of the Oracle7 ODBC Setup dialog box.

USELSX Statement to Enable LS:DO

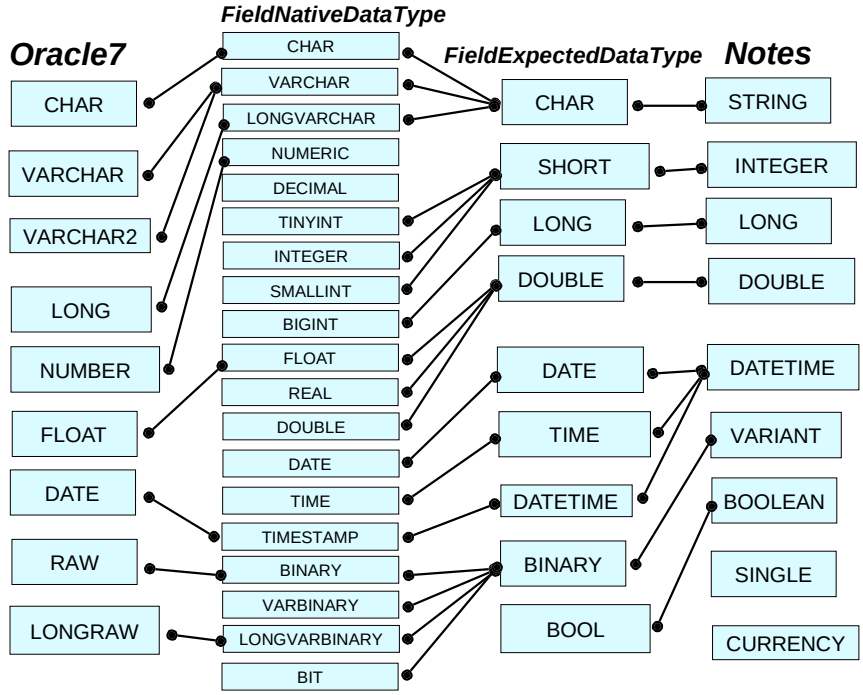
The following statement must be specified in the Define (Globals) Event (Declarations) within Notes:

```
Uselsx "*LSXODBC"
```

Note The leading "*" tells LotusScript to use the class registry to look up the path of the LS:DO dynamic library being loaded. This is a platform-independent way of loading LS:DO since each operating system uses different methods.

Mapping Data Types Between RDB and Notes DB

The following diagram shows the data type mapping between an Oracle database and a Domino database through the LS:DO:



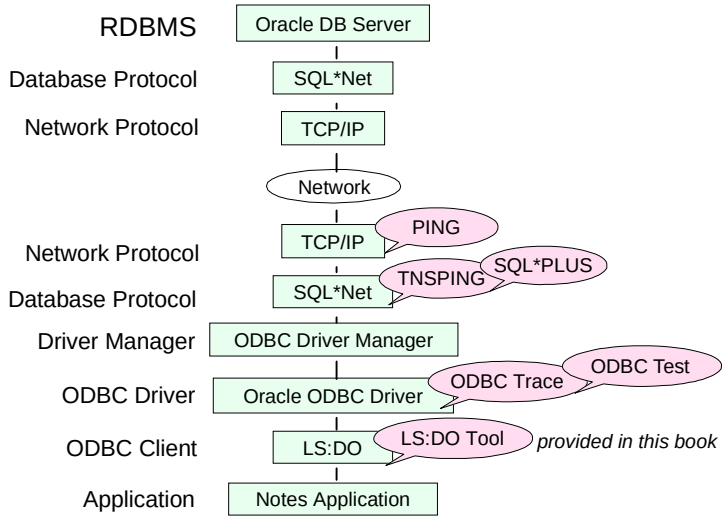
How to Trace and Debug LS:DO

In this section, we will briefly touch on a few ways to debug and trace applications that employ the LS:DO.

The structure for a connection between a Notes application using the LS:DO and the target RDBMS is the same as in all ODBC-compliant systems.

The following diagram shows each connection layer and the respective component. If a connection cannot be established for some reason, the configuration of each of the components must be verified one by one.

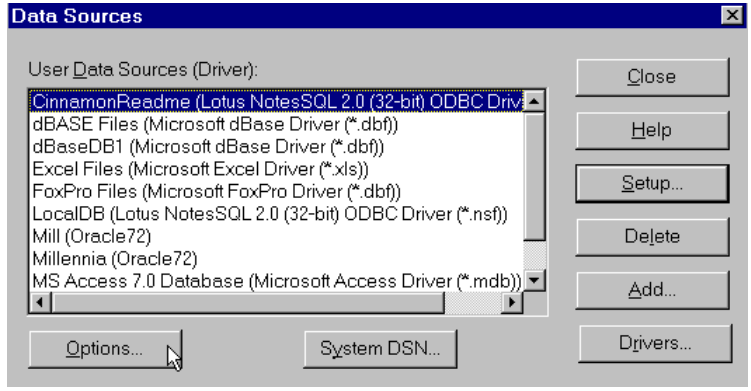
The appropriate debugging or tracing tool is indicated by the caption circle for each component.



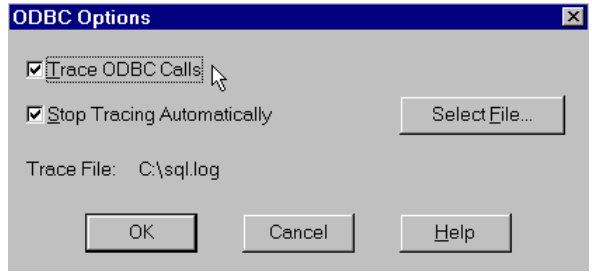
Using the ODBC Trace Option

Trace information issued by the ODBC API can be gathered using the trace function in the ODBC Administrator program. To do this, follow these steps:

1. Run the ODBC Administrator program and click Options.…



2. Select the Trace ODBC Calls check box. Click OK, and leave the ODBC Administrator program.



Trace descriptions of the ODBC API calls in your program(s) using LS:DO are saved in the "C:\sql.log" text file as a default. You may change the log file name by clicking the Select File button in the dialog box above. An example of some of the trace output is shown below:

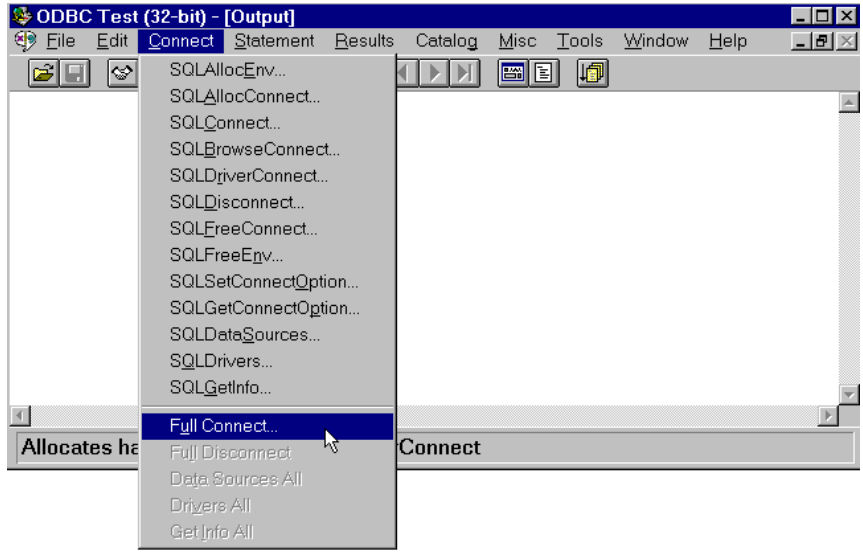
SQLAllocConnect(henv004993F0, phdbc00483E0C);
SQLConnect(hdbc00483E0C, "dBaseDB1", -3, "", -3, "", -3);
SQLGetInfo(hdbc00483E0C, 11, rgbInfoValue, 4, pcbInfoValue);
SQLGetInfo(hdbc00483E0C, 21, rgbInfoValue, 4, pcbInfoValue);
SQLAllocStmt(hdbc00483E0C, phstmt00488110);

Using the ODBC Test Tool

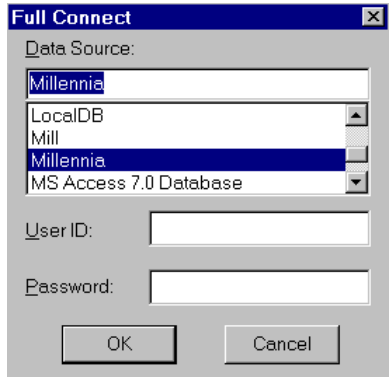
If you are using the Microsoft ODBC driver SDK, you can use the ODBC Test tool which allows you to issue ODBC API function calls. In the following example, we will test whether the connection to the database is configured properly and, if so, try other calls to debug each of the layers.

1. Double-click the following icon to start the ODBC Test program.

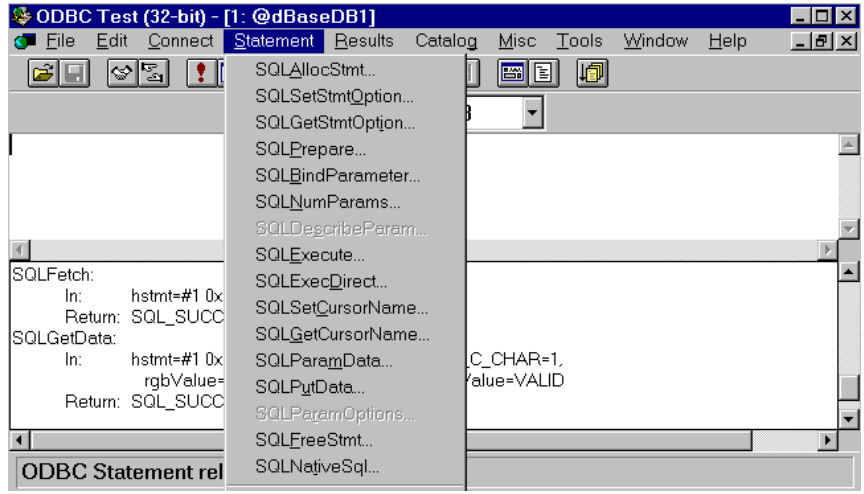
2. Choose Connect - Full Connect from the menu bar.



3. You are prompted for the connection specifics. From the list box, choose the Data Source you want to test the connection to. These are the data sources you specified with the ODBC driver manager. In our example, we chose the Millennia Oracle7 instance data source. If security is enforced, you must specify the User ID and/or a password.



4. You will see the results of the connection. If successful, you can select from the other ODBC API function calls from the menu to test the functionality of the connection and the results of the calls:



LS:DO Class Library

The classes that make up the LotusScript:Data Object provide you with the following benefits:

- **Connection sharing**

  Connections are cached to avoid the added overhead of establishing a connection. In addition, since it is defined as an independent object, one connection object can be used by multiple LotusScript SQL calls.

- **Multiple query and result sets**

  You can define multiple query objects to generate multiple result set objects which can all be executed against the same connection, and manipulated from the same script.

- **Bi-directional scrolling over result set**

  The ODBCResultSet object provides a scrolling cursor with methods for navigating to the next, previous, first, and last rows.

- **Result set search**

  The LocateRow method of the ODBCResultSet object provides the ability to search for specific rows within the result set based on specified criteria. This search capability executes faster than multiple queries or comparing values from multiple rows in LotusScript.

- **Cached results**

  The query result in the ODBCResultSet object is optionally cached in memory (default setting), so it can be accessed later by other events in the form, increasing performance and reducing DBMS connection time. In addition, the cached result set gives you the ability to locate records using LocateRow later.

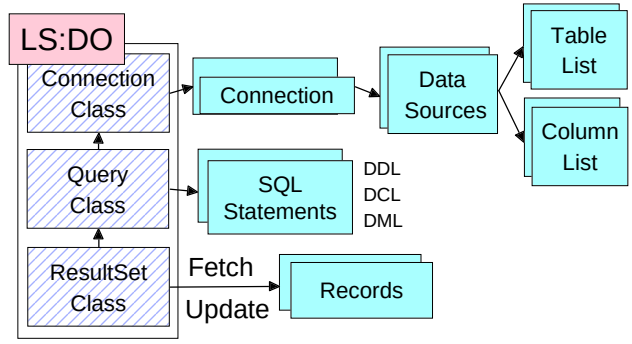
- **Update services**

  Updates to back-end DBMSs through a generic ODBC interface are limited to SQL statements, where the user must ensure that the row to be updated contains a unique record reference or can be otherwise uniquely accessed through a cursor. LS:DO extends this capability by permitting individual items in a result set to be modified without use of an SQL statement using the SetValue method. These changes are then updated to the back end database all at once.

- **Driver transparency**

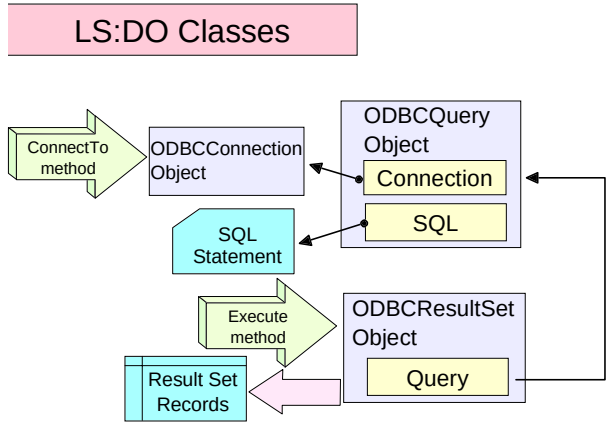
  Although different vendors' ODBC drivers support varying conformance levels, the LotusScript:Data Object assesses these differences and often provides the same level of behavior across all drivers and databases. The developer does not have to write separate scripts for separate drivers.

The following figure represents the manner in which a LotusScript program would use each class in an application access in a database:



Relationship Among Classes

The three classes in the LS:DO are tightly related to one another as shown in the following diagram:



Event Handling

If needed, you can create event-handling subroutines for some ODBC methods. An event-handling subroutine you create is called according to the behavior of an appropriate ODBC method, after the On Event statement is issued.

In the following example, an event handler, named presub1 is called, before the ListDataSources method is called.

1. On Event statement

```
Dim connection As New ODBCConnection
On Event BeforeListDataSources From connection Call presub1
```

2. Event handler

```
Sub presub1(Source As ODBCConnection)
     '** Write your event handling script here
End Sub
```

Note Your event handler must be in the scope where the event occurs.

ODBCConnection Class

The ODBCConnection class allows you to establish a connection. It also allows you to access some database catalog information, such as data source lists, table lists, procedures lists, and so on.

Property
The following table shows the properties of the ODBCConnection Class:

Property	*Data Type*	*Read/Write*	*Argument*
AutoCommit	Boolean	R/W	
CommitOnDisconnect	Boolean	R/W	
DataSourceName	String	R	
IsConnected	Boolean	R	
IsSupported(option)	Boolean	R	option: DB_SUPP_ASYNCHRONOUS DB_SUPP_CURSORS DB_SUPP_PROCEDURES DB_SUPP_READONLY DB_SUPP_SILENTMODE DB_SUPP_TRANSACTIONS
SilentMode	Boolean	R/W	

Note Boolean is not a predefined data type in LotusScript. However, you can use a constant value (TRUE and FALSE) as a Boolean data type.

The following table shows the ODBCConnection methods with the corresponding arguments and events.

Method	*Argument*	*Return Value Type*	*Error Constant*	*Event*
CommitTransaction		Boolean	DBstsNCON	BeforeCommit Transaction AfterCommit Transaction
ConnectTo(source$ [, userID$, password$])		Boolean	DBstsCANF DBstsSVRQ DBstsCCON DBstsACCS	BeforeConnect AfterConnect BeforeConnectTo AfterConnectTo
Disconnect		Boolean	DBstsNCON	BeforeDisconnect AfterDisconnect TransactionUpdate

continued

Method	*Argument*	*Return Value Type*	*Error Constant*	*Event*
GetError		Constant *1		
GetErrorMessage([error%])	error%: DB_LAST ERROR or Constants *1	String		
GetExtendedError Message([error%])	error%: DB_LAST ERROR or Constants *1	String		
ListDataSources		Array of String		BeforeListData Sources AfterListData Sources
ListFields([tableName$])		Array of String	DBstsNCON DBstsNCOL	BeforeListFields AfterListFields
ListProcedures([source$ [, userID$, password$])		Array of String	DBstsNCON DBstsACCS	BeforeList Procedures AfterList Procedures
ListTables([source$ [, userID$, password$]])		Array of String	DBstsNCON DBstsACCS	BeforeListTables AfterListTables
RollbackTransaction		Boolean	DBstsNCON	BeforeRollback Transaction AfterRollback Transaction

*1 Error number list is shown in the ODBCResultSet Class section, later in this chapter.

Note Most of the pre-defined data types in LotusScript are represented by the following suffix types:

Data Type	*Suffix*
Currency	@
Double	#
Integer	%
Long	&
Single	!
String	$

Sample Uses of the ODBCConnection Class:

1. To get a data source list registered by the ODBC administrator, use the ListDataSources method of the ODBCConnection class.

```
Dim con As New ODBCConnection
Dim dl As Variant
dl = con.ListDataSources
'** Keywords is a field name in which a data source list is
saved
keyDoc.Keywords = dl
```

2. To get a table list owned by a database, use the ListTables method of the ODBCConnection class.

```
Dim con As New ODBCConnection
Dim tl As Variant
'** sampleDB1 is a database name registered in this example
tl = con.ListTables("sampleDB1")
'** Keywords is a field name in which a table list is saved
keyDoc.Keywords = tl
```

   Note When the ListTables method is issued, the SQLConnect ODBC API is called in LS:DO before getting a table list; so you don't need to execute the ConnectTo method.

3. To get a column name list owned by a table, use the ListFields method of the ODBCConnection class.

```
Dim con As New ODBCConnection
Dim Clist As Variant
Dim status As Variant
'** sampleDB1 is a database name registered in this example
status = con.ConnectTo("sampleDB1")
'** courses is a table name in the sampleDB1 database
CList = con.ListFields("courses")
'** Keywords is a field name in which a column list is saved
keyDoc.Keywords = Clist
```

ODBCQuery Class

The ODBCQuery class is used to hold the ODBCConnection object in which a connection is established, and to hold a SQL statement you want to use to perform the inquiry. The SQL statement is parsed through the ODBC driver which your application requires.

Property
The properties of this class are shown below:

Property	*Data Type*	*Read/Write*
Connection	ODBCConnection Object	W
QueryExecuteTimeOut	Integer	R/W
SQL	String	R/W
UseRowID	Boolean	R/W

The ODBCQuery class provides the following methods:

Method	*Argument*	*Return Value*
GetError		Constant
GetErrorMessage([error%])	error%: DB_LASTERROR or Constants *1	String
GetExtendedErrorMessage ([error%])	error%: DB_LASTERROR or Constants *1	String

*1 Error number list is shown in the ODBCResultSet Class section, later in this chapter.

Sample Uses of the ODBCQuery Class:
This sample shows the execution of a SQL statement using the Execute method in ODBCResultSet. The following steps are performed before carrying out the Execute method. The Connection method and the SQL property in ODBCResultSet class are also used in this example.

```
Dim con As New ODBCConnection
Dim qry As New ODBCQuery
Dim res As New ODBCResultSet
Dim status As Variant
'** sampleDB1 is a database name registered in this example
status = con.ConnectTo("sampleDB1")
Set qry.Connection = con
'** courses is a table name in the sampleDB1 database
qry.SQL = "select * from courses"
Set res.Query  = qry
```

ODBCResultSet Class
The ODBCResultSet class has many functions that are used to handle records which are termed result sets. A result set holds the retrieved records of a SQL query which is specified with the ODBCQuery object.

Property
The following table shows the properties available with the ODBCResultSet Class:

Property	*Data Type*	*Read/Write*
CacheLimit	Integer	R/W
CurrentRow	Integer	R/W
FetchBatchSize	Integer	R/W
HasRowChanged	Boolean	R
IsBeginOfData	Boolean	R
IsEndOfData	Boolean	R
IsResultSetAvailable	Boolean	R
MaxRows	Integer	R/W
NumColumns	Integer	R
NumRows	Integer	R
Query	ODBCQuery Object	W
ReadOnly	Boolean	R/W

Methods
The methods of the ODBCResultSet class can be categorized into the following areas:

- **SQL execution and transaction**: these methods are used to issue an SQL statement and to commit or roll back a transaction.

Method	*Argument*	*Return Value Type*	*Error Constant*	*Event*
Close(option)	Option: DB_CLOSE DB_COMMIT DB_ROLLBACK	Boolean		BeforeClose AfterClose
ExecProcedure (name$, arg$)		Boolean	DBstsNCON DBstsODBC	BeforeExec Procedure AfterExecProcedure
Execute([option])	Option: DB_CANCEL	Boolean	DBstsODBC	BeforeExecute AfterExecute AsynchOperation Complete

continued

Method	*Argument*	*Return Value Type*	*Error Constant*	*Event*
Transactions (option)	Option: DB_COMMIT DB_ROLLBACK	Boolean		BeforeTransactions AfterTransactions

- **Result set row navigation and location**: these methods are used to locate a cursor on a result set which is produced by the Execute method.

Method	*Argument*	*Return Value Type*	*Error Constant*	*Event*
FirstRow		Boolean	DBstsINVR	BeforeFirstRow AfterFirstRow BeforeRowPositionChange AfterRowPositionChange
LastRow		Boolean		BeforeLastRow AfterLastRow BeforeRowPositionChange AfterRowPositionChange
LocateRow(column, value$ [, column, value$, ...])	column is Integer or String	Boolean	DBstsCARR DBstsEOFD DBstsNODA	BeforeLocateRow AfterLocateRow BeforeRowPositionChange AfterRowPositionChange
NextRow		Boolean	DBstsINVR DBstsEOFD	BeforeLocateRow AfterLocateRow BeforeRowPositionChange AfterRowPositionChange
PrevRow		Boolean	DBstsINVR DBstsCARR	BeforePrevRow AfterPrevRow BeforeRowPositionChange AfterRowPositionChange

- **Accessing column values**: these methods are used to access specific column values and to check column properties.

Method	*Argument*	*Return Value Type*	*Error Constant*	*Event*
GetValue(column [, variable])	column is Integer or String	Variant	DBstsINVC DBstsNODA DBstsCNVR	BeforeGetValue AfterGetValue
IsValueAltered(column)	column is Integer or String	Boolean	DBstsINVC	
IsValueNull(column)	column is Integer or String	Boolean	DBstsINVC	
SetValue(column, value)	column is Integer or String	Boolean	DBstsRDON DBstsRDEL DBstsINVC DBstsCNVR DBstsNODA	AfterSetValue BeforeSetValue

- **Result set row modification operations**: these methods enable you to dynamically add and delete rows from within the result set. Furthermore, you can retrieve the row status and you can update the altered result set in the database.

Method	*Return Value Type*	*Error Constant*	*Event*
AddRow	Boolean	DBstsAHVR DBstsRDON DBstsNOEX	BeforeAddRow AfterAddRow
DeleteRow (tableName$)	Boolean	DBstsINVR DBstsNUNQ DBstsRCHG DBstsRDON	BeforeDeleteRow AfterDeleteRow RowContentsChanged TransactionsPending
GetRowStatus	DB_UNCHANGED DB_ALTERED DB_UPDATED DB_DELETED DB_NEWROW	DBstsNODA	
UpdateRow	Boolean	DBstsRDON DBstsRDEL DBstsCXIN DBstsNUNQ DBstsRCHG DBstsRUNC DBstsUPDB	BeforeUpdateRow AfterUpdateRow TransactionsPending RowContentsChanged

- **Column attributes operations**: these methods allow you to access information about the column attributes.

Method	*Argument*	*Return Value Type*	*Error Constant*
FieldExpectedDataType (column [, dataType])	column is Integer or String. dataType: DB_TYPEUNDEFINED DB_CHAR DB_SHORT DB_LONG DB_DOUBLE DB_DATE DB_TIME DB_BINARY DB_BOOL DB_DATETIME	DB_TYPEUNDEFINED DB_CHAR DB_SHORT DB_LONG DB_DOUBLE DB_DATE DB_TIME DB_BINARY DB_BOOL DB_DATETIME	DBstsINVC
FieldID(columnName$)		Integer	DBstsINVC
FieldInfo(column)	column is Integer or String	Array with elements *1	DBstsINVC
FieldNativeDataType (columnID%)		String *2	DBstsINVC
FieldID(column)	column is Integer or String	Constant	DBstsINVC
FieldName(column)	column is Integer or String	Integer	DBstsINVC

- **SQL parameter operations**: these methods are used to define new SQL parameters and to retrieve the values of those already existing.

Method	*Argument*	*Return Value Type*	*Event*
GetParameter(parameter)	parameter is Integer or String	Variant	BeforeGetParameter AfterGetParameter
GetParameterName (parameterID%)		String	BeforeGetParameterName AfterGetParameterName
NumParameters		Integer	
SetParameter(parameter, value$)	parameter is Integer or String	Boolean	BeforeSetParameter AfterSetParameter

Error Operations
These methods are used to deal with error messages.

Method	*Argument*	*Return Value Type*
GetError		Constant *3
GetErrorMessage([error%])	error%: DB_LASTERROR or Constants *3	String
GetExtendedErrorMessage([error%])	error%: DB_LASTERROR or Constants *3	String

The following three tables contains information referenced in the above ODBCResultSet class tables about:

- Return Value Constants of the FieldInfo Method
- Return Value Constants of the FieldNativeDataType Method
- Error Constants

Reference *1 - Return Value Constants of the FieldInfo Method

DB_INFO_AUTOINCREMENT	DB_INFO_NULLABLE
DB_INFO_CASESENSITIVE	DB_INFO_PRECISION
DB_INFO_COLUMNID	DB_INFO_READONLY
DB_INFO_COLUMNNAME	DB_INFO_SCALE
DB_INFO_COMPUTED	DB_INFO_SEARCHABLE
DB_INFO_DISPLAYSIZE	DB_INFO_SETTABLE
DB_INFO_EXPECTED_DATATYPE	DB_INFO_SQLDATATYPE
DB_INFO_LENGTH	DB_INFO_TABLENAME
DB_INFO_MONEY	DB_INFO_UNSIGNED
DB_INFO_NATIVE_DATATYPE	

Reference *2 - Return Value Constants of the FieldNativeDataType Method

SQL_CHAR	SQL_FLOAT	SQL_TIMESTAMP	SQL_LONGVARBINARY
SQL_NUMERIC	SQL_REAL	SQL_VARCHAR	SQL_BIGINT
SQL_DECIMAL	SQL_DOUBLE	SQL_BINARY	SQL_TINYINT
SQL_INTEGER	SQL_DATE	SQL_VARBINARY	SQL_BIT
SQL_SMALLINT	SQL_TIME	SQL_LONGVARCHAR	

Reference *3 - Error Constants

DBstsSUCCESS	DBstsINVC	DBstsDSTY	DBstsTMPL	DBstsRDON
DBstsFAIL	DBstsNCOL	DBstsDRVN	DBstsBROW	DBstsRCHG
DBstsMEMF	DBstsBADP	DBstsFITY	DBstsCANF	DBstsRUNC
DBstsNCON	DBstsODBC	DBstsFILT	DBstsCNVR	DBstsCXIN
DBstsCCON	DBstsLIBM	DBstsINST	DBstsCNVD	DBstsAHVR
DBstsNOEX	DBstsSNFD	DBstsNODR	DBstsHSTMT	DBstsCPAR
DBstsINVR	DBstsINTR	DBstsNAUT	DBstsSQLP	DBstsNIRC
DBstsCARR	DBstsACCS	DBstsNOSV	DBstsINTE	DBstsRDEL
DBstsNODA	DBstsTYPE	DBstsNAPE	DBstsUPDB	
DBstsEOFD	DBstsENTR	DBstsSVRQ	DBstsNUNQ	

Sample Programs Using the ODBCResultSet Class

In this example we get a column value list.

```
Dim con As New ODBCConnection
Dim qry As New ODBCQuery
Dim res As New ODBCResultSet
Dim status As Variant
'** sampleDB1 is a database name registered in this
example
status = con.ConnectTo("sampleDB1")
Set qry.Connection = con
'** courses is a table name in the sampleDB1 database
qry.SQL = "select * from courses"
Set res.Query  = qry
Call res.execute
Dim num As Integer
num = 0
Dim vl As Variant
Redim vl(num)
Do
'** name is a column name in the courses table
```

```
        vl(num) = res.GetValue("name")
        num = num + 1
        Redim Preserve vl(num)
Loop While res.NextRow
'** CValue is a field name in which a value list is saved
gDoc.CValue = vl
```

Server Side Processing for Web Applications

When you need to access external databases via a Web browser and the Domino server you can use the LS:DO.

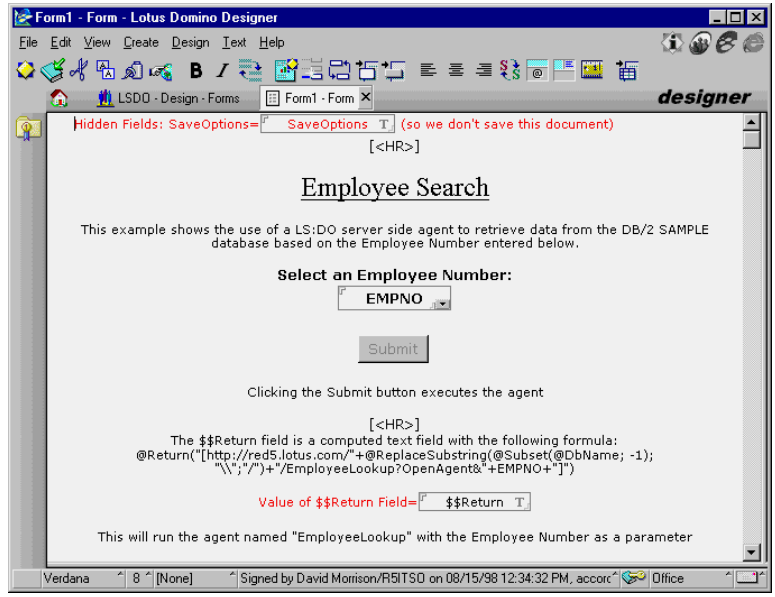
The principle of accessing the data remains the same; you must define an ODBC connection to your data source and you must write the LS:DO code to access the database in the same way. However, the LS:DO LotusScript code needs to be placed in an agent that is run via a URL command and the display of the data needs to be formatted in HTML.

To better explain how you can use the LS:DO within your Web application, we will use an example that queries data from the DB/2 SAMPLE database and displays it in a Web page. We will also demonstrate how to drill down further into the database with subsequent queries. In our example we use information from the EMPLOYEE table to display information on employees based on their employee number.

1. We assume that you have already set up an ODBC connect using the ODBC manager to the EMPLOYEE table within the DB/2 SAMPLE database. To set up an ODBC connection, see the example earlier in this chapter.
2. Create a new form in your database and add the following fields and buttons:

Field Name	*Field Type*	*Field Code*
SaveOptions	Computed Text	0 (Zero)
EMPNO	Editable Text	
$$Return	Computed Text	@Return("[http://yourservernameinhere/" + @ReplaceSubstring(@Subset(@DbName;1);"\\";"/") + "/EmployeeLookup?OpenAgent&"+EMPNO+"]")
Submit	Button	None

3. Below is an example of the new form in Domino Designer.



4. Create an agent and name it EmployeeLookup, check the Shared check box, select Manually From Agent List in the When Should this Agent Run field, select All Documents in Database from the Which document(s) should this agent run on field, select LotusScript from the Run field and add the following code:

   Click Options and enter:

```
USELSX "*LSXODBC"
```

   Click Initialize and enter:

```
Sub Initialize
  Set session = New NotesSession
  Set conn = New ODBCConnection
  Set query = New ODBCQuery
  Set data = New ODBCResultSet
  Set query.connection = conn
  Set data.query = query
  Set doc = session.DocumentContext
  Set db = Session.CurrentDatabase

  conn.SilentMode = True
  USERNAME$ = "DB2Admin"
  PASSWORD$ = "password"

  URLString = doc.Query_String(0)
```

```
  URLLength = Len(URLString)
  ParamPosition = Instr(URLString, "&") + 1
  WebParam = Mid(URLString, ParamPosition,
URLLength-ParamPostion)
     'Print WebParam

  If Not conn.ConnectTo("SAMPLE", USERNAME$, PASSWORD$)
Then
        Print "Not OK, Could not Connect!"
        error% = Conn.GetError
        message$ = Conn.GetErrorMessage(error%)
        extendedMessage$
Conn.GetExtendedErrorMessage(error%)
        Print message$ & "<br>"
        Print "Error Code: " & Str$(error%)
        Print "Extended Error: " & ExtendedMessage$ & "<HR>"
        Exit Sub
  End If
  query.SQL = "SELECT * FROM EMPLOYEE WHERE EMPNO='" &
WebParam & "'"
  If Not data.Execute Then
        Print data.GetExtendedErrorMessage,,
data.GetErrorMessage
        Exit Sub
  End If
  Do
        data.NextRow
        FirstName = data.GetValue("FIRSTNME", firstName)
        LastName = data.GetValue("LASTNAME", lastName)
        MidInit = data.GetValue("MIDINIT", MidInit)
        WorkDept = data.GetValue("WORKDEPT", WorkDept)
        PHONENO = data.GetValue("PHONENO", PhoneNo)
        HIREDATE = data.GetValue("HIREDATE", HireDate)
        JOB = data.GetValue("JOB", Job)
        EDLEVEL = data.GetValue("EDLEVEL", EdLevel)
        SEX = data.GetValue("SEX", Sex)\
        BIRTHDATE = data.GetValue("BIRTHDATE", BirthDate)
        SALARY = data.GetValue("SALARY", Salary)
        BONUS = data.GetValue("BONUS", Bonus)
        COMM = data.GetValue("COMM", Comm)

        Print "<HEAD><BODY>"
        Print "<H3>This is the information for employee: " &
WebParam & "</H3>"

        Print "First Name: " & firstName & "<BR>"
        Print "Initials: " & MidInit & "<BR>"
        Print "Last Name: " & lastName & "<BR>"
        Print "<BR>"
```

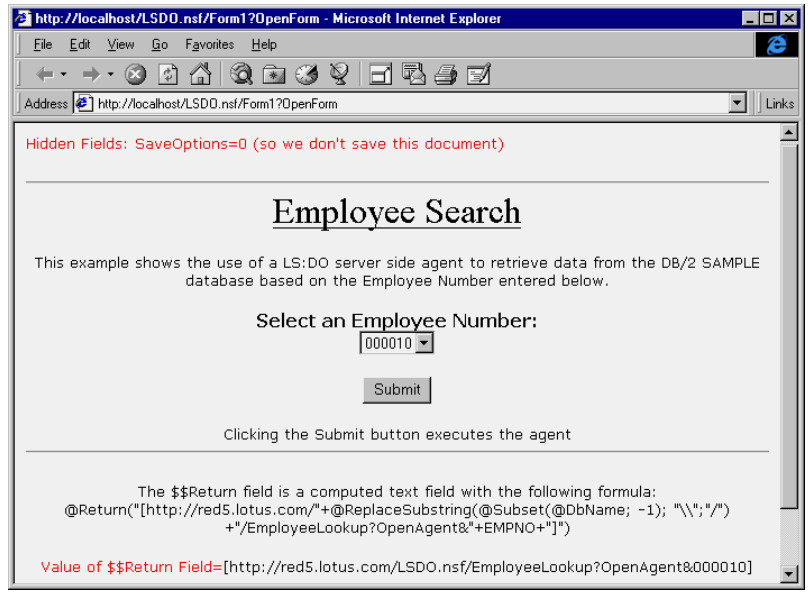
```
        Print "Work Department: <a
href=./DeptLookup?OpenAgent&" & workdept & ">" & workdept &
"</a>" & "<BR>"
        Print "Phone Number: " & PhoneNo & "<BR>"
        Print "Hire Date: "& HireDate & "<BR>"
        Print "Job: " & Job & "<BR>"
        Print "Education Level: "& EdLevel & "<BR>"
        Print "<BR>"
        Print "<BR>"
        Print "Thank-You"
  Loop Until data.IsEndOfData

  data.Close(DB_CLOSE)
  conn.Disconnect
End Sub
```

5. Open your Web browser and compose a new document using the form you created. For our server, the URL was:

   `http://red5.lotus.com/LSDO.nsf/Form1?open`

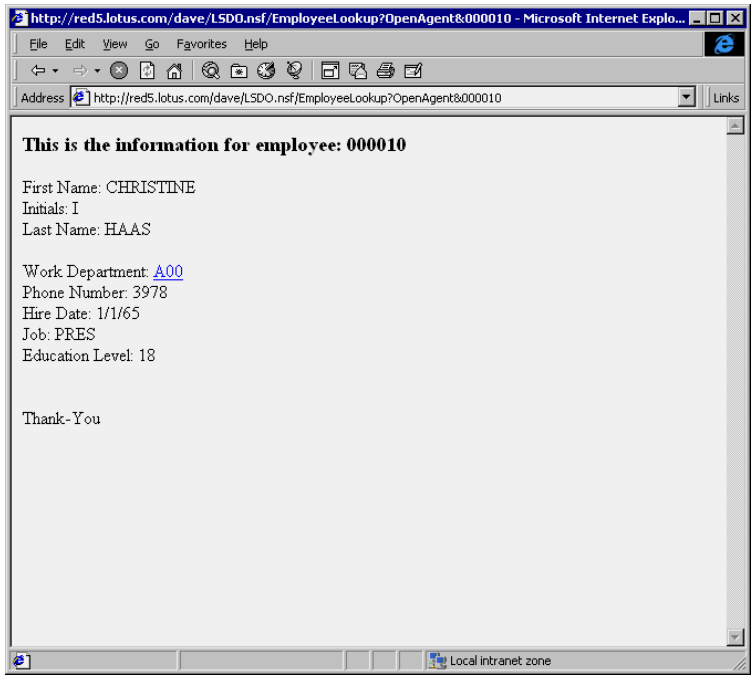
You should see the following in your browser:



Note Some of the hidden fields are actually displayed on this screen to show you their computed values.

6. Enter an employee number, some valid numbers in the EMPLOYEE table of the SAMPLE database are 000010, 000020, 000030, 000050. Click the Submit button on the form.

7. After a few seconds you will see the following on the screen. (We chose employee number 000010.)



How Does This Work?

If we take a closer look at the steps we took to create this example, the first part of the process is triggered when the user clicks the Submit button from their Web browser. Since we have a $$Return field in our form, the value of this is evaluated and then executed on the Domino server.

When the user selects an employee number the $$Return field evaluates to the following URL, which is processed by the Domino Web server and runs the EmployeeLookup agent with a parameter of 000010:

```
http://red5.lotus.com/LSDO.nsf/EmployeeLookup?OpenAgent&00010
```

As the agent is initiated, one of the first things it does is parse the command line that was passed to it via the DocumentContext method of the NotesSession which gives us access to the CGI variable URLString. The following code extract shows this:

```
' Get a handle to the current doc in memory
Set doc = session.DocumentContext
' Retrieve the CGI variable URLString
URLString = doc.Query_String(0)
' URLString should equal ?OpenAgent&000010
```

```
' Calculate how long the string is
URLLength = Len(URLString)
' Find the position of the & in the string
ParamPosition = Instr(URLString, "&") + 1
' Now we can extract the employee number from the string
WebParam = Mid(URLString, ParamPosition,
URLLength-ParamPostion)
' If we need to we can send the parameter to the Web browser
' to check it.
'Print WebParam
```

Now that we know the employee number we can use the LS:DO to query the EMPLOYEE table in the SAMPLE database and extract all the information we require. This is done with the following SQL query:

```
SELECT * FROM EMPLOYEE WHERE EMPNO='" & WebParam & "'"
```

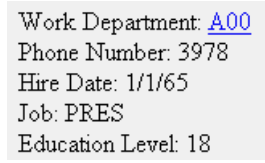
The final thing to do is output the information onto the Web browser. This is done using the LotusScript Print command and a combination of HTML tags and the variables we assigned.

Digging Deeper

If you look back at the EmployeeLookup agent code we created earlier, you will see a line that looks like this:

```
Print "Work Department: <a href=./DeptLookup?OpenAgent&" &
workdept & ">" & workdept & "</a>" & "<BR>"
```

This line of code creates an HTML link to another agent in the database called DeptLookup. If you look carefully at the figure below, you will see how this looks.



Notice that the line beginning with Work Department displays its value as a URL. Clicking this hotspot will run the agent DeptLookup on the Domino server with a parameter of A01. The code for this agent is very similar to the EmployeeLookup agent but this time it retrieves a list of all employees that work in the same department.

The code for the DeptLookup Agent is shown below:

```
Sub Initialize
      Set session = New NotesSession
      Set conn = New ODBCConnection
      Set query = New ODBCQuery
```

```
      Set data = New ODBCResultSet
      Set query.connection = conn
      Set data.query = query
      Set doc = session.DocumentContext
      Set db = Session.CurrentDatabase

      conn.SilentMode = True
      USERNAME$ = "DB2Admin"
      PASSWORD$ = "password"

      URLString = doc.Query_String(0)

      URLLength = Len(URLString)
      ParamPosition = Instr(URLString, "&") + 1
      WebParam = Mid(URLString, ParamPosition,
URLLength-ParamPosition)
     'Print WebParam

      If Not conn.ConnectTo("SAMPLE", USERNAME$, PASSWORD$)
Then
            Print "Not OK, Could not Connect!"
            error% = Conn.GetError
            message$ = Conn.GetErrorMessage(error%)
            extendedMessage$ =
Conn.GetExtendedErrorMessage(error%)
            Print message$ & "<br>"
            Print "Error Code: " & Str$(error%)
            Print "Extended Error: " & ExtendedMessage$ & "<HR>"
            Exit Sub
      End If
      query.SQL = "SELECT * FROM EMPLOYEE WHERE WORKDEPT='" &
WebParam & "'"
      If Not data.Execute Then
            Print data.GetExtendedErrorMessage,,
data.GetErrorMessage
            Exit Sub
      End If
      Print "<HEAD><BODY>"
      Print "<H3>These are other employees that work in
department " & WebParam & "</H3>"
      Print "<TABLE border="1">"
      Print "<TR>"
      Print "<TD>FirstName</TD>"
      Print "<TD>Init</TD>"
      Print "<TD>LastName</TD>"
      Print "<TD>Department</TD>"
      Print "<TD>Phone</TD>"
      Print "<TD>HireDate</TD>"
      Print "<TD>Job </TD>"
      Print "<TD>EdLevel</TD>"
      Print "<TR>"
```

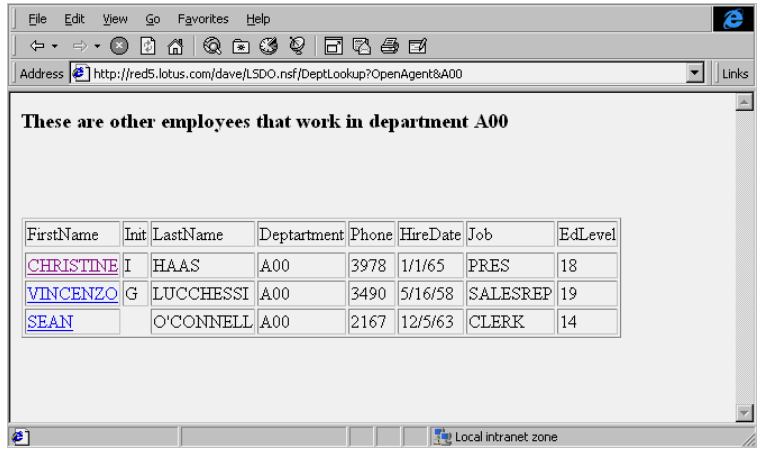
```
      Do
            data.NextRow
            EmpNo = data.GetValue("EMPNO", Empno)
            FirstName = data.GetValue("FIRSTNME", firstName)
            LastName = data.GetValue("LASTNAME", lastName)
            MidInit = data.GetValue("MIDINIT", MidInit)
            WorkDept = data.GetValue("WORKDEPT", WorkDept)
            PHONENO = data.GetValue("PHONENO", PhoneNo)
            HIREDATE = data.GetValue("HIREDATE", HireDate)
            JOB = data.GetValue("JOB", Job)
            EDLEVEL = data.GetValue("EDLEVEL", EdLevel)
            SEX = data.GetValue("SEX", Sex)
            BIRTHDATE = data.GetValue("BIRTHDATE", BirthDate)
            SALARY = data.GetValue("SALARY", Salary)
            BONUS = data.GetValue("BONUS", Bonus)
            COMM = data.GetValue("COMM", Comm)
            Print "<TR>"
            Print "<TD><a href=./EmployeeLookup?OpenAgent&" &
EmpNo & ">" & firstName & "</a>" & "</TR>"
            Print "<TD>" & MidInit  & "</TR>"
            Print "<TD>" & lastName  & "</TR>"
            Print "<TD>" & workdept   & "</TR>"
            Print "<TD>" & PhoneNo  & "</TR>"
            Print "<TD>" & HireDate  & "</TR>"
            Print "<TD>" & Job  & "</TR>"
            Print "<TD>" & EdLevel  & "</TR>"
            Print "</TR>"
            Print "<BR>"
      Loop Until data.IsEndOfData
      Print "</TABLE>"
      Print "</BODY></HEAD>"

      data.Close(DB_CLOSE)
      conn.Disconnect
End Sub
```

Notice that this agent formats the output using an HTML table. This is particularly good for displaying tabular information back to a Web browser and is fairly simple once you understand the HTML tags.

HTML Tag	*Description*
<TABLE>	Defines the start of a new table
<TR>	Defines the start of a new row
<TD>	Defines the start of a new column
</TD>	Ends the current column
</TR>	Ends the current row
</TABLE>	Ends the table

Below is a figure of how the form looks when displayed in a Web browser:



Again we have added a small piece of code in this agent that allows the user to retrieve more information by clicking a name in the table and running the EmployeeLookup agent again. This is achieved with the following line of code:

```
Print "<TD><a href=./EmployeeLookup?OpenAgent&" & EmpNo & ">"
& firstName & "</a>" & "</TR>"
```

With a little imagination it is possible to give your employees or customers real-time access to your company's relational databases and the capabilities to drill down through the information from a Web browser.

Running Multiple Instances of an Agent

When Domino is being used as a Web server to access and display data from external sources via the LS:DO and LotusScript agents, you need to add a line into the NOTES.INI file on the Domino server.

```
DominoAsynchonizeAgents=1
```

This enables an agent to be run by more than one person at the same time. By default the Domino server only runs one copy of an agent at a time and queues other requests.

Using @DB Functions to Access Other Databases Through ODBC

@DBCommand, @DBLookup and @DBColumn are Notes functions that enable you to access RDBMSs which use the underlying ODBC interface. The @DB formulas are read-only.

The basic purpose of these functions is to create value lists for keyword fields. @DBLookup and @DBColumn can be used to query a relational database; @DBCommand is only used for executing stored procedures. @DBCommand does not return result sets. If you need a more customized and more complex query, LS:DO is a better option.

When to Use

Lotus Notes provides fast and easy-to-use read access to ODBC-compliant DBMSs via @DB functions. Notes @DB functions give developers the power of three frequently-used query tasks:

- **Generating Keyword Lists**

  The @DBColumn function in the Notes formula language generates Notes keyword lists from internal, as well as external, data sources. The same function supports keyword value lookups in tables stored in a DBMS through ODBC. For example, a Notes @DBColumn field formula can present a keyword list of customer names stored in a DBMS table when composing a document in a Notes customer contact tracking database.

- **Performing Lookup Operations**

  The @DBLookup function looks up a value in one field based on the value of a related field. For example, it will look up a customer phone number in a DBMS when given a customer name in Notes. Like @DBColumn, @DBLookup works both with other Notes databases and with external data sources through ODBC. The @DBColumn and @DBLookup functions can be used in other Notes formula contexts as well, such as input validation or translation formulas.

- **Launching External DBMS Stored Procedures**

  Database procedures and insert statements can be triggered with the @DBCommand function.

Note Some of these functions inherently involve a delay before they complete; so in order to set user expectations, it is sometimes a good idea to code the functions behind a button so that the user expects some delay before the function is completed.

How to Use @DB Functions

The @DB functions are summarized in the following table:

Functions	*Descriptions*	*Equivalent SQL*
@DBColumn	Generates a keyword list. Returns a specified column for all rows in the specified table.	SELECT DISTINCT column_name FROM table_name
@DBLookup	Performs a lookup. Returns a specified column value in the row that matches the specified condition.	SELECT column FROM table WHERE condition
@DBCommand	Triggers stored procedures in the external database.	(any SQL statement)

@DBColumn

The @DBColumn syntax is:

```
@DBColumn( "ODBC": Cache ; DataSource  ; UserID1 : UserID2 ;
Password1 : Password2 ; TableName ; ColumnName : NullHandling ;
Distinct : Sort )
```

Parameters:

@DBColumn("ODBC":	*Description*	*Choice*	*Optional*
Cache,	Inquiry Cache	"Cache" (Default) "NoCache"	X
DataSource,	Database resource name		
UserID1:UserID2,	User IDs		X
Password1:Password2,	Passwords		X
TableName,	Table Name		
ColumnName:	Column Name		
NULLHandling,	Null Handling	"Fail" "Discard" (Default) "ReplacementValue"	X
Distinct:	Remove duplicate values	"Distinct"	X
Sort)	Sort Direction	"Ascending" "Descending"	X

@DBLookup

The @DBLookup syntax is:

```
@DBLookup( "ODBC": Cache ; DataSource  ; UserID1 : UserID2 ;
Password1 : Password2 ; TableName ; ColumnName :
NullHandling ; KeyColumn ; Key ; Distinct : Sort )
```

Parameters:

@DBLookup("ODBC":	*Description*	*Choice*	*Optional*
Cache,	Inquiry Cache	"Cache" (Default) "NoCache"	X
DataSource,	Database resource name		
UserID1:UserID2,	User IDs		X
Password1:Password2,	Passwords		X
TableName,	Table Name		
ColumnName:	Column Name		
NULLHandling,	Null Handling	"Fail" "Discard" (Default) "ReplacementValue"	X
KeyColumn,	Column Name to be looked into		
Key,	Search String in KeyColumn		
Distinct:	Remove duplicate values	"Distinct"	X
Sort)	Sort Direction	"Ascending" "Descending"	X

@DBCommand

The @DBCommand syntax is:

```
@DBCommand( "ODBC": Cache ; DataSource  ; UserID1 : UserID2 ;
Password1 : Password2 ; SQL ;  NullHandling )
```

Parameters:

@DBCommand("ODBC":	*Description*	*Choice*	*Optional*
Cache,	Inquiry Cache	"Cache" (Default) "NoCache"	X
DataSource,	Database resource name		
UserID1:UserID2,	User IDs		X
Password1:Password2,	Passwords		X
SQL	SQL Statement		
NULLHandling)	Null Handling	"Fail" "Discard" (Default) "ReplacementValue"	X

Summary

This chapter covered NotesSQL, the Domino JDBC driver and LotusScript Data Object (LS:DO). We discussed the circumstances where each method should be applied and also gave some examples of their use.

Chapter 15
Domino Global WorkBench

Domino Global WorkBench is a set of software tools that helps you manage the localization (translation) of the design elements in Domino databases, especially Web site databases. It also includes synchronization features that help you manage the content of localized databases across languages. Also, when the design of a database changes, you can use the update features of Domino Global WorkBench to transmit the changes easily through to the localized versions.

Domino Global WorkBench is part of Domino Designer but has its own installation program.

Who Benefits from Domino Global WorkBench

Typical users of applications prepared by Domino Global WorkBench are companies that want to reach customers in several countries through the World Wide Web and companies with offices in several countries that need localized intranet applications.

What is Localization?

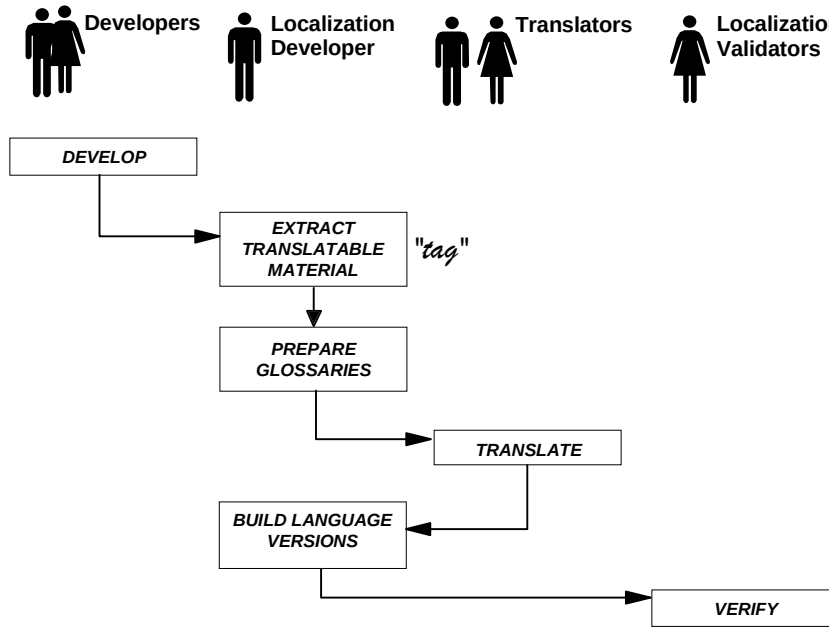
"Localizing" a database (.NSF file) or database template (.NTF file) means creating a version of it in another language. In general, localization involves:

1. Analyzing and sometimes modifying the design to make the remaining localization tasks simpler and identifying the text that is not to be translated (for reasons of functionality).
2. Translating the text in the design elements. All text seen by users is translated, as well as some text that is not seen by users.
3. Modifying the layout. Translations are often significantly longer than the original text, and this means that certain parts of the design, for example tables and navigators, will usually have to be adjusted.

Which Processes Support Domino Global WorkBench

Domino Global WorkBench is used by the application developer during the development and maintenance process and by the content provider during the ongoing content updating process across languages.

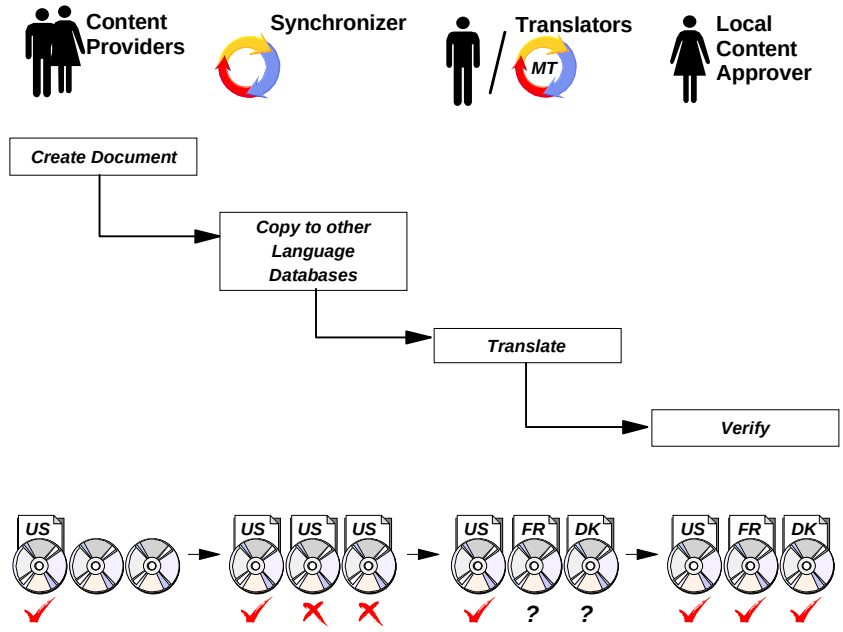
The following figure shows an example of a localization process during development and maintenance and the roles involved in it:



The process in the figure is just a simple example. Domino Global WorkBench does not require a specific process to be followed, but offers full flexibility in integrating with the process you are using for your development and content creation.

Note The people shown in the figure represent different roles. You don't need a huge development organization to use Domino Global WorkBench. One person may cover several roles; for example, a developer may also be the localization developer as well as the translator.

The next figure shows an example of how Domino Global WorkBench can support the creation of content for several languages through its synchronization technique:



For a new document (if it is marked as translatable) the synchronizer creates a copy of it in each of the other languages supported by the application (or site). Using Domino workflow, an automated process can be created where the new language documents are assigned to a translator or to machine translation. After the translation, the workflow process can bring the documents on to be controlled for the correct language and content before making them visible to the users of the application or site. If you create synchronized unilingual databases, you also have the option of including a "language switchbar." This is an element that is added, during building, to any forms that are marked "Translatable." It provides doclinks to other language versions of documents created from the form. Users simply click the language they want to see on the switchbar.

Domino Translation Object

To support translation of content Lotus offers Domino Translation Object 1.02 (DTO) which is included with the Domino Designer and Domino Server 5.0.1.

The DTO provides full access to Machine Translation services using the complete control and flexibility of a structured programming language:

LotusScript and Java. The DTO consists of 2 classes for use with LotusScript — NotesTransObj and TransObj — that come complete with a powerful set of properties and methods to add translation capabilities to Domino based applications, connected to best of breed Machine Translation Engines, via the Alis Translator for Lotus Domino.

To learn more about Domino Translation Object go to the following Web site

`http://www.lotus.com/international`

We will cover Domino Global WorkBench from the developers' perspective of application translation in the following sections of this chapter:

- Concepts, Databases, and Tools in Domino Global WorkBench.

  This section introduces the most important elements of Domino Global WorkBench.

- Localizing an Application.

  This section walks through an example of the process that the localization developer must go through to localize an application.

- Preparing Your Database - Tips for Developers.

  To minimize any rework on the application during the localization process, the designer and developer must focus on the fact that the application is to be translated into several languages from the beginning of the design and development. This section provides tips that make it easier to design and develop translatable Domino applications.

Note Domino Global WorkBench offers very rich functionality. This chapter is not exhaustive. For a detailed description of all the functions and features of the product, refer to the documentation that comes with Domino Global WorkBench.

Concepts, Databases, and Tools in Domino Global WorkBench

This section gives an overview of the different types of databases you encounter when working with Domino Global WorkBench. We will also explain the concept of tagging, how the WorkBench fits into the translation work, and the synchronizer technique.

Domino Global WorkBench Databases

You will encounter several types of databases in Domino Global WorkBench. They are:

Source Database
A source database is the completed application (or part of it) in the original language (the reference language). If the developer updates the application, the source database must be updated by the new version.

Tagged Database
The tagged database is a copy of the source database where all translatable text and pictures have been exported and replaced with unique tags. This database is an interim product. It allows you to generate multiple language databases.

The glossary stores the terms that have been exported from the source database, together with a reference to the tags that were inserted instead. The glossary also contains a set of all the terms for each language to be supported. Translation of the terms can be done using the glossary or the terms can be exported for translation using another tool and then imported again. When building a language version, Domino Global WorkBench will match the tags in the tagged database with the terms for the chosen language in the glossary.

A glossary can also contain already translated terms. Domino Global WorkBench can identify these terms in the source database and tag them with the tags for the existing terms without creating new duplicate terms.

Glossaries can be assigned to a database or a collection of databases. You can also assign more than one glossary per database.

When working with Domino Global WorkBench, the user can choose to have information and error messages written to the report database.

The language database is the output from Domino Global WorkBench. The design elements are taken from the tagged database and the tags are replaced with the translated terms in the chosen language from the glossary. The language database can store multiple language versions in the same database or one database can be created for each language being built.

What is Tagging?

Tagging is the process of replacing a translatable piece of text or image from an original application with a unique identifier which is called a tag. At the same time an entry is created in a Glossary that associates the tag with the text or image it identifies.

When we talk about a piece of text we mean any user visible string that has a meaning by itself, that is, if you translate it, it still makes sense.

New glossary entries are not always created during tagging. The product builder may choose to use existing terms and tags from a glossary when tagging an application.

The WorkBench

The WorkBench is the primary tool for the person acting as localization developer. This is where databases are tagged for translation and where language databases are defined and built.

The WorkBench has different areas (panes) dedicated to the different types of databases being worked with. There is also a common area for design elements in the selected application database where you can select/deselect for processing, display the properties of specific design elements, and so on.

The WorkBench includes panes for:

- Source databases (original application databases)
- Glossary database(s) for the chosen source database
- A tagged database version of the chosen source database
- Language database(s) for the chosen source database

You can optionally also show a log window where messages are written during processing.

The options available in the WorkBench depend on which kind of databases you select. For example, if you select a tagged database, Domino Global WorkBench will display a button to update the corresponding databases. When going through the example in the next section you will see how the WorkBench coaches you through the process.

The Project Manager

An application being translated may consist of many databases that need to be tracked. Furthermore, several applications may be in the process of being translated at the same time. Domino Global WorkBench has a Project Manager component to handle this. In the Project Manager you specify which databases belong to a given project. You can have as many projects as you want and you can easily switch from one project to another using the Project pane in Domino Global WorkBench or by going through the menus.

The Standalone Tagger

The standalone Tagger allows you to adjust the tagging in a tagged database manually, working in the Notes client. In earlier versions of the translation tools, this tool was called The Populator. You can use the Tagger to manually tag terms that were missed during the full database tagging process.

Caution The Tagger allows you to change text in the tagged database. Be aware that if you do this, your tagged database (and the language databases built from it) will then be out of step with your source database. If you want to change text in the database, the recommended process is to change it in the source database and then update the tagged and language databases using the WorkBench.

The Synchronizer

Domino Global WorkBench also includes synchronization features that help you set up a workflow process to handle documents to be translated in existing databases. You can mark each form in an existing language database in one of three ways:

- **Translatable**

  In the language database(s), any document created from this form will be copied automatically to other languages and marked for translation.

- **Global**

  In the language database(s), any document created from this form will be copied automatically to other languages but will not be marked for translation.

- **Local**

  In the language database(s), any document created from this form will not be copied to other languages.

A Synchronize agent, run regularly in one of the synchronized databases, does the necessary copying. It also handles changes to existing synchronized documents.

You can synchronize between unilingual databases (provided they are to be held in the same directory) or multilingual databases. If you are using unilingual databases, you can also choose to build a language Switchbar into synchronized documents. This allows users to access synchronized documents in other languages.

Localizing an Application

In this section we will walk through the steps required by the localization developer to localize a Domino application.

We will refer to the same application throughout the section, but you could go through the steps using your own application.

The application we will use to illustrate Domino Global WorkBench is one that our fictitious company, Millennium Entertainment, wants to use to make information, such as press releases, available to consumers in several languages through the Web. To accomplish this, Millennium Entertainment will translate their current English WebNews application into several other languages. Each language will have its own database for news.

Using a workflow process for approvals, the application will be accessed by Web browsers and Notes clients before information is released to the Web.

The example covers:

- Setting up the project and tagging the database
- Translating the text in the glossary
- Building language databases

Not all Domino Global WorkBench functions will be explained in the example. Please refer to the Help in Domino Global WorkBench for a more detailed description of those functions not covered in the example.

Setting up the Project and Tagging the Database

Setting up the project and tagging the source database is work that would normally be performed by the localization developer of the application. The output will be a glossary ready for translation into the chosen languages.

Before Launching the WorkBench

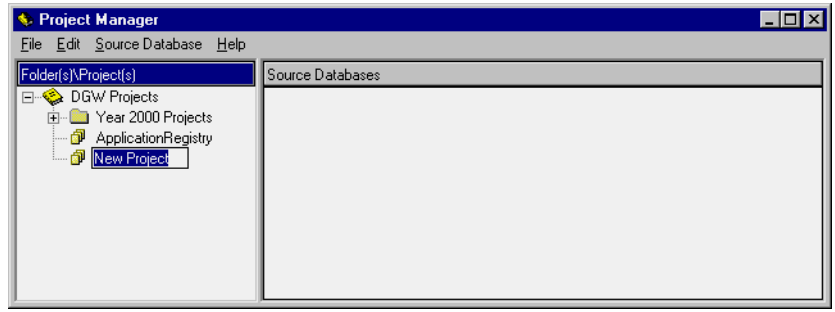
Depending on the development process, the first thing the localization developer should do when he receives an application database for translation from the developer is to create a synopsis database for it. The synopsis database is a good tool for checking whether the database has been properly prepared for translation. For example, design elements without an alias can be identified, enabling the developer to correct such omissions before spending time tagging the database.

When the quality of the source database has been assured, it's time to launch the WorkBench.

Creating a Project

A new project is created in the WorkBench using the Project Manager.

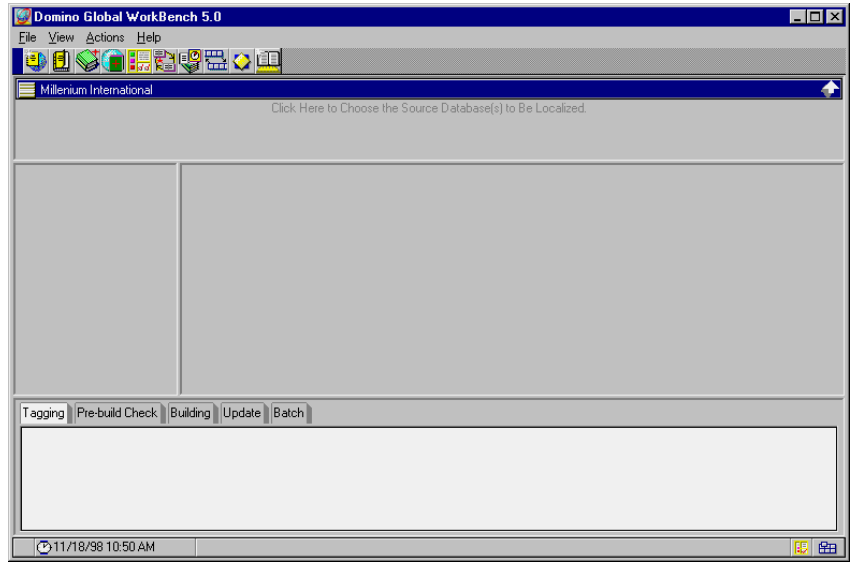
1. There are several ways to open the Project Manager window. One method is to select New Project from the File menu on the WorkBench. This opens the Project Manager window and creates a new project in one operation:



In this example the project is called *Millennium International*. In the Project Manager you can add/remove source databases for a project and rename or delete a project. The Project Manager also allows you to organize your projects in folders and subfolders which can be helpful if you work with several projects at one time.

2. Make this new project the active one in the WorkBench by double clicking it in the left pane of the Project Manager window. This minimizes the Project Manager window and focuses on the new (empty) project in the WorkBench.

   The WorkBench looks like this:

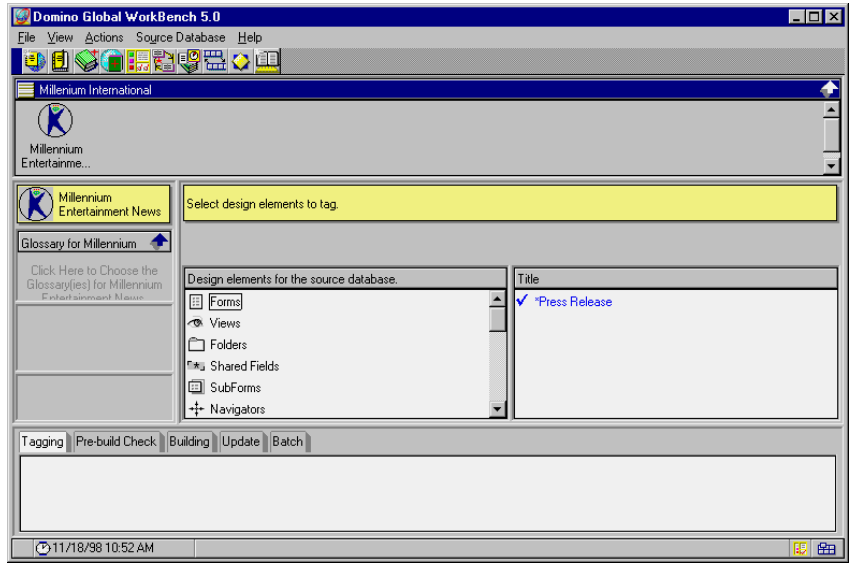


Notice the text saying Click Here to add source databases in the pane below the SmartIcons. The WorkBench "coaches" the user in adding the necessary databases to the project.

Note For the sake of simplicity, we will assume that all the databases we specify during this example are stored locally.

3. Click the Source Database pane. A file open dialog appears.
4. Specify the filename of the source database. In this example the database and pathname will be: *mei\meinews.nsf*. Select the database and then click OK.

   Now the WorkBench looks like this:



You can specify which glossaries to use from the WorkBench, but you cannot create a new glossary database using the WorkBench, so we will have to switch to the Notes client to create the glossary.

Creating a New Glossary

1. In the Notes client select File - Database - New.
2. In the New Database dialog, specify that you want to create a database based on the template *DGW 5.0 Glossary*. Name the database *Millennium Glossary* and give it the filename *mei\mei-glos.nsf*. We will keep all databases related to this project in the subdirectory *mei* in the Domino data directory.
3. When creating a new glossary database based on the Domino Global WorkBench 2.0 Glossary template, you will be prompted for the following information during the creation:

   - **Description**

     Enter *Millennium International Project*.

- **Reference language**

  Pick the language of the source database from a list. In our example we selected *English (United States)*.

- **Languages to enable**

  You select which languages you want to enable from the list box. When a new term is created in the glossary, a copy of it for each language will be created. However, at this time **do not enable** the languages that will be your target languages. This is because you will add comments to the terms in the reference language after you have tagged the source database and this should be done before language copies of the terms are created.

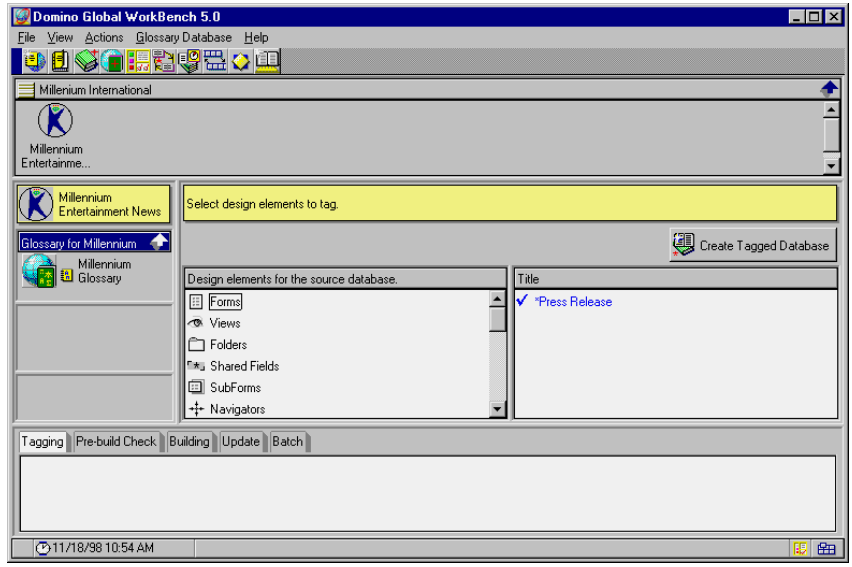
  Instead select the language *Pseudo*.

  The glossary gives you the ability to manipulate the terms in Pseudo. You can reverse all terms or extend all terms in the Pseudo language with a given percentage (say 30%). You can then build a language database to quickly identify untranslated text or to see how the user interface is affected by translations that are longer in the reference language.

Caution If you have already done translations using Notes Global Designer R4.6 and you want to use the glossaries you created then, you must first update the glossaries to the new format for Domino Global WorkBench. Refer to the Domino Global WorkBench documentation for a description of how to update the glossaries.

4. Leave the glossary database and go to the WorkBench.
5. Click the glossary pane where it says Click Here To Choose The Glossary For Millennium Entertainment News and specify the glossary that you just created. You must also specify whether the glossary is to be used as:
   - The Source Database Only
   - All Source Databases In This Project
   - All New Projects

Select All Source Databases In This Project.



6. Notice the Create Tagged Database button. You are now ready to tag your source database, but in order to capture the messages from the tagging you will first need to create a report database using the Notes client.

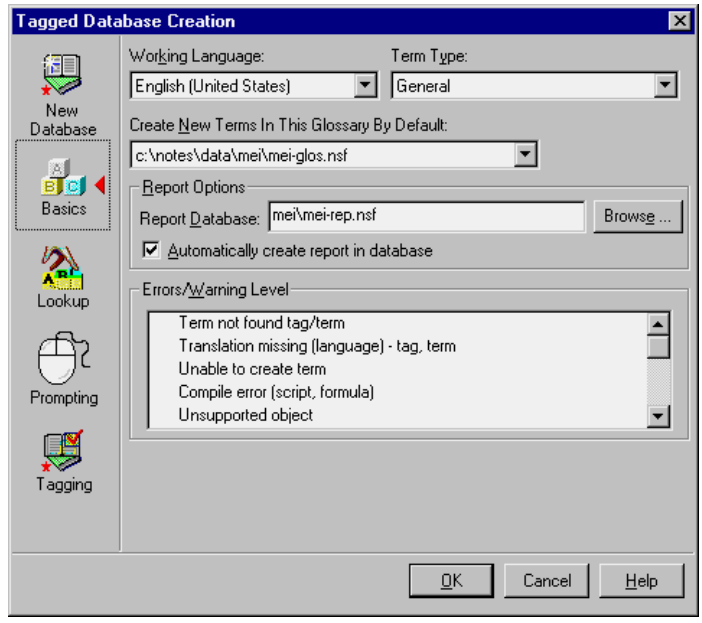
Creating a Report Database

1. In the Notes client menu, select File - Database - New. In the New Database dialog, specify that you want to create a database based on the template DGW 5.0 Report.
2. Name the database Millennium DGW Reports and give it the path and filename mei\mei-rep.nsf.
3. Switch back to the WorkBench.

Tagging a Database

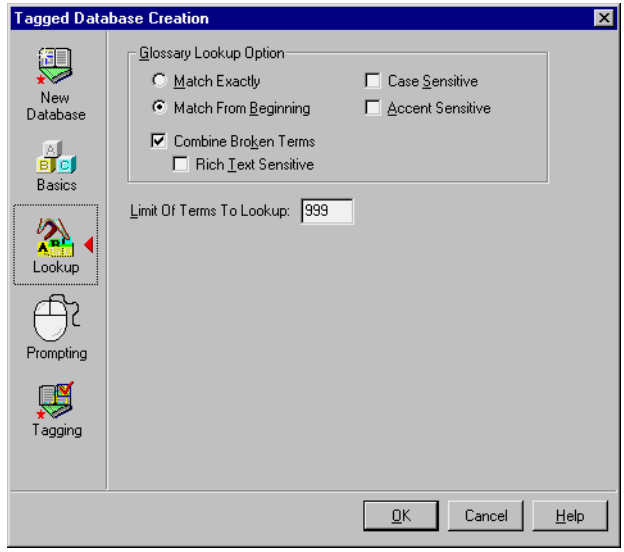
1. Click the button Create Tagged Database. A dialog box with five sections appears. The following section will discuss each of these tabs in turn.
2. The first section is called New Database. You must fill out the path and filename for the tagged database. Enter **`mei\meinewtg.nsf`**.

3. Then click the Basics section on the left side of the dialog box. It will look like this:

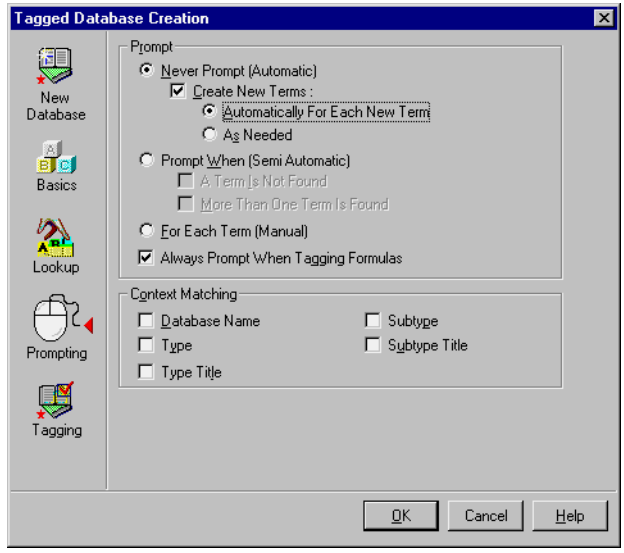


4. In Report Options, specify the report database that you just created and make sure that Automatically Create Report in Database is checked.
5. You can also select what kind of messages should be written to the report database in the list box named Errors/Warning Level. Select all items in the list box.

6. Click the Lookup icon on the left side of the dialog box to move on.

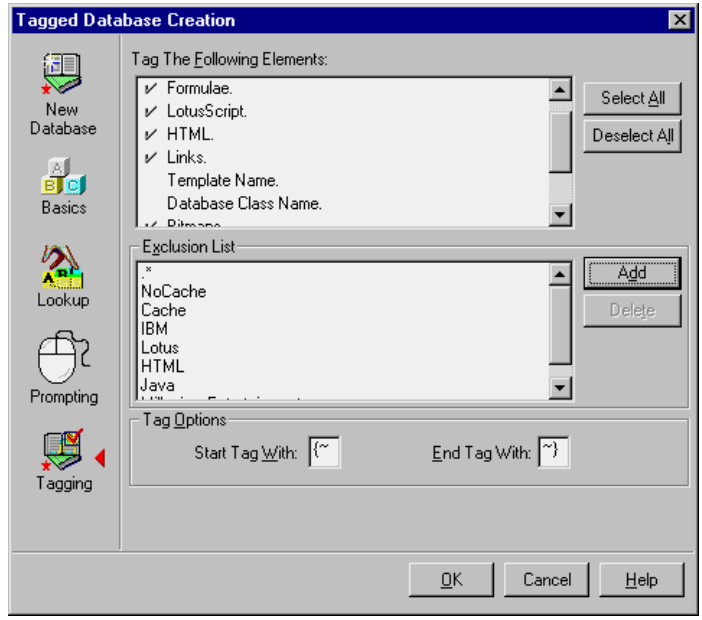


7. The Glossary Lookup Options are important if you use a glossary that already contains translated terms and you want to match those terms with the terms in the source database. You are going to create a glossary from scratch, so just click the Prompting icon on the left side of the dialog box to move on.



8. In Prompting, you can decide when you want to be prompted during the translation process as to what should be tagged.
9. Some of the options are only relevant if you have an existing glossary that contains terms with language translations. In this example, your glossary is empty, so select the following options:
    - Never Prompt
        - Create New Terms
            - Automatically for Each New Term

    Leave the option Always Prompt When Tagging Formulas checked.
10. The section with Context Matching also refers to glossaries containing translated terms. You can select options here to make the lookup and matching of terms more specific. With an empty glossary as in our example, these options are not relevant.
11. Click the Tagging icon on the left side of the dialog box to move on.



12. In the Tagging section, you can decide what you *do and don't want* to tag. In the upper list box you select/deselect whether you want to tag elements such as formulas, LotusScript, HTML and so on.

    Leave the default selections in the list.

    Note If you select to have the database title tagged, and you have an application with several databases, make sure that you understand enough of the target languages being translated so that you can distinguish one translated database from another.

    Note You can also deselect certain design elements such as a special form or all agents from being tagged. This is done in the WorkBench.

13. Below the list box with elements to tag there is the Exclusion List. Here you can specify terms that are not to be tagged. Untagged text will be copied unchanged to language databases when you build. The wildcard * (asterisk) represents any text.

14. If you have adopted a system of prefixes for alias names, you can prevent them from being tagged by entering the prefixes here, followed by an asterisk. For example, fa_* prevents the tagging of any reference to aliases that have the prefix fa_. This provides a reliable way of preventing the tagging of alias names in formulas, LotusScript or JavaScript.

    Our example database uses a simple prefix system where all text that should not be translated is prefixed by a dot.

    For example, a form alias looks like this:

    ```
    .PressReleaseForm
    ```

15. To avoid references to aliases being tagged, you must add a prefix to the Exclusion List. Click the Add button, and enter this in the prompt that appears:

    ```
    .*
    ```

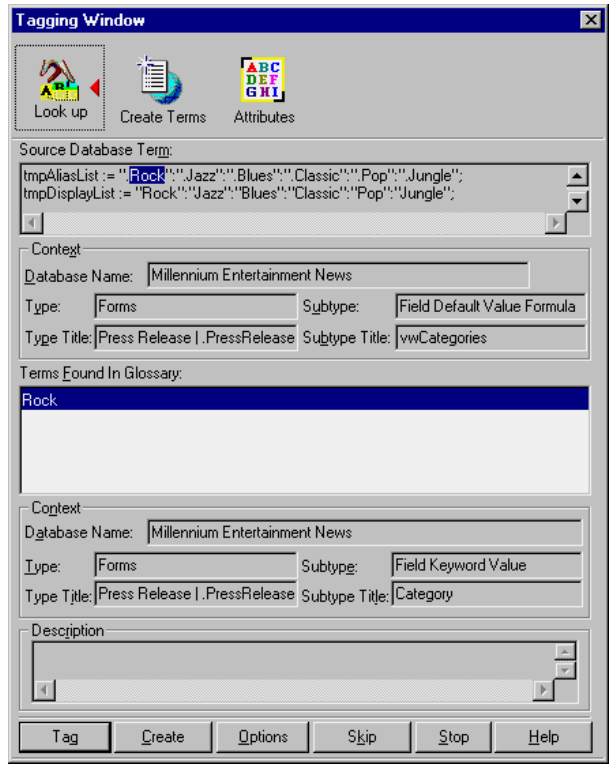
16. Click OK and the prefix is now in the exclusion list.

17. Occurrences of the company name should not be translated either. Add **Millennium Entertainment** to the exclusion list as well.

18. You are now ready to start tagging. Click OK and enter your password if prompted.

19. Watch the Tagging window for messages that will appear indicating the progress of the tagging.

When a formula with text in it is encountered the following will appear:



Here it is a computed field called vwCategories that has the following formula:

```
tmpAliasList := ".Rock":".Jazz":".Blues":".Classic":".Pop";
tmpDisplayList := "Rock":"Jazz":"Blues":"Classic":"Pop";
@Replace(Category; tmpAliasList;tmpDisplayList)
```

There is another field named Category on the form where the user selects a category from a keywords list. Aliases are used in the keyword list, so an alias such as **.Rock**, with the dot prefix, is the actual value of this category field. However, to display the category in a view, a match between the alias and its "plain text" representation must be made and this is what this formula does.

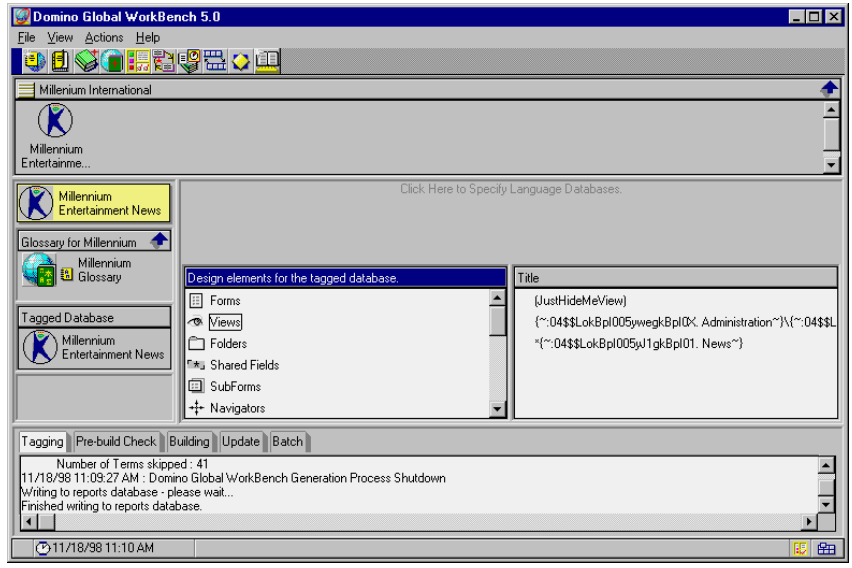
The calculation is done and stored in a field on the document when it is saved. The calculation could also be placed in a view column, but then it would affect performance because it would have to be calculated for all documents every time the view is opened.

The strings prefixed with a dot in the formula have been skipped and the first "plain text" term is selected. If the application developer has adhered strictly to the convention of prefixing all strings referenced by code with a dot, you should be able to click Create and Tag every time you are prompted for a string in a formula. However, in the case of a candidate string that should not be translated, just press the Skip button.

Note If you accidentally click the Create and Tag button on a term that should not be translated, you can "repair" this later in the glossary by marking the term as "Prevent Translation."

Caution Use only the technique described above for translating category elements when the element list is static. If you need the category list to be dynamic you should store it in a profile or setup document.

20. When the tagging is done, the tagged database will appear in the WorkBench:



21. You should open the reports database in Notes and check if there any unusual messages. You can browse all the messages or create reports with Error Summary or Untranslated Summary.

22. When you have a successfully tagged database, go to the glossary to mark terms that should not be translated and to prepare the pseudo translation.

Marking Terms in the Glossary as "Do Not Translate"

Even though you have specified the prefixes used in naming design elements and other names not to be translated in the tag exclusion list, you may still encounter terms in the glossary that should not be translated.

For example, you entered the name *Millennium Entertainment* in the exclusion list, but it turns out that the name *Millennium* also appears in the application and it is now in the glossary as a term. However, you can prevent the translators from being able to translate it. To do this, open the glossary. Select the term and click the action button labeled "Prevent Translation."

This makes sure that the term remains unaltered in any language version of the application.

Running Checks Using Pseudo-Translation

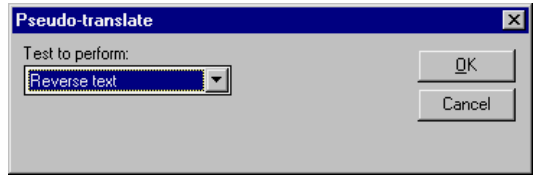
Before sending the terms in the glossary off for translation, the localization developer must check for untranslated terms and potential sizing problems. The glossary helps you in this by enabling automatic translations of the terms in the pseudo language. You have two options:

1. **Reverse terms** allows you to identify untranslated terms in the application.
2. **Expand terms** allows you to check for potential sizing problems in the application.

First check for untranslated terms.

Performing a Reverse Pseudo-Translation

1. Open the Glossary by double-clicking it in the WorkBench if it is not already open.
2. Click Glossary Management on the main navigator and then click Pseudo-translate. The Pseudo-translate dialog box will appear:
3. Switch from Expand in the list box to Reverse and click OK.



4. The Pseudo terms are now being reversed as shown in the following example:

```
Enter Subject
```

appears in the reverse pseudo translation as

```
tcejbuS retnE
```

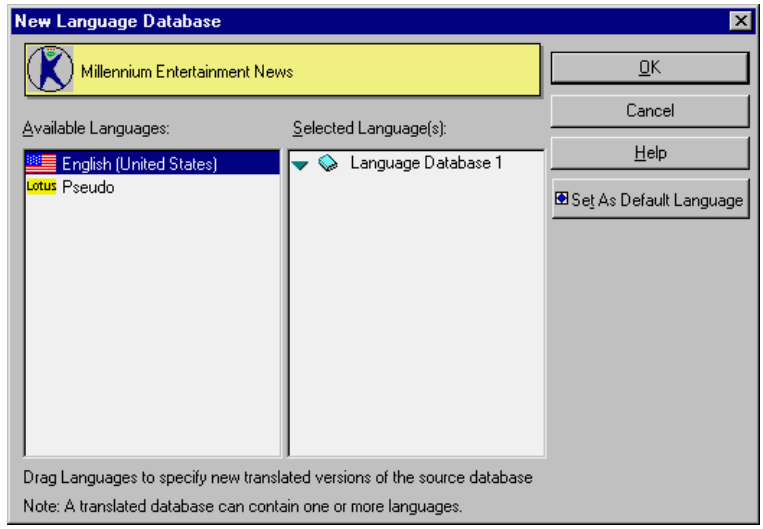
This means that it will be easy to detect terms that have not been translated into reverse pseudo.

Note After a term has been pseudo-translated it is marked as translated and will not be changed in subsequent pseudo-translations.

5. Close the Glossary and go back to the WorkBench to build the language version of the application in reverse pseudo.

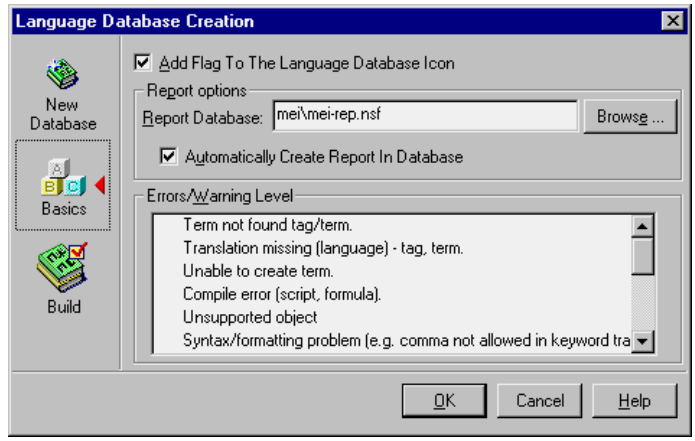
Building the Application in Reverse Pseudo Language

1. In the WorkBench select the tagged database and then select Click Here To Specify Language Databases. The New Language Database dialog box appears:



2. Select Pseudo in the Available Languages list box. Drag it over to the Selected Language(s) list box and drop it under Language Database 1. Click OK.
3. Press the button in the WorkBench that says Build Language Database.
4. A Language Database Creation dialog box appears. It has three sections. In the New Database section, specify the location and filename for the new database. Call it *mei\meinewps.nsf*.

5. Click Basics on the left side of the dialog box to move to the next section. It looks like this:



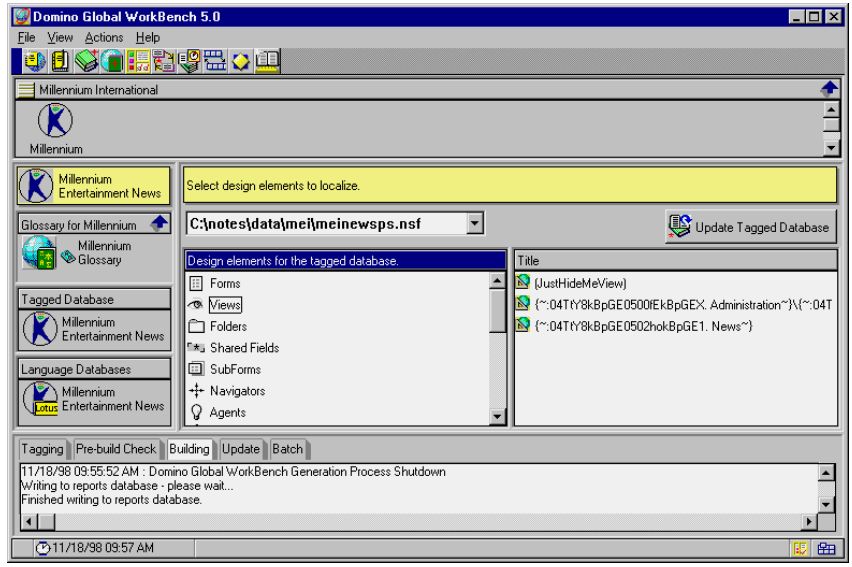
6. Leave Add Flag To The Language Database Icon checked. This option can be a big help in quickly identifying a certain language version of the database when working with several languages. Specify that you want all messages written to the report database by selecting each of them in the list.
7. Click Build on the left side of the dialog box.

   In the Build section you specify whether you want the new language database to be created as a replica or as a copy of the tagged database. You can have the ACL from the tagged database copied to the language database.
8. This is also where you can specify if you want to use the synchronizer technology that allows for automatic copying of new documents created in one language database to the other language versions of this database.

   Select the following:

   - Language Database is A Copy
   - Copy The ACL

Click OK to start the building of the language database. Enter your password if prompted. When the building is complete, the WorkBench will look like this:



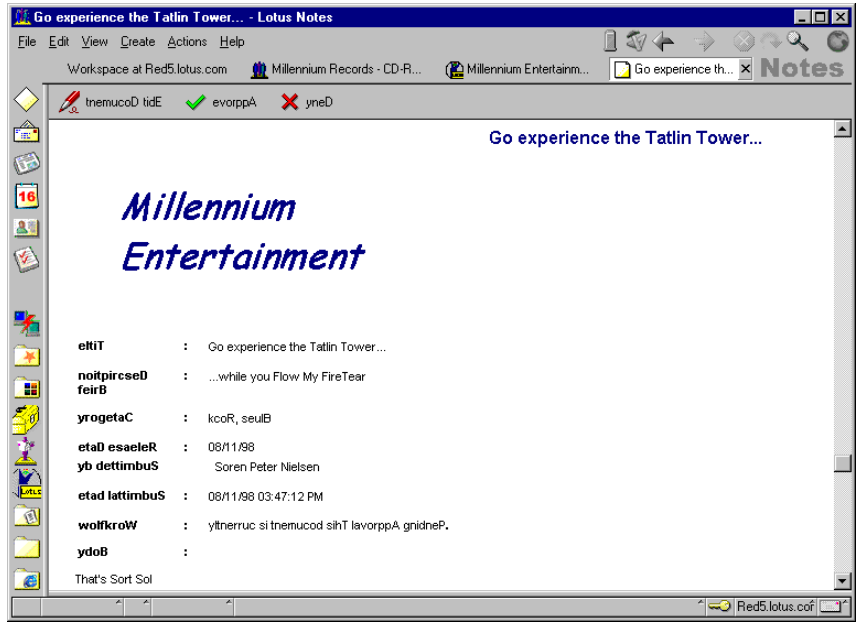
9. Check the report database to see if anything unusual has occurred during the build process.
10. Open the Pseudo language database by double clicking it in the WorkBench. This is just a quick check to see that the database opens all right and seems, at a first glance, to be working.

The pseudo language database is now ready to be inspected for terms that have been missed during tagging or that have been marked as Do Not Translate in the glossary.

Checking the Reverse Pseudo Application for Untranslated Terms

Basically, the same test case scenarios that the application went through before being given over to translation should be rerun. All visible design elements (forms, views, pages, and so on) should be inspected for untranslated terms.

Here is an example of a form:



This form is OK because the text that is readable was either excluded from translation (Millennium Entertainment) or it is part of the data in the document, for example, the subject, name of author, and so on.

Note The order of views, elements in lists, and so on will be juggled around, For example, if you have these three views in your database:

```
1. News

2. Approval Status

X. Administration
```

they will appear in this order in the Pseudo Reverse language database:

```
noitartsinimdA .X

sutatS lavorppA .2

sweN .1
```

However, if you have numbered your views, it should be quite easy to determine which is which.

Handling Untranslated Terms

If you encounter untranslated terms do the following:

If the term has been skipped during tagging you can tag it individually in the Notes client using the Tagger, or you can Re-build/Update the tagged database using the WorkBench.

If the term has been erroneously marked with Prevent Translation in the Glossary, go into the glossary and mark it as Allow Translation. Then select Pseudo-translate using the Reverse option again.

This will reverse only those terms that have been created during the latest tagging and terms that have been marked Allow Translation. The Pseudo terms that have already been reversed are marked as translated and will not be touched by this new Pseudo translation.

As you have made changes to the translations in the glossary you must rebuild the language database (rather than just update it). In the WorkBench, select the Pseudo language database and then on the Language Database menu, select Re-Build Language Database.

Check the language database for untranslated terms again and then move on to check for sizing problems.

Checking for Size Problems

When translating from the reference language into another language more space is required for the text. For example, when translating from English to German, expect up to 30% expansion in the text. To be able to quickly test how the application looks and behaves in a language that requires more space for text, use the pseudo translate function in the glossary.

Recreating the Pseudo Terms in the Glossary

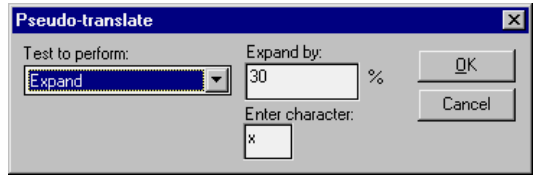
Since you already have translated your terms into Reverse Pseudo, we will not be able to change those terms into Expanded Pseudo language. Therefore, you need to perform a small "trick" to enable expanded pseudo-translation: you will have to remove the Pseudo language from the glossary and then enable it again.

1. Open the Glossary by double-clicking it in the WorkBench. Click Glossary Management in the main navigator.
2. Click Select/Deselect languages in the navigator. The Available languages view opens. Scroll down and select the Pseudo document.
3. Then click Deselect. You will be warned that documents for the selected language will be deleted. Click OK.
4. Now, with Pseudo still selected, click Select and then click OK. New documents for pseudo are now being created with status as untranslated.

You can now proceed with the pseudo-translation that expands the terms.

Performing an Expanded Pseudo-Translation

1. Go to Glossary Management in the glossary if you are not already there. Then, click Pseudo-translate in the navigator.



2. In the Pseudo-translate dialog box accept the default values that specify expanding the text by 30% by clicking OK.

   If there are many terms in the glossary, this may take a little while. Wait until the Pseudo-translate dialog box disappears before proceeding.

3. Go back to the WorkBench to perform the Expanded Pseudo language build.

Building the Application in Expanded Pseudo Language

Click the language database that you built when checking for untranslated terms and select Re-build Language Database from the Language Database menu on the WorkBench.

Caution Do not select Update Language Database as this will not pick up the new translations from the glossary.

Checking the Expanded Pseudo Application for Sizing Problems

When checking for untranslated terms repeat the scenario you used to check the application for any sizing problems.

Note Make sure you use a display resolution that is the same as the one that the target users will have on their PCs.

Here is one example of a sizing problem using the Notes client where the translated action buttons are being cut off at the right side of the window.

In the reference language of the example application, the action buttons look like this:

There is sufficient space for the buttons on the action bar. However, if the text in all labels is expanded by 30%, the action bar will look like this; the rightmost button is cut off:

With the new option to scroll action buttons in R5.0, the user will not be cut off from any of the functions, but the idea of giving users instant access to commonly used functions by action buttons suffers a bit if they have to scroll to get to the function.

The fairly long labels on each button in this example could be fixed by using shorter labels and/or noting to the translators to watch out for the length of their translations. However, for other sizing problems you may have to ask the developer to change the design, for example, by combining several action buttons into one cascaded action button.

If you have found any sizing problems that require changes to the source database, you must update or rebuild your tagged database and the language database and then check again.

When you can validate that all translatable terms in the application have been tagged and that no sizing problems seem to exist, the localization developer can continue preparing the terms in the glossary for the translators.

Preparing the Glossary for Translation

Before enabling the languages in the glossary that you want to translate your application into, the following must be done:

- Add comments to terms
- Prevent translation of terms that should not be translated

Note It is important that you add comments to the terms before enabling other languages because comments added afterwards will not be duplicated automatically to the language versions of the term.

Often, the translators will not have an intimate knowledge of the application they are translating or they may not even have access to the application they are translating. In such situations it is important to add good comments to the terms in the glossary to explain the context in which the terms are used.

The glossary may also contain terms that should not be translated. Typically these will be:

- **References to aliases in formulas, LotusScript, or JavaScript.**

  These must not be translated if functionality is to be preserved. Typically they are references to forms, views, agents, and the like. For example, in a formula such as @command([ChangeView]; "mainview") the text mainview is usually an alias.

  In our example we use aliases with the standard dot prefix. We included this prefix in the WorkBench's exclusion list when creating the tagged database so we should not encounter any of these.

- **Text strings that never display, and therefore do not need to be translated**.

  For example, the names of hidden views, code comments, company names, and so on.

  Names of companies and products that must be the same in all languages should be entered in the taggers exclusion list. However, once such a name has been added to the glossary it might be just as easy to prevent it from being translated in the glossary as it is to rebuild the tagged database and then delete the name from the glossary.

  For code comments, the developer should apply the text style DO_NOT_TRANSLATE to the comment during translation as this will keep them from being tagged. However, if code comments have made it into the glossary, the easiest method is to mark them with Prevent Translation.

It is a good idea to keep entries in the glossary for these *do not translate* terms because you will then not have to think about them again. For example, when processing a revised version of the database you could leave them untagged, with no glossary entry, but you would then encounter them again, as *new* terms, when you create a tagged database from a new version of the database.

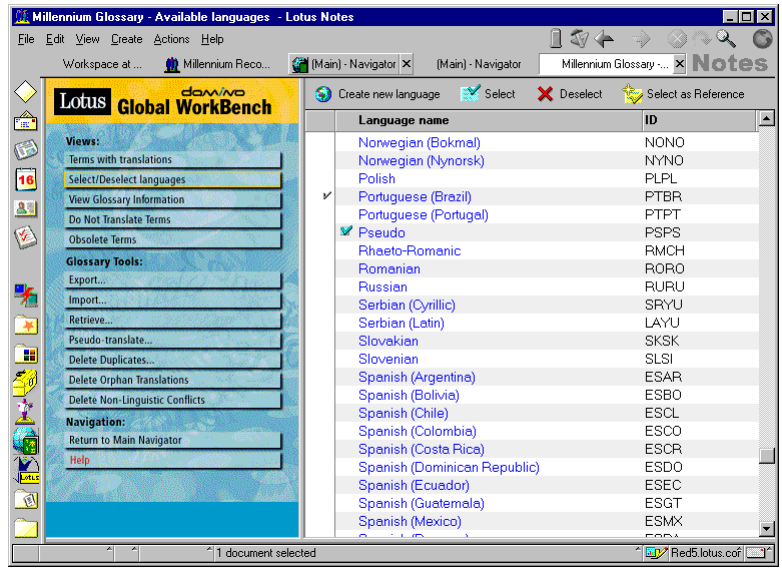
Enabling Languages in the Glossary

When you have finished adding comments and marking *do not translate* terms, it is time to enable the languages that you want your application translated into.

1. Go to the Glossary Management navigator in the glossary. Click Select/Deselect Languages in the navigator. The Available Languages view opens. Select the target languages for the application, for example Chinese, Danish, French, German, Italian, Portuguese, and Spanish.

   Note Most major languages exist in several variants depending on the country where they are spoken. You should check with your local experts as to whether you need to create an area/country specific variant of your language databases. If this is the case, you may want to start out with enabling one variant of the language and then do the translation to the other variants based on the first translation for that language.

In our example we will perform one translation for each language. For example, for Portuguese we selected Portuguese (Brazil).



2. When you have selected all the target languages click Select. You will be warned that documents for the selected languages will be created and it may take some time. Click OK.

The language terms are now ready for translation.

Translation

When a glossary is ready for translation you can either let the translators do the translation in the glossary itself (or a replica) or you can export the terms on a language basis for translation by other tools outside the glossary.

The glossary offers functionality to assist in the translation and review of translated terms.

Assigning Roles to Translators and Reviewers in the Glossary

If the translation is being done in the glossary database, you must assign ACL roles to the translators and reviewers to control the level of access that they are allowed to the glossary database. The roles are:

- **Translator**

  Users assigned as translators can edit and save translation documents, but they cannot create new terms and they have no access to management or review functions.

- **Approver**

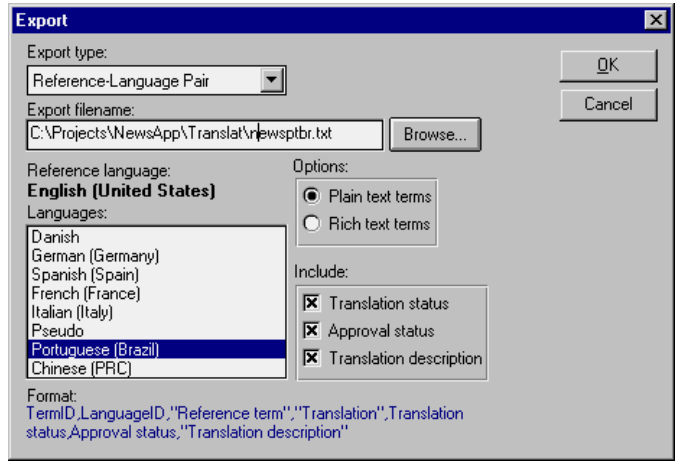
  Users assigned as approvers have the same rights as translators, but can also mark translations as approved.

Note Users not assigned to either role have access to all the functions of the glossary.

Exporting Languages for Translation Using Other Tools

If the translation of a language is to be done using a tool other than the glossary, you must export it.

1. On either of the main navigators in the glossary, click Export. The Export dialog box appears:



2. Select the language to export and the export type.

   You can export both reference language text and translation language text, or just the translation language text. Even if you perform the translation in the glossary you will want to export the translated term for spell checking.

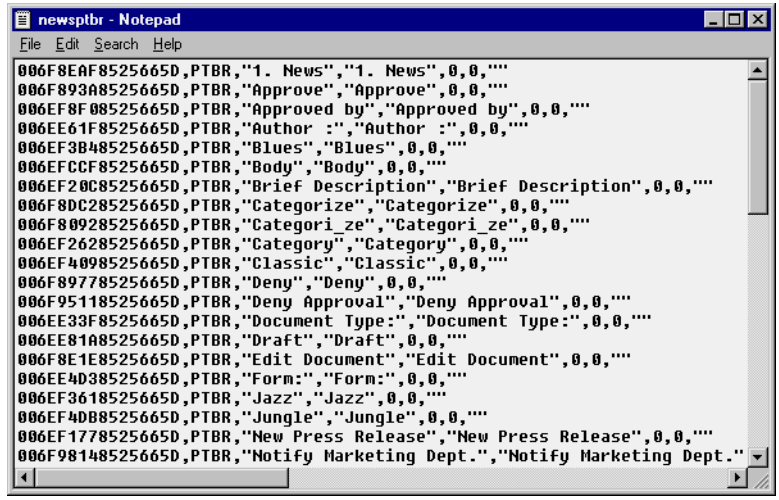
3. Enter a filename for the exported file.

4. Choose whether to export text from plain text terms or rich text terms.

   Export can export text only as plain text, even when it comes from glossary entries whose format is Rich Text.

5. (Optional) Choose additional information to include in the export: Translation status, Approval status, and/or Translation description.

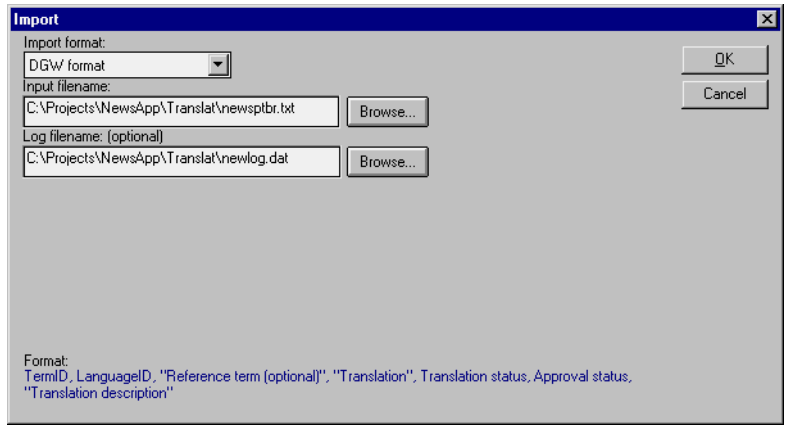
The following screen shows an example of the first lines in an exported file:



Importing Translated Terms

When a language that has been exported to a file for translation comes back translated, you must import the translated terms to the glossary.

1. On either of the main navigators in the glossary, click Import. The Import dialog box appears:



For Import format you can choose between Domino Global WorkBench format and custom format. Custom format allows you to import terms stored in almost any format. In this example we will stick to the Domino Global WorkBench format.

2. Specify the filename and click OK. The translated terms being imported update the existing language terms documents.

Note Make sure that the character set used by the terms you are importing is supported by the operating system and the Domino software that you are running.

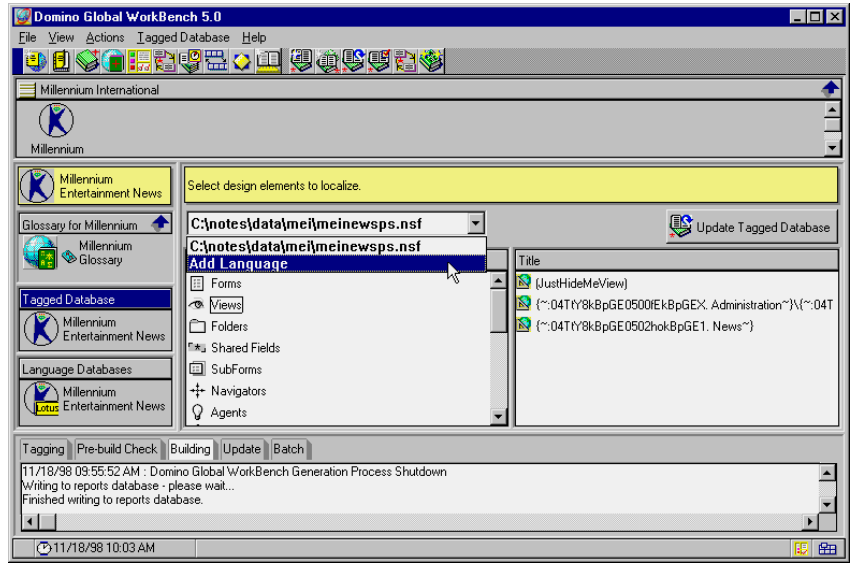
All translations should be reviewed. Whether the review is done on the translated terms in the glossary or on an exported version of the terms depends on your process for translation.

When all terms for a language have been marked as translated or reviewed, you are ready to build the application for that language.

Building Language Databases

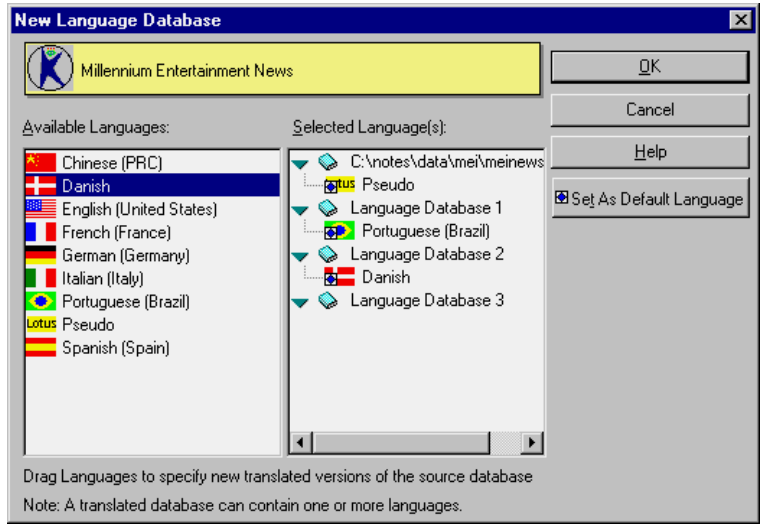
You build a language version of the application in the same way that you built the Pseudo language version.

1. In the WorkBench, select the tagged database.
2. Click the drop down arrow next to the pseudo language database name. You will see the item Add Language as shown below:



3. Click Add Language and a dialog box will be displayed.
4. Select the language that you are ready to build from the Available Languages list box.
5. Drag it over to the Selected Language(s) list box and drop it under Language Database 1.

If you are ready to build for several languages, repeat step 5. Since you are creating one database for each language, you must drop each language under its own database as shown below:

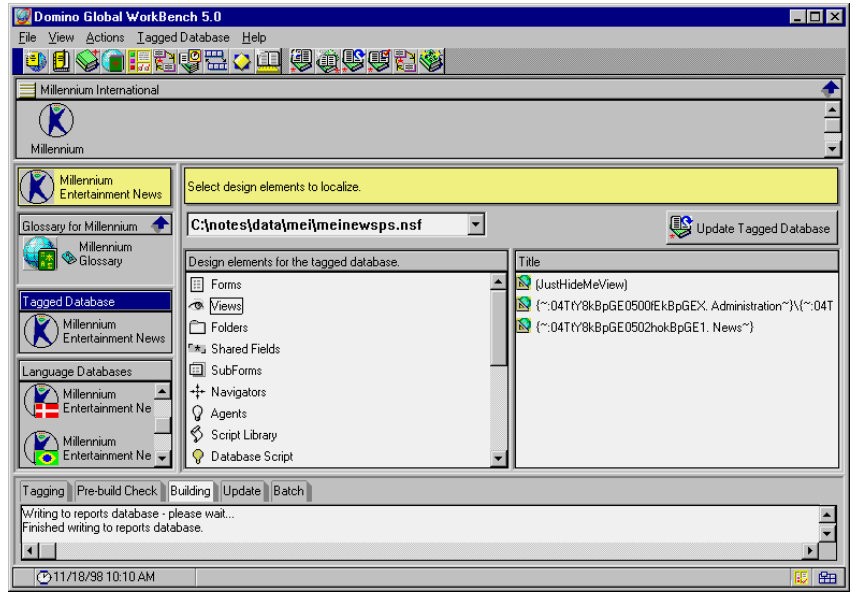


6. Click OK. The drop down list on the WorkBench now has one of the new language databases selected and you can now build them one by one. Press the Build Language Database button now visible to the right of the drop down list.

7. The Language Database Creation dialog box appears. Fill out the information in the same way as you did when creating the Pseudo language database, for example, name the Danish version mei\meinewda.nsf and so on.

   Repeat the building of language databases until you have built for all your translated languages.

The WorkBench will now look like this:



8. Remember to review the report database for any error or warning messages. Do some preliminary tests in the language versions and then send them off to the localization validators.

Should the localization validators require any changes to the application or the translations, you can easily apply the updates to the language versions using the WorkBench later on. For now, you have completed the basic cycle of translating a Domino application.

Skipping Terms During Tagging Versus Marking Terms as "Do Not Translate"

During the example above you have seen that there are several ways to handle terms that must not be translated:

- Define terms and parts of terms in the tagging options exclusion list.
- Assign the paragraph style DO_NOT_TAG.
- Skip the terms when the tagger prompts as to whether it should be tagged or not.
- Mark a tagged term in the glossary as "Do Not Translate."

You should exclude as much as you can using the exclusion list and the DO_NOT_TAG style.

You should allow any non-translatable term that cannot be excluded using either an exclusion list or the DO_NOT_TAG style to be tagged and then mark it in the Glossary as "Do Not Translate." This is because you probably will have to rebuild or update the tagged database several times during the initial preparation for translation.

If you are prompted for a term during the tagging process and you select to skip the term, you will be prompted for that term every time the source database is tagged. If you put it in the glossary and mark it as "Do Not Translate," you will not be prompted for it again.

The Difference Between Updating and Rebuilding

When working with tagged databases or language databases, the WorkBench displays an update button. However, sometimes you have to rebuild a database instead of updating it. The following section explains the difference between the two functions.

Updating a Database

When choosing to update a database, the WorkBench updates only those design elements that have changed in the database it was derived from. If the WorkBench detects new untagged text in the updated design, it will tag it automatically or bring up the tagger window (depending on the tagging options you have selected).

The update process will not detect changes to the terms in the glossary. If you need to have new translations included in your language database you must choose to rebuild it.

Rebuilding a Database

You should choose to rebuild a database instead of updating it when:

- You want to change the selection of elements for tagging or building.
- You want to change the tagging or building options.
- You have made changes to the translations in the glossary(ies).

When you rebuild a database, the WorkBench recreates it entirely from scratch. If you have made any changes to the design of a tagged database or a language database being rebuilt, those changes will be lost. In any case, you should really only change the design in the original database and then carry the changes forward to the other databases using the update or rebuild function in the WorkBench.

Preparing Your Database — Tips for Developers

This section of the Domino Global WorkBench chapter explains how to design and implement a database that will be as simple as possible to localize. It also discusses what you can do to prepare an existing database for use with Domino Global WorkBench.

When designing database applications that will be translated for international environments, keep the following points in mind. If you pay attention to these, the localization task will be quicker, easier, and less expensive.

In General

Expansion

You are likely to encounter expansion problems if your working language (the language in which your database application is developed) is one of the "shorter" languages, such as English. Translated text can be up to 30% longer than English, so if you are working in English try to leave space for expansion when you design forms, views, and so on.

Text expansion often causes problems in tables. Translated tables often have to be reformatted, and this can be very time-consuming. If you need to use tables, design them with expansion in mind.

The length of currency symbols varies from country to country. For example, US and UK currency symbols are single characters ($ and £), but in Italy the currency symbol is sometimes represented as "Lit." (five characters). The size of numbers will vary from country to country. For example, a house might cost £100,000 (eight characters) in the UK the equivalent in Italy could be Lit. 200,000,000 (16 characters). In a case like this, you might need to leave space in your design for a heading such as "All numbers in 000s."

Note You should also be aware of whether your application is going to be used in countries that are part of the European Monetary Union. These countries are replacing their local currency with the common Euro-currency. It may be a requirement for your application to display amounts in Euro as well as in the local currency that it is replacing.

Action Buttons

Action buttons are sized dynamically by Domino. If you display several of them, using close to the full screen width, you may find that after translation the rightmost one or two will disappear from view.

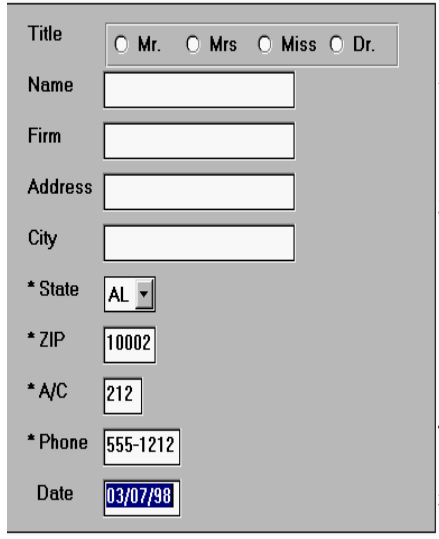
Format

The following are typically different in different countries. On a small scale: date formats, number formats, and postal codes. On a larger scale: formats of addresses, formats of letters.

In general Domino is aware of local formats, and so Notes clients in two different countries may display dates and numbers differently.

Caution Do not store date and time values as strings because their interpretation depends on the language settings of the particular workstation.

Even though Domino will handle the format of data in individual fields, you must also consider formats where several fields are combined. The most obvious example is the format of addresses. Take a look at the following figure:



Fill in the form.....

Prof.
Archibald Bennington-Loughly K.B.E.
Managing Director
Marolt Industries (Europe) PLC
Marolt House
24, North Warwick Close
Hanslow
Midhampton-on-the-Blaines
North Suffolk
MH22 E6A
ENGLAND

Tel. +44 18643 279102

3rd November 1998

You have an address entry form with a format that can be used by a user living in the USA. Fields marked with the asterisk * must be filled in. Our example user (Professor Archibald Bennington-Loughly K.B.E) lives in the United Kingdom. Try to see how you can map his address information to the displayed entry form. Some problems that you may encounter could be:

- Where can I specify a title of Professor?
- Which part of the address goes into the Address field and which goes in the City field?

- State and Area Code are required — but we have no state information or area code.
- ZIP is restricted to 5 characters — our ZIP requires 7 characters.
- How should the date 3rd November 1998 be entered? As 03/11/98 or 11/03/98?

Even in countries that speak almost the same language, you must consider address formats.

There are several solutions you can consider:

- Create computed subforms where you include the correct format depending on information about the user's country of residence.

  You need to make sure that you have the information about the country before you display the form with the computed subform, and you also must consider what to do for countries where you have no subform to include.
- Make sure that all fields required by different formats are available in the entry form.

  This may make the form look cluttered and users may be irritated by having to go through fields that are not relevant to them.
- Use fewer fields, for example Name, Address, Country, Telephone and then leave it to the user to enter their data in the format that is correct for their country.

  This will not allow you to structure the data very much. You will, for example, not be able to list people by zip code.

The approach that you use depends on the application requirements. Do you need structured input because you need to do some processing on the data later? Is it very important for you to do input validation on the address, and so on. Unfortunately, there is no single answer to how address entry forms should be designed in a multinational environment.

Note Just making sure that you have the fields for all the required data is not always enough. In Japanese Kanji, a character has multiple pronunciations depending on the context. A pronunciation may be spelled by multiple Kanji characters, so for searching and sorting in Japanese you need to have pronunciation fields in addition to Kanji fields.

Culture

Be careful not to offend your customers. References that are not considered offensive in your country may be so in other countries. Your users are likely to be scattered across the world, belonging to many religions and races, and holding many different beliefs.

Examples that make good sense in one country may not be understood at all in another. An example involving a knowledge of baseball or cricket would not make sense to many people.

Acronyms

Acronyms can be very convenient when creating an application and are often used to save space on a form or view. In general they will need to be translated, like any other piece of text. Bear in mind that:

- There may be no corresponding acronym in the target language so the text will have to be represented in full.
- Sometimes acronyms have negative connotations in other languages.
- Translators will need to know what the acronym stands for. Add an explanatory comment in the application itself or, preferably, in the glossary.

Here are a few examples of familiar acronyms. To an English speaker the translated versions are not always as recognizable as the acronym itself:

UNO: United Nations Organization

ONU: Organisation des Nations Unis

US: United States

EU: Estados Unidos

Graphics

If you use graphics in your design, try to use international ones. A symbol that is very familiar in one country (for example, a symbol for the rail network) may be meaningless in other countries.

Where possible, use graphics without text as they will usually then require no localization. If graphics need to contain text, do what you can to help translators:

- Make sure that the source files are available in a format where the text can easily be translated. It is usually easiest for the translators if the text is held in a separate "layer."
- Keep any subsidiary files that may be useful, and make them available to the translators. For example, if you set up a sample database in order to make a screen shot of a particular form or view, it will generally be useful for the translators to have a copy of it.
- Use common fonts for the text, and let the translators know the names of the fonts and the sizes you have used. If you plan to use an unusual font, make sure the translators have access to it.
- Allow space for expansion (see above).

Use Aliases

Wherever you can, use aliases. This ensures that displayed text can be translated without disturbing the functionality of the database. The tagging function recognizes aliases and does not tag them; it tags only the displayable text.

In particular, you should alias:

- Agent names
- Folder names
- Keyword field values (see Note at the end of this section)
- Navigator names
- View names
- Form names

When referencing design elements in formulas and LotusScript, use the aliases whenever possible. If, for some reason, you cannot use an alias, make a note so that localization groups are aware that a name is being used.

There are different ways to handle aliases:

1. Adopt a system of standard prefixes for aliases. Then it will immediately be clear when a formula is using an alias. For example, you will see something like this:

   ```
   SELECT (Form=".New" | Form=".Current")
   ```

   Here, the dot-prefix '.' is used to indicate an alias. The person tagging the database will know that pieces of text with the recognized prefix are aliases and should not be translated, while other references are not aliases and should be translated.

   Using the dot-prefix on all aliases is recommended because strings with the prefix are immediately recognized as being non-translatable while maintaining the readability of the code for the developer.

   If your development organization already uses a more elaborate naming standard where aliases for different design elements have different prefixes, you can stay with that naming standard. Each prefix used for aliases just needs to be added to the taggers exclusion list.

   The important thing is to have a consistent standard and to apply it from the very start of the development cycle.

2. You can also use numbers as aliases. Normally, where aliases are text strings rather than numbers, automatic tagging does not tag them. However, it *will* tag references to them in formulas (it cannot distinguish them from other displayable text) unless you use a naming convention for aliases with a prefix that has been entered in the taggers exclusion list. The person doing the tagging then has to take some action: either leave the tag in the formula text, but mark the term "Prevent Translation" in the glossary; or remove the tag.

However, automatic tagging *always* ignores numbers, so if an alias is a number, automatic tagging will not tag the alias *and* it will not tag references to it in formulas. The translators task is simpler; there are no tagged references in formulas to consider.

Using numbers as aliases may not always be convenient, but it does help translators. Consider using them where possible, for example, for keywords.

3. The last approach is to make aliases exactly the same as the original text. You may have to choose this option when preparing an existing application for translation. Formulas will automatically use the alias, and the original text can safely be tagged for translation. However, there are one or two drawbacks:

   - There will be no visual indication within the formulas that aliases are being referenced. For example, in the following formula the text "New" and "Current" could be form names or they could be aliases of form names:

     ```
     SELECT (Form="New" | Form="Current")
     ```

     If everything has been properly aliased, it will be safe for translators to treat these as aliases, and either not tag them or, preferably, tag them but mark their glossary entries as "Prevent Translation." The only way to be sure whether or not an alias is being used is to check the form itself.

   Note There are cases, for example, in a simple action where a reference is *always* to a name, never its alias.

   - Each alias will be the same length as the text it is aliasing, and this may be inconveniently long, for example, when working in formulas.

See the section on Creating Aliases later in this chapter for more information.

Temporary Variables

These do not display and therefore do not need to be tagged. However, sometimes they will *look* like displayable text, so it is advisable to use a standard prefix to distinguish them. For example:

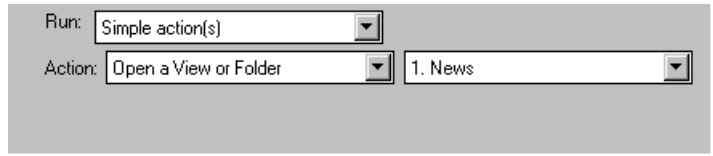
```
@Set("tmpFullName"; Title + " " + LastName)
```

You can avoid the tagging of temporary variables by using a prefix like "tmp" to denote a variable and then adding that prefix to the exclusion list in the tagger.

Use Formulas Instead of Simple Actions

In Domino Designer, it is often possible to reference other design elements through a "simple action." You choose a design element from a drop-down list. However, the list shows only the element *names*, not their aliases (if any), and this can cause problems for Domino Global WorkBench.

It helps if a reference like this:



is changed to one like this:

Note The dot-prefix ".", seen here, is the one recommended for all design element aliases.

Hide-When Formulas

Hide-when formulas can be associated with text in a database. When they evaluate to TRUE, the text does not display. These are held in the same way as text attributes (they are accessed through the text InfoBox), and they are preserved by the WorkBench, provided Combine Broken Terms and Rich Text Sensitive are selected on the Lookup panel in the Tagging Options dialog box. They are not generally preserved by the manual tagging process. You should always use aliases in conditional hide/show formulas.

Where you use them, make a note of them and inform the localization team.

Exclude Paragraphs From Translation Using the DO_NOT_TAG Style

If there are paragraphs in your database that you know will never need to be translated, for example, comments about the design, you can make Domino Global WorkBench ignore them by giving them a paragraph style called DO_NOT_TAG. You can define the style in any way you like; it is the name of the style that the WorkBench looks for.

To assign the DO_NOT_TAG style:

1. Open a design document that contains text you want to exclude from translation.
2. Put the cursor in the section of text, right-click and choose Text Properties.
3. In the InfoBox, click the Styles tab.
4. Click Create Style.
5. Enter DO_NOT_TAG, and check Make Style Available for All Documents.
6. Click OK.
7. Put the cursor in the next section of text you want to exclude from translation, and select DO_NOT_TAG in the InfoBox.
8. Repeat Step 7 for other text that is to be excluded from translation.

There are other ways of excluding text from translation once the database has been designed. See Text that must not (or need not) be translated.

Avoid Shared/Personal on First Use Folders or Views

These are hard to localize. If you do use them in an application that will be multilingual, they may have to be left untranslated.

When a user opens a shared/personal on first use folder or view for the first time, Domino creates a private folder or view with the same name (which the user can customize) and hides the public one. If the name of the public folder or view changes for any reason, for example by translation, Domino will no longer be able to locate the original private version and will display the public one again. This will create a new private copy when opened.

Concatenated Sentences

Avoid constructing sentences using formulas such as:

```
"Today is "+@weekday(@today)+" "+@month(@today)+"
"+@day(@today)+" "+@year(@today)
```

The length of this will vary according to the local language and setup, and in many languages the order of the elements will be different (so the translators would have to change the formula, not just translate the text). In this example you could use a simpler format:

Date: TodaysDate

where TodaysDate is a time field calculated using the formula @Today. Avoid doing the same sort of thing with fields, for example:

Created by [Author] on [Date] for [Customer]

Translating a sentence like this may involve changing the order of the fields, and that means additional work and may introduce errors. A simpler format would be:

Created by:	Author
Creation date:	Date
Customer:	Customer

Use @ReplaceSubstring to Construct Sentences with Variables

If you cannot use a format as suggested above because of space restrictions, layout guidelines or some other reason, you should construct the concatenated sentence using temporary variables and the @ReplaceSubstring function.

Look at this example. Instead of writing:

```
tmpFirstLine := Subject + "  by  " + @Name([CN];From);
```

where the translator just gets the string " by " to translate, you should use this approach:

```
tmpSourceText := "%Subject%  by  %Author%";
tmpAuthor := @Name([CN];From);
tmpFirstLine := @ReplaceSubstring(tmpSourceText;
"%Subject%":"%Author%"; Subject:tmpAuthor);
```

In addition to using temporary variables with the "tmp" prefix, an application may also use sentence variables that have the character "%" before and after them. In the arguments to @ReplaceSubstring, you first specify the string with the source sentence where sentence variables are placeholders for outside values (referenced by tmpSourceText). Next, you specify a list with all the sentence variables used (%Subject%:%Author%) and as the last argument, you enter a list with the real values to be substituted into the source sentence (Subject:tmpAuthor).

The translator gets the string "%Subject% by %Author%" to translate. The translator will be able to see the context in which the word "by" is to be translated and will also have the freedom to change the construction of the sentence, for example, by putting the name of the author first in the sentence.

The translator must be informed that words with a prefix and postfix of, for example, "%" are sentence variables and must *not* be translated, but can be moved at will in sentences.

You should not put the prefix for these sentence variables in the taggers exclusion list because this also will exclude concatenated sentences starting with a sentence variable (as in the example above) from being tagged.

Handling Translatable Lists

If your application contains lists that the user or administrator should be able to update dynamically, you must store them in a profile or setup document. Leave only lists that will remain static within the application to be translated.

When a text list is tagged for translation, each value is entered in the glossary as an individual term. You must make sure to comment the terms so that the translator knows which terms belong to the same list.

An alternate way to handle translatable lists is to store them as one string and then split that string up into the different list elements at runtime using @Explode. Here is an example of how to do this:

```
tmpDisplayList := "Rock#Jazz#Blues#Classic#Pop";
@Explode(tmpDisplayList; "#")
```

In this example a text list is stored as one string where each list element is separated using the character #. The output from this formula is the text list:

```
"Rock":"Jazz":"Blues":"Classic":"Pop"
```

However, when tagging the database only one term is entered in the glossary:

```
"Rock#Jazz#Blues#Classic#Pop"
```

Thus the developer will be able to see which elements belong to the same list.

Caution The developers must be instructed not to touch the separator character in the list because an accidentally deleted separator will concatenate two list elements into one and change the working of the application. If using concatenated lists, the localization developer should check them carefully when returned from translation before building the application.

Keywords

Keep Them Simple

If you use keyword fields, try to keep the keywords simple. They are difficult to translate because field name text is tagged separately from the keyword values, so they end up as separate glossary entries. The translator sees several, apparently unconnected, pieces of text (although there will be some context information to help). In many languages there may need to be grammatical agreement between them.

For example, the field:

Schedule status:	Updated/Not updated

will be harder to translate than:

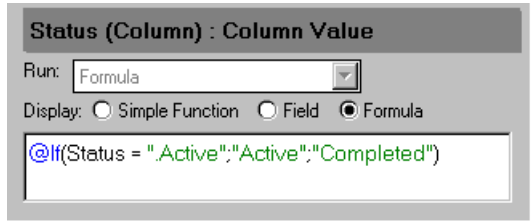
Schedule updated?	Yes/No

Using Formulas for Keywords

If you can, avoid entering keywords using a formula. Instead, use the static items option "Enter choices (one per line)" in the keyword field InfoBox. Automatic tagging will correctly tag keywords, provided that they are static and aliased. However, it will not tag them correctly in formulas, unless they are named with a prefix that has been entered into the taggers exclusion list.

Displaying in Views

If you want a view to display a keyword field do not refer to the keyword field directly, because that will display aliases, not the keywords, and aliases are not tagged or translated. Instead use the Formula option:



Here, a keyword field named Status has the following keywords and aliases:

```
Completed | .Completed
Active | .Active
```

The dot prefix "." has been used to denote a keyword alias. With ".*" entered in the taggers exclusion list, only the values "Completed" and "Active" will be tagged for translation.

Caution Using formulas in view columns instead of referencing fields may affect view performance. As an alternative in single language databases, you can reference a hidden computed field in your form where you enter a formula that computes the text to display in the view. For example, you can have a hidden computed field called vwCategory with the following formula:

```
tmpAliasList := ".Rock":".Jazz":".Blues":".Classic":".Pop";
tmpDisplayList := "Rock":"Jazz":"Blues":"Classic":"Pop";
@Replace(Category; tmpAliasList;tmpDisplayList)
```

If the field Category contains the text list ".Rock":".Blues" then the field vwCategory will contain "Rock":"Blues".

However, if you update the keywords field Category using an agent, take care to update the hidden computed field vwCategory as well.

Note If your application will be multilingual, you have to do the calculation of which text string to display 'on the fly' using view column formulas.

Numbers as Aliases

You may want to consider using numbers as aliases for keywords. However, it may be more difficult to maintain code where numbers are used as aliases.

Fonts

When designing a database, check that the fonts you use:

- Are available to the people who will be using your database. When the Notes client encounters an unknown font it uses a suitable alternative, but this may have an adverse affect on the layout and/or readability of your application.
- Contain all the characters needed for the languages you will be translating into. If a character, for example an accented character, is not available in a font, an unknown character symbol (a vertical bar) will display instead.

For example, Japanese characters will usually display as unknown characters in European environments. This is not a data corruption problem; the Notes client keeps your data secure and will display it properly wherever it can.

Pay Attention to Length Limitations

Domino R5.0 has certain limitations on the length of particular sections of text. This means that if the translated version of a section of text exceeds the allowed length, there will be a truncation in the built database (in that language). Keep text for these items (especially navigator button captions) as short as you can.

Maximum Lengths

Database Title:	32 characters
Navigator Button Caption:	31 characters
View Title:	64 characters*
View Column Headings:	80 characters
View Comment:	255 characters
Form Title:	64 characters

Field Help:	255 characters
Folder Title:	80 characters
Folder Column Headings:	80 characters
Folder Comment:	255 characters
SubForm Title:	64 characters
Navigator Title:	64 characters

* For multilingual applications, the total length of translated cascaded view titles must not exceed 66 characters (including the backslash). If this limit is exceeded, the application may crash without warning for any view accessed by db.getview() that follows after the view in question (including the view in question). For non-cascaded view titles, the total length may not exceed 63 characters. Longer view titles will be truncated to 63 characters with the usual warning in the report database.

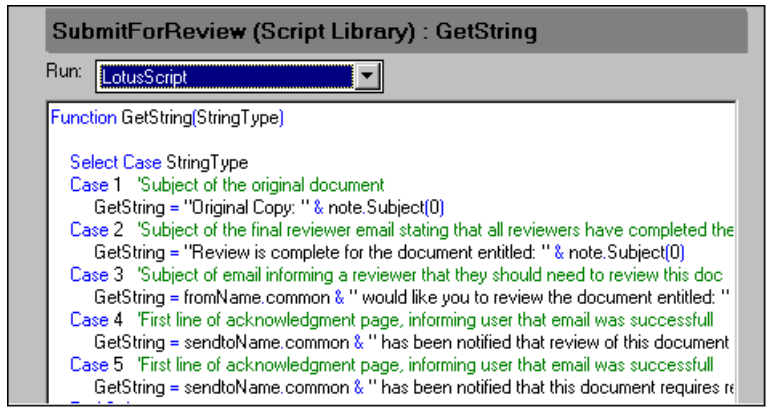
LotusScript

Domino Global WorkBench 5.0 recognizes and tags LotusScript. It tags all text it finds between standard string delimiters with the exception of text that is part of the LotusScript language. Standard delimiters are double quotes ("), vertical bars (|) and braces ({ }).

If you use standard alias prefixes as recommended above, you can make the WorkBench ignore references to them in LotusScript automatically. However, there may be other references that must not be tagged, such as file names. Usually the simplest way to handle these is to let the WorkBench tag them, but mark them “Prevent Translation” in the glossary.

To help the translators, put all the text in one place and store it in the database or template as a script library.

In the following example, a function GetString is used to store text. Each text string has been given an explanatory comment (the text beginning with a single quote) by the database designer, which is very helpful for translators.



Keep Translators Informed

If there are elements of your design that are never displayed and also are not referenced by any displayed text that requires translation, like temporary variables used in constructing sentences, inform the translators.

Note Text that need not (or must not) be translated, is usually tagged anyway, but marked Prevent Translation in the glossary. This ensures that it will always be recognized, and always treated as text to exclude from translation. It *could* be left untagged, but then when an updated version of the database is autotagged, it would not be found and would be presented to the person doing the tagging as new text.

If the functionality of your database depends on any pieces of text that are not aliased (for example, file names or simple action references), inform the translators. Special precautions will need to be taken when tagging these, and some of them may have to be kept in the original language. Use REM statements in formulas to indicate such references. It will also help translators if you include REMs to explain the function of more complex formulas.

If you have used hide-when formulas in your design where translatable text is referenced, tell the translators where they are.

Preparing an Existing Database

This section explains what you can do to prepare an application that has already been developed and now needs to be localized.

Make a Backup

First make a backup replica of the database. You can do this by either:

- Using the DOS copy command to copy the .NSF file.
- Selecting File - Replication - New Replica.

Caution Do not use the File - Database - New copy command. This creates a copy, but it has a different replica ID and therefore cannot replicate with the original database or any of its existing replicas.

Create a Design Synopsis

Create and print a design synopsis. This will list the database design elements and the text they contain. You can use this to identify and mark design elements that might need changing:

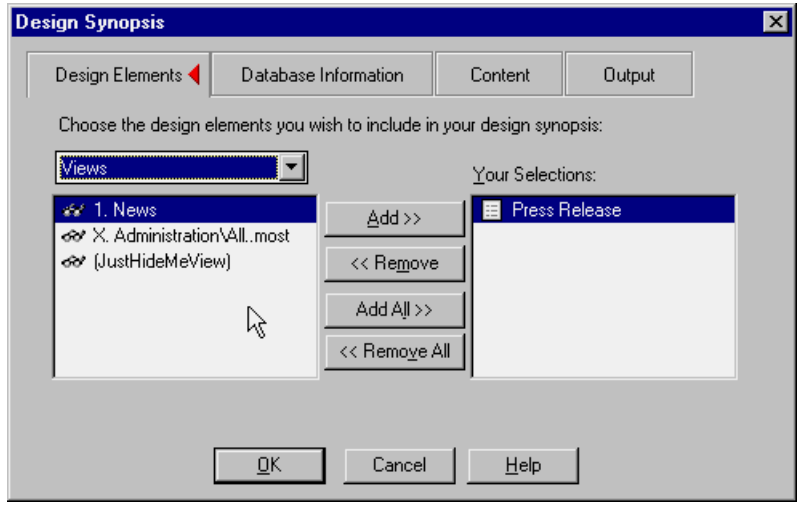
- If you are able to change the design of the database, consider making changes along the lines suggested in the guidelines above.
- If you are not able to change the design of the database, you should at least make sure that all views, forms, and navigators are aliased.

Note Keep the design synopsis. You can use it to identify the pieces of text in the database that need to be tagged.

The procedure is:

1. Open the database in Domino Designer if it is not already in the most recently used list.
2. Expand the database in Domino Designer's most recently used list so you can see all the different types of design elements.

3. Click the element named Synopsis. The following dialog box appears:

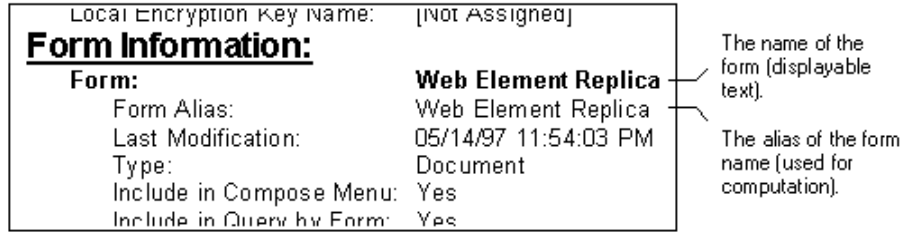


4. In the Design Synopsis dialog box, you can specify which parts of the database and how much information per part you want the synopsis to describe.
5. On the last tab of the Design Synopsis dialog box, you can specify to have the synopsis written to a database for later referral or shown directly in a window.
6. Print the synopsis — either from the window it is displayed in or from the database. Since it can be long, you may want to review it in electronic form (and maybe reduce the font size) before printing.

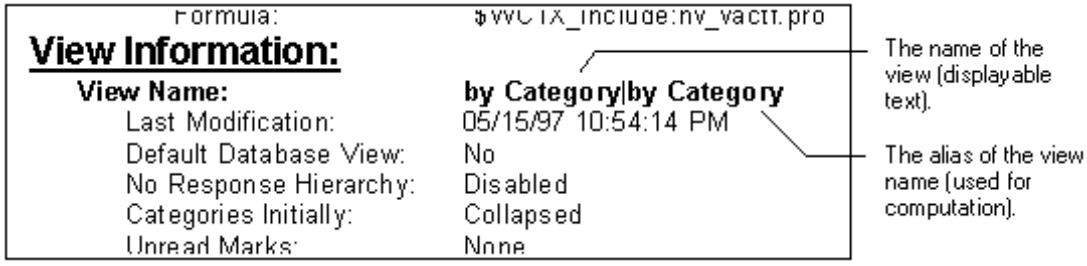
You should now mark on the design synopsis:

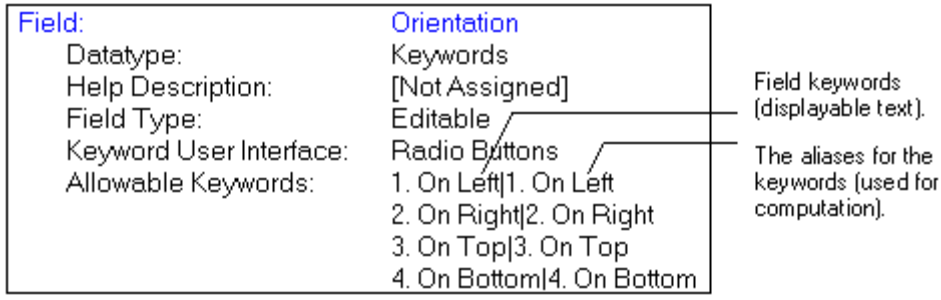
- Any cases where a design element has an alias, but a reference to it (for example in a formula) is using its display name, not the alias. In such cases you must either:
  - Change the reference to point to the alias, or
  - Make sure the design element name is not translated.
- Any of the following that have *not* been aliased:
  - Form names
  - Folder names
  - View names
  - Field keyword values, defaults, input validation and translation values
  - Navigator names

If an item has an alias, you will either see an entry in an Alias field:



or you will see one or more vertical bars in the name:





7. Now create any missing aliases. To assign an alias to various types of design elements, refer to the description of the individual elements in this book.

Summary

Domino Global WorkBench is part of Domino Designer R5.0. It enables you to translate your application into several languages. You can either have one language per database, or combine several languages in the same database and then let the user decide which language to use. Domino Global WorkBench also includes technology to synchronize the translation of database content among several languages.

Extra care must be taken when designing applications for an international audience. Avoid hard coding of strings to be translated, be careful with string length limits in titles, and so on. Issues like how to format an address should also be considered.

Chapter 16
Domino Ad Hoc Workflow

This chapter describes one of the most powerful, yet surprisingly simple, features of Domino, ad hoc workflow. We will cover the basics of what workflow is and why you would want to use it in your applications, and some of the key tasks of designing a Domino workflow application, and we'll take a detailed look at an Approval Cycle template.

Note The Approval Cycle template used in this chapter is from the Domino R4.6 template. The template is no longer included in Domino R5.0, but is available in electronic format on the Internet from the redbooks Web server.

Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG245331`

Alternatively, you can go to the redbooks Web site at:

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

Select Additional Materials and open the file that corresponds to the redbook form number.

Note Lotus also now has a dedicated workflow product called Domino Workflow 2.0. It offers an easy and intuitive rules and roles-based workflow management system for the Lotus Domino environment. This includes an intuitive visual workflow model that enables companies to develop and deploy flexible and robust workflow applications faster and more efficiently. To learn more about Domino Workflow, point your Web browser to:

`http://www.lotus.com/workflow`

What is Workflow?

Workflow is best described using a paper-based environment. Take a simple order form for some goods as an example. Initially, the order form must be be filled in by the customer requesting the goods. This form is then sent to the company where it is received into the sales department. A salesperson may check that the order is valid and then pass it over to the credit control department to check whether the customer is able to pay for the goods. If the credit control department agrees, the order is routed to the warehouse,

where the requested goods are checked against stock levels and then sent out to the customer. When the customer receives the goods, they send the money to pay for them.

In the above process, the order form has been passed from customer to department to another department and then back to the customer. At any point, any of the departments could have rejected the order form for one reason or another and the order would not have been processed.

What we have just described is a workflow process: a piece of paper being moved around a defined process that has a 'status' associated with it at each point in that process. For example, at the beginning, the order form is a *new* order; as it is passed through the departments it is in a *pending authorization* status; through to *delivered, awaiting payment*, and *completed.* At any time the form could enter a status of *rejected* and then would not travel any further through the process.

Using Domino, it is fairly straightforward to create an application that copies this paper-based process. You can create a form that looks almost exactly the same as the paper-based form, you can define the route that the form must take through each of the departments, and you can define the 'status' for the form at any one time.

The advantages of using Domino for workflow over a paper-based system are enormous. Not only can you reduce the amount of paper that is used in an office but you can speed up the authorization process. You can remind people if a document is overdue for authorization, you can create tailored views so that each department only sees the documents that relate to them, and you can add privacy and security to the documents.

Workflow Design Considerations

Before jumping in and developing a workflow application, it is extremely important to make sure that you have a good understanding of the entire workflow process, from beginning to end.

Ensure that you visit each department involved in the process and fully understand the procedure that they follow to complete a particular operation. This also includes what happens if the form is rejected, if a piece of information is missing, or if the form needs to be passed back a step to the previous department. There are usually some obvious steps in the process that are easy to find; make sure that you question each department thoroughly and find the steps that are not so obvious.

At the end of the discussions with each department, you should at least have a list of all the items of information that they require to move the form onto the next stage, where the information comes from, and the order in which the form must travel through each of the departments.

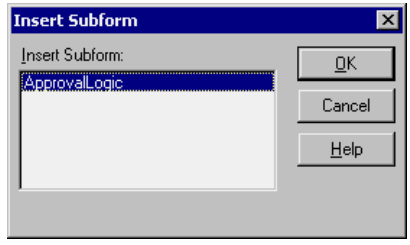
Creating a Database Using the Approval Cycle Template

We will describe an example that uses the Approval Cycle Template from Domino R4.6. This template contains workflow features that are very easy to incorporate into an application and make it possible to have a workflow application working in a very short period of time. See the start of this chapter for instructions on how to get the template if you want to try the example yourself. Once you have the file you must copy it to the data directory of your Notes client or the Domino server you are working on.

To create a database using the template, follow these steps:

1. Select File - Database - New from the menu.
2. The New Database dialog box is displayed. Select the server where your database is to be created and enter a database title and file name.
3. Select the Approval Cycle template from the list and click OK.

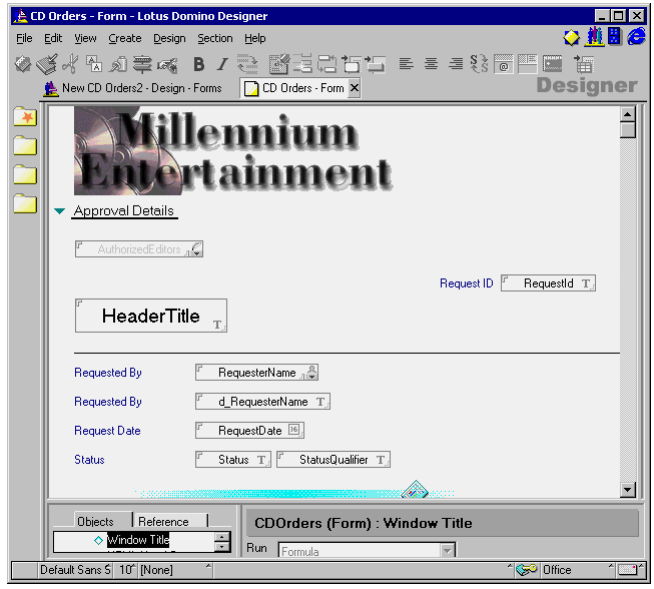
   Note If the Approval Cycle template is not displayed, change the template server from local to that of a Domino server to which you have access to by clicking Template Server.
4. Open Domino Designer by choosing View - Design from the menu.
5. Create a new form in the database by clicking the New Form action button.
6. You will see the following dialog box, select the ApprovalLogic subform and click OK. This subform contains all the workflow logic needed for your application.

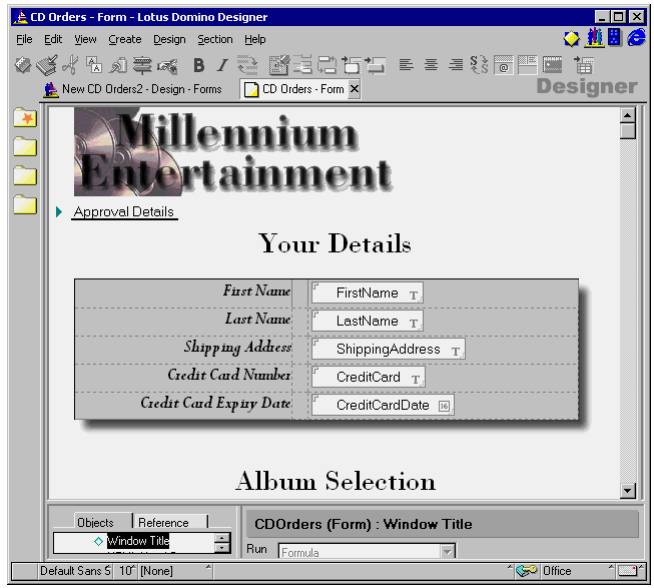


7. The form design is displayed in Domino Designer with the ApprovalLogic subform inserted.

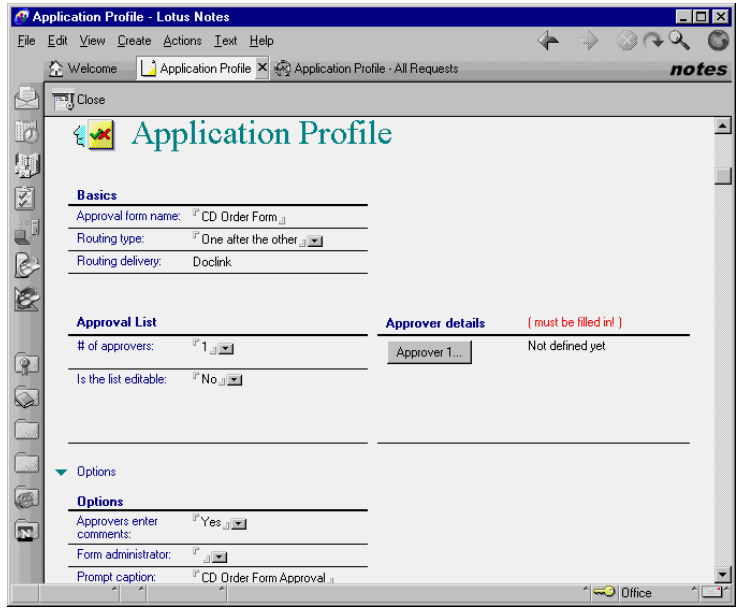
   Tip As the subform takes up quite a lot of space on the screen, create a collapsible section from the subform. Select the whole subform with the mouse, and choose Create - Section - Standard from the menu. Add a title by choosing Section - Section Properties from the menu bar to display the InfoBox, and add your own title.

8. Add your own fields for your workflow application under the ApprovalLogic subform and save the form. Following is an example of a workflow form. The first example shows the subform section expanded and the second shows the section collapsed.





9. After creating your form, the next step is to create an Application Profile. In Domino, select the Application Profiles view and click Create Application Profile. The application profile document is displayed:



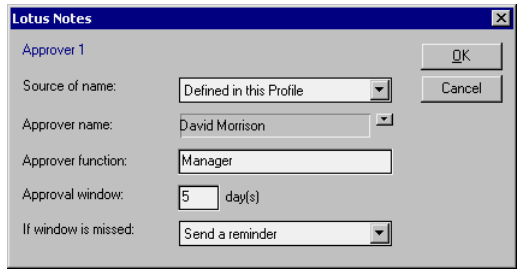
10. In the Basics section, enter the name of the form you just created; this can be either the full name or the form alias.

    Note This must be exactly the same name as the form name.

11. If your workflow document needs to be authorized in an escalated process, select One After The Other from the Routing Type keyword field, or if the document can be authorized by many people at once, select the All At Once option. An escalated process is one where, for example, a customer requests a credit amount above their limit, so the form is routed to a different sub-process for approval before returning to the original process.

12. In the Approval List section, specify the number of approvers. The range is from 1 to 5. In our example we are using just one approver and the list is not editable.

13. Click the button for each of the approvers to display the Approver dialog box. It looks like this:



14. Click the arrow next to the source name to display the available choices. The choices are as follows:

    - Defined in this profile
    - Entered on the form by the submitter
    - Retrieved from a database

    This last option gives you the ability to create your own list of approver names from another database. If you use this option, make sure that the names are valid Notes names, as they will need to be sent in an e-mail.

    In our example, we have left the option as Defined in this Profile and selected a name from the Public Address Book.

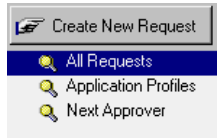
15. Select the appropriate title for the approvers in the Approver Function field.
16. In the Approval Window field, specify how long the approver can take to handle the approval request.
17. In the If Window is Missed field, specify the action to be taken when the approval window has passed.
18. Click OK to return to the Application Profile form.
19. In the Options section, you may modify the terminology used for each stage of the approval cycle.

    Note You must have exactly the same number of stages as the form displays and they must correspond to the terms on the left.

20. Click the Close button and save the new document.

Using the Workflow Document

1. To use the new workflow document, select the All Request view from the view list and click Create New Request.



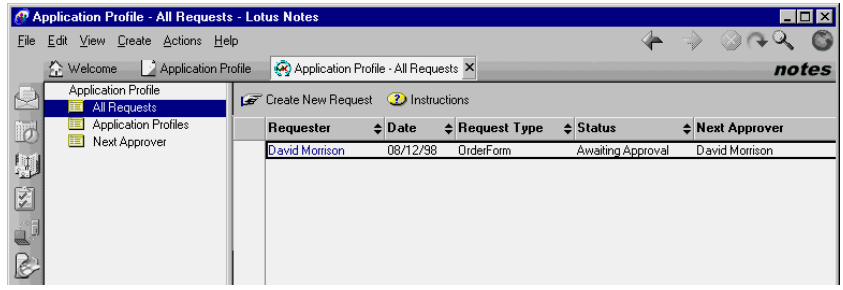
2. The New Request dialog box is displayed. Select your workflow form from the list and click OK.

   Note The form names are taken from the Approval Form Name field of the Application Profile.

3. Complete the fields in the form and when you are ready, click Submit for Approval.

   The following message is displayed, notifying you that an e-mail has been sent to the first approver.

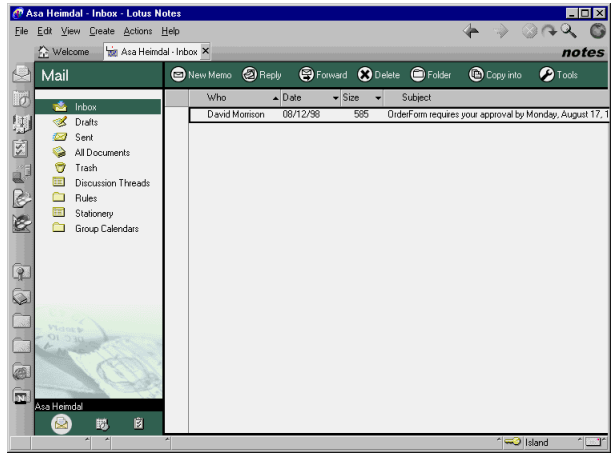
4. Click OK to close the message box. You can now see the new request in the view with the current status Awaiting Approval. It looks like this:



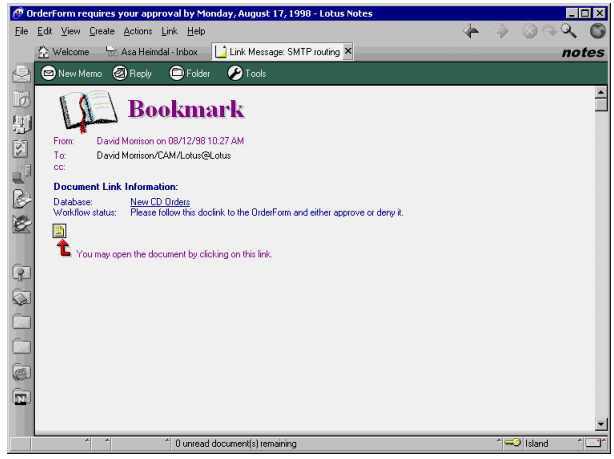
Working With the Request

Assuming the role of Manager, you can now open your mailbox and process the request.

1. Open your mailbox.
2. You can see new mail listed in the Inbox view. This is how the view looks when new approval requests arrive:



3. Open the mail item to display the approval document.



To open the approval request, click the document link icon above the red arrow. The approval document will open.

4. If you are the next approver for this document you will see two action buttons at the top of the screen marked Approve and Deny:

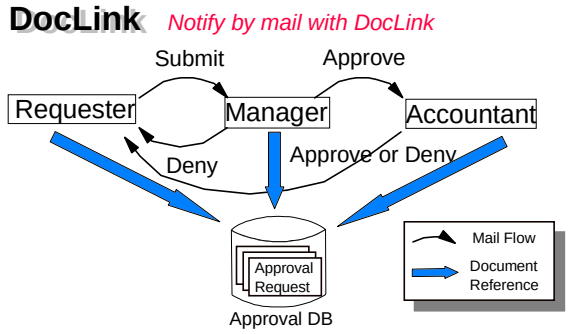
Clicking on either of these buttons displays the Comments dialog box, where you can enter any comments you may have or reasons why you are rejecting a request. When you close the Comments dialog box by clicking OK, an e-mail is sent, either to the next approver or back to the originator of the request if you are the last approver.

Approval Cycle Database: Design

The following section explains the design of the Approval Cycle database.

How Does a Form Flow?

The following figure shows the Doclink type of form flow, which involves three users.



All the notifications to process a flow are sent by e-mail containing a document link to an approval request document located in the Approval Cycle database. The approvers can see the same document submitted by a requester for processing. Only the current approver can take an action (Approve or Deny) to deal with the request. The network path, server access authorization, and appropriate database access controls are needed to share the same database for the workflow.

How Is the Approval Cycle Database Organized?

Relationship Between Forms

The Approval Cycle template has two forms:

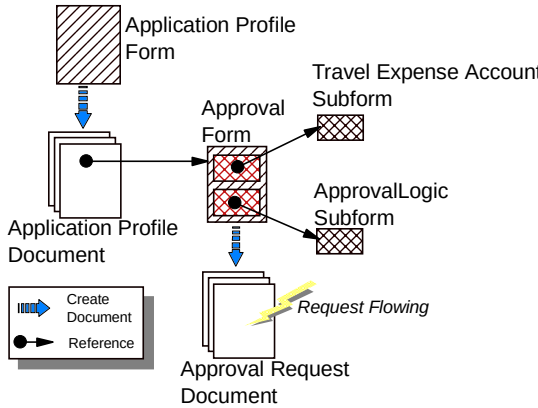
- Application Profile Form
- ApprovalLogic Subform

It is recommended that you create at least one document and one other form to use this template.

An Application Profile document must be based on the Application Profile form and it must describe the workflow properties and specify an Approval form. The Approval form must include the ApprovalLogic subform and your workflow contents, for example, an Order Form.

An Approval Request document is created from the Approval form every time you start a workflow request. Approval status information is shown in the Approval Request document.

The following figure shows the relationship between the forms:



Procedure Calling Sequence and Event Handling

Approval Request Form

An Approval Request form consists of two design elements:

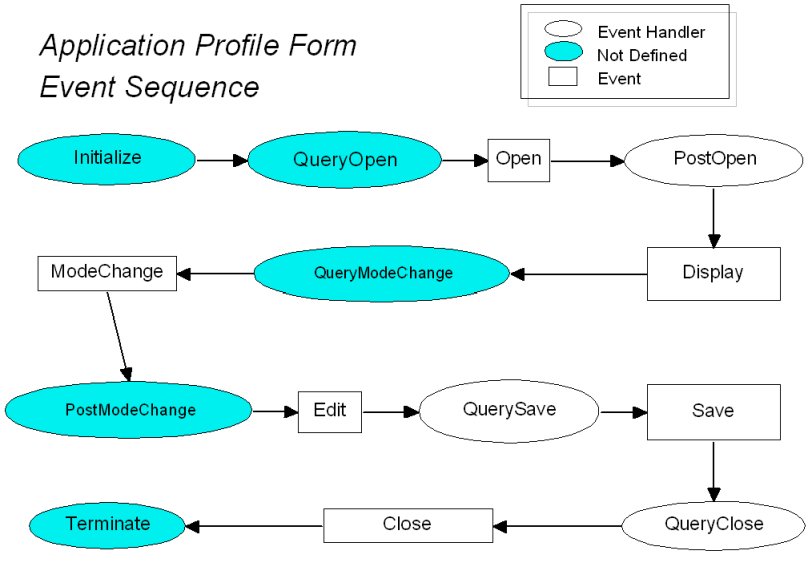
- ApprovalLogic subform
- The form you design to contain your own specific fields

The main workflow form contains some Notes agents, but no LotusScript programs. The ApprovalLogic subform contains a large number of LotusScript programs and many Domino macros, which control the approval request workflow.

Note When you add procedures or modify scripts, be careful to avoid conflicts of variables between global variables and implicit variables used in the ApprovalLogic subform.

Tip If you write LotusScript programs, you should use Option Declare, user-defined classes, and user-defined data types to make the programs clear and safe.

When you create the Approval Request document, the form events defined in the ApprovalLogic subform are inherited into the main form. The picture below shows which events are used and the sequence in which they are triggered.



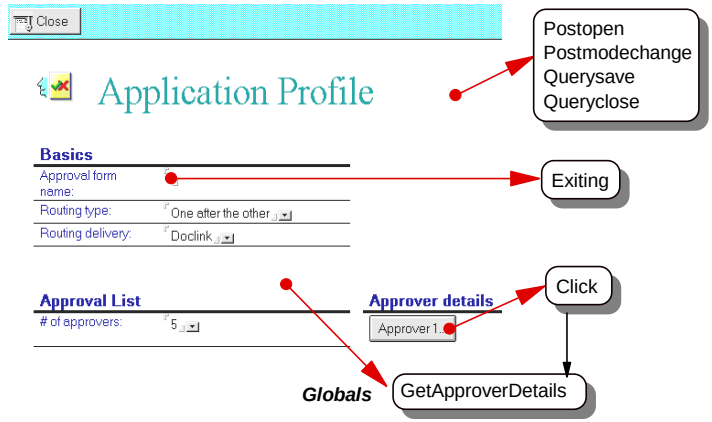
Eclipses displayed in gray are not performed because there are no scripts defined for them.

Note You can use the Domino debugger to see some of the sequences of real-time events. You will not be able to debug the LotusScript code that is executed via the @Function language. For example, you will not be able to debug the Approve and Deny action buttons as they start by executing commands in the formula language. For more information on the debugger, see Chapter 10: Programming for Domino.

Application Profile Form

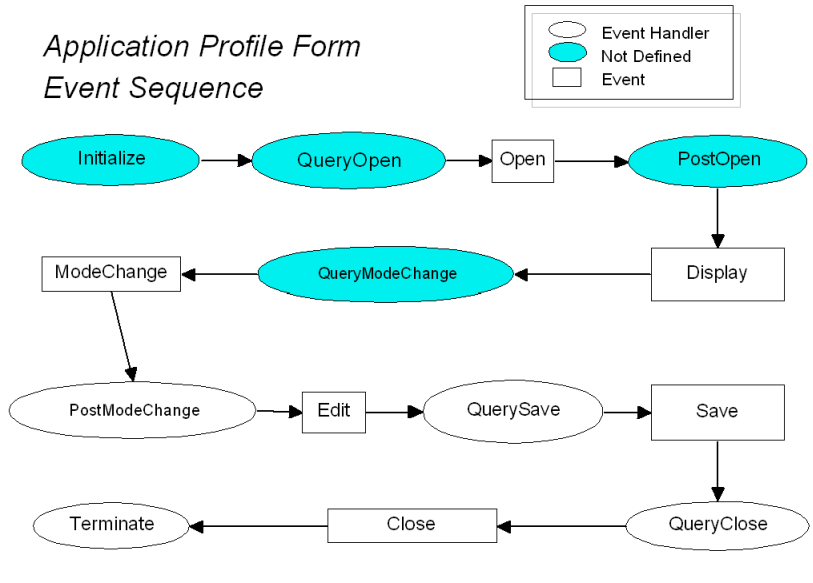
The Application Profile form contains some event handlers, which are included in forms, fields, and buttons. The Exiting event is performed when

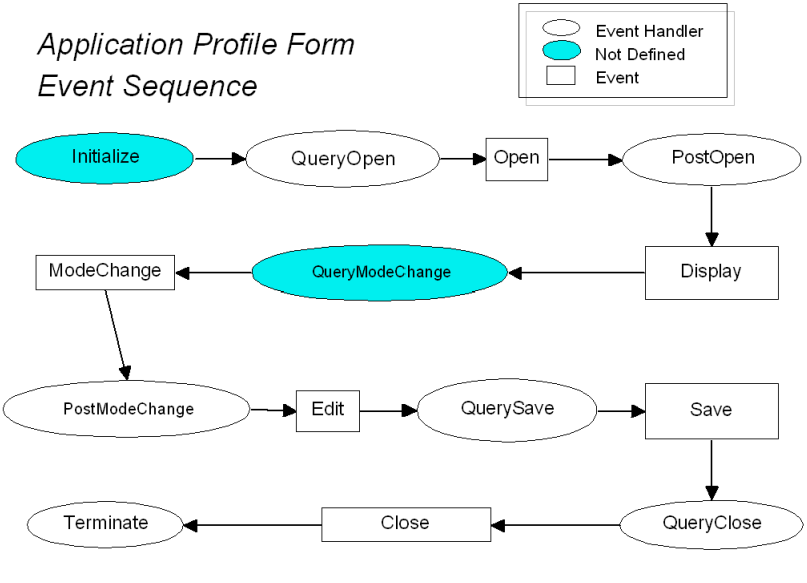
you try to move the cursor from the Approval form name field to another field, as shown in the following figure:



You cannot move the cursor without typing a name, because the Exiting event handler checks the field contents. The Approver button has a Click event handler which is called by a mouse click trigger. The handler calls the global routine GetApproverDetails, which displays a dialog box created by a layout region in the (Approverinfo) form.

The following picture shows an event sequence which occurs when you create the Application Profile document:





Eclipses shown in gray are not performed, because there is no program defined for them.

You can use the Domino debugger to see the sequence of real-time events. For more information on the debugger, see Chapter 10: Programming for Domino.

Approval Cycle Database: Agent

The approval application requires an agent to deal with the due date expiration in this example. When the due date has passed and the approver has not taken any action, the agent processes the approval request depending on the criteria specified in the application profile for the due date.

A Closer Look at the ApprovalLogic Subform

The following section explains how the ApprovalLogic subform works and shows you some of the techniques that you can use in designing your own workflow applications.

We assume here that you have already created your workflow form and have set up an Application Profile document for it.

Note The LotusScript that follows may not look exactly the same as that in the template, as it has been formatted for better clarity.

The Major Fields

The ApprovalLogic subform uses many variables to perform its operation, but there are a few that require particular attention:

Field Name	*Field Type*	*Description*
AuthorizedEditors	Authors	Controls who currently has the ability to edit the document. This is modified in the QuerySave form event and changed from the original author to the next approver, and finally to the form administrator.
NextApprover	Text	Contains the name of the next approver in the workflow process.
Status	Text	Contains the current status of the workflow document as designated in the Application Profile document.
Action	Text	Controls which action button (submit, approve, deny) the current user has clicked and performs the appropriate functions. This field is not displayed on the form, but created via the action buttons.

Creating a New Request

1. To create a new request, the user clicks Create New Request in the All Requests view. This does three things:

   - It builds a list of available Application Profiles from the ApplicationProfiles view and displays them in a dialog box where the user can make selections.
   - It sets an environment variable, *DocType*, to the value of the selected document.
   - It composes the selected document.

   ```
   tList := @DbColumn(""; ""; "ApplicationProfiles"; 1);
   List := @If(@IsError(tList); @Return(@Prompt([OK]; _
   "Error"; "An error has occurred. Please try again.")); _
   tList);
   ENVIRONMENT DocType := @Prompt([OKCANCELLIST]; "New _
   Request"; "Choose one of the following:"; @Subset(List; _
   1); List);
   @PostedCommand([Compose]; DocType)
   ```

2. When the new workflow document is being created, the LotusScript *PostOpen* event is triggered. This code first checks to see if the new document was created via the Create New Request action button by checking the value of the environment variable DocType. If the environment variable does not exist or is blank, then the user is not allowed to proceed.

```
If source.IsNewDoc Then
  DocType = s.GetEnvironmentString("DocType")
  Call s.SetEnvironmentVar("DocType", "None")
  If DocType = "None" Or Isempty(DocType) Then
       Messagebox "This Document must be created via"
                  "the Create New Request action."_
       & " Please remove it from the Create Menus.", 0,_
    "Error"
       Continue = False
       source.Close
       Exit Sub
  End If
```

3. Still within the PostOpen event of the form, LotusScript then finds the selected Application Profile document and copies all the fields into the new document being created. If the profile document cannot be found, the user is informed and is not allowed to continue.

```
  Set view = db.GetView("Application Profiles")\
Set profile = view.GetDocumentByKey(DocType,False)
  If profile Is Nothing Then
       Messagebox "This application will not execute"&_
"correctly without an application profile.",
0 + 64, "Design Error"
  doc.close
       Print
       Exit Sub
  Else
       profile.RemoveItem("Form")
       Call profile.CopyAllitems(note)
  End If
  FormAdmin = note.FormAdmin
  If FormAdmin(0) = "" Then note.FormAdmin = db.Managers
  note.RequesterName = s.CommonUserName
End If
```

Note The form name is removed from the Application Profile document before the fields are copied into the new document. If this was not done, the new document would open using the Application Profile form rather than the correct workflow form. Although the form field is removed from the Application Profile document, it is not actually deleted, as the document is not saved.

4. When a new document is being created, the subroutine InitializeNewDoc() is called from the forms PostOpen event to set up the following approver fields: Approver Names (only if not defined on the Application Profile), Status, Approval Dates, and Comments.

5. If the list of approvers has been defined in the Application Profile as being stored in another database, the GetApproverNames() LotusScript function is executed to retrieve the list of names from the database and the view specified in the Application Profile.

   Note All the fields have been updated in the back-end document, rather than the UIDocument.

6. At the end of the PostOpen event, the fields in the UI document are reloaded from the back-end database and then refreshed to reset the hide-when formulas and computed fields.

```
source.Reload
source.Refresh
```

7. When the document is refreshed, the PostRecalc event for the form is triggered. This event checks whether the current user is trying to edit the approver list and, if not, exits.
8. The new document is then displayed for the user to complete as required.

In summary, when a new workflow document is created, the PostOpen form event validates whether the request was generated from the Create New Request action button by using environment variables. The Application Profile document is located and its fields copied into the new document, and the list of approver fields is updated.

Submitting a New Form for Approval

1. When a new workflow document is completed as required, the user clicks Submit for Approval. This performs the following functions.
   - It checks to see if the field ApprName contains the value "Entered when Submitted." This value is copied from the Application Profile document and indicates that the user has not selected the list of people to send this document to for approval. If they have not made the selection, then an error message is displayed, and they are required to return to the document and enter the approvers' names.
   - The action button sets the value of the field *Action* to the value *Submit*. The field SaveOptions is then set to the value "1" to force the document to be saved and not display the "Do you want to Save this Document" dialog box to the user.

- The document is then saved and closed.

```
@If(@Contains(ApprName; "Entered when submitted");
@Return(@Prompt([OK]; PromptTitle; "Please use the \"Edit
Approver List\" button to enter a valid approver
name.")); "");
FIELD Action := "Submit";
FIELD SaveOptions := "1";
@PostedCommand([FileSave]);
@PostedCommand([FileCloseWindow])
```

2. Before the document is closed, the QuerySave() form event is triggered by the @PostedCommand([FileSave]) function from the action button.
   - This function first sets the global variable DocWasSaved to true so that the QueryClose event knows to remove some fields.
   - It is possible that the user simply wanted to save the document rather than submit it for approval, so a check is made to see if the document was saved via the Submit for Approval action button by testing whether the document contains the Action field:

     ```
     If Not note.HasItem("Action") Then Exit Sub
     ```

   - Next, the IdentifyUser() subroutine is called to rebuild the list of remaining approvers by identifying those approvers that still have a status of none.
   - Using a case statement, the QuerySave event then checks to see which action button the user clicked (Submit, Approve, or Deny) by testing the value of the field Action:

     ```
     Select Case Action(0)
     Case "Submit"
          If (Status(0) = StatusList(4)) Or (Status(0) = _
     StatusList(5)) Then
               For n = 1 To 5
                    note.RemoveItem("ApprStatus")
                    note.RemoveItem("ApprDate")
                    note.RemoveItem("ApprComment")
               Next
          End If
          note.RequestDate = dt.LSLocalTime
          note.Status = StatusList(2)
          SetNextApprover
          SetDueDate
          SendNotification
     ```

- If the action is Submit, the function SetNextApprover() is called to identify the next approver. This subroutine checks to see which routing method has been defined (either Serial or Parallel) and sets the next approver to the first person in the array built from the IdentifyUser() subroutine:

```
Select Case RoutingMethod(0)
'If RoutingMethod is serial then NextApprover is simply
'the next in the list
Case "Serial"
     If NextAppr > Ubound(ApprName) Then
'StatusQualifier is only used while the approval cycle is
'active
          note.RemoveItem("StatusQualifier")
          LastApprover = True
     Else
          NextApprover = ApprName(NextAppr)
          note.NextApprover = NextApprover
          note.StatusQualifier = "by " & NextApprover
     End If
Case "Parallel"
     Select Case Action(0)
'If submitting - the next approver is all approvers
     Case "Submit"
          note.NextApprover = ApproverList
'If Approving - remove CurrentUser from the ApproverList
'and reset NextApprover to the new list
     Case "Approve"
          ListMax = Ubound(ApproverList)
          Redim tmpList(ListMax)
          x = 0
          For y = 0 To ListMax
               If ApproverList(y) <> CurrentUser Then
                tmpList(x) = ApproverList(y)
                x = x + 1
               End If
          Next
          If x = 0 Then
               tmpList(x) = "None"
               LastApprover = True
          End If
          note.NextApprover = tmpList
     End Select
End Select
```

- The subroutine SetDueDate() is called next to set the due dates by which the approver(s) must authorize the document. For a parallel workflow document all the due dates for each of the remaining authorizers are set, and for a serial workflow document, only the next approver's due date is set:

```
Select Case RoutingMethod(0)
'If Parallel - adjust all dates
Case "Parallel"
      For n = 0 To ListMax
            Adjustment = ApprWin(n)
            dt.AdjustDay(Int(Adjustment))
            tmpList(n) = dt.LSLocalTime
            dt.SetNow
      Next
'If Serial - only adjust the date for the Next Approver
Case "Serial"
      DueDate = note.DueDate
      Redim tmpList(NextAppr)
      x = 0
      Forall d In DueDate
            tmpList(x) = d

            If (tmpList(x) = "") Then
                  tmpList(x) = dt.LSLocalTime
            End If

            x = x + 1
      End Forall
      Adjustment = ApprWin(NextAppr)
      dt.AdjustDay(Int(Adjustment))
      tmpList(NextAppr) = dt.LSLocalTime
End Select
note.DueDate = tmpList
```

- The SendNotification() subroutine is then called to mail the details of the workflow document to the next approver. This subroutine creates a new mail document and sets up the required Domino mail fields in order for the document to be mailed. If the workflow document is set up as a serial document, all approvers are sent an e-mail, if it is set up as a parallel workflow document, only the next approver is e-mailed:

```
Select Case RoutingMethod(0)
'If it is Serial - mail gets sent to the NextApprover
      Case "Serial"
            SendTo = NextApprover
            Recipient = NextApprover
'If it is Parallel - mail gets sent to all approvers
      Case "Parallel"
            SendTo = ApproverList
```

```
                Recipient = "all Approvers"
  End Select
  Subject = WorkflowObject(0) & " requires your approval by
  " & Format(DueDate(NextAppr), "Long Date")
  maildoc.DueDate = Format(DueDate(NextAppr), "Long Date")
  maildoc.FlowStatus = "Please follow this doclink to the "
  & WorkflowObject(0) & " and either approve or deny it."
  'Put a doclink in the Body field and populate the other
  fields on the Bookmark mail form
  Call rtitem.AppendDocLink(note, "Doclink to " &
  WorkflowObject(0))
  maildoc.InheritedDbTitle = db.Title
  maildoc.Form = "Bookmark"
  maildoc.SendTo = SendTo
  maildoc.Subject = subject
```

- The mail message is then sent:

```
Call maildoc.Send (False)
```

- Finally, the UI document is then reloaded from the back-end document to update the UI document fields.

3. After the mail item is sent and the QuerySave event has finished, the @PostedCommand([FileCloseWindow]) function is performed. This triggers the QueryClose form event. This event cleans up any temporary fields that were used in the document:

```
'Remove the Action field and any field that begins with d_
'This includes all temporary fields and all display only
'fields
'(a back-end save will save computed for display fields
'unless you do this)
  note.RemoveItem("Action")
  ItemList = note.Items
  Forall n In ItemList
        If Left(n.Name, 2) = "d_" Or Left(n.Name, 2) = "D_"
Then n.Remove
  End Forall
```

4. The QueryClose event then executes the ResetAuthorNames() subroutine. This subroutine removes the document author from the AuthorizedEditors Authors field and adds the name of the next approver:

```
Select Case NewStatus(0)
'If approvals are done - the FormAdmin is the only editor
  Case StatusList(6), StatusList(7)
        note.AuthorizedEditors = FormAdmin(0)
'If approvals are not started - the Requester is the only
'editor
  Case StatusList(0), StatusList(1)
```

```
        note.AuthorizedEditors = RequesterName(0)
    Case Else
    tmpList(0) = FormAdmin(0)
    x = 1
    Forall n In NextApprover
        tmpList(x) = n
        x = x + 1
    End Forall
    note.AuthorizedEditors = tmpList
End Select
```

5. The document is then finally closed.

In summary, when Submit is clicked, the field Action is set to Submitted and the QuerySave event is triggered. The QuerySave event sets up a list of remaining approvers, sets the next due date and sends an e-mail to the next approver(s). Finally, the QueryClose event cleans up the document and sets the author of the document to be the next approver.

Approving a Request

When an approver clicks Approve, the following events occur:

1. On the Approve action button, the value of the field Action is set to Approve and a dialog box is displayed to enable the approver to enter any comments. The document is then saved and closed:

```
@Command([EditDocument]; "1");
FIELD Action := "Approve";
FIELD SaveOptions := "1";
@If(CommentsAllowed = "Yes";
@DialogBox("EnterComments"; [AutoVertFit] : [AutoHorzFit]);
PromptTitle);
@PostedCommand([FileSave]);
@PostedCommand([FileCloseWindow])
```

2. The @PostedCommand([FileSave]) command triggers the QuerySave form event. The QuerySave event updates the current approver status, date and comment fields:

```
Select Case Action(0)
Case "Approve"
'Set approval status for the current approver
  ApprStatus = note.ApprStatus
  ApprStatus(CurrentApprover) = StatusList(3)
  note.ApprStatus = ApprStatus
'Set approval date for the current approver
  ApprDate = note.ApprDate
  ApprDate(CurrentApprover) = dt.LSLocalTime
  note.ApprDate = ApprDate
'Set comment value for the current approver
```

```
  NewComment = note.d_ApprComment
  ApprComment = note.ApprComment
  ApprComment(CurrentApprover) = NewComment(0)
  note.ApprComment = ApprComment
'Set the next approver, the expiration and send mail to the
appropriate person(s)
  SetNextApprover
```

3. The QuerySave event then executes the SetNextApprover() subroutine. This subroutine iterates through the remaining approvers to select the next approver or, if this is the last approver, sets the LastApprover variable to be true:

```
Case "Approve"
  ListMax = Ubound(ApproverList)
  Redim tmpList(ListMax)
  x = 0
  For y = 0 To ListMax
       If ApproverList(y) <> CurrentUser Then
            tmpList(x) = ApproverList(y)
            x = x + 1
       End If
  Next
  If x = 0 Then
       tmpList(x) = "None"
       LastApprover = True
  End If
  note.NextApprover = tmpList
End Select
```

4. If the current approver is the last approver, then the field NextApprover is removed from the document:

```
If LastApprover Then
  If Action(0) = "Approve" Then note.Status = StatusList(7)
  note.RemoveItem("NextApprover")
End If
```

5. Returning to the QuerySave event, an e-mail is then sent by executing the SendNotification() subroutine if the selected routing method is set up as serial or if this is the last approver. If the routing method was set up as parallel, then each person would already have been sent an e-mail when the document was originally submitted:

```
If RoutingMethod(0) = "Serial" Then
  If Not LastApprover Then SetDueDate
End If
If RoutingMethod(0) = "Serial" Or LastApprover Then
SendNotification
```

6. The SendNotification() subroutine first checks to see if this is the last approver. If so, the document originator is sent an e-mail informing them that their workflow request was successful. If there are more approvers, then the next approver is sent an e-mail.
7. The QuerySave event then finishes and the Approve action button @Command([FileCloseWindow]) is executed, which triggers the QuerySave form event. Again, this subroutine cleans up any temporary variables and sets the document author to that of the next approver.

In summary, when an approver clicks Approve, the current approver fields are updated, the next approver is identified, and an e-mail is sent either to the next approver or to the document originator if there are no remaining approvers. Finally, if there are further approvers, the AuthorizedEditors field is updated to the name of the next approver.

Denying a Request

When an approver denies a request the following events occur:

1. The Deny action button sets the value of the Action field to Deny, prompts the approver for the reason for denying the request, and saves and closes the document:

```
@Command([EditDocument]; "1");
FIELD Action := "Deny";
FIELD SaveOptions := "1";
@If(CommentsAllowed = "Yes";
@DialogBox("EnterComments"; [AutoVertFit] : [AutoHorzFit]);
PromptTitle);
@PostedCommand([FileSave]);
@PostedCommand([FileCloseWindow])
```

2. When the @PostedCommand([FileSave]) command is executed, the QuerySave form event is triggered. When the action is set to Deny, the QuerySave event sets the document status field to Deny, removes the NextApprover field, and e-mails the document originator by calling the SendNotification() subroutine.
3. The @PostedCommand([FileCloseWindow]) command is then executed. This triggers the QueryClose form event which removes any temporary fields from the document and calls the ResetAuthorNames() subroutine.
4. The ResetAuthorNames() subroutine sets the value of the document author field, AuthorizedAuthors, to the form administrator designated in the Application Profile document.

Summary

In this chapter we have discussed how to create, set up, and use the Approval Cycle template. We have also looked in detail at the LotusScript code used within the ApprovalLogic subform to better understand how an ad hoc workflow application can be designed.

Appendix A
Domino URLs

Domino uses URLs to access servers, databases, and other components of a Web site, and display them to Web users. Knowing Domino URL commands allows you to design links or enter commands directly into a browser to navigate a Domino site or to reach specific components quickly.

Domino URL Command Syntax

Domino URL commands have the following syntax:

```
http://Host/Database/DominoObject?Action&Arguments
```

where:

Host

DNS entry or an IP address

Caution You cannot use the Domino Server name unless it has a DNS entry.

Database

Database can be one of the following:

- The database file name with the path relative to notes\data.
- The database Replica ID.

DominoObject

A Domino construct (a view, document, form, navigator, agent, etc.)

Action

The desired operation on the specified Notes object, for example, ?OpenDatabase, ?OpenView, ?OpenDocument, ?EditDocument, ?OpenForm, ?ReadForm and so on.

Arguments

A qualifier of the action. For example, Count = 10 combined with the ?OpenView action limits the number of rows displayed in a view to 10.

Use the following guidelines when working with Domino URLs:

1. Special identifiers used in Domino URL commands include: $defaultView, $defaultForm, $defaultNav, $searchForm, $file, $icon, $help, $about. These special identifiers are described in the following sections of this appendix.
2. *DominoObject* can be any of the following:
   - For a database, it is the database name or replicaID itself.
   - For other objects, the Domino object name or alias, universal ID, NoteID or special identifier. For example, to specify a view in a URL, you can use any of the following: the view name, view universal ID, view Note ID, or $defaultView.

   A Domino object name and universal ID are identical in all replicas of a database, but the NoteID will probably change in database replicas. Therefore, it is best to use the Domino object name or universal ID in URLs.

   Note We will not show NoteID and universal ID in this section.
3. Action can be explicit or implicit.
   - Examples of explicit actions include ?OpenServer, ?OpenDatabase, ?OpenView, ?OpenDocument, ?OpenForm, and ?EditDocument.
   - Examples of implicit actions include ?OpenDocument, ?OpenView, and ?OpenDatabase.
   - If you do not specify an action, Domino defaults to the ?Open action.
4. Append the Login argument to any Domino URL to require user authentication.

   Because URLs may not contain spaces, use the **+** (plus sign) as a separator. For example:

   ```
   http://www.testR5.com/discussion.nsf/By+Author
   ```
5. Separate arguments with & (ampersands). For example:

   ```
   http://www.testR5.com/leads.nsf/By+Salesperson?OpenView&Exp
   andView
   ```
6. Separate hierarchical names with **/** (slashes). For example, to open a view named Docs\By Author in a database named Discussion, enter:

   ```
   http://www.testR5.com/discussion.nsf/Docs/By+Author
   ```

Opening Servers, Databases, and Views

The following commands access servers, databases, views, About documents, help documents, and database icons.

OpenServer

Syntax:

```
http://Host
```

Example:

```
http://www.testR5.com/?OpenServer
```

OpenDatabase

Syntax:

```
http://Host/DatabaseFileName?OpenDatabase
```

```
http://Host/DatabaseReplicaID?OpenDatabase
```

Examples:

```
http://www.testR5.com/leads.nsf?OpenDatabase
```

```
http://www.testR5.com/sales/discussion.nsf?OpenDatabase
```

```
http://www.testR5.com/852562F3007ABFD6?OpenDatabase
```

OpenView

Syntax:

```
http://Host/Database/ViewName?OpenView
```

```
http://Host/Database/$defaultview?OpenView
```

Note ViewName can be also an alias of the view.

Examples:

```
http://www.testR5.com/leads.nsf/By+Salesperson?OpenView
```

```
http://www.testR5.com/leads.nsf/$defaultview?OpenView
```

Arguments (Optional)
Append optional arguments to refine the URL. Combine any of the following arguments for the desired result.

- **Start** = *n* Where *n* is the row number to start with when displaying the view. The row number in a hierarchical view can include subindexes; for example, Start=3.5.1 means thst the view will start at the third main topic, subtopic 5, document 1.
- **Count** = *n* Where *n* is the number of rows to display.
- **ExpandView** Displays the view in expanded format.

- **CollapseView** Displays the view in collapsed format.
- **Expand = *n*** Where *n* is the row number to display in expanded format in a hierarchical view. Do not combine this argument with the ExpandView or CollapseView0 arguments.
- **Collapse = *n*** Where *n* is the row number to display in collapsed format in a hierarchical view. Do not combine this argument with the ExpandView or CollapseView arguments.
- **StartKey= string** Open a view starting from the first document that matches the key. The key is selected on the first sorted column.

Examples:

```
http://www.testR5.com/leads.nsf/By+Salesperson?OpenView&ExpandView
```

```
http://www.testR5.com/leads.nsf/By+Salesperson?OpenView&Start=3&Count=15
```

Note To open the first document in a view, use keyword $First and the following syntax: http://host/database/view/$First.

OpenAbout

Use the OpenAbout command to access the "About database" document.

Syntax:

```
http://Host/Database/$about?OpenAbout
```

Examples:

```
http://www.testR5.com/leads.nsf/$about?OpenAbout
```

OpenHelp

Use the OpenHelp command to access the Help document.

Syntax:

```
http://Host/Database/$help?OpenHelp
```

Examples:

```
http://www.testR5.com/leads.nsf/$help?OpenHelp
```

OpenIcon

Use the OpenIcon command to access the database icon.

Syntax:

```
http://Host/Database/$icon?OpenIcon
```

Examples:

```
http://www.testR5.com/leads.nsf/$icon?OpenIcon
```

Login Argument

Append the Login argument to any Domino URL to force user authentication, regardless of the database access control list. This ensures that anonymous Web users who weren't initially prompted for a name and password when they entered the site, are required to supply a name and password to complete tasks that require user identity.

Note Do not use this argument to let a Web user switch login. In fact if a user has already logged in with a certain UserID and Password, the login argument will be ignored. The only way to re-login as a new user is to close and then restart the browser.

Syntax:

You can use this argument with any URL, but the most common are:

```
http://Host?OpenServer&Login
```

```
http://Host/Database?OpenDatabase&Login
```

Examples:

```
http://www.testR5.com?OpenServer&login
```

```
http://www.testR5.com/sales/leads.nsf?OpenDatabase&login
```

Opening Framesets, Pages, Forms, Navigators, and Agents

The following commands open framesets, pages, forms, navigators, and agents in a database.

OpenFrameset

Syntax:

```
http://Host/Database/FramesetName?OpenFrameset
```

Note FramesetName can be also an alias of the frameset.

Examples:

```
http://www.testR5.com/products.nsf/Product?OpenFrameset
```

OpenPage

Syntax:

```
http://Host/Database/PageName?OpenPage
```

Note PageName can be also an alias of the page.

Examples:

```
http://www.testR5.com/products.nsf/Product?OpenPage
```

OpenForm

Syntax:

```
http://Host/Database/FormName?OpenForm
```

```
http://Host/Database/$defaultform?OpenForm
```

Note FormName can be also an alias of the form.

Examples:

```
http://www.testR5.com/products.nsf/Product?OpenForm
```

```
http://www.testR5.com/products.nsf/$defaultform?OpenForm
```

Arguments (Optional):
ParentUNID = The *Universal ID* of the parent document, to respond to or to inherit from. Remember that if you are composing a response document or you want to inherit formulas from another document on the Web, you cannot select the parent document.

Syntax for using this argument:

```
http://Host/Database/FormUniversalID?OpenForm&ParentUNID
```

Examples:

```
http://www.testR5.com/products.nsf/40aa91d55cle4c8285256363004d
c9e0?OpenForm&ParentUNID=6bc72a92613fd6bf852563de001f1a25
```

Note You can also use the ?OpenRead or ?ReadForm command to open a form only in read mode. This is useful when you don't want to display the submit button.

OpenNavigator

Syntax:

```
http://Host/Database/NavigatorName?OpenNavigator
```

```
http://Host/Database/$defaultNav?OpenNavigator
```

Examples:

```
http://www.testR5.com/products.nsf/Main+Navigator?OpenNavigator
```

```
http://www.testR5.com/products.nsf/$defaultNav?OpenNavigator
```

Note $defaultNav opens the folder navigator in a database.

OpenAgent

Syntax:

```
http://Host/Database/Agentname?OpenAgent
```

Examples:

```
http://www.testR5.com/sales/leads.nsf/Process+New+Leads?OpenAge
nt
```

Opening, Editing, and Deleting Documents

The following commands manipulate documents in a database:

OpenDocument

Syntax:

```
http://Host/Database/View/DocumentKey?OpenDocument
```

DocumentKey contains the contents of the first sorted column in the specified view. For more information, see the section on opening documents by key later in this chapter.

Note The *View* is a necessary parameter because Domino uses the *Form Formula* of a view to determine the form to use when displaying the document (either using a Notes Client or a browser). If this formula is set to nothing, Domino uses the form written in the "Form" field of the document.

Examples:

```
http://www.testR5.com/products.nsf/By+Part+Number/PC156?OpenDoc
ument
```

EditDocument

Syntax:

```
http://Host/Database/View/Document/?EditDocument
```

Example:

```
http://www.testR5.com/products.nsf/By+Part+Number/PC156?EditDoc
ument
```

DeleteDocument

Syntax:

```
http://Host/Database/View/Document?DeleteDocument
```

Example:

```
http://www.testR5.com/products.nsf/By+Part+Number/PC156?DeleteD
ocument
```

CreateDocument

The CreateDocument command is used as the POST action of an HTML form. When the user submits a form, Domino obtains the data entered in the form and creates a document.

Syntax:

```
http://Host/Database/Form/?CreateDocument
```

Example:

```
http://www.testR5.com/products.nsf/part?CreateDocument
```

SaveDocument

The SaveDocument command is used as the POST action of a document being edited. Domino updates the document with the new data entered in the form.

Syntax:

```
http://Host/Database/View/Document?SaveDocument
```

Example:

```
http://www.testR5.com/products.nsf/a0cefa69d38ad9ed8525631b006582d0/4c95c7c6700160e2852563df0078cfeb?SaveDocument
```

Opening an Anchor Link

As seen in the previous section on Domino Links, a new syntax exists for opening a document at a specified area of its text.

Syntax:

```
http://Host/Database/View/Document?OpenDocument#AnchorLabel
```

Example

```
http://localhost/RedBook.nsf/66aa0bd809ee8316852564d8004e7ddc/503d4ee771078042852564e400598a8e?OpenDocument#Paragraph+3
```

The Anchor label is created when you build the anchor inside the linked document. If no name has been entered for that link, a default number is added (Example: "#_0").

Opening Documents by Key

The following commands allow you to open a document by key, or to generate a URL to link to a document by key.

Using Domino URLs to Access a Document

To open a document by key, create a sorted view with the sort on the first key column. You then can use a URL to open the document:

Syntax:

```
http://Host/DatabaseName/View/DocumentName?OpenDocument
```

Example:

```
http://www.testR5.com/register.nsf/Registered+Users/Jay+Street?OpenDocument
```

Where View is the name of the view, and DocumentName is the string, or key, that appears in the first sorted or categorized column of the view. Use this syntax to open, edit, or delete documents, and to open attached files.

Domino returns the first document in the view with a column key that exactly matches the DocumentName.

There may be more than one matching document; Domino always returns the first match. The key must match completely for Domino to return the document. However, the match is not case-sensitive or accent-sensitive.

Note that the view can be a view Note ID, UNID, or view name. In addition, the implicit form of any of these commands will work when appropriate. (EditDocument and DeleteDocument must be explicit commands).

Advantages of Using Keys Instead of Universal ID

Let's suppose that the following two URLs refer to the same document, which is the personal document of "Jay Street" (for instance):

```
http://www.testR5.com/register.nsf/
Registered+Users/Jay+Street?OpenDocument
```

```
http://www.testR5.com/register.nsf/
a0cefa69d38ad9ed8525631b006582d0/
4c95c7c6700160e2852563df0078cfeb?OpenDocument
```

The first URL is, of course, much more understandable than the second URL and therefore, easier to remember. However, this is not the only advantage of using keys to refer to Domino objects.

Imagine that the "Jay Street" document has been deleted and replaced by a new copy. The reasons for using this event might be, for example, where all documents of that kind are deleted and rebuilt every night by an agent that takes updated data from an external data source. In this situation, the first URL continues to reference the "Jay Street" document, while the second URL fails because the Universal ID of the new document will be different from the former. This means that if a user has stored the first URL as a bookmark he will have no problem finding his document again, and so will avoid repeated searches.

However, you should also consider the following implications of using URL by keys:

- A developer must use @formulas to calculate every URL using keys, and must hide or inhibit all the URLs automatically generated by Domino. This means that you don't need to copy and paste links to Domino objects.
- The access to documents is faster when using Universal ID than using keys, as the server must read the view index to reach the document when using keys.

Opening Image Files, Attachments and OLE Objects

The following commands open files and objects within a document:

OpenElement

Use the ?OpenElement command to access file attachments, image files, and OLE objects. This is very useful when writing passthru HTML; for example, when you need to display an image that is attached into another Notes document you can use the URL described here.

Using OpenElement with File Attachments

Syntax:

```
http://Host/Database/View/Document/$File/Filename?OpenElement
```

```
http://Host/Database/View/Document/$File/
InternalFileName/Filename?OpenElement
```

Example:

```
http://www.testR5.com/lproducts.nsf/
By+Part+Number/SN156/$File/spec.txt?OpenElement
```

If more than one attached file has the same name, the URL includes both the internal file name as well as the external name. Since the internal file name is not easily determined, make sure that all attached files have unique names.

Domino treats all file attachment OpenElement commands as implicit commands, because some browsers require that the URL ends with the attached file name.

Using OpenElement with Image Files

It is used to retrieve an image that is an *imported* image into a Notes field.

Syntax:

```
http://Host/Database/View/Document/FieldName/
FieldOffset?OpenElement&FieldElemFormat=ImageFormat
```

FieldOffset is represented by *xx.yy*, where *xx* is the field number, and *yy* is the byte offset into the field.

ImageFormat is either GIF or JPEG. If the FieldElemFormat is not entered, Domino assumes that the image file format is GIF.

To see an example, try to import a GIF image into a RichText field and launch the preview in browser. Next, look at the HTML source for that image. You should find something like this:

```
<IMG SRC="/database/view/document/FieldName/xx.yy
?OpenElement&FieldElemFormat=GIF>
```

Using OpenElement with OLE Objects

Syntax:

```
http://Host/Database/View/Document/FieldName/
FieldOffset/$OLEOBJINFO/FieldOffset/obj.ods?OpenElement
```

Note The current URL syntax for referencing images and objects in Notes documents — specifically the FieldOffset — makes it impractical to create these URLs manually. As an alternative, you may paste the actual bitmap or object in place of the reference, create URL references to files stored in the file system, or attach the files to the documents.

Searching for Text with Domino Search URLs

The following commands allow you to search a Domino site or to search individual databases within a Domino site:

SearchSite

Syntax:

```
http://Host/Database/[$SearchForm]?SearchSite[ArgumentList]
```

Where *$SearchForm* and *ArgumentList* are optional arguments. The special identifier *$SearchForm* indicates that Domino will present a search site form for search input. If this identifier is provided, ArgumentList is ignored.

Examples:

```
http://www.testR5.com/mercsrch.nsf/$SearchForm?SearchSite
```

```
http://www.testR5.com/mercsrch.nsf/?SearchSite&
Query=product+info+requests;1;;0;FALSE
```

SearchView

Syntax:

```
http://Host/Database/View/[$SearchForm]?SearchView[Argument
List]
```

Where *$SearchForm* and *ArgumentList* are optional arguments.

The special identifier $SearchForm indicates that Domino will present a search view form for search input. If this identifier is provided, the ArgumentList is ignored.

Examples:

```
http://www.testR5.com/products.nsf/By+Product+Number/
$SearchForm?SearchView
```

```
http://www.testR5.com/products.nsf/By+Product+Number/
?SearchView&Query=PC156;3;;0;TRUE;
```

Argumentlist (Optional)

The ArgumentList must contain the Query argument; in addition, it may contain any or all of the other arguments.

- **Query** = the search string
- **Count**=[*n*]
- **SearchFuzzy**=[TRUE, FALSE], default = FALSE
- **SearchOrder**=[1,2,3] default = 1.
  - 1 = "By Relevance"
  - 2 = "By Date Ascending"
  - 3 ="By Date Descending"
- **SearchMax**=[n], 0 default= 0 (meaning all)
- **SearchWV**=[TRUE, FALSE], default = TRUE
- **Start**=[*n*]
- **OldSearchQuery**

  Repeats the last query.

Note The SearchThesaurus argument is ignored by the R5.0 search machine.

Appendix B
Shortcuts

In this appendix we have put together a list of shortcuts that are available when working with Domino.

Caution Some of these shortcuts are language dependent. The shortcuts listed are based on the US English version of Domino. They may vary in other language versions. Refer to the documentation if you are in doubt.

Workspace Keys

Action	*Shortcut*
Get context-sensitive help	F1
Exit Notes	ALT+F4
Log off Notes (revoke password login but leave Notes running)	F5
Restore Notes program window to default size	ALT+F5
Cycle through open panes	F6
Cycle through open windows	CTRL+F6
Update all fields in current document, view, or workspace	F9
Rebuild all views in current document, view, or workspace	SHIFT+F9
Minimize active window and cascade other active windows	CTRL+F9
Maximize all open windows	CTRL+F10
Access the menu bar so you can use arrow keys to choose commands	F10 or ALT
Create a new memo	CTRL+M
Send a memo	ALT+3
Create a new database	CTRL+N
Open a database (add a database to workspace)	CTRL+O
Close a document	ESC

continued

Action	*Shortcut*
Cancel a server operation	CTRL+BREAK
Open the Scan Unread dialog box	TAB
Enter and follow a URL to a World Wide Web site	CTRL+L
Move from one workspace tab to another	right arrow or left arrow
Move from any workspace tab except Replicator to the first workspace tab	HOME
Move from any workspace tab except Replicator to the Replicator tab	END
Move from one database icon to another	arrow
Move a database icon	CTRL+SHIFT+arrow To stop moving, press ENTER.
Delete a database icon, or mark document for deletion	DEL
Move from one Replicator entry to another	up arrow or down arrow
Move from any Replicator entry to the first entry	HOME or PAGE UP
Move from any Replicator entry to the last entry	END or PAGE DOWN

Function Keys

Action	*Shortcut*
Get context-sensitive Help	F1
Enlarge text to next available point size	F2
Reduce text to next available point size	SHIFT+F2
Go to next selected document	F3
Go to previous selected document	SHIFT+F3
Go to next unread document	F4
Go to previous unread document	SHIFT+F4
Exit Notes	ALT+F4
Log off Notes (revoke password login but leave Notes running)	F5
Restore Notes program window to default size	ALT+F5
Cycle through open windows	CTRL+F6
Indent the first line in a paragraph	F7
Outdent first line in a paragraph	SHIFT+F7

continued

Action	*Shortcut*
Indent entire paragraph	F8
Outdent entire paragraph	SHIFT+F8
Update all fields in current document, view, or workspace	F9
Rebuild the current view	SHIFT+F9
Rebuild all views in current database	SHIFT+CTRL+F9
Minimize active window and cascade other active windows	CTRL+F9
Maximize all open windows	CTRL+F10
Access the menu bar so you can use arrow keys to choose commands	F10
Access the action bar so you can use number keys to choose buttons	ALT

Dialog Boxes

Action	*Shortcut*
Go to the next option or set of options	TAB
Go to the previous option or set of options	SHIFT+TAB
Highlight next item in a list box or set of options	down arrow or right arrow
Highlight previous item in a list box or set of options	up arrow or left arrow
Accept the default or highlighted selection(s)	ENTER
Cancel any changes and close the dialog box	ESC

Keys for Editing Documents or Designing Domino Objects

Action	*Shortcut*
Select contents	CTRL+A
Bold selected text	CTRL+B
Copy selected text or object	CTRL+C
Edit a document (does not apply to forms or subforms)	CTRL+E
Find and replace	CTRL+F
Find next and replace	CTRL+G
Italicize selected text	CTRL+I

continued

Action	*Shortcut*
Format paragraphs (margins, tabs, line spacing, and so on)	CTRL+J
Format text (font, size, color, and so on)	CTRL+K
Change selected text to normal	CTRL+T
Underline selected text	CTRL+U
Insert page break	CTRL+SHIFT+L
Paste	CTRL+V
Cut selected text or object	CTRL+X
Clear selected text or object	DEL
Undo last action	CTRL+Z
Save the current element	CTRL+S
Close the current element	CTRL+W
Close or open (toggle) the InfoBox	ALT+ENTER

Appendix C
CORBA Internals

In Chapter 11 we described how the information is directed between the client and the server. We know that CORBA architecture must reside in both the client and the server and that the IIOP protocol transfers the information between the client and that server over the TCP/IP-network. Furthermore, we know that the ORB enables the requests between client and server. This section will explain the CORBA components in Domino so that you can better understand what is really happening when you use CORBA applets in your application.

CORBA is made up of a number of different components that make this communication possible. The following section introduces these CORBA components:

CORBA Client Components

The client is any application requiring remote services from a CORBA-enabled server. With Domino, a Java program doesn't have the local APIs, so it must use local CORBA stubs that invoke the corresponding remote request on the server.

Client IDL Stubs

From the client's perspective, these local stub classes are the same as the actual implementations. Internally these stub classes act as a local proxy for the remote server objects and define how clients invoke corresponding remote services on servers. Once a successful request has been made to instantiate a remote server object, a reference ID to that object is returned. Future method requests on that remote object are sent with the reference id to the remote server and executed on the remote object via the CORBA server components with data being returned if required. This is seamless to the programmer.

The client stubs are created by first defining your server interfaces using the CORBA Interface Definition Language (IDL). This IDL is language-independent syntax and it defines the types of objects, their attributes, the methods they export, and the method parameters. There must be one IDL definition per interface. You can think of an interface as a class definition without the implementation. From a Domino API perspective, there is one IDL definition field per Domino Object Model C++ class that we wish to expose to CORBA.

The IDL file is pre-compiled into the language of the client and the server by a CORBA pre-compiler. This would require two compiles for the Domino implementation, one for the Java client and the second for the server C++ Domino APIs. The client and server-side IDL stubs are then produced.

Within Domino 5.0, the Java client IDL stubs are contained in the lotus.domino package in the NSCO.jar file. This jar file also contains the Java client ORB classes. This jar file must be imported with the applet classes into Domino or accessible to a Java standalone application.

The client stub also includes code that encodes and decodes (marshaling) the requested operation and its parameters into a flattened message format, General Inter-ORB Protocol (GIOP), that can be sent to the server. GIOP is the message and data format protocol for communications across the TCP/IP. GIOP defines message formats for all ORB request/reply semantics. This message is passed to the client ORB.

Client ORB
The ORB (Object Request Broker) is the transportation bus for CORBA object requests to and from remote objects.

The transport protocol between ORBS, Internet Inter-ORB Protocol (IIOP), specifies how GIOP messages are exchanged over the TCP/IP network. You don't need to worry about GIOP, because it is mapped to TCP/IP.

Note To be CORBA 2.0 compatible, an ORB must support GIOP over TCP/IP.

The Domino R5.0 Java client ORB is a lighter version for server ORB and performs the following functions:

- HTTP tunneling allows the client side to pass through firewalls using HTTP tunnels.
- Security allows the client to use SSL to create authenticated sessions with the server.

Client Side Objects (CSO)
When a user uses a CORBA applet, the CORBA components, including the Client Side Objects (CSO), are loaded onto the client. These classes feature caching, helper and holder classes, a binary compatibility layer, and the ability to run on other protocols, for example, DCOM. These are transparent to the developer. Helper classes contain methods that manipulate IDL types. A helper Java class is defined for each IDL Type and interface. Holder classes provide the parameter passing modes that Java doesn't provide.

Note This option is not yet available in Domino R5.0, but it is coming in a future release.

CORBA Server Components

On retrieving IIOP requests, the server ORB uses the Basic Object Adapter in combination with the Implementation Repository to pass parameters and method requests to the required server object via the server stubs. The server stubs communicate with the actual Object Implementation, which in Domino's case is the C++ back-end DOM classes.

Basic Object Adapter (BOA)

BOA is a run-time core ORB communication service for instantiating server objects, passing requests and assigning object IDs (object references).

Server IDL Stubs

The server stub provides interfaces to each service provided by the server. In Domino there will be a server stub for each back-end DOM C++ class. These C++ stubs are created using an IDL compiler as described in the client IDL stub. In other words, the IDL compiler translates IDL to your target language source code (skeletons) and then this skeleton calls your objects.

Another function of the R5.0 ORB is the static loading of the DOM server IDL stubs via a shared library at startup.

Implementation Repository

This is a runtime repository for object information, such as the classes that a server supports, and which objects are instantiated and their IDs.

Domino R5.0 Implementation

The R5.0 ORB allows the Domino Object Model to load and respond to client IIOP requests. Its primary use will be for doing client-side processing in Domino Web applications.

The R5.0 ORB will be a modified version of the IBM ORB. Some of the original IBM ORB will be stripped out. Improvements have been made to address such issues as scalability. Other customer created ORBs may also run on the Domino server.

This ORB will be released as a DLL (NORB.DLL). To load the server ORB (and other CORBA components outlined above) into memory on the Domino server, the administrator loads the DIIOP server task (ndiiop.exe). This server addin process links with the ORB (norb.dll) and loads it into memory. This process can be loaded at the server startup by placing it in the notes.ini ServerTasks line.

```
ServerTasks = ...., DIIOP
```

The server ORB listens to IIOP requests via an available port to HTTP. This port can be changed via the server document.

Communicating

When the user opens their applications via the Web CORBA starts to run. In the following section, we explain in which order these CORBA components run, and what they do when the user opens the CORBA applets in the Domino application.

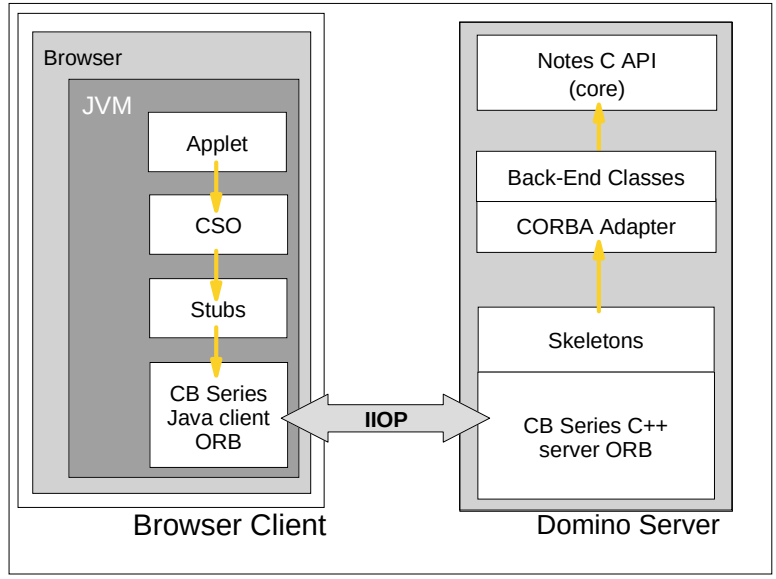
From the client to the server (the request):

1. First, the client application calls the stub class method.
2. The stub passes the request and parameters to the client ORB.
3. The client ORB then packages the request and parameters into the IIOP and sends it to the server ORB.
4. The server ORB now unpacks it and passes it to the static server stubs (server skeleton).
5. The skeleton now calls the server object class method and the method runs.

From the server to the client (the response):

1. When the server object methods have run, the result (if there is any), it is passed to the server skeleton.
2. The skeleton then passes it to the server ORB.
3. The server ORB packages the information/result to the IIOP and sends it to the client ORB.
4. The client ORB unpackages the information that was sent and passes it to the waiting stub.
5. The stub passes the information/result to the calling application.
6. The information/result is shown by user.

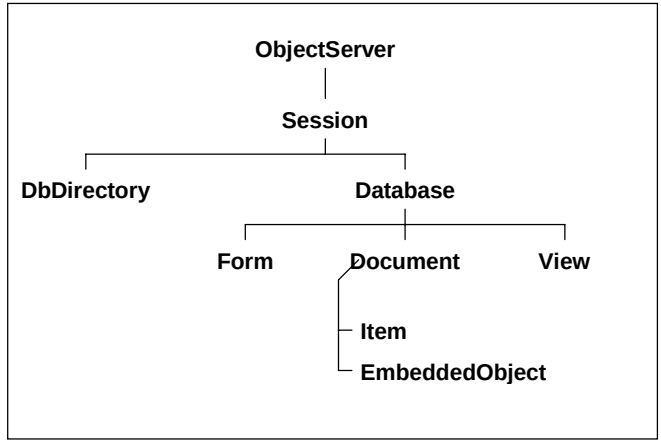
The following figure shows how the Java applet on a browser communicates with the Domino R5.0 Server:



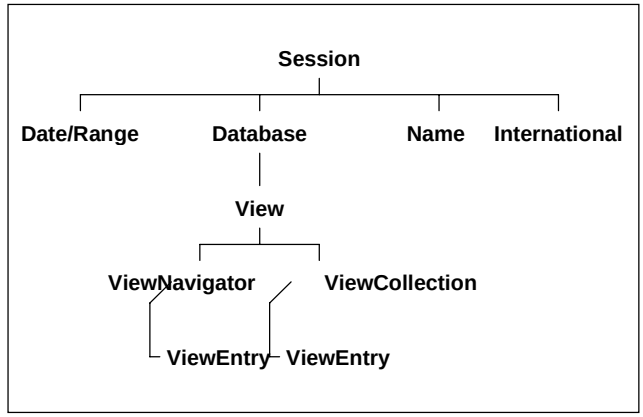
CORBA Objects

The following two figures illustrate the CORBA objects and show what Domino supports, and how you can add some additional function to your application using CORBA.

Object model in hierarchy order:



Client-Side Object in hierarchy order:



Special Notices

This publication is intended to help you develop applications using Lotus Domino Release 5.0.

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

The following are trademarks of Lotus Development Corporation in the United States and/or other countries:

LotusScript®
Lotus SmartSuite®
Notes Mail®
NotesPump
NotesSQL
Lotus®
Lotus Domino

Lotus Notes®
RealTime Notes
SmartIcons®

The following terms are trademarks of other companies:

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

File name	*Description*
5331c13a.nsf	DECS DB2 Sample A. Database used in the examples in the first part of Chapter 13: DECS
5331c13b.nsf	DECS DB2 Multi-Value Data Sample 5.0.1. Database used in the last example of Chapter 13: DECS
Approve4.ntf	Approval Cycle (R4). Template referenced in Chapter 16: Domino Ad-hoc Workflow

How to Get the Web Material

To get the Web material point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG245531`

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."

- *Using VisualAge for Java to develop Domino applications*, IBM form number SG24-5424, Lotus part number CT6ENNA
- *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
- *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
- *Lotus Solutions for the Enterprise, Volume 1. Lotus Notes: An Enterprise Application Platform*, IBM form number SG24-4837, Lotus part number 12968
- *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
- *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:

- *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 CT7QTNA
- *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 CT7QWNA
- *Lotus Notes and Domino R5.0 Security Infrastructure Revealed,* IBM form number SG24-5341, Lotus part number CT6TPNA
- *Lotus Notes and Domino: 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: Case Studies on Lotus cc:Mail to Lotus Domino and Lotus Notes*, IBM form number SG24-5100, Lotus part number 12992
- *Lotus Notes and Domino: The Next Generation in Messaging. Moving from Novell GroupWise to Lotus Notes and Domino*, IBM form number SG24-5321, Lotus part number CT7NNNA
- *High Availability and Scalability with Domino Clustering and Partitioning on Windows NT*, IBM form number SG24-5141, Lotus part number CT6XMIE
- *From Client/Server to Network Computing, A Migration to Domino*, IBM form number SG24-5087, Lotus part number CT699NA
- *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
- *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
- *Lotus Domino for S/390 Performance Tuning and Capacity Planning*, IBM form number SG24-5149, Lotus part number CT6XNIE
- *Porting C Applications to Lotus Domino on S/390*, IBM form number SG24-2092, Lotus part number AB1720
- *Managing Domino/Notes with Tivoli Manager for Domino, Enterprise Edition, Version 1.5*, IBM form number SG24-2104
- *Measuring Lotus Notes Response Times with Tivoli's ARM Agents*, IBM form number SG24-4787, Lotus part number CT6UKIE
- *Using ADSM to Back Up Lotus Notes*, IBM form number SG24-4534

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.

CD-ROM Title	*Collection Kit Number*
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
Application Development Redbooks Collection	SK2T-8037
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
AS/400 Redbooks Collection	SK2T-2849
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Networking and Systems Management Redbooks Collection	SK2T-6022
System/390 Redbooks Collection	SK2T-2177

How to Get ITSO 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.

- **Redbooks Web Site** `http://www.redbooks.ibm.com`
  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:

	e-mail address
In United States:	usib6fpl@ibmmail.com
Outside North America:	Contact information is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada (toll free)	1-800-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information for customers may be found at http://www.redbooks.ibm.com/ and for IBM employees at http://w3.itso.ibm.com/.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access `MyNews` at `http://w3.ibm.com/` for redbook, residency, and workshop announcements.

IBM Redbook Fax Order Form

Please send me the following:

Title	Order Number	Quantity

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

$$Return, 252
$$Return field, 70
  link to Web pages, 70
$$Search, 251
$$SearchSiteTemplate, 261
$$SearchTemplate, 256
$$SearchTemplateDefault, 257
$$ViewBody, 257
$about, 638
$defaultform, 640
$defaultNav, 640
$File, 644
$First, 638
$help, 638
$icon, 638
$OLEOBJINFO, 645
$Readers Field, 220
$SearchForm, 645
& arguments separator, 636
($All) view name, 111
($Inbox) view name, 111
($Sent) view name, 111
($Trash) view name, 111
(Declarations), 55
  for fields, 75
(Options), 55
  for fields, 75
@ClientType, 233
@Command
  ToolsRunMacro, 193
@DB Functions, 556
  @DBColumn, 556
  @DBCommand, 557
  @DBLookup, 557
@DBColumn, 518, 555, 556
@DBCommand, 518, 555, 557
@DBLookup, 518, 555, 557
@Functions, 264, 268, 285, 298, 519
  new in R5, 265
@URLOpen, calling agents, 194
@UserName, 232
@UserNameList, 233
@UserRoles, 232

Numbers

123
  OLE, 386

A

About Database document, 176
Access Control
  @ClientType, 233
  @UserName, 232
  @UserNameList, 233
  @UserRoles, 232
  authors field, 222
  combining Readers and Authors fields, 223
  controlled access sections, 224
  developing a plan, 240
  distinguishing true security features, 244
  documents, 219
  field editor access security option, 224
  forms, 216
  Hide-When formulas, 225
  hiding the design of a database, 234
  HTML and other files, 237
  password field, 234
  pasting documents into database, 234
  programming considerations, 232
  reader access, 220
  using directory link files, 214
  using encryption for field security, 225
  using outline control, 214
  using sgnatures for security, 234
  views, 215
Access list for views and folders, 149
Accessing Notes from Visual Basic, 498
Accessing RDBMS from Notes, 518
ACL, 204
  access options, 208
  anonymous access, 209
  changing the ACL programmatically, 212
  enforce consistent ACL, 211
  example, 275
  maximum Internet name and password access, 212
  public access users, 218
  roles, 210
  settings for creating agents, 177
  user and server access levels, 205
  user types, 207
Action, 635
  adding to navigators, 154
  bar, 121
  buttons, 99
  drop down, 100
  in navigators, 151
  object, 284
Activate
  OLE, 380
Advanced templates, 36
Agent log, 188
Agent Manager, 189
Agents
  capturing CGI variables, 194
  changing documents, 197
  debugging, 188
  disabling, 189
  restricted, 178
  running multiple instances, 554
  scheduling, 180
  selecting documents, 182
  shared, 179
  simple actions, 183
  testing, 187
  unrestricted, 178
  WebQueryOpen, 193
  WebQuerySave, 193
Alias
  framesets, 164

in keyword lists, 575
of form, 44
prefix, 574
use aliases, 597
used in localization, 597
Aligning
fields, 83
paragraph, 67
tables, 86
AllDocuments property, 276
Alt - W - #, 12
Alternate text
hotspots, 103
Anonymous access, 209
Anonymous form, 47
API
for Notes, 341
Applet security
CORBA, 365
Applets
rich text field, 68
Application Profile form, 619
Application templates, 27
Approach
OLE, 386
Approval Logic subform, 619
Archiving
documents, 35
Argument passing to LSX, 408
Arguments, 635
Authors Field, 222
Auto launch, 49
Auto refresh a field, 329

B

Back-end classes, 298
Background color
forms, 49
Background graphic
in navigators, 151
Backup and Restore, 240
BASIC, 268
Border
of cells, 87
of tables, 89
Breakpoints, 316
Built-in functions, 312
Button class, 270
Button object, 282
Buttons
action bar, 99
in navigators, 151

C

C programming language
header files for LSX, 419
Calendar views
conflict marks, 117
creating, 117
Categories
hiding in views, 113
Categorizing a view, 137
Cells
border of, 87
colors, 87
in tables, 84
CGI, 194. *See Also* Common Gateway Interface
Class library, LS:DO, 533
Class method arguments, LSX, 425
Class property arguments, LSX, 427
Click event
example, 283
Collapse, 638
CollapseView, 638
Collapsible sections, 82
Colors
for fields, 67
for views, 112
of cells, 87
unread documents, 136
Column headings for views, 112
COM, common object model, 423
Common Data Security Architecture (CDSA), 240
Common Gateway Interface, 103
table of variables, 103
using, 70
Common Object Model, 423
Compile errors, 313
Computed subforms, 78
Computed text, 97
Configuring
DECS, 441
Conformance levels, 472, 525
Connection Server Administrator database, 441
Containment hierarchy, 273
Controlled Access Section, 224
Cookies
features, 335
syntax, 334
Copy Database dialog box, 32
Copying
views, 108
CORBA, 338, 346
CORBA/IIOP, 357
and Domino, 359
and getSession( ), 362
applet security, 365
benefits of using CORBA, 357
coding the applet, 361
CORBA architecture, 358
extending the AppletBase class, 362
importing applets, 364
Interface Definition Language (IDL), 358
Object Request Broker (ORB), 358
security options, 361
Count, 637
Create Java agent, 353
CreateObject
OLE, 378
Creating
a navigator, 152
agents, 177
an LSX, 395
background in navigators, 152
buttons, 121
calendar views, 117
collapsible section, 82
computed subform, 78
DECS activity document, 446
folders, 148
forms, 43
framesets, 163
language database, 578
language databases, 589
layout region, 80
nested tables, 84
new database, 28, 32
pages, 158
report database, 569, 570
resources, 173
tables, 83
views, 108
Ctrl - Tab, 12
Currency symbols, 593
CurrentDatabase property, 274
Customized Authentication, encryption, signing
APIs, 239
Customized search form
creating, 252

D

Data type mapping, 529
Data types of an LSX, 406
Database, 635
Database access facilities, 518
Database record, 276
Database script, 176
Databases
  creating new, 28, 32
  encrypting, 32, 35
  encryption, 29
  inherit design, 36
  localizing, 559
  multi database search type, 34
  multilingual, 565
  name & address book type, 34
  preparing for localization, 566, 593, 607
  properties, 33
  removing subforms, 77
  size limit, 29
  tagging, 564, 570
  unilingual, 565
Date Picker
  embedded, 93
Date/Time values in views, 141
Debugging
  agents, 188
  debug mode, 315
  debugger, 313
  OLE, 374
DECS
  accessing multiple tables, 449
  administrator navigator, 442
  configuring, 441
  connecting to db2, 439
  connection server administrator database, 441
  creating an activity document, 446
  data sources, 438
  loading, 437
  real-time connection, 443
  running an activity, 448
  shutting down, 437
  verifying setup, 440
Default focus for fields, 66
Default form, 46, 277
Default value
  for fields, 74
Default view, 112
Design elements
  not allowed in layout regions, 80
  pages, 157
  supported in tables, 83
  translation, 559
Design Pane, 11
Design, inherit, 36
Designing
  an LSX, 423
DisableRole method, 275
Disabling agents, 189
Disabling the debugger, 314
Do Not Translate
  Domino Global WorkBench, 577
DO_NOT_TAG style, 599
Document fields
  $UpdatedBy, 277
  form, 277
Document retrieval, 39
DocumentContext, 194
Documents
  archiving, 35
  changing associated form, 198
  sorting in views, 116, 143
Domino
  clients, 4
Domino Administrator R5.0, 6
Domino Application Server, 1
Domino Design Elements, 15
Domino Enterprise Connectivity Services, 437
Domino Enterprise Server, 2
Domino Global WorkBench
  alias in keywords list, 575
  checking for size problems, 582
  concatenated sentences, 600
  creating a new glossary, 568
  creating a project, 566
  creating a report database, 570
  do not translate, 577
  exporting languages for translation using other tools, 587
  handling keywords, 602
  language database, 563
  language switchbar, 561
  localizing an application, 565
  prefix for alias names, 574
  prevent translation, 576
  processes supported, 559
  pseudo-translation, 577
  rebuilding, 592
  report database, 563
  source database, 563
  synchronization between several languages, 561
  tagged database, 563
  tagging, 563
  tagging a database, 570
  the Project Manager, 564
  the Standalone Tagger, 564
  the Synchronizer, 565
  the WorkBench, 564
  translatable lists, 602
  untranslated terms, 580
  updating, 592
Domino Mail Server, 1
Domino profile, 335
Domino Security Architecture, 202
Domino Server Family, 1
Domino Server Services, 2
Domino Translation Object, 561
Domino Web Server API, 239
DoVerb
  OLE, 380
DTO, 561

E

Embedded
  date picker, 93
  elements, 92
  file upload control, 96
  folder pane, 94
  navigators, 93
  OLE object, 380
  outline, 170
  outline control, 93
  scheduling control, 94
  SQL, 472
  views, 93, 138
EmbedObject
  OLE, 379
Enabling the debugger, 314
Encryption, 29
  creating an encryption key, 226
  databases, 35
  field security, 225
Enforce Consistent ACL, 211
Entering event
  for fields, 75
Enterprise integration technologies, 25
Error handler, 313
Evaluate function, 300
Events
  action object, 284
  button object, 282
  click, 283
  exiting, 281

for field object, 280
for fields, 74
forms, 53
Initialize, 55
not available for subforms, 77
onClick, 54
OnDblClick, 54
onHelp, 54
onKeyDown, 54
onKeyPress, 54
onKeyUp, 54
onLoad, 54
onMouseDown, 54
onMouseMove, 54
onMouseOut, 54
onMouseOver, 55
onMouseUp, 55
onReset, 55
onSubmit, 55
onUnload, 55
Postmodechange, 55
Postopen, 55, 278
Postrecalc, 55
PostSave, 55
Queryclose, 55
Querymodechange, 55
Queryopen, 55
QuerySave, 55, 302
sequence of, 285, 289
terminate, 55
WebQueryOpen, 54
WebQuerySave, 54
Events in Notes, 278
Examples
accessing a field, 274, 275
accessing an ACL entry, 275
click event, 278, 283
cookie, 335
evaluate function, 301
exiting event, 281
Java agent, 353
JavaScript, 329, 331, 332, 333
messagebox statement, 320
postopen event, 278
print statement, 318
testing an LSX, 413
using Java Notes classes, 354
Exclusion list
for tagging, 574
using prefixes, 574
Execution Control List (ECL), 234
Exiting event
example, 281
for fields, 75
Expand, 638
Expanded pseudo language, 582
ExpandView, 637
Exporting
a view, 145
for translation, 586, 587
Extending Notes with other products
Notes API, 341

F

Field class, 270
Field Editor Access Security Option, 224
FieldOffset, 644
Fields
(Declarations), 75
(Options), 75
aligning, 83
allow values not in list, 65
authors, 62
checkbox, 62
colors, 67
combobox, 62
computed, 64
computed for display, 64
computed when composed, 65
creating, 56
date, 61
default focus, 66
default value, 74
deleting, 198
dialog list, 62
editable, 64
entering event, 75
events, 74
exiting event, 75
fonts, 67
formula, 62
help information, 66
hide delimiters, 65
HTML attributes, 74
Initialize event, 75
input translation, 74
input validation, 74
listbox, 62
multi-value separator, 66
names, 62
number, 62
object, 280
onBlur event, 74
onChange event, 74
onClick event, 74
onFocus event, 74
onKeyDown event, 74
onKeyPress event, 74
onKeyUp event, 74
onMouseDown event, 74
onMouseMove event, 74
onMouseOut event, 74
onMouseOver event, 74
onMouseUp event, 75
onSelect event, 75
password, 62
properties, 63
radio button, 62
readers, 62
renaming, 198
replication, 124
rich text, 62
security, 66
shared, 60
terminate event, 75
text, 61
time, 61
using in NotesSQL, 491
File upload control
embedded, 96
Folder pane
embedded, 94
Folders
access list, 149
creating, 148
description, 107
Fonts
for fields, 67
Footer, 35
Form Access List, 217
Form flow in Approval Cycle, 619
Forms, 43
alias, 44
anonymous form, 47
auto launch, 49
background color, 49
create dialog, 45
creating, 43
default form, 46
events, 53
frameset option, 49
generate HTML for all fields, 48
header and footer, 50
importing graphics, 50
inherit entire document, 48
inherit values, 47
keyboard shortcut, 44
merge replication conflicts, 47
naming, 44
properties, 43

refresh fields automatically, 46
removing subforms, 77
response, 45
response to response, 45
security, 51
select for display, 79
store form in document, 46
title, 52
using resources, 174
using with NotesSQL, 486
version control, 46
Formulas, 264, 285
hide-when, 297
using in localization, 599
Formulas, using in agents, 185
Framesets, 163
creating, 163
for forms, 49
launched by page, 160
sizing, 165
Freelance
OLE, 387
Front-end classes, 298
Full text index
indexing, 39
location, 250
multi-databases, 260
search, 39
size, 251
updates, 251
Functions, 290

G

Generate HTML for all fields, 48
GetEmbeddedObject
OLE, 379
GetFirstDocument method, 274, 276
GetFirstItem method, 274
GetNextDocument method, 276
GetObject
OLE, 378
GetSession( )
and CORBA, 362
GetView method, 274
Global WorkBench, 559
Graphic files
supported in layout regions, 81

H

Header, 35
Header and footer, 50
Help
for fields, 66
HelpRequest, 54
Hidden views
showing, 123
Hide-when formulas, 255, 297
Hiding
views, 111
views from Notes users, 140
views from Web users, 140
Horizontal rules, 97
Host, 635
Hotspots, 99
alternate text, 103
HTML
attributes for body field, 74
HTML body attributes, 53
HTML head attributes, 53
using in views, 128
HTTP Basic Authentication, 227

I

IDE, 267, 268
ILsiADTControl structure, 423
ImageFormat, 644
Importing
translation, 588
Importing a view, 147
Importing graphics
into forms, 50
Importing Java applets, 364
Indexing, full-text, 39
Info List, 13
Inherit entire document, 48
Inherit values, 47
Initialization of an LSX, 417
Initialize event, 55
for fields, 75
Input from Web in rich text, 68
Input translation, 276
Input translation for fields, 74
Input validation, 276
Input validation for fields, 74
Installing
an LXS, 415
NotesSQL, 479
Integrated Development Environment, 267, 268
Integration of an LSX, 417
Internet authentication
when to use Internet security, 230

J

Java, 192
using in agents, 186
using JDBC, 506
Java agent, 346, 353
creating an agent, 353
Java applet, 346
Java files, 343
Java Notes classes, 352, 354
Java program, 348
JavaScript, 320, 328
add to form or field, 328
Domino object model, 323
Example 2, 331
Example 3, 331
Example 4, 332
Example 5, 333
JS header, 54
library, 328
LiveConnect, 337
onBlur, 321
onChange, 59, 320, 321, 330, 332
onClick, 321
onDblClick, 321
onFocus, 321
onHelp, 321
onKeyDown, 321
onKeyPress, 321
onKeyUp, 321
onLoad, 54, 321
onMouseDown, 321
onMouseMove, 321
onMouseOut, 321
onMouseOver, 321
onMouseUp, 321
onReset, 55, 321
onSubmit, 55, 321
onUnload, 55, 321
JavaScript function, 329
JDBC, 506
connecting to Domino, 509
JDBC-ODBC bridge, 506
JS Header, 54

K

Keyboard shortcut
forms, 44
Keywords
in localization, 602
using in NotesSQL, 493

L

Language Database
creating, 578
Domino Global WorkBench, 563
Language databases
creating, 589
Language switchbar
Domino Global WorkBench, 561
Launch Buttons, 12
Launch options, 37
Launching pages, 160, 162
Layout regions, 80
LCTEST
testing DB2 connectivity, 441
testing DECS, 440
Loading
DECS, 437
Localization
checking size problems, 582
currency symbols, 593
date and time values, 594
limits, 604
preparing a database, 566
preventing translation, 577
process, 560
pseudo language, 569
roles, 560
setting up a project, 566
skipping vs. tagging terms, 591
translation roles, 586
untranslated terms, 582
what is, 559
Localization glossary
enabling languages, 585
setting, 568
translating, 584
Localization process, 560
Localizing
localizing an application, 565
Logical errors, 314
Login, 639
Login argument, 636
LotusScript, 192, 278, 285, 298, 344, 620
(Declarations), 55
(Options), 55
benefits of, 268
client API, 416
DocumentContext property, 194
extending the architecture, 416
functions, 290
instance handle, 416
performance, 290
programming tips, 290
subroutines, 290
using in agents, 186
using in navigators, 154
LotusScript:Data Object, 518, 519, 521
LS:DO, 518, 519, 521
architecture, 523
class library, 533
tracing and debugging, 529
using from the Web, 546
when to use, 524, 555
LS:DO and ODBC, differences, 524
LSX, 269
arguments passing, 408
C header files for, 419
class registration utility, 419
creating, 395
data types, 406
design considerations, 423
initialization of, 417
installing, 415
integration, 417
loading, 392
object creation, 418
object deletion, 418
portability, 424
prerequisites, 393
registering classes, 420
registration, 415
termination, 419
testing, 413
LSX Toolkit, 392
utilities, 394
LSXBASE.CPP, 419
LSXBASE.HPP, 419
LSXCOMM.CPP, 419
LSXCOMM.HPP, 419
LSXREG, 394
LSXRUN, 394, 414
LSXTEST, 394, 413

M

Mapping data types, 529
Mapping names for NotsSQL, 485
Margins
tables, 89
Merge replication conflicts, 47
Messagebox statement, 320
Microsoft ODBC driver, 531
MS Word
OLE, 382
Multi-database full text indexed, 260
Multilingual Applications Support, 24
Multi-value separator
for fields, 66
Multi-valued fields in NotesSQL, 491

N

Naming
a form, 44
an agent, 179
framesets, 164
pages, 159
Naming rules in NotesSQL, 488
Naming views, 132
National Language Support
Domino Global WorkBench, 559
Navigator class, 270
Navigators
description, 107
embedded, 93
Nested tables, 84
creating, 84
New Database dialog box, 28
New Features of Domino Designer, 17
Notes API, 341
Notes classes, 268, 278
button, 270, 278
examples, 274
field, 270
hierarchical relation, 273
navigator, 270
NotesACL, 270
NotesACLEntry, 270, 275
NotesAgent, 271
NotesDatabase, 270, 274, 276
NotesDateRange, 271
NotesDateTime, 271
NotesDbDirectory, 270
NotesDocument, 271, 274, 276
NotesDocumentCollection, 271, 276
NotesEmbeddedObject, 271
NotesForm, 272
NotesInternational, 272
NotesItem, 271, 274
NotesLog, 271
NotesName, 272
NotesNewsLetter, 272
NotesRichTextItem, 271
NotesSession, 270, 274, 275
NotesTimer, 272
NotesUIDatabase, 270

NotesUIDocument, 270
NotesUIView, 270
NotesUIWorkspace, 270
NotesView, 271, 274
NotesViewColumn, 271
Notes R5.0, 4
Notes/FX, 46
NotesACL class, 270
NotesACLEntry class, 270, 275
NotesAgent class, 271
NotesDatabase class, 270, 274, 276
NotesDateRange class, 271
NotesDateTime class, 271
NotesDbDirectory class, 270
NotesDocument class, 271, 274, 276
NotesDocumentCollection class, 271, 276
NotesEmbeddedObject class, 271
NotesForm class, 272
NotesInternational class, 272
NotesItem class, 271, 274
NotesLog class, 271
NotesName class, 272
NotesNewsLetter class, 272
NotesObject, 635
NotesOutline, 272
NotesOutlineEntry, 272
NotesRegistration, 272
NotesReplication, 272
NotesRichTextItem class, 271
NotesRichTextParagraphStyle, 272
NotesRichTextTab, 272
NotesSession class, 270, 274, 275
NotesSQL, 471
advantages, 477
connecting to data source, 484
data source configuration, 481
installing, 479
limitations, 497
new features, 471
prompting for password, 484
when to use, 478
NotesSQL features, not supported, 473
NotesTimer class, 272
NotesUIDatabase class, 270
NotesUIDocument class, 270
NotesUIView class, 270
NotesUIWorkSpace class, 270
NotesView class, 271, 274
NotesViewColumn class, 271
NotesViewEntry, 272
NotesViewEntryCollection, 272
NotesViewNavigator, 273
Nothing, special value, 276
Numbers in views, 142

O

Object browser
viewing LSX classes, 392
Objects View, 13, 53
ODBC, 471, 472, 519
unique index, 489
when to use, 520
ODBC access flow, 520
ODBC Administrator, 530
ODBC and LS:DO, differences, 524
ODBC data sources, registering, 527
ODBC drivers, 526
ODBC SQL grammar, 493
exceptions, 496
ODBCConnection, 521, 535
ODBCQuery, 521, 538
ODBCResultSet, 521, 539
OldSearchQuery, 646
OLE, 268, 367
1-2-3, 386
activate, 380
and Excel, 369
Approach, 386
CreateObject, 378
debugging, 374
determining the classname, 381
DoVerb, 380
embedding OLE objects, 380
EmbedObject, 379
Freelance, 387
GetEmbeddedObject, 379
GetObject, 378
MS Word, 382
SmartSuite, 385
troubleshooting, 387
Visual Basic, 372
Word Pro, 386
On Error statement, 312
onBlur, 330
onBlur event
for fields, 74
onChange event
for fields, 74
onChangeIndex
#, 330
onClick event, 54
for fields, 74
OnDblClick event, 54
onFocus event
for fields, 74
not triggered, 75
onHelp event, 54
onKeyDown event, 54
for fields, 74
onKeyPress event, 54
for fields, 74
onKeyUp event, 54
for fields, 74
onLoad event, 54
onMouseDown event, 54
for fields, 74
onMouseMove event, 54
for fields, 74
onMouseOut event, 54
for fields, 74
onMouseOver event, 55
for fields, 74
onMouseUp event, 55
for fields, 75
onReset
JavaScript, 55
onReset event, 55
onSelect event
for fields, 75
onSubmit event, 55
onUnload event, 55
Option Declare, 293
Option Public, 290, 293
Outline control
embedded, 93
Outlines, 167
embedding, 170

P

Pages, 157
background, 160
creating, 158
design elements not supported, 158
launching, 160, 162
naming, 159
Paragraphs
aligning, 67
ParentUNID, 640
Pass-Thru-HTML in views, 128
Password Field, 234
Performance, 298, 302
of forms, 297
of LotusScript, 290
Personal views, 120
Personal-on-first-use views, 120

Portability of an LSX, 424
Postmodechange event, 55
Postopen, 278
Postopen event, 55
  example, 278
Postrecalc event, 55
PostSave event, 55
Prerequisites for LSX, 393
Preventing Printing, Forwarding and Copying of Documents, 219
Preventing tagging, 574
Preview
  in Web browser, 58
Print options, 35
Print statement, 318
Profile document, 335
Programmer's Pane, 13
Programming a Java program, 352
Properties
  fields, 63
  forms, 43
  of databases, 33
Pseudo language, 569
Pseudo translation
  Domino Global WorkBench, 577
  expanded type, 583
  reverse type, 577
Public Access
  users, 52, 218
  views, 114

Q

Query, 646
QueryClose event, 55
QueryModeChange event, 55
QueryOpen event, 55
QueryOpen form event, 193
QuerySave event, 55
QuerySave form event, 193

R

RDBMS, accessing from Notes, 518
Read Access List
  form, 220
Reader Access, 220
Real-time data access, 522
Rebuilding a translated database, 592
Reference View, 14
Refresh fields automatically
  in forms, 46
Refresh method, 276, 301
Registering an LSX class, 417
Registering ODBC data sources, 527
Registration of an LSX, 415
Reload method, 301
Removing subforms
  from databases, 77
  from forms, 77
Report Database
  creating, 569, 570
  Domino Global WorkBench, 563
Report options, for tagging, 571
Reserved fields, 124
Reserved names
  for views, 111
Resources, 173
  actions, 176
  applets, 175
  database script, 176
  icons, 176
  images, 173
  script libraries, 176
  shared fields, 175
Response
  forms, 45
Response documents
  indenting in views, 112, 142
Response message to Web users, 70
Response to Response, 45
  forms, 45
Restricted agents, 178
Resume statement, 312
Rich text, 276
Rich text field
  applet, 68
  using in NotesSQL, 492
Roles, 210
Run-time errors, 312, 313

S

SaveOptions field
  example, 626
Scheduling agents, 180
Scheduling control
  embedded, 94
Script Area, 14
Script libraries, 293
Search, 39
Search arguments, 646
Search Builder
  for agents, 182
  views, 109
Search conditions for agents, 183
Search related URLs, 248
Search scope, 259
Search site database, 258
Search site index, 250, 258
Searching
  $$ SearchSiteTemplate, 261
  $$Return, 252, 255
  $$Search, 251, 254
  $$SearchTemplate, 256
  $$ViewBody, 257
  default search form, 251
  full text index, 249, 260
  search arguments, 256
  search result form, 256
  search site index, 250
  search site URLs, 249
  search view URLs, 248
  security, 250
  views, 247, 262
  Web search Advanced, 251, 261
  Web search simple, 251, 261
  web site, 247, 262
SearchMax, 646
SearchOrder, 646
Secure Sockets Layer (SSL), 228
Security
  application security, 201
  for fields, 66
  forms, 51
  public access users, 52
  searching, 250
Security with cookie, 335
Setting
  localization glossary, 568
  window title, 53
Shared actions, 176
Shared field
  debugging, 290
Shared views, 120
Shutting down
  DECS, 437
Signatures, 234
Signing a Database, 237
Simple actions, 263, 290
Simple actions, using in agents, 183
Size limit
  databases, 29
Sizing framesets, 165
SmartSuite
  OLE, 385
Sorting documents in views, 116
Source Database
  Domino Global WorkBench, 563

SQL Grammar Conformance Level, 473
srchsite.ntf, 258
Start, 637
StartKey, 638
Store form
in document, 46
removing, 198
Storing
documents in databases, 276
Subforms, 75
Submit button, 99
Subroutines, 290
Synchronization between several languages
Domino Global WorkBench, 561
Synchronize agent, 565
Synchronizer, 579
Domino Global WorkBench, 565

T

Tables, 82
borders, 89
HTML options, 92
margins, 89
style, 87
Tagged Database
Domino Global WorkBench, 563
Tagger
standalone, 564
Tagging
databases, 564, 570
Domino Global WorkBench, 563
tagging a database, 570
Target Frame, 54
Templates, 27
advanced, 36
approval cycle, 619
Temporary variables
using in localization, 598
Terminate event, 55
for fields, 75
Testing
a navigator, 154
agents, 187
an LSX, 413
The Event Model, 278
Title
of forms, 52
Troubleshooting
OLE, 387

U

UNICODE, 424
Unique index for ODBC, 489
Universal Relation in NotesSQL, 485
Unread documents
views, 136
Unrestricted agents, 178
Untranslated Terms, 582
Domino Global WorkBench, 580
Updating a translated database, 592
URL for Domino objects
anchor link, 642
and frames, 637
CreateDocument, 641
DeleteDocument, 641
EditDocument, 641
keywords based URL, 642, 644
login argument, 639
OpenAbout, 638
OpenAgent, 640
OpenDatabase, 637
OpenDocument, 641
OpenElement, 644
OpenForm, 639, 640
OpenHelp, 638
OpenIcon, 638
opening attachments, 644
opening imported images, 644
opening OLE objects, 645
OpenNavigator, 640
OpenServer, 637
OpenView, 637
SaveDocument, 642
SearchSite, 645
SearchView, 645
syntax, 635
using keys vs universal ids, 643
Use statement, 294
UseLSX statement, 392, 528
User activity, 35
User scripts, 267, 293
Using
Universal Relation, 486
Using Database document, 176
Using pages, 161
Using the debugger, 317

V

Variables
declaration of, 290, 293
naming of, 291
Version control, 46
forms, 46
View
embedded, 93
View Access List, 215
View columns
HTML in column formulas, 130
HTML in headers, 129
icons in headers, 129
Views
access list, 149
adding HTML to, 130
advanced tab, 112
alias, 111, 132
calendar type, 117
cascading, 109, 133
changing name, 132
collapsing on open, 138
color, 112
column headings, 112
column properties, 114
copying, 108
creating, 108
default Web layout, 126
description, 107
embedding, 138
exporting, 145
formatting date/time values, 141
formatting numbers, 142
hiding, 111, 133
hiding empty categories, 113
hiding from Notes users, 140
hiding from Web users, 140
importing, 147
in localization, 600
indenting response documents, 142
InfoBasic tab, 111
inspecting form fields, 123
Java applet for display, 127
naming, 132
open as last-used, 138
pption tab, 111
Pass-Thru-HTML, 128
properties, 111
public access, 114
reserved names, 111
restrictions with NotesSQL, 490
search builder, 109
security tab, 113
showing hidden views, 123
showing icons in columns, 115

showing response documents, 112
sorting documents, 116, 143
sorting rules, 144
style tab, 112
type of, 109
unread documents, 136
used as index in NotesSQL, 489
using categories, 136
using HTML, 128
using on the Web, 125
using with NotesSQL, 486
Web navigation, 131
View templates
menu actions, 131
Visual Basic, 268, 498, 524
OLE, 372
using NotesSQL, 474

W

Web
using agents, 192
Web Users
defining, 231
WebQueryClose, 285
WebQueryOpen, 193, 285
WebQueryOpen event, 54
WebQuerySave, 193
WebQuerySave event, 54
Window Tabs, 12
Window title, 53
Word Pro
OLE, 386
Work pane, 11
WorkBench
Domino Global WorkBench, 564
Workflow, 611
concepts, 611
properties, 620
WYSIWYG, 158

ITSO Redbook Evaluation

Lotus Domino Release 5.0: A Developer's Handbook SG24-5331-01

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. 1999

Printed in the U.S.A.

SG24-5331-01

Part No. CC7EDNA

Lotus Domino Release 5.0: A Developer's Handbook