

Mastering XPages

A Step-by-Step Guide to XPages
Application Development and
the XSP Language

Martin Donnelly, Mark Wallace,
Tony McGuckin

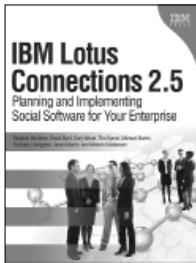


FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Related Books of Interest

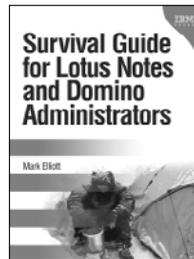


IBM Lotus Connections 2.5 Planning and Implementing Social Software for Your Enterprise

By Stephen Hardison, David Byrd, Gary Wood, Tim Speed, Michael Martin, Suzanne Livingston, Jason Moore, and Morten Kristiansen

ISBN: 0-13-700053-7

In *IBM Lotus Connections 2.5*, a team of IBM Lotus Connections 2.5 experts thoroughly introduces the newest product and covers every facet of planning, deploying, and using it successfully. The authors cover business and technical issues and present IBM's proven, best-practices methodology for successful implementation. The authors begin by helping managers and technical professionals identify opportunities to use social networking for competitive advantage—and by explaining how Lotus Connections 2.5 places full-fledged social networking tools at their fingertips. *IBM Lotus Connections 2.5* carefully describes each component of the product—including profiles, activities, blogs, communities, easy social bookmarking, personal home pages, and more.



Survival Guide for Lotus Notes and Domino Administrators

By Mark Elliott

ISBN: 0-13-715331-7

Mark Elliott has created a true encyclopedia of proven resolutions to common problems and has streamlined processes for infrastructure support. Elliott systematically addresses support solutions for all recent Lotus Notes and Domino environments.

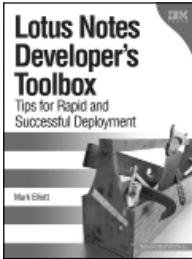
Survival Guide for Lotus Notes and Domino Administrators is organized for rapid access to specific solutions in three key areas: client setup, technical support, and client software management. It brings together best practices for planning deployments, managing upgrades, addressing issues with mail and calendars, configuring settings based on corporate policies, and optimizing the entire support delivery process.



Listen to the author's podcast at:
ibmpressbooks.com/podcasts

Sign up for the monthly IBM Press newsletter at
ibmpressbooks/newsletters

Related Books of Interest

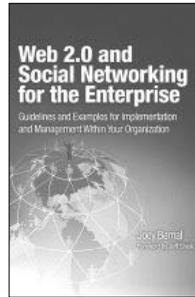


Lotus Notes Developer's Toolbox Tips for Rapid and Successful Deployment

By Mark Elliott

ISBN-10: 0-13-221448-2

Lotus Notes Developer's Toolbox will help you streamline and improve every phase of Notes development. Leading IBM Lotus Notes developer Mark Elliott systematically identifies solutions for the key challenges Notes developers face, offering powerful advice drawn from his extensive enterprise experience. This book presents best practices and step-by-step case studies for building the five most common types of Notes applications: collaboration, calendar, workflow, reference library, and website.



Web 2.0 and Social Networking for the Enterprise Guidelines and Examples for Implementation and Management Within Your Organization

By Joey Bernal

ISBN: 0-13-700489-3

This book provides hands-on, start-to-finish guidance for business and IT decision-makers who want to drive value from Web 2.0 and social networking technologies. IBM expert Joey Bernal systematically identifies business functions and innovations these technologies can enhance and presents best-practice patterns for using them in both internal- and external-facing applications. Drawing on the immense experience of IBM and its customers, Bernal addresses both the business and technical issues enterprises must manage to succeed.



Listen to the author's podcast at:
ibmpressbooks.com/podcasts

IBM
Press™

Visit ibmpressbooks.com
for all product information

Related Books of Interest



The Social Factor Innovate, Ignite, and Win through Mass Collaboration and Social Networking

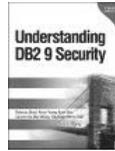
By Maria Azua

ISBN: 0-13-701890-8

Business leaders and strategists can drive immense value from social networking “inside the firewall.” Drawing on her unsurpassed experience deploying innovative social networking systems within IBM and for customers, Maria Azua demonstrates how to establish social networking communities, and then leverage those communities to drive extraordinary levels of innovation.

The Social Factor offers specific techniques for promoting mass collaboration in the enterprise and strategies to monetize social networking to generate new business opportunities.

Whatever your industry, *The Social Factor* will help you learn how to choose and implement the right social networking solutions for your unique challenges...how to avoid false starts and wasted time...and how to evaluate and make the most of today’s most promising social technologies—from wikis and blogs to knowledge clouds.



Understanding DB2 9 Security

Bond, See, Wong, Chan

ISBN: 0-13-134590-7



DB2 9 for Linux, UNIX, and Windows

DBA Guide, Reference, and Exam Prep, 6th Edition

Baklarz, Zikopoulos

ISBN: 0-13-185514-X

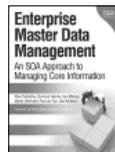


The Art of Enterprise Information Architecture

A Systems-Based Approach for Unlocking Business Insight

Godinez, Hechler, Koenig, Lockwood, Oberhofer, Schroeck

ISBN: 0-13-703571-3

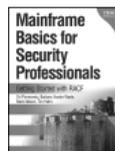


Enterprise Master Data Management

An SOA Approach to Managing Core Information

Dreibelbis, Hechler, Milman, Oberhofer, van Run, Wolfson

ISBN: 0-13-236625-8



Mainframe Basics for Security Professionals

Getting Started with RACF

Pomerantz, Vander Wee, Nelson, Hahn

ISBN: 0-13-173856-9

Sign up for the monthly IBM Press newsletter at
ibmpressbooks/newsletters

This page intentionally left blank

Mastering XPages

This page intentionally left blank

Mastering XPages:

A Step-by-Step Guide to XPages
Application Development and the
XSP Language

**Martin Donnelly, Mark Wallace,
and Tony McGuckin**

IBM Press
Pearson plc

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

ibmpressbooks.com

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

© Copyright 2011 by International Business Machines Corporation. All rights reserved.

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

IBM Press Program Managers: Steven M. Stansel, Ellice Uffer

Cover design: IBM Corporation

Associate Publisher: Dave Dusthimer

Marketing Manager: Stephane Nakib

Executive Editor: Mary Beth Ray

Publicist: Heather Fox

Senior Development Editor: Christopher Cleveland

Managing Editor: Kristy Hart

Designer: Alan Clements

Senior Project Editor: Lori Lyons

Technical Reviewers: Maureen Leland, John Mackey

Copy Editor: Sheri Cain

Indexer: Erika Millen

Senior Compositor: Gloria Schurick

Proofreader: Kathy Ruiz

Manufacturing Buyer: Dan Uhrig

Published by Pearson plc

Publishing as IBM Press

IBM Press offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact

U. S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com.

For sales outside the U. S., please contact

International Sales

international@pearson.com.

The following terms are trademarks of International Business Machines Corporation in many jurisdictions worldwide: IBM, Notes, Lotus, Domino, Symphony, Quickr, Sametime, Lotusphere, Rational, WebSphere, LotusScript, and developerWorks. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Oracle, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft, Windows, ActiveX, and Internet Explorer are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Library of Congress Cataloging-in-Publication Data

Donnelly, Martin, 1963-

Mastering XPages : a step-by-step guide to XPages : application development and the XSP language / Martin Donnelly, Mark Wallace, Tony McGuckin.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-13-248631-6 (pbk. : alk. paper)

1. Internet programming. 2. XPages. 3. Application software—Development. 4. Web site development. I. Wallace, Mark, 1967- II. McGuckin, Tony, 1974- III. Title.

QA76.625.D66 2011

006.7'6—dc22

2010048618

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671 3447

ISBN-13: 978-0-13-248631-6

ISBN-10: 0-13-248631-8

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.

Second Printing: July 2011

*I dedicate this book to the memory of my dear sister Anne,
the brightest and the best.*

—Martin

*For Dee, Sam, and Becky: I couldn't have contributed to this book without the
support, encouragement, and unending patience of my wonderful wife.*

Thank you, Dee.

—Mark

*I want to thank some great people for my involvement in this book.
First, it would not have happened without the encouragement and direction of
my lead architect (and co-author) Martin; thank you for the great opportunity.
Second, I want to thank my development manager, Eamon, and senior technical
architect, Phil, who had to keep things going without a full-time engineer, and
yet both remained upbeat throughout the process.*

*Finally, I dedicate my contribution to this book to my parents, family, and
especially my wife, Paula, and daughter, Anna-Rose, for putting up with a
part-time husband and dad—I love you both!*

—Tony

Contents

Foreword by Philippe Riand	xx
Preface	xxiv
Part I: Getting Started with XPages	1
Chapter 1 An Introduction to XPages	3
XPages Fundamentals	3
Brand New Technology?	4
A Different Development Paradigm	5
The More Things Change, the More Things Stay the Same	7
New Horizons	7
Conclusion	8
Chapter 2 Getting Everything You Need	9
Downloads, Versions, and Locations	9
Installing Domino Designer	10
Installing Client Fix Packs	11
Client Configuration	11
Quick Tour of Domino Designer	12
Domino Designer Welcome Screen	13
Domino Designer Perspective	14
Creating a New Application	15
Creating an XPage	16
Previewing in the Notes Client	18
Previewing in a Web Browser	18
Adding a Control to an XPage	21
Conclusion	22

Chapter 3	Building Your First XPages Application	23
	Laying the Foundations	24
	Forms and Views	26
	Building an XPages View	31
	Completing the CRUD	36
	Conclusion	42
Part II:	XPages Development: First Principles	43
Chapter 4	Anatomy of an XPage	45
	What Exactly Is an XPage?	46
	Understanding XSP Tag Markup	47
	Getting Started with XML	47
	XPages XML Syntax	50
	Simple Properties	52
	Complex Properties	54
	Complex Values	54
	Computed Properties	55
	Data Binding	59
	XPages Tags	60
	Data Sources	61
	Domino Document	61
	Domino View	62
	Data Context	63
	Controls	64
	Editing Controls	64
	Command Controls	70
	Selection Controls	74
	Display Controls	82
	File-Handling Controls	84
	Containers	87
	Panel	87
	Table	90
	View	91
	Data Table	94
	Repeat	95
	Include Page	99
	Tabbed Panel	99
	Section	100
	XPage Resources	101
	Script Library	101
	Style Sheet	103
	Resource Bundle	104

Dojo Module	105
Generic Head Resource	106
Metadata Resource	106
Converters	107
Validators	110
Simple Actions	118
Client-Side Scripting	125
HTML Tags	127
Conclusion	128
Chapter 5 XPages and JavaServer Faces	129
What Is JavaServer Faces?	130
JSF Primer	131
How Does XPages Extend JSF?	138
XML-Based Presentation Tier	141
Request Processing Lifecycle	142
User Interface Component Model	143
Standard User-Interface Components	148
Value Binding and Method Binding Expression Evaluation	152
XPages Default Variables	154
Conclusion	156
Chapter 6 Building XPages Business Logic	157
Adding Business Logic	157
Using the xp:eventHandler Tag	160
Simple Actions	167
Change Document Mode	168
Confirm Action	169
Create Response Document	170
Delete Document	171
Delete Selected Documents	172
Execute Client Script	173
Execute Script	173
Modify Field	174
Open Page	175
Publish Component Property	176
Publish View Column	177
Save Data Sources	179
Save Document	180
Set Component Mode	182
Set Value	183
Action Group	184

Using JavaScript with XPages	186
Server-Side JavaScript	186
Client JavaScript	206
Conclusion	211
Part III: Data Binding	213
Chapter 7 Working with Domino Documents	215
Domino Document Data Source	216
Creating and Editing Documents	219
Controlling URL Parameter Usage	220
Creating Response Documents	220
Executing Form Logic	224
Managing Concurrent Document Updates	227
Multiple Document Data Sources	228
Document Data Source Events	231
Common Data Source Properties	233
Miscellaneous Data Source Properties	234
Working with Domino Documents—Programmatically!	235
Simple Actions	235
JavaScript	236
Rich Documents	238
Conclusion	242
Chapter 8 Working with Domino Views	243
databaseName Property	245
View Data Source Filters	246
categoryFilter Property	246
search, searchMaxDocs Properties	249
parentId Property	251
ignoreRequestParams Property	252
keys, keysExactMatch Properties	253
Other View Content Modifiers	256
startKeys Property	256
expandLevel Property	257
A Page with Two Views	259
requestParamPrefix Property	260
When Is a View Not a View?	261
Go Fetch! Or Maybe Not...	262
loaded, scope Properties	263
postOpenView, queryOpenView Properties	263
Caching View Data	265
Sorting Columns	270
Conclusion	271

Chapter 9	Beyond the View Basics	273
	Pick a View Control, Any View Control	273
	The View Control: Up Close and Personal	276
	Column Data Like You've Never Seen Before	277
	Simple View Panel Make Over	279
	Working with Categories	293
	View Properties and View Panel Properties	301
	Data Table	305
	Building a Mini Embedded Profile View using a Data Table	311
	Repeat Control	316
	A Repeat Control Design Pattern	317
	Nested Repeats	318
	The Rich Get Richer	320
	Some Fun with the Pager	321
	Conclusion	324
Part IV:	Programmability	325
Chapter 10	Custom Controls	327
	Divide and Conquer	328
	Getting Started with Custom Controls	329
	Using Property Definitions	337
	Property Tab	340
	Validation Tab	343
	Visible Tab	345
	Property Definitions Summary	346
	Using the compositeData Object	346
	Send and You Shall Receive	352
	Multiple Instances and Property Groups	355
	Custom Control Design Patterns	357
	Aggregate Container Pattern	357
	Layout Container Pattern	358
	Conclusion	365
Chapter 11	Advanced Scripting	367
	Application Frameworks	367
	AJAX and Partial Refresh	369
	Partial Refresh: Out-of-the-Box Style!	369
	Partial Refresh: Doing-It-My-Way Style!	376
	Event Parameters	384
	Dojo Integration	386
	dojoTheme and dojoParseOnLoad Properties	387
	dojoModule Resource	388
	dojoType and dojoAttributes Properties	389
	Integrating Dojo Widgets and Extending the Dojo Class Path	390

Working with Traditional Notes/Domino Building Blocks	401
Working with @Functions, @Commands, and Formula Language	402
Working with Agents, In-Memory Documents, and Profile Documents	405
Managed Beans	412
Conclusion	419
Chapter 12 XPages Extensibility	421
How to Create a New User Interface Control	422
Example Component	423
Let's Get Started	424
Create the Initial Application	424
Add Package Explorer to the Domino Designer Perspective	424
Add a Java Source Code Folder	426
Building a Component	428
Create a UI Component Extension Class	428
Create Tag Specificaton (.xsp-config) for the UI Component Extension	431
Create a Renderer and Register It in the Application Configuration (faces-config.xml)	434
Quick Test Application to Verify Everything Is OK So Far	437
Working with Component Properties	438
Component Properties and Attributes	438
Adding a Property to a Component	439
State Holder: Saving State Between Requests	440
Specifying Simple Properties	440
Inheriting xsp-config Properties	441
Create the Initial xsp-config Definitions	446
Create base.xsp-config	446
Create an Interface to Match the Group Property Definition in base.xsp-config	450
Revisit the Component Properties in Domino Designer	452
Specifying Complex Properties	453
Complete the xsp-config for the UISpinner Component	464
Complete the UI Component Extension, UISpinner	473
Complete the Renderer UISpinnerRenderer	477
Create a Sample Application Using the UISpinner Component	483
Take Your New UI Component Extension for a Test Drive	483
Create a Backing Bean	483
Register the Backing Bean	486
Create the Final Test Application	486
Nice Look and Feel	491
Test to Ensure That It All Works!	491
Where to Go From Here	491
XPages Extensibility API Developers Guide	492
XPages Extension Library	492
IBM developerWorks	492
Conclusion	493

Chapter 13	XPages in the Notes Client	495
Think Inside the Box		496
Getting Started with XPages in the Notes Client		498
3, 2, 1...Lift Off		499
Bookmarks		501
Working Offline		503
One of These Things Is Not Like the Other		507
Other Subtle Differences		508
XPages: A Good Notes Citizen		511
Introducing enableModifiedFlag and disableModifiedFlag		513
Keeping Tabs on Your Client Apps		516
Notes Links Versus Domino Links		520
Some Debugging Tips		525
XPages and Composite Applications		528
Making a Component of an XPages Application		529
Is Anyone Out There? Creating a Component that Listens to Your XPages Component		531
Assembling a Composite Application: Aggregating the XPages Discussion		
Component and Notes Google Widget		533
Hey, This Is a Two-Way Street! A Component May Receive and Publish Events!		536
Further Adventures with Composite Applications		540
Part V:	Application User Experience	541
Chapter 14	XPages Theming	543
It Used to Be Like That...But Not Anymore!		543
Styling with Style!		545
Setting the Style Property Manually		550
Understanding How the Style Property Is Used		551
Computing the Style Property		552
Styling with Class!		552
Getting Something for Nothing!		553
Understanding How the styleClass Property Is Used		559
Computing the styleClass Property		561
Working with Extended styleClass and style Properties		563
Theming on Steroids!		567
What Is a Theme?		567
What Can You Do with a Theme?		568
Understanding Theme Architecture and Inheritance		569
Working with a Theme		576
Theme Resources		587
Resource Paths		597
Theme Properties, themeId, Control Definitions, and Control Properties		606
Conclusion		620

Chapter 15 Internationalization	621
Using Localization Options	622
Localization with Resource Bundle Files	623
Setting Localization Options	624
Testing a Localized Application	626
Working with Translators	628
Merging XPage Changes	631
Gotchas!	633
Localizing Computed Expressions and JavaScript	636
Adding a Resource Bundle	637
Localizing Computed Expressions	638
Localizing Client-Side JavaScript	639
Localizing Script Libraries	640
Server-Side Script Libraries	640
Client-Side Script Libraries	641
International Enablement	643
Locales in XPages	644
Deprecated Locale Codes	648
Conclusion	650
Part VI: Performance, Scalability, and Security	651
Chapter 16 Application Performance and Scalability	653
Golden Rules	654
Understanding the Request Processing Lifecycle	655
GET-Based Requests and the JSF Lifecycle	656
POST-Based Requests and the JSF Lifecycle	656
Reducing CPU Utilization	658
GET- Versus POST-Based Requests	658
Partial Refresh	663
Partial Execution Mode	665
Reducing Memory Utilization	668
HTTPJVMMaxHeapSize and HTTPJVMMaxHeapSizeSet Parameters	669
xsp.persistence.* Properties	669
dataCache Property	670
Conclusion	672
Chapter 17 Security	673
Notes/Domino Security and XPages	673
Server Layer of Security	674
Application Layer of Security	675
Design Element Layer of Security	677
Document Layer of Security	684

Workstation ECL Layer of Security	686
Useful Resources	687
Let's Get Started	687
Creating the Initial Application	687
Implementing ACLs	689
Sign the XPages with Your Signature	690
Programmability Restrictions	691
Sign or Run Unrestricted Methods and Operations	692
Sign Agents to Run on Behalf of Someone Else	692
Sign Agents or XPages to Run on Behalf of the Invoker	693
Sign Script Libraries to Run on Behalf of Someone Else	693
Restricted Operation	693
XPages Security Checking	695
NSF ClassLoader Bridge	695
XPages Security in the Notes Client	696
Execution Control List (ECL)	697
Active Content Filtering	699
Public Access	702
Setting Public Access for XPages	703
Checking for Public Access in XPages	703
SessionAsSigner	704
Troubleshooting XPages Java Security Exceptions	706
Conclusion	707
Part VII: Appendixes	709
Appendix A XSP Programming Reference	711
XSP Tag Reference	711
XSP Java Classes	712
Notes/Domino Java API Classes	714
XSP JavaScript Pseudo Classes	715
Appendix B XSP Style Class Reference	719
XSP CSS Files	719
XSP Style Classes	720
Appendix C Useful XPages Sites on the Net	727
Index	729

Foreword: Revolution Through Evolution

I never got a chance to meet the inventors of Notes®, but these guys were true visionaries. Their concepts and ideas of 20 years ago still feed today’s buzz. They invented a robust “NO SQL” data store, provided a social platform with collaboration features, and made the deployment and replication of applications easy...it is certainly no accident that Notes became so popular! Backed by a strong community of passionate developers dedicated to the platform, it elegantly solves real problems in the collaboration space by bringing together all the necessary components. As a developer, it makes you very productive.

Lotus Notes is also a fabulous software adventure and definitely a model for other software projects. At a time when technology evolves at unprecedented speed, where new standards appear and deprecate quickly, Lotus Notes adapts by keeping up to date. Over the past 20 plus years, Notes/Domino® has continually embraced diverse technologies in different domains: HTTP, XML, JavaScript™, Basic, Java™, POP/IMAP, LDAP, ODBC, just to name a few...this makes it unique in the software industry. Best of all, this is done while maintaining full compatibility with the previous releases. This reduces the risk for IT organizations and makes their long-term investment safer. Applications that were built about two decades ago on top of Windows® 2 (remember?) can be run without modification on the latest release of Notes/Domino, using any modern 64-bit operating system, including Linux® and MAC-OS! Continuity is the master word here, paired with innovation.

But, the world evolves. Software platforms in the old days were just proprietary, providing all the features they required by themselves. The need for integration wasn’t that high. However, as IT has matured over time, most organizations nowadays rely on heterogeneous sets of software that have to integrate with each other. Starting with version 8, the Notes client became a revolutionary integration platform. Not only does it run all of your traditional Notes/Domino applications, but it also integrates a Java web container, provides a composite application framework, embeds Symphony™, offers connectors to Quickr®, Sametime®, Lotus Connections, and so on. This was a great accomplishment—kudos to the Notes team.

At the same time, a parallel evolution saw the emergence of a more web-oriented world. An increasing set of applications, which traditionally required a specific proprietary client, started to become fully available through just a regular web browser. Google is certainly deeply involved in this mutation. New frameworks, languages, and libraries were designed to support this new no-deployment model. So, what about Notes/Domino? How can it be remain relevant in this new, ever-changing world? Of course, the Domino server includes an HTTP server that goes all the way back to R4.5. But, although it allows you to do pretty much everything, the cost of developing a web application, and the amount of required experience, was prohibitive. Moreover, the development model uses a proprietary page-definition language that is not intuitive for newcomers to the platform. Although not insurmountable, this was certainly a significant barrier to entry. It became clear that Domino web-application development (including Domino Designer) needed the same kind of revolution that the Notes client had undergone. True to our core values, however, this had to really be an evolution, where existing investment could be preserved, while throwing open the door to the new world. In essence, a revolution through an evolution.

During this time, I was leading a team at IBM® working on a development product called Lotus Component Designer (LCD). Its goal was to provide a Notes/Domino-like programming model on top of the Java platform, targeting the Lotus Workplace platform. It included most of the ingredients that made Notes/Domino a successful application development platform, while at the same time being based upon standard technologies: Java, JavaServer Faces (JSF), and Eclipse. Designed from the ground-up to generate top-notch web applications, it included a lot of new innovations, like the AJAX support, way before JSF 2.0 was even spec'd out. What then could have been a better fit for Notes/Domino app dev modernization? The asset was solid, the team existed, and the need was great, so it became the natural candidate for integration into Domino. An initial integration was achieved in a matter of a few weeks, and this is how the XPages story started!

When I joined the Notes Domino team four years ago (yes, time is running fast!), my mission was to make that revolution happen, starting with web applications. Taking over such a mission was intimidating because Domino has such a fabulous community of developers with unrivaled experience who obviously know much more about the product than I ever could. In fact, one of our business partners recently showed me a picture of five key employees and pointed out that they collectively represent more than 80 years of Notes/Domino development experience! In addition to this, the Lotus Notes/Domino development team is a well-established one, with mature processes and its own unique culture and habits. The XPages team was not only new to this world, but located geographically on the other side of it—in Ireland! The challenge thus became one of gaining acceptance, both internally and externally. This was a risky bet, because people might have easily just rejected the XPages initiative and pushed for another solution. But, we were pleasantly and encouragingly surprised. The first reactions were very positive. There was definitely room to deliver the innovation that the community so badly needed.

Notes/Domino 8.5 was the first release developed using an agile methodology. As it happened, that perfectly suited a new technology like XPages. It allowed us to communicate a lot

with the community, share design decisions, get advice, and modify our development plan dynamically. We had been, and still are, listening closely to the community through many and varied sources like blogs, wikis, forums, and of course, direct communication. We are most definitely dedicated to putting our customers in a winning situation. Everything we do is toward this goal: We truly understand that our success is our customers' success.

In this area, the XPages development team showed an impressive commitment. For example, we organized not one, but two workshops in our development lab 6 months before releasing the product! And it paid off: We introduced happy customers on stage at Lotusphere® 2009, a mere 15 days after the official release of the Domino 8.5. Their testimonials were encouraging and have not been proved wrong since, as the XPages adoption curve moves ever onward and upward. Many XPages-based solutions were shown at Lotusphere 2010, and Lotusphere 2011 promises to be another great stage with a lot of already mature solutions waiting to be announced. The team also wrote numerous articles in the Domino Application Development wiki, recorded many videos, and has been responsive on the different forums. This is also a major change where the development team is not isolated in its sterilized lab, but interacting positively with the broader community. The revitalization of openNTF.org is another example. The number of its monthly hits shows just how successful it is. Many partners have told me that they always look for already available reusable components before deciding to develop their own, and openNTF is just a fantastic resource in this regard.

So, what's next? Are we done? Certainly not! We have new challenges coming in, particularly with the next generation of browsers and platforms. We need to evolve XPages to generate applications that can take advantage of the new client capability. We need XPages to be tightly integrated with the rest of IBM Collaboration Services portfolio (a.k.a. Lotus portfolio). We need to support the new devices, such as smartphones and tablet PCs. We want to make sure that XPages plays a leading role with the next generation of Lotus Software (code name Vulcan). But, beyond the technology, we also have the challenge of transforming the way we create and deliver software. We want to make the Notes/Domino technology more open. We want to make the development process more transparent. We want to get feedback earlier, and we even want the community to contribute to that effort. We're all here to make it better, aren't we? The answer, in my opinion, is to open source some parts of the platform. OpenNTF is becoming our innovation lab, delivering technology early, breaking the regular release cycles. It allows us to be responsive to the community needs and then integrate the components later in the core product. Recently, we successfully experienced this with the new XPages Extension Library. The feedback we received was very positive, so we want to continue in this direction. Stay tuned...Notes/Domino is the platform of the future!

Finally, this story wouldn't have happened without a great XPages and Domino Designer team. For the quality of the work, the innovation path, the willingness to take on new challenges, the customer focus...well, for many aspects, this team is seen as exemplary in the broader Lotus organization. I really feel lucky and proud to be part of it. This book's three authors are also key members. Each one of them has worked on different areas of XPages; the gang of writers cannot

be better staffed. Martin is the team lead in Ireland, and he designed the Notes client integration and the data access part. Mark is a core runtime expert, and he has been involved since the early prototypes. Tony is our applications guy, in charge of the new generation of template applications. He has also been successful on many customer projects. Finally, helping them is Jim Quill, our security expert and general XPages evangelist. With this book, you definitely get the best of the best! I have no doubt that you'll learn a lot by reading it, whether you're a beginner or an XPages hacker.

Enjoy, the story has just begun!

Philippe Riand
XPages Chief Architect

Preface

XPages made its official public debut in Notes/Domino version 8.5, which went on general release in January 2009. At the annual Lotusphere conference that same month in Orlando, Florida, XPages was featured directly or indirectly in a raft of presentations and workshops, including the keynote session itself, as the technology was introduced to the broad application-development community. Throughout the conference, it was variously described as a new framework for Web 2.0 development, a strategic move to reinvigorate the application-development experience, a standards-based runtime that would greatly boost productivity for the Domino web developer...to quote but a few! Fancy claims indeed, but then again, Lotusphere has always been the stage that heralded the arrival of the “next big things” in the Notes/Domino community.

Fast forward to the present time: It’s fair to say that all these claims (excluding maybe one or two made much, much later into those Floridian evenings) were prophetic and XPages, as a technology, is indeed living up to its promise in the real world. Evidence of this is all around us. A vibrant XPages development community has evolved and thrives. Respected bloggers wax enthusiastic about the latest XPages tips and tricks. XPages contributions abound in OpenNTF.org, while the Notes/Domino Design Partner forum sees a steady flow of questions, comments, and, of course, requests for new cool features.

A recurring pattern evident in the flow of requests is the call for better documentation. XPages is a powerful Java runtime with a host of rich and sophisticated features that runs the entire app dev gamut. In the Notes/Domino 8.5 release, would-be XPages developers were left to their own devices to get up to speed with the technology. Typical approaches for the resourceful newbie developer included foraging for XPages programming patterns in the standard Notes Discussion template (which shipped with an out-of-the-box XPages web interface), scouring the limited Help documentation, and sharing random enablement materials that had started to appear on the web. Although all these, along with a sizable dollop of developer ingenuity, often worked remarkably well for those with large reserves of determination, the value of a single source of XPages information cannot be understated. This book’s goal is to fill that gap and provide a single comprehensive guide that enables readers to confidently take on, or actively participate in, a real-world XPages application-development project.

Approach

This book's objective is to impart as much practical XPages knowledge as possible in a way that is easy for the reader to digest. The authors seek to cover all aspects of the XPages development spectrum and to engage the reader with hands-on problems wherever possible. Most chapters come with a sample application that provides plentiful exercises and examples aimed at enabling you to quickly and efficiently solve everyday real-world use cases. These resources are located on the web at www.ibmpressbooks.com/title/9780132486316, so waste no time in downloading before getting started!

Tinker, Tailor, Soldier, Sailor?

Our Diverse Reading Audience

Although XPages is a new technology that offers a development model familiar to the average web developer (and the above-average ones, too!), many traditional Notes/Domino development skills can also be harnessed to good effect. One challenge in writing this book is that no single developer profile really defines the reader audience. For example, is the typical reader a web-application developer coming to the Notes/Domino platform or a Notes/Domino web developer wanting to learn XPages? In fact, since the release of Notes version 8.5.1, the reader may well be a Notes client application developer seeking to write new XPages applications for the Notes client or customize web applications that can now be run offline in that environment. Finally, a fourth category of reader may be the novice developer, for whom all this stuff is pretty much new! Which one are you? Or you may indeed be graced with the fine talents of bilocation and can appear in two of these camps at once!

Anyway, suffice to say that there inevitably will be aspects to several topics that are peculiar to a particular category of audience. Such content will typically be represented in this book as sidebars or tips in the context of the larger topic. Other cases might merit a dedicated section or chapter, such as Part IV, "Programmability," which contains a chapter that deals with all the details of XPages in the Notes client, while Part VI, "Performance, Scalability, and Security," has an entire chapter dedicated to the topic of application security.

Other Conventions

Any programming code, markup, or XSP keywords are illustrated in numbered listings using a fixed width font.

User-interface elements (menus, links, buttons, and so on) of the Notes client, Domino Designer, or any sample applications are referenced using a **bold** font.

Visual representations of the design-time experience or runtime features are typically captured as screen shots and written as numbered figures, using superimposed callouts where appropriate.

How This Book Is Organized

This book is divided into seven parts to separately address the many different aspects of XPages software development in as logical a manner as possible:

Part I, “Getting Started with XPages”: This part gets you familiar with XPages at a conceptual level to get you up and running quickly with the technology and get you comfortable with the overall application development paradigm.

- **Chapter 1, “An Introduction to XPages”:** Here, you are introduced to the history of XPages and given some high-level insights into its design principles in order for you to understand exactly what it is and what it is not. This is all about giving you the right context for XPages by defining the problems it solves, the technologies on which it is based, and where it might go in the future.
- **Chapter 2, “Getting Everything You Need”:** This chapter concerns itself with the practical business of obtaining, installing, and configuring your XPages development environment and successfully walking you through your first “Hello World” XPage!
- **Chapter 3, “Building Your First XPages Application”:** This chapter aims to provide a breadth-first hands-on experience of building a simple web application using the XPages integrated development environment (a.k.a Domino Designer). This is really just an introductory practical to get your feet wet and ensure you are comfortable with the basics of the application development model before diving any deeper.

Part II, “XPages Development: First Principles”: This part is mostly architectural in nature and aims to give you an appreciation of what’s happening under the XPages hood. This is an essential prerequisite to some of the more advanced topics, like XPages performance and scalability.

- **Chapter 4, “Anatomy of an XPage”:** This chapter examines the XSP markup language and gives a simple example of all the standard elements (controls and such) that can be used in an XPage. It provides a great broad-based view of XPages basics.
- **Chapter 5, “XPages and JavaServer Faces”:** This chapter looks at JavaServer Faces (JSF), which is the web-application development framework on which XPages is based. It looks at some core JSF design points and how XPages leverages and extends the framework.
- **Chapter 6, “Building XPages Business Logic”:** This chapter is a primer for XPages programmability. It introduces the various tools that can be used to implement XPages business logic so that you will be ready to work with the practical examples that are coming down the pike.

Part III, “Data Binding”: This part is really about how XPages reads and writes Notes data. XPages comes with a library of visual controls that are populated at runtime using a process known as data binding. The mechanics of the data binding process is explored in depth for Notes views and documents.

- **Chapter 7, “Working with Domino Documents”:** This chapter focuses on reading and writing Notes documents via XPages. Advanced use cases are explored and *every* design property on the Domino document data source is explained and put through its paces using practical examples.
- **Chapter 8, “Working with Domino Views”:** In this chapter, the Domino view data source is dissected and examined, property by property. Again, practical exercises are used to drive home the material under discussion
- **Chapter 9, “Beyond the View Basics”:** Working with Notes/Domino views is a large subject area, so much so that it demands a second chapter to cover all the details. This chapter looks at the various container controls that are available in the standard XPages control library, whose job it is to display view data in different formats and layouts in order to support a myriad of customer use cases.

Part IV, “Programmability”: This part covers the black art of programming—essentially how to code your applications to do everything from the most basic user operation to writing your own controls that implement completely customized behaviors. This part concludes with a look at XPages in the Notes client and considers cross-platform application development issues.

- **Chapter 10, “Custom Controls”:** This chapter explains the “mini-XPage” design element that is the custom control. It explains how to leverage the custom control in order to “componentize” your application and then maximize the reuse of your XPages development artifacts.
- **Chapter 11, “Advanced Scripting”:** Advanced scripting is an umbrella for many cool topics, like AJAX, Dojo, @Functions, agent integration, managed beans, and so forth. This is a must for anyone looking to add pizzazz to their XPages applications.
- **Chapter 12, “XPages Extensibility”:** This chapter explains how to use the XPages extensibility APIs to build and/or consume new controls. This is an amazingly powerful feature that has only recently become available and is well worth exploring once you have mastered XPages fundamentals.
- **Chapter 13, “XPages in the Notes Client”:** XPages in the Notes client initially explains how you can take your XPages web applications offline and then goes on to highlight how you can take advantage of powerful features of the client platform itself, and how to manage applications that run in both environments.

Part V, “Application User Experience”: This part is all about application look and feel. You learn not just how to make your apps look good and behave well, but how to do so for an international audience!

- **Chapter 14, “XPages Theming”:** This chapter teaches you how to manage the appearance and behavior of your application’s user interface. It provides an in-depth look at ad-hoc XPages application styling using cascading style sheets, as well as the main features of the standard XPages UI themes, and explains how to create your own customized themes.

- **Chapter 15, “Internationalization”:** Read this chapter to learn how your XPages applications can be translated so that they look, feel, and behave as native applications in any geographical locale.

Part VI, “Performance, Scalability, and Security”: Up to this point this book has concentrated on the skills and tools you need to know to develop state-of-the-art collaborative applications. Part VI shifts to deployment and what you need to do to make sure your applications meet customer expectations in terms of performance, scalability, and security.

- **Chapter 16, “Application Performance and Scalability”:** This chapter highlights various tips and tricks that will enable you to tune your XPages application for optimal performance and scalability in various deployment scenarios.
- **Chapter 17, “Security”:** Learn about application security issues and considerations and see how XPages integrates with the Domino server and Notes client security models.

Part VII, “Appendixes”

- **Appendix A, “XSP Programming Reference”:** This appendix points to a collection of definitive reference sources that describe all the details of the XSP tags, Java and JavaScript classes. It provides examples of how to use these resources to find the information you need.
- **Appendix B, “XSP Style Class Reference”:** This appendix identifies all the standard XPages CSS files and style classes used to build XPages application user interfaces. It’s an essential quick reference for Chapter 14.
- **Appendix C, “Useful XPages Sites on the Net”:** A snapshot of the authors’ favorite XPages websites at the time of writing. This list of sites should help you find whatever it is you need to know about XPages that isn’t found in this book.

Acknowledgments

This book was a new and eventful journey for all three authors as none of us had been down the book-writing road before. At times, the trip became a little more arduous than we had anticipated, but we received a lot of help from some great people along the way. We first want to thank our contributing author and colleague in IBM Ireland, Jim Quill, who we press-ganged at the eleventh hour and cajoled into writing a couple of chapters on the specialized topics of extensibility and security, respectively. Jim duly delivered, and we could not have met our project deadlines without him—just goes to show, a friend in need is a friend indeed!

We are happy to say that we are still on speaking terms with our two excellent and dedicated technical reviewers, Maureen Leland and John Mackey. Thanks to you both for keeping us honest and being positive and insightful at all times.

A sincere thank you to those who helped get this book proposal off the ground—especially Eamon Muldoon, Pete Janzen, and Philippe Riand, for their encouragement and advice along the way.

We are indebted to Maire Kehoe who always parachutes in for us to solve thorny problems at the drop of a hat—where would we be without you! Padraic Edwards and Teresa Monahan deserve our kudos for helping out on composite application use cases, and to Teresa again for her CK Editor brain dump. And because all the authors are based in Ireland, you can well imagine that we took every opportunity to lean on the other members of the XPages runtime team at the IBM Ireland lab. For that help, we want to collectively thank Brian Gleeson, Brian Bermingham, Darin Egan, Dave Connolly, Edel Gleeson, Gearóid O'Treasaigh, Lisa Henry, Lorcan McDonald, Paul Hannan, and Willie Doran.

We want to express our thanks to Robert Perron for some articles and documentation utilities that we are glad to leverage in a couple of places in this book. Thanks also to Thomas Gumz for some collaborative demo work we did at a dim and distant Lotusphere that is still worthy of print today! We are privileged to say there is a long list of folks at IBM past and present who have helped push the XPages cause forward over its eventful course thus far. Thanks to Azadeh Salehi, Bill Hume, Brian Leonard, Dan O'Connor, Dave Kern, David Taieb, Girish P. Baxi, Graham O'Keeffe, Ishfak Bhagat, Jaitirth Shirole, Jeff deRienzo, Jeff Eisen, Jim Cooper, John Grosjean,

John Woods, Kathy Howard, Margaret Rora, Matthew Flaherty, Mike Kerrigan, Na Pei, Peter Rubinstein, Russ Holden, Santosh Kumar, Scott Morris, Simon Butcher, Simon Hewett, Srinivas Rao, Steve Castledine, Steve Leland, Tom Carriker, Xi Pan Xiao, and Yao Zhang. Apologies to any IBMers accidentally omitted; let us know and we'll be sure to include you in the reprints!

To our friends at IBM Press—in particular Mary Beth Ray, Chris Cleveland, Lori Lyon, and Gloria Schurick—it may be a well-worn cliché, but it truly was a pleasure working with you guys! And on the IBM side of that relationship, we echo those sentiments to Steven Stansel and Ellice Uffer.

Finally, a great big THANK YOU, as always, to our customers and business partners, particularly the early adopters who got behind XPages at the get-go and made it the success that it is today!

About the Authors

The authors of this book have a number of things in common. All three hail from Ireland, work for the IBM Ireland software lab, and have made significant contributions to the development of XPages over the past number of years.

Martin Donnelly is a software architect and tech lead for the XPages runtime team in IBM Ireland and has worked on all three XPages releases from Notes/Domino 8.5 through 8.5.2. Prior to this, Martin also worked on XFaces for Lotus Component Designer and on JSF tooling for Rational® Application Developer. In the 1990s while living and working in Massachusetts, he was a lead developer on Domino Designer. Now once again based in Ireland, Martin lives in Cork with his wife Aileen, daughters Alison, Aisling, and Maeve, and retired greyhounds Evie and Chelsea. Outside of work, he confesses to playing soccer on a weekly basis, and salmon angling during the summer when the opportunity presents itself.

Mark Wallace is a software architect working in the IBM Ireland software lab. In the past, he worked on the XSP runtime, which was developed for Lotus Component Designer and subsequently evolved into the XPages runtime. He has a keen interest in programming models and improving developer productivity. Mark has worked in Lotus and IBM for more than 15 years on various products and is currently working on Sametime Unified Telephony. Mark lives in Dublin with his wife and two children and spends as much time as possible in the Ireland's sunny south east enjoying fishing and kayaking with his family.

Tony McGuckin is a senior software engineer in the IBM Ireland software lab. Having studied software engineering at the University of Ulster, he began his career with IBM in 2006 working in software product development on the component designer runtime before moving into the XPages core runtime team. When not directly contributing to the core runtime, Tony is busy with software research and development for the next generation of application development tooling, and also engaging directly with IBM customers as an XPages consultant. Tony enjoys spending time with his wife and daughter, and getting out into the great outdoors for hill walking and the occasional chance to do some hunting in the surrounding hillsides of his native County Derry.

Contributing Author

Jim Quill is a senior software engineer for the XPages team in IBM Ireland. He is relatively new to the Notes/Domino world, joining IBM just over two years ago at the tail end of the first XPages release in Domino 8.5. Previous to IBM, Jim enjoyed more than 13 years at Oracle Ireland. There, he worked in areas such as product development and database migration technology, and he was both principal software engineer and technical architect for a number of internal Oracle® support systems. Jim lives in the coastal village of Malahide, north County Dublin, with his wife and four children. When not acting as the kids' taxi, he continues to play competitive basketball...way past his retirement date.

This page intentionally left blank

Beyond the View Basics

Because the preceding chapter concentrated exclusively on the gory details of data retrieval from Domino views, it's only fitting that this chapter focuses on the fine art of presenting view data in XPages. Once again, a modified version of the Discussion template is used as the sample application. In fact, for this chapter, you need two samples, namely **Chapter9.nsf** and **Chapter9a.nsf**. You need to download these resources now from the following website and load them up in Domino Designer so that you can work through all the examples provided: www.ibmpressbooks.com/title/9780132486316.

You will see how this standard template uses the View and Repeat controls to best effect when displaying view data, and extra XPages have been added to show off some new tips and tricks. You will also learn how to extend and modify the behaviors of the view controls using JavaScript, Cascading Style Sheets (CSS), and so on. If you work through all the examples as you read along, you will have consummate expertise on this topic by the end of this chapter!

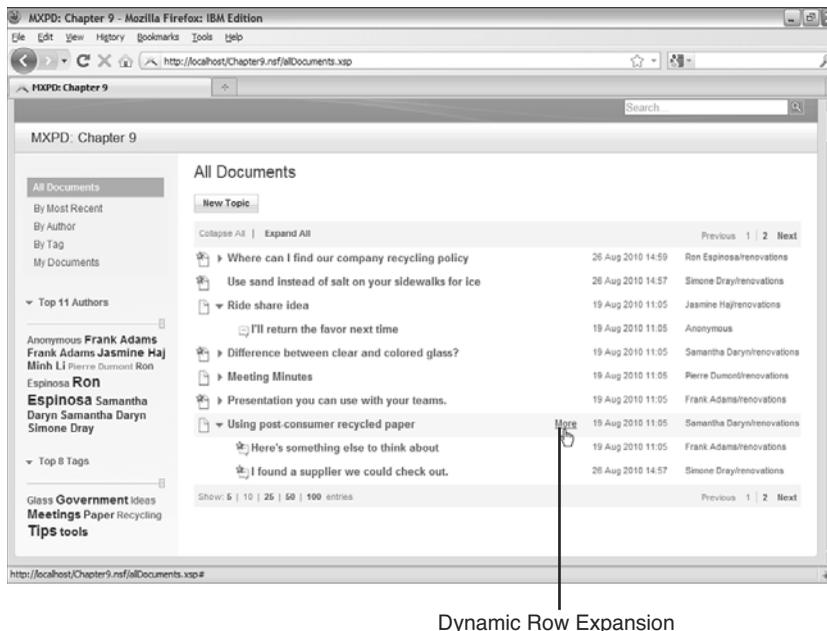
XPages provides three standard controls for presenting Domino view data, namely the View, Repeat control and Data Table. You will find all three on the **Container Controls** section of the palette in Designer. You have already done some work with these controls, mostly with the View control, although you have only used the basic properties up until now. You will see here how to put some of the lesser known properties to good use to solve some more advanced use cases. Perhaps it is best to start, however, with an explanation of why there are three different view presentation controls in the first place!

Pick a View Control, Any View Control

When it comes to presenting view data, we all have our individual preferences! For some use cases, a view with a strictly tabular format where rows and columns crisscross to form a rigidly

ordered grid layout is what's required. In other scenarios, a more free-form view layout of summary information that allows end users to dynamically dive deeper into the underlying data is the order of the day. In terms of providing off-the-shelf controls to meet these demands, no one-size-fits-all solution exists. In other words, separate specialized renderers are required to handle what are wildly different layout requirements, and each renderer has its own unique set of properties and behaviors that cater to those particular use cases.

Rather than simply describing various alternative view layouts, it is useful for you to see real-world use cases firsthand. As usual, the sample application can be readily called upon to demonstrate different view presentation examples. For example, explore the **All Documents** view on the main page of the application, and then compare its look and feel to one of the other views in the main navigator, such as **By Tag**, **By Author**, **By Most Recent**, and so on. Some key differences should come to your attention immediately. Chief among these is the interesting capability of the **All Documents** view to dynamically expand and collapse row content inline. That is, as you hover over any particular row, you are presented with **More** and **Hide** links, depending on the current state of the row content. If the row is collapsed, clicking the **More** option effectively injects an extra row of detail into your view, showing an abstract of the underlying document and presenting options to compose a reply or to switch to a view of documents that contain the same tags. Figure 9.1 summarizes this feature.



Dynamic Row Expansion

Figure 9.1 Sample Discussion application using repeat control to render all documents view

The other views do not have this capability and instead display content on a strict one-document-per-row basis. The data in these views is typically organized according to a specific criterion, say by category, author, or date, and feature the standard document link navigators for some of the columns in each row. You will no doubt recognize these behaviors as built-in properties of the View control, and you have already implemented a view sample similar to these in Chapter 3, “Building Your First XPages Application.” That first sample demonstrated that you could build simple views using a View control in a matter of minutes. Although it also is possible to build sophisticated view renderings with the View control (as you’ll soon see), there are some things it is simply not designed to do—dynamic inline row insertion/deletion being a case in point.

The fancy dynamics shown in Figure 9.1 are achieved using a Repeat control. This container control iterates or “repeats” over every row in the view data source to which it is bound. Any control that is added to the Repeat container (by default it is empty) can be bound to a column in the backend view. The iterative read cycle that occurs at runtime then ensures that all contained controls display the appropriate column value *once* for every row in the view. Thus, you have a totally free-form means of laying out view data, where nothing is predefined but anything is possible. The presentation content is totally dependent on the controls you choose to add to the Repeat container. It is not required to be structured within an HTML table for example—something you are stuck with when using the View control or Data Table controls whether you like it or not. Also, Repeat controls can be nested within each other, meaning that different data sources can be navigated as part of one overall view presentation.

All this, of course, means the Repeat control is an incredibly powerful and flexible tool for displaying view data—that’s the upside! The downside is that you must define all the content and layout data yourself; in other words, it can be a lot of work depending on what you want to achieve. The View control, on the other hand, is somewhere toward the other end of the scale—a View control can be built quickly using easy point-and-click operations, but the end result is more restrictive than is the case with a Repeat control. Again, depending on what you want to achieve, the View control may be the correct instrument to use—a simple case of choosing the right tool for the right job!

To see how the various view controls have been employed in the Discussion template, you can search the Discussion template for the tags `xp:viewPanel`, `xp:repeat` and `xp:dataTable` (in Designer, type `Ctrl-H` and specify the literal tags in the **File Search** tab, as shown in the previous chapter). The View control is used in all the aforementioned XPages (**By Tag, By Author, By Most Recent**) and in **AuthorProfileView.xsp**. If a user has registered a profile in the application, the **Author Profile** custom control is one of three views displayed when the user’s name is picked from the author cloud. The Repeat control is used for the **All Documents** page, the presentation of both the tag and author clouds (as shown in Figure 9.1), and to build the response document chain displayed when editing a document that is contained in a hierarchy.

Interestingly, although perhaps not surprisingly, the search for `xp:dataTable` results in no hits—at least this is true in the out-of-the-box template; however, you can find matches in **Chapter9.nsf** because a Data Table example has been added for your convenience. The absence of the `xp:dataTable` tag from the Discussion template and from most other real-world application (at least in this author’s experience) is because it offers neither the convenience of a View control nor the flexibility of a Repeat control. In essence, it is like a limited version of both controls and, thus, tends to be left out in the cold when it comes to more sophisticated application development scenarios. It is, however, useful for prototyping and for simple use cases, and we examine a sample Data Table later in this chapter. First, however, it’s time to take a closer look at the intricacies of the View control.

The View Control: Up Close and Personal

In this book, the *View control* is commonly referred to as the *View Panel*. This reference emanates from the markup tag used for the View control, i.e. `<xp:viewPanel>`, and it comes in handy when its necessary to disambiguate the view control from the backend Domino view that serves as its data source. In any case, the terms “View control” and “View Panel” can be used interchangeably and refer to the visual control that renders the view data.

The View Panel is a rich control with an abundance of properties and subordinate elements, such as pagers, columns, data sources, converters, and so on. Some of its properties are generic insofar as they are also shared by other controls in the XPages library to support common features like accessibility, internationalization, and so forth. For the most part, this chapter concentrates on the other properties as they are more directly relevant to view presentation, while the generic properties are addressed separately in other chapters.

In any case, the View Panel properties used in the examples up to now have been few in number and basic in nature. The upcoming examples start to pull in more and more properties in order to tweak the look and feel of your views. As usual, you learn these by way of example, but before you dive in, it is useful to summarize the View Panel features that have already been covered and provide the necessary reference points should you need to recap. The forthcoming material assumes that you are proficient with the topics listed in Table 9.1, although more detailed information may be provided going forward.

Table 9.1 viewPanel Features Previously Discussed

Feature	Chapter Reference: Section	Description
viewPanel Designer: Drag & Drop	Chapter 3: Building an XPages View	Creating a View control from controls palette Working with the view binding dialog
viewColumn property: displayAs	Chapter 3: Building an XPages View	Linking View control entries to underlying Notes/Domino documents
viewColumn property: showCheckBox	Chapter 3: Completing the CRUD	Making view entries selectable for executable actions
viewPanel <code><xp: pager></code>	Chapter 4: View	Basic description of View control with pager information
viewPanel property: facets	Chapter 4: Facets	General introduction to <code>facets</code> , including simple examples using view pagers
viewPanel Designer: appending columns	Chapter 8: Caching View Data	Adding a new column to a View control and computing its value using server-side JavaScript

Column Data Like You've Never Seen Before

So, start the next leg of this View Panel journey of discovery by creating a new XPage, say **myView.xsp**. Drop a View Panel from the control palette to view and bind it to the **All Documents** view when the helper dialog appears. Deselect all but three columns of the backend view—retain **\$106**, **\$116**, and **\$120**. These are the programmatic names that have been assigned to the view columns; XPages allows you to use either the column's programmatic name *or* the view column title to identify the column you want to include in the View control. Not all view columns have titles, however! Click **OK** to create the View Panel.

When you preview this raw XPage, you see the **Date** and **Topic** fields as expected, along with what can best be described as some gobbledygook wedged in between those columns, as shown in Figure 9.2.

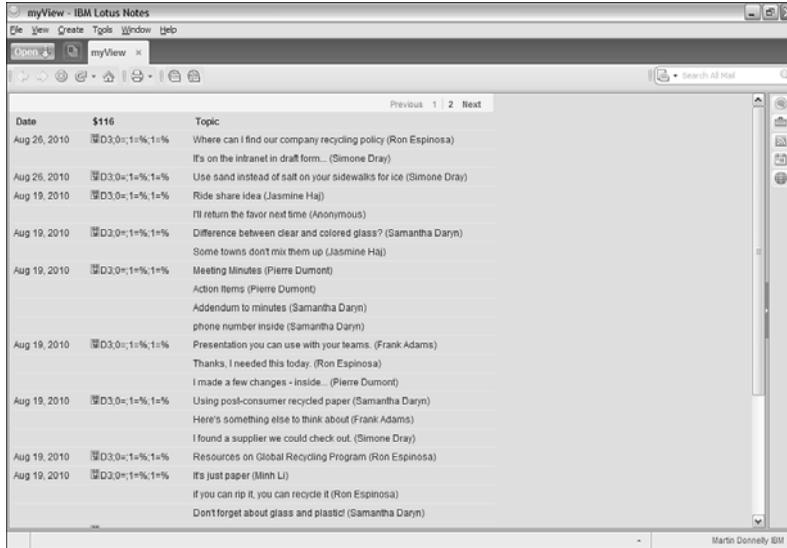


Figure 9.2 Columns from All Documents view displayed in a View Panel

It is not unreasonable to question what exactly this **\$116** column represents. The formula behind the column in the backend view looks like this:

```
@If (!@IsResponseDoc ;@DocDescendants ( " " ; "% " ; "% " ) ; " " )
```

In the regular Notes client, this column displays the number of descendant documents for all root level documents. To decipher the code, the `@DocDescendants` function is only applied when `!@IsResponseDoc` evaluates to true, meaning when the current document is *not* a response document, or in other words, for top-level documents only. The `" % "` within the parameter strings are replaced with the actual number of descendant documents at runtime. According to the Help documentation, `@DocDescendants` is among a class of `@Functions` that are restricted in their applicability and cannot be run from web applications. The function is described as returning “special text,” which is computed for client display only, not actually stored in the view, cannot be converted to a number, and so on. Other `@Functions`, such as `@DocNumber` and `@DocChildren`, present the same issues (you can find a more complete list in the Designer help pages). Designer itself attempts to preclude such columns from selection in the View Panel binding dialog, and the Java API `getColumnValues()` method, which is used to populate the View Panel row data, also tries to “null out” any autogenerated values that are contained in a row. However, these `@Functions` can be embedded in conditional logic and thus can be difficult to detect in advance. As a result, you might occasionally see spurious results like this appearing in views you are working on. So, what to do?

Because you cannot always work with *all* types of data contained in Domino views, you might need to create a modified version of a view in order to match your design criteria. Remember that the root of this problem is that the data defined in such columns is not actually contained

in the backend view, but it is possible that the underlying documents have fields that hold the required information or perhaps the information you need can be deduced using one or more fields. Thus, you could modify the backend view or create a new version that contains the column values you require based on fetching or computing the information by alternative means.

In the more immediate short term, however, you need to remove the offending column from the View Panel. This can be done in Designer in a number of different ways. You can highlight the column in the **Outline** panel or in the WYSIWYG editor and use the right-mouse **Delete** menu to remove the column—you appended a new column back in Chapter 8, “Working with Domino Views,” in much the same way. Alternatively, you can find the `<xp:viewColumn>` tag that is bound to **\$116** in the source pane and delete the markup directly from there.

Simple View Panel Make Over

Many presentational issues can be taken care of directly at the XPages level without any modifications to underlying the Domino view! For example, you are not restricted to the column order defined in the Domino view. You can reorder the columns in a View Panel by simply cutting and pasting the `<xp:viewColumn>` tags in the source pane—try this now in **myView.xsp**. Also, the date format of what is now or soon to be the second column can be modified in the XPages layer using a component known as a *converter*—this is the same component you used in Chapter 4, “Anatomy of an XPage,” when working with the Date Time Picker examples. To do this, click the **Date (\$106)** column in the WYSIWYG editor, select the **Data** property sheet, and change the **Display type** from “String” to “Date/Time.” Then, change the **Date style** from “default” to “full,” as shown in Figure 9.3.

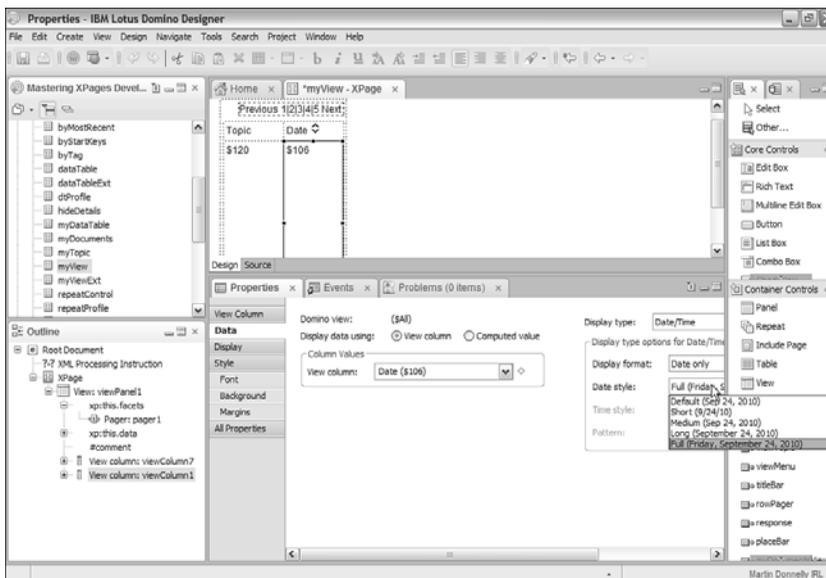


Figure 9.3 Applying a date converter in the View Panel

Listing 9.1 shows the markup generated from the cut/paste operation and the addition of the date converter.

Listing 9.1 viewPanel Markup with Reordered Columns and Alternative Date Formatting

```
<xp:viewPanel rows="30" id="viewPanel1">
  <xp:this.facets>
    <xp:pager partialRefresh="true"
      layout="Previous Group Next"
      xp:key="headerPager" id="pager1">
    </xp:pager>
  </xp:this.facets>
  <xp:this.data>
    <xp:dominoView
      var="view1"
      viewName="($All)">
    </xp:dominoView>
  </xp:this.data>
  <!-- Reordered columns so that Topic is first -->
  <xp:viewColumn columnName="$120" id="viewColumn7">
    <xp:viewColumnHeader value="Topic" id="viewColumnHeader7">
    </xp:viewColumnHeader>
  </xp:viewColumn>
  <xp:viewColumn columnName="$106" id="viewColumn1">
    <!-- Present full date like "Thursday, August 26, 2010" -->
    <xp:this.converter>
      <xp:convertDateTime type="date" dateStyle="full">
      </xp:convertDateTime>
    </xp:this.converter>
    <xp:viewColumnHeader value="Date" id="viewColumnHeader1">
    </xp:viewColumnHeader>
  </xp:viewColumn>
</xp:viewPanel>
```

Now that you've turned the view presentation on its head, you might as well look at its runtime rendition. All going well, you see a View Panel like the one shown in Figure 9.4.

You're not done yet, however! Albeit a simple View Panel, it is still possible to dress this puppy up a little further and add some extra behaviors.

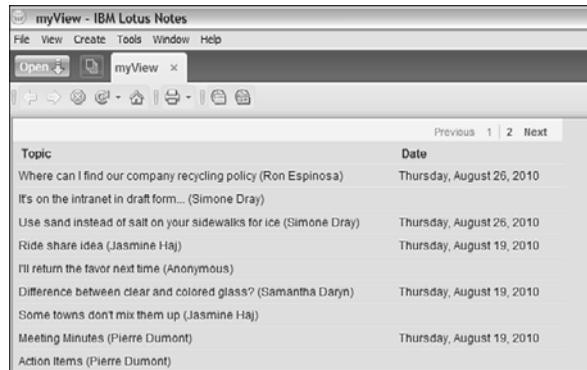


Figure 9.4 An alternative XPages view of All Documents

The World Is Flat???

An obvious limitation of the View Panel shown in Figure 9.4 is that the document hierarchy is not shown. The **Topic** column is just a flat list of entries that does not reflect their interrelationships in any way. To show the various threads in this view, all you need to do is click the **Topic** column in Designer, select the **Display** property sheet, and check the **Indent Responses** control. Reload the page after doing this, and you find that all parent documents now have “twistie” controls that can be used to expand or collapse its own particular part of the document tree. If you don’t like the standard blue twisties, feel free to add your own! Some extra images have been added as image resource elements to **Chapter9.nsf**, so if you want to try this feature out, you can simply assign **minus.gif** and **plus.gif** from the list of image resources in the application as the alternative twisties, as shown in Figure 9.5, although I’m sure you can come up with more interesting ones than these! Whatever alternative images are specified in this property sheet would also be applied to the twistie controls used for expanding and collapsing category rows, if you were working with a categorized view. Category views are discussed in the section, “Working with Categories.”

Linking the View Panel to its Documents

In Chapter 3, you learned to use the **Check box** feature shown in Figure 9.5 to enable row selection by the end user. You also learned to display the contents of the **Topic** column as links and to bridge it to **myTopic.xsp** by explicitly nominating that XPage as `pageName` property for the View Panel itself. Select the **Show values in this column as links** feature for **Topic** column again now, but omit nominating **myTopic.xsp** as the target XPage on this occasion. Preview the page and click any link—do you know just why this happens to magically work?

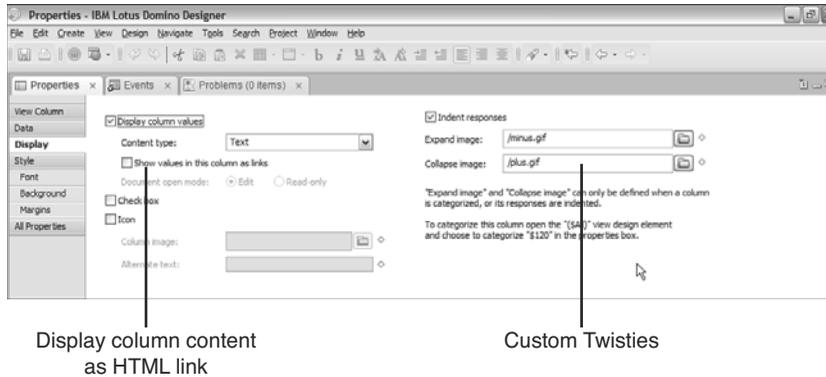


Figure 9.5 View Column Display Property sheet

The clue is in the View Panel's default link navigation option shown in Figure 9.6. When no page is explicitly nominated, XPages looks in the form used to create the underlying documents for a hint as to what XPage it should use. The form in question in this scenario is **Main Topic** and, if you open it in Designer and inspect its properties, you see a couple of interesting options, as highlighted in Figure 9.7.

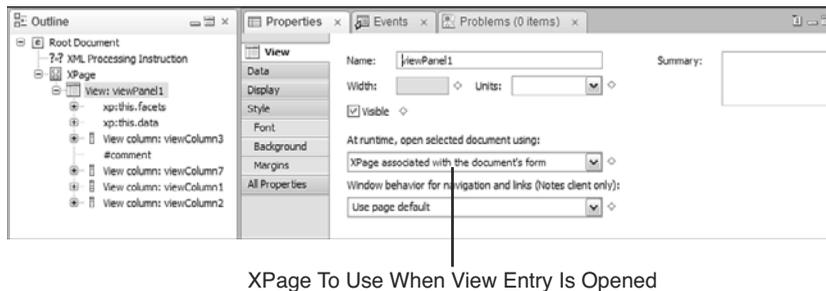


Figure 9.6 View Panel Basic Property panel

You can basically choose to override the form associated with a document on the web and on the client by opting to substitute an XPage instead in either or both environments. For the purposes of this chapter only, **Main Topic** has been updated to use **myTopic.xsp** as an alternative on both platforms, and thus, it is resolved as the go-to XPage when a column is clicked in the View Panel.

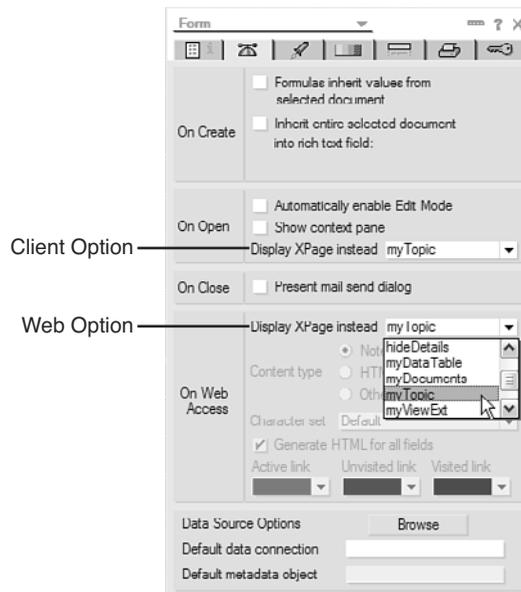


Figure 9.7 Form Properties Infobox: Display XPage Instead property

TIP **Display XPage instead** can be used to incrementally phase in XPages application implementations. If you are migrating an application to XPages, it might be possible to replace subsets of functionality that have been encapsulated in forms with XPages code, and then use pull these blocks into your application on a piecemeal basis using this feature.

There was originally just one **Display XPage instead** property. Since XPages was first made available on the web before being released on the Notes client, many customers converted their application's web implementation to XPages, but still had the original client application in place. When running the application natively on the client, they did not want to suddenly start seeing XPages appearing in place of forms! This feature was revamped in 8.5.2 to allow XPages and non-XPages implementations of an application to run harmoniously on separate platforms.

Although **Display XPage instead** certainly has its uses, the more common practice in the app dev community would appear to favor having an explicit XPage `pageName` navigation setting on the View Panel itself.

There is, in fact, a third strategy that can be employed to resolve what XPage is used when opening a document, and it is perhaps the simplest of them all! If you give the XPage the same name as the form used to create the document, it is chosen as a last resort if the other two options come up blank. This can be a useful approach if you are closely mimicking the original application implementation in XPages and if the application is simple enough to support such one-to-one design element mappings.

But, what of the remaining features in Figure 9.5? You just learned a second way to handle the **Show values in this column as links** option, and the **Check box** feature was already explored in Chapter 3. The **Display column values** checkbox merely serves to hide the column value retrieved from the view. This is potentially useful if you want to retrieve the column value but display something else based on what's actually contained in the column. In my experience, this property is not widely used as there are other (perhaps easier) ways of computing column values. We work through some examples of this shortly in the course of this View Panel makeover. On the other hand, if you simply want to conceal a column, you need to deselect the **Visible** checkbox in its property sheet, which sets `rendered="false"` in the underlying `<xp:viewColumn>` tag.

This just leaves the **Icon** and **Content type** in the view column **Display** panel, so you can learn now how to further enhance this simple makeover by putting those properties to work.

Decorating Your Columns with Images

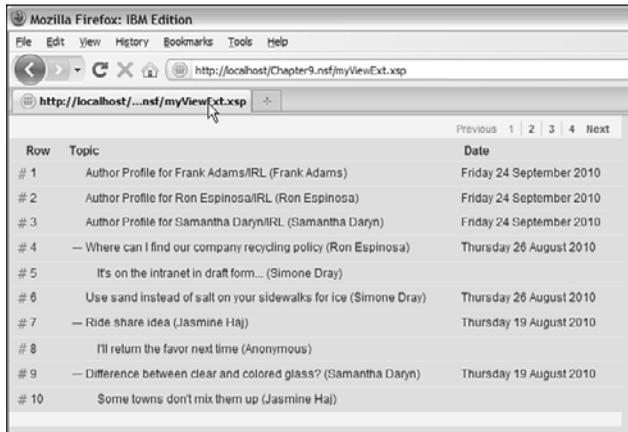
Any column in a View Panel can display an image as well as its column value. To add an image to a view column, you can simply check the **Icon** control (refer to Figure 9.5 to find the control, if needed) and type the name of the image resource or use the image browser dialog to locate it. It is good practice to enter some alternative text in case the image cannot be resolved at runtime and to facilitate screen readers and so on. The view column properties behind these two Designer choices are called `iconSrc` and `iconAlt`, respectively. You can implement a simple example as follows:

1. Insert a new column before the first column in the View Panel. You can use the **View > Insert Column** main menu when the **Topic** column is selected.
2. Check the **Icon** checkbox in the **Display** property sheet and add `/hash.gif` as the nominated image resource (you can also browse for this image resource). This image has already been added to **Chapter9.nsf** for your convenience.
3. Add `Index` as the alternative text.
4. Add `indexVar="rowIndex"` to the `<xp:viewPanel>` tag in the **Source** pane. You can also do this via the View Panel's **Data** category in the **All Properties** sheet.
5. Add the following server-side JavaScript snippet to compute the column's value:

```
var i:Number = parseInt(rowIndex + 1);  
return i.toPrecision(0);
```

In summary, you added an image to the new column and along with some alternative text. The `indexVar` property keeps a count of the rows in the View Panel as it is being populated. The `indexVar` property is used here as a simple row number to display in the UI. The JavaScript applied in step 5 simply increments each row index by 1 (it is a zero-based index) and ensures that no decimal places are displayed. Finally, to give the new column a title, click the view column header in the WYSIWYG editor and enter some text, say `Row`, as the label. Now, you can

preview or reload the page to see the results (all this has been done for you in `myViewExt.xsp`, if you want to look at the final creation), which should closely match Figure 9.8.



The screenshot shows a Mozilla Firefox browser window displaying a table with 10 rows. The table has three columns: Row, Topic, and Date. The rows contain author profiles and dates. The browser's address bar shows the URL `http://localhost/Chapter9.nsf/myViewExt.xsp`.

Row	Topic	Date
# 1	Author Profile for Frank Adams/IRL (Frank Adams)	Friday 24 September 2010
# 2	Author Profile for Ron Espinosa/IRL (Ron Espinosa)	Friday 24 September 2010
# 3	Author Profile for Samantha Daryn/IRL (Samantha Daryn)	Friday 24 September 2010
# 4	— Where can I find our company recycling policy (Ron Espinosa)	Thursday 26 August 2010
# 5	It's on the intranet in draft form... (Simone Dray)	
# 6	Use sand instead of salt on your sidewalks for ice (Simone Dray)	Thursday 26 August 2010
# 7	— Ride share idea (Jasmine Haj)	Thursday 19 August 2010
# 8	I'll return the favor next time (Anonymous)	
# 9	— Difference between clear and colored glass? (Samantha Daryn)	Thursday 19 August 2010
# 10	Some towns don't mix them up (Jasmine Haj)	

Figure 9.8 Computed View Panel column using `iconSrc`, `iconAlt` and `indexVar` properties

This is all well and good except that the icon displayed is static in nature; observe that it is the same for each row (the hash symbol gif). Although it is a computable property, `iconSrc` does not have access to the `View Panel` var or `indexVar` properties, so it difficult to do something dynamic with it, such as select the image resource based on a particular row column value for example. This might be addressed in a future release.

But fear not, as a dynamic solution can still be provided by using the **Content type** option on the same **Display** panel. To implement an example of applying images based on row content, work through the following instructions:

1. Append a new column to the end of the View Panel using the **View > Append Column** main menu.
2. In the **Display** panel set the **Content type** to HTML.
3. In the Source pane, add `var="rowData"` to the `<xp:viewPanel>` tag to gain access to the current row via server-side JavaScript while the View Panel is being populated.
4. On the Data property sheet, add the following server-side JavaScript snippet to compute the column's value property:

```
var i:number = rowData.getDescendantCount();
if (i < 10) {
    return ("

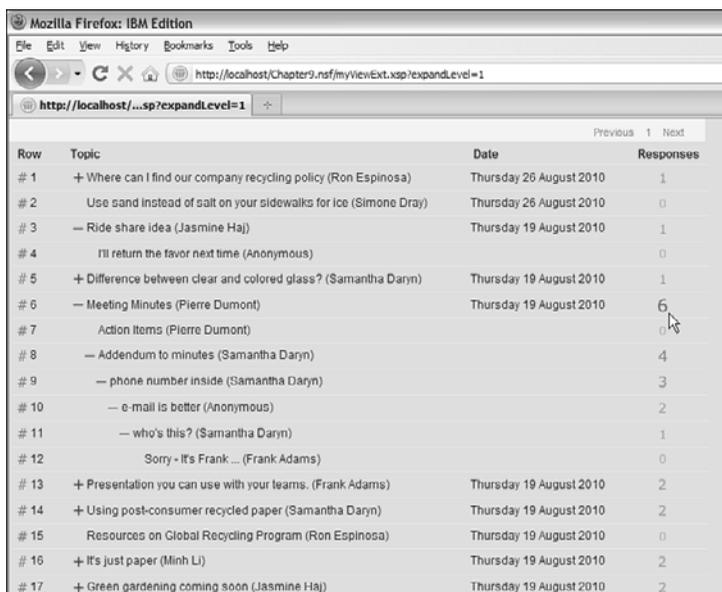
```

5. Move to the **Events** tab for this column and for the only defined event, `onclick`, add another server-side JavaScript snippet:

```
if (rowData.getDescendantCount() > 0) {
    rowData.toggleExpanded();
}
```

As you can see, the column value is set using server-side JavaScript in step 4. An HTML image tag is returned with the `src` value determined by the number of documents in the row's document hierarchy, 1 descendant document means "1.gif" is used, 5 descendant documents means "5.gif" is used, and so on. Because you set the column's content type to HTML, the image tag is simply passed through to the browser as is. Moreover, the image is clickable (unlike the image added via the `iconSrc` property) and fires an expand/collapse event for any non-leaf entry, such as when the entry has any responses, thanks to the code you added in step 5.

The column header label should be set to Responses, and the content of the column can be quickly centered using the **Alignment** button on the column **Font** property panel. Reload the page and see the new runtime behavior for yourself. The rendering of this column is also shown in Figure 9.9. Note that the `expandLevel=1` data source setting discussed in the previous chapter was used here (via a URL parameter) to initially collapse all rows. Some were then expanded to create a good example.



Row	Topic	Date	Responses
# 1	+ Where can I find our company recycling policy (Ron Espinosa)	Thursday 26 August 2010	1
# 2	Use sand instead of salt on your sidewalks for ice (Simone Dray)	Thursday 26 August 2010	0
# 3	- Ride share idea (Jasmine Haj)	Thursday 19 August 2010	1
# 4	I'll return the favor next time (Anonymous)		0
# 5	+ Difference between clear and colored glass? (Samantha Daryn)	Thursday 19 August 2010	1
# 6	- Meeting Minutes (Pierre Dumont)	Thursday 19 August 2010	6
# 7	Action Items (Pierre Dumont)		0
# 8	- Addendum to minutes (Samantha Daryn)		4
# 9	- phone number inside (Samantha Daryn)		3
# 10	- e-mail is better (Anonymous)		2
# 11	- who's this? (Samantha Daryn)		1
# 12	Sorry - It's Frank ... (Frank Adams)		0
# 13	+ Presentation you can use with your teams. (Frank Adams)	Thursday 19 August 2010	2
# 14	+ Using post-consumer recycled paper (Samantha Daryn)	Thursday 19 August 2010	2
# 15	Resources on Global Recycling Program (Ron Espinosa)	Thursday 19 August 2010	0
# 16	+ It's just paper (Minh LI)	Thursday 19 August 2010	2
# 17	+ Green gardening coming soon (Jasmine Haj)	Thursday 19 August 2010	2

Figure 9.9 Computed View Panel column using computed pass-through HTML content

So, this time, the image resource in the **Responses** column indeed varies depending on the response count for each row entry. It might not be too evident in the printed screen shot, but the color of the images darken and increase in pixel size as the numbers increase. Thus, the rows with more responses get more emphasis in the UI (similar in concept to the tag cloud rendering) on the basis that they represent busier discussion threads and are, therefore, likely to be of more interest to forum participants. If the number of response documents exceeds nine, an ellipses image (`n.gif`) is shown instead. Add more documents yourself and create deep hierarchies to see how this View Panel rendering works in practice—interesting all the same to see what can be achieved by tweaking a few properties and adding some simple lines of JavaScript code!

Some Final Touches

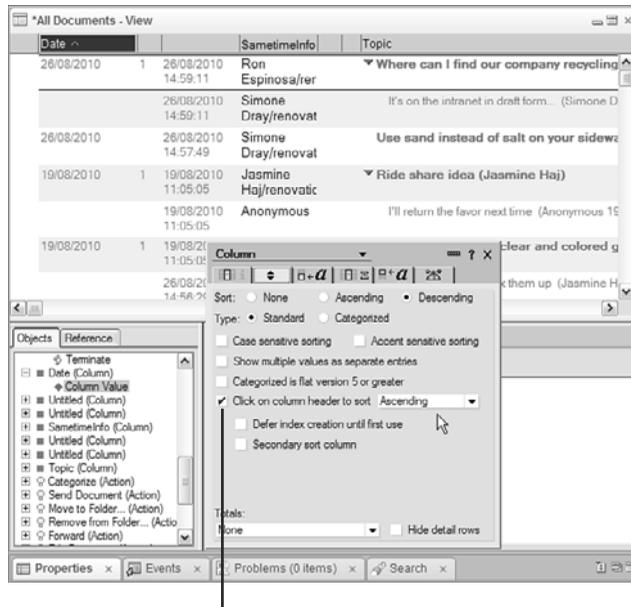
Before completing our sample rendering of the **All Documents** view, there are some final miscellaneous features to apply and some other behaviors to observe. First, when used in native client mode, the backend **All Documents** view can be sorted by clicking the **Date** column. This sorting facility is not in evidence as yet in the XPages View Panel, so you must learn how to enable it.

The first thing to understand is that it is the backend view itself that performs the sorting. It is not performed client-side in XPages itself, and any attempt to do so is invariably inefficient and performs poorly as applications scale. Don't go there—leave the sorting operation to the view itself.

To enable the sort feature in the View Panel, you need to select the required view column header in the WYSIWYG editor and activate its property sheet. You see a **Sort column** checkbox that you need to check. If this is disabled, it means that the column as defined in the backend view does not have any sorting capability; Designer looks up the column design properties and enables or disables this option appropriately. Figure 9.10 shows the view column property that defines sorting capability.

If the column you want to sort in XPages is not defined, as shown in Figure 9.10, you need to either update the view design or create a new modified copy of the view to work with going forward. After the backend sort property *and* the XPages sort property are enabled, the View Panel displays a sort icon in the header and performs the sort operation when clicked by the user. Figure 9.11 shows the **All Documents** view after being resorted via the View Panel (oldest documents are now first).

TIP A view can lose its sorting capability after certain filters are applied. For example, if you perform a full-text search on a view, the resulting document collection is not sortable. In 8.5.2, the View Panel sort icons are removed when it displays the results of a full text search. In previous releases, the icons remained enabled, thus implying that the result set was sortable when, in fact, it was not. This is a commonly requested feature, however, and might be addressed in a future release.



Column can be sorted by user

Figure 9.10 View Column infobox with sorting capability enabled

Now complete this particular make over by selecting the View Panel and selecting its **Display** property sheet. Check the **Show title** and **Show unread marks** controls, and change the number of maximum number of rows from the default of 30 to 10. Figure 9.12 shows the property sheet with these changes applied.

Clicking **Show title** places a View Title component into the header of the View Panel. You can then click this component directly in the WYSIWYG editor and then set its label and other properties via the component's property sheet. This results in a `<xp:viewTitle>` tag being inserted into the View Panel facets definition; for example:

```
<xp:viewTitle xp:key="viewTitle" id="viewTitle1"
    value="All Documents - Make Over Complete!">
</xp:viewTitle>
```

The View Panel also has a `title` property defined on the `<xp:viewPanel>` tag. This is merely exposing the `title` attribute of the underlying HTML table element that is used to construct the View Panel when rendered at runtime. If you enter a value for this property, it is passed through to the browser as part of the `<table>` HTML markup. For a visible view title, you need to use the **Show title** property and not this `title` property.

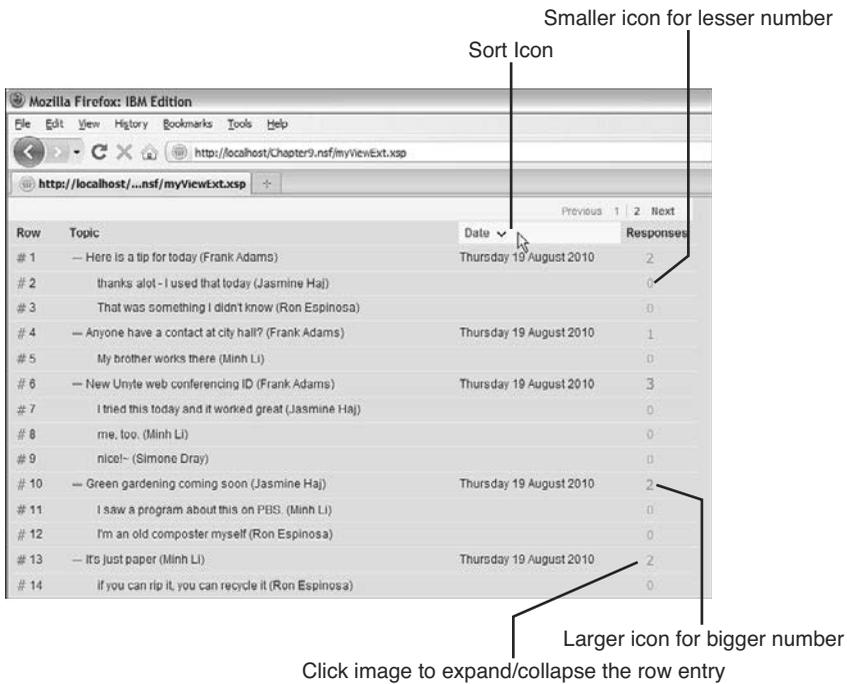


Figure 9.11 View Panel with all documents resorted by date in ascending order

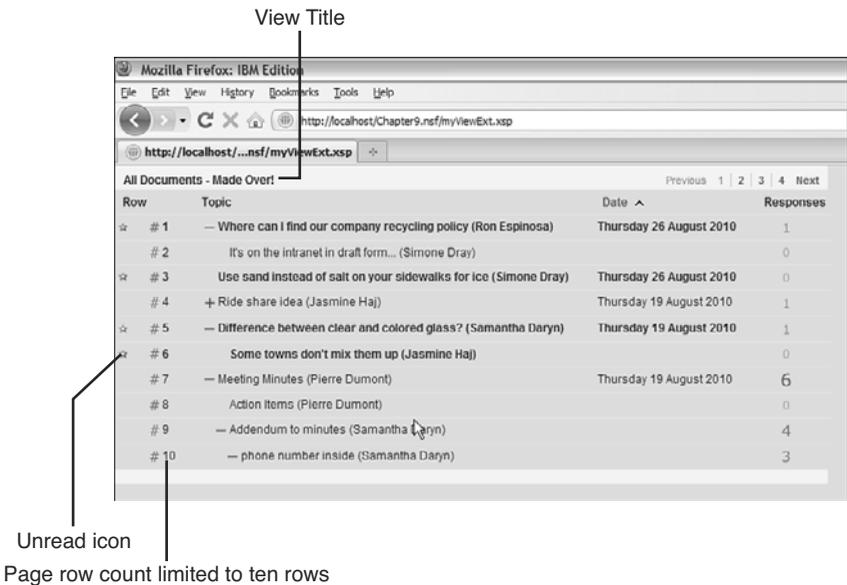


Figure 9.12 View Panel with title, unread marks, and a row count of ten documents

Secondly, if your unread view entries are not displayed as unread (no unread icon is displayed), this is most likely because the Domino server is not maintaining unread marks for the application—keeping track of read/unread documents is optional. You can ascertain the status of this feature in Designer via the **Application Properties > Advanced** property sheet. Look for the **Maintain unread marks** checkbox in the top-left corner.

The `rows` property that controls the maximum number of entries displayed in a view at any one time (set to 10) is exposed directly in the regular Discussion template UI. For example, the footer of the **All Documents**, **By Tag**, and **By Author** views conveniently *lets the user choose* the number of entries to display, as shown in Figure 9.13.

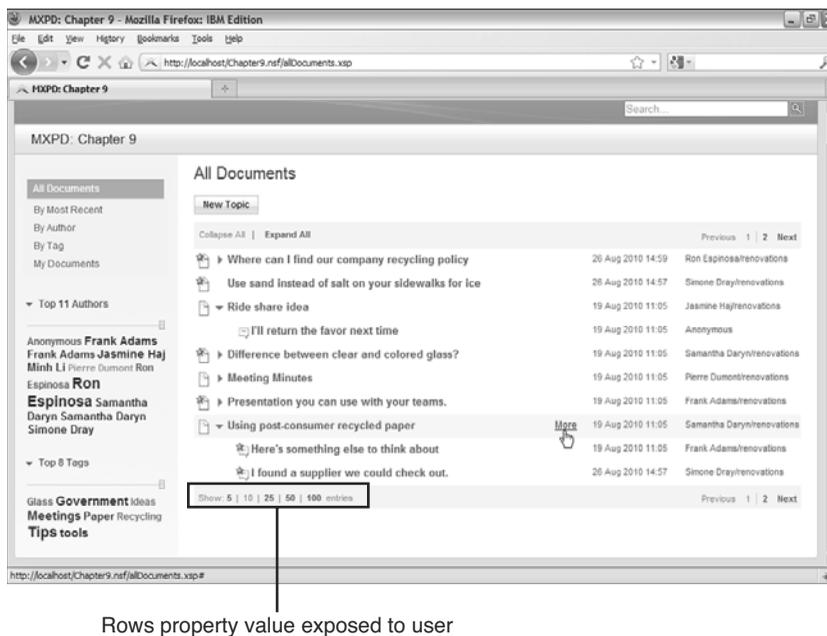


Figure 9.13 Rows property exposed as user option in view footer

Listing 9.2 provides the entire View Panel markup, along with comments in case you had difficulty applying any of the many and varied features discussed in this section. It is also included in **Chapter9.nsf** in the **myViewExt.xsp XPage**.

Listing 9.2 View Panel: Complete Source for Make-Over Exercise

```
<xp:viewPanel rows="10" id="viewPanell" var="rowData"
  indexVar="rowIndex" showUnreadMarks="true">
  <xp:this.facets>
    <xp:pager partialRefresh="true"
```

```

        layout="Previous Group Next"
        xp:key="headerPager" id="pager1">
</xp:pager>
<!-- View Panel Title -->
<xp:viewTitle xp:key="viewTitle" id="viewTitle1"
        value="All Documents - Made Over!">
</xp:viewTitle>
</xp:this.facets>
<xp:this.data>
    <xp:dominoView var="view1" viewName="($All)">
    </xp:dominoView>
</xp:this.data>
<!-- Static Column Image # -->
<xp:viewColumn id="viewColumn3"
    iconSrc="/hash.gif"
    iconAlt="Row Number Symbol">
<xp:this.facets>
    <xp:viewColumnHeader xp:key="header"
        id="viewColumnHeader3" value="Row">
    </xp:viewColumnHeader>
</xp:this.facets>
<!-- Compute Row Number -->
<xp:this.value><![CDATA[#{javascript:
    var i:Number = parseInt(rowIndex + 1);
    return i.toPrecision(0);}]]>
</xp:this.value>
</xp:viewColumn>
<!-- Reordered columns so that Topic is before Date -->
<!-- Use custom twistie images for expand/collapse -->
<xp:viewColumn columnName="$120" id="viewColumn7"
    indentResponses="true"
    collapsedImage="/plus.gif"
    expandedImage="/minus.gif">
    <xp:viewColumnHeader value="Topic"
        id="viewColumnHeader7">
    </xp:viewColumnHeader>
</xp:viewColumn>
<!-- Present full date like "Thursday, August 26, 2010" -->
<xp:viewColumn columnName="$106" id="viewColumn1">
    <xp:this.converter>

```

```

        <xp:convertDateTime type="date" dateStyle="full">
        </xp:convertDateTime>
    </xp:this.converter>
    <xp:viewColumnHeader value="Date"
        id="viewColumnHeader1"
        sortable="true">
    </xp:viewColumnHeader>
</xp:viewColumn>
<!-- Dynamic Column Images - 1.gif thru 9.gif -->
<!-- inline CSS to center img -->
<xp:viewColumn id="viewColumn2"
    contentType="HTML"
    style="text-align:center">
    <xp:this.facets>
        <xp:viewColumnHeader xp:key="header"
            id="viewColumnHeader2" value="Responses">
        </xp:viewColumnHeader>
    </xp:this.facets>
    <!-- Compute image name based on response count -->
    <xp:this.value><![CDATA[#{javascript:
        var i:number = rowData.getDescendantCount();
        if (i < 9) {
            return (";)

```

Working with Categories

Just like sorting, categorization is handled by the backend view itself and not by XPages. For a column to be treated as a category, the column type must be set to **Categorized** in the view column properties infobox; refer to the **Type** radio button option shown in Figure 9.10, which allows columns to be defined as **Standard** or **Categorized**.

The View Panel merely presents category rows and columns and renders them so they can be expanded and collapsed as required. The expansion and contraction of category rows works the same as it does for indented responses. Note also that the state of both category rows and document hierarchies is maintained as you navigate through the view data. For example, as part of the final make over, you restricted the number of rows presented in the View Panel to ten elements (remember `rows="10"`). This caused more pages to be displayed in the view pager contained in the header. If you expand and collapse some categories or response hierarchies on any given View Panel page and then navigate forward and backward via the pager, you find that the display state of these rows is maintained and then redisplayed on your return exactly as you had left them. This statefulness is a great built-in feature of XPages and something often lacking in other web applications...try the same view navigation exercises using the classic Domino web engine.

In any case, categorization becomes more interesting when two or more category columns are in a view. To provide some working examples of this, a modified form and view were added to **Chapter9.nsf**, namely the **Main Topic2** form and the **subCats** view. A small number of documents with multiple categories have also been created in the sample application so that examples can be quickly constructed. You do not see these documents in the **All Documents** view because the view selection formula on the (**\$All**) view only displays documents created using the **Main Topic** form, and thus excludes those created using **Main Topic2**. Figure 9.14 shows the sample multicategory documents when the **subCats** view is previewed in the client.

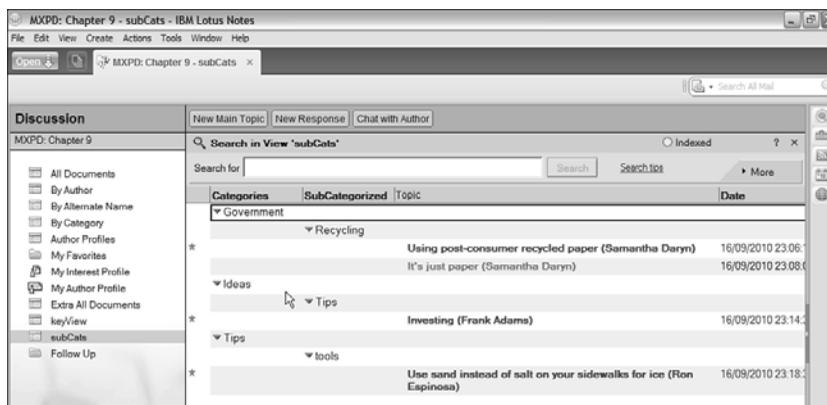
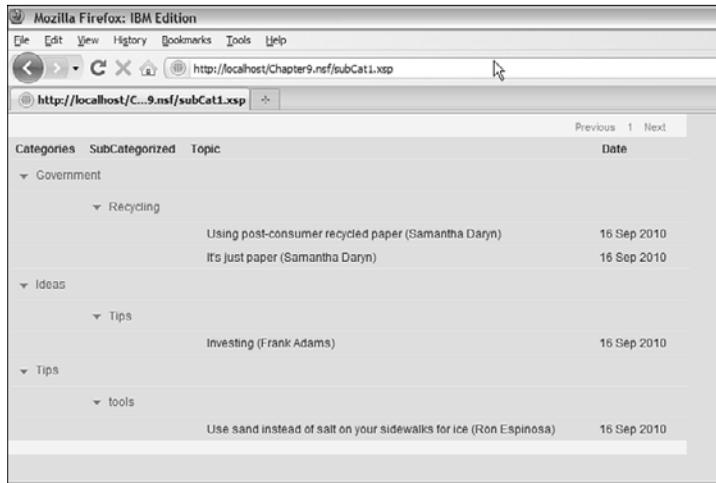


Figure 9.14 Domino view with subcategories

Figure 9.15 shows an XPage named **subCat1.xsp**, which is a default rendering of the **subCats** view. By “default rendering,” I mean that a View Panel control was simply dropped on an XPage and all the columns in the **subCats** view were accepted for inclusion—nothing more than that.



The screenshot shows a Mozilla Firefox browser window displaying a web page. The address bar shows the URL `http://localhost/Chapter9.nsf/subCat1.xsp`. The page content is a table with the following structure:

Categories	SubCategorized	Topic	Date
Government	Recycling	Using post-consumer recycled paper (Samantha Daryn)	16 Sep 2010
		It's just paper (Samantha Daryn)	16 Sep 2010
Ideas	Tips	Investing (Frank Adams)	16 Sep 2010
Tips	tools	Use sand instead of salt on your sidewalks for ice (Ron Espinosa)	16 Sep 2010

Figure 9.15 View Panel with subcategories

If you experiment with the XPages View Panel and the Notes view, you find that the presentation and behavior of both are identical. The category columns are automatically rendered as action links with twistie icons, both of which serve to expand and collapse the category row. Apart from this specialized behavior, all the regular column properties described thus far can also be applied to category columns, they can be reordered within the View Panel so they are not contiguous, and so on.

Although adding two or more categorized columns to a view is one way of implementing subcategorization, an alternative method seems to be a common practice. That is, instead of having multiple categorized columns in the view, which map to fields in the underlying form, the view has just one category column but it can support multiple categories through the use of a “category\subcategory” data-format notation. Thus, if a user enters something like “Government” as a category value, this is interpreted as a top-level category. However, if “Government\Recycling” is entered by the user into the Categories field when creating a document, the document is categorized in a “Recycling” subcategory within the top-level “Government” category.

To provide an example of this, an alternative sample NSF is provided for this chapter, namely **Chapter9a.nsf**. Some of the sample documents contained in **Chapter9.nsf** have been recategorized in the manner just described (which is why you need a separate database). Figure 9.16 shows an example of a redefined category field as inspected in a Notes infobox and how these updated documents are displayed in the Notes client.

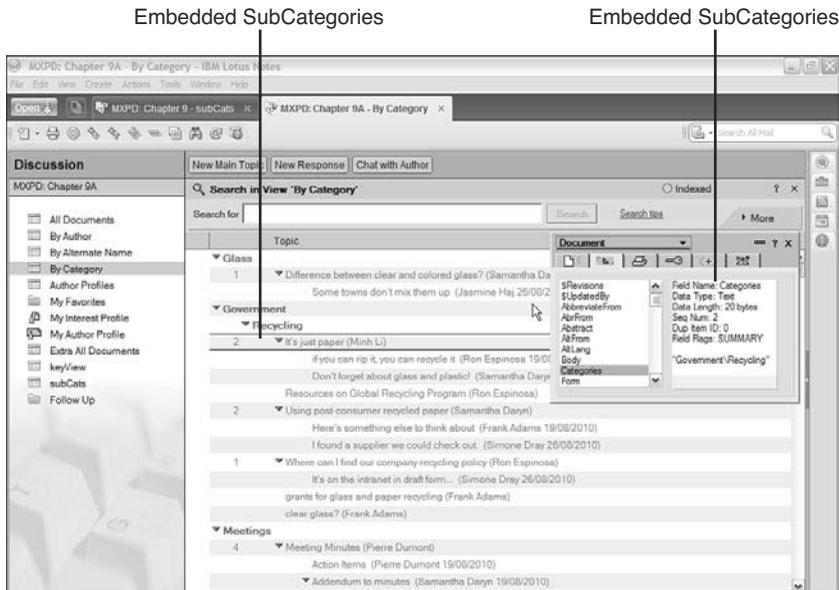


Figure 9.16 Category field containing hierarchical categories

Observe that the Notes client view indents the new subcategories tucked in under the main categories. You have little or no control over this particular rendering because it is built-in view behavior. However, if you repeat the exercise described for Figure 9.15 and create an XPages View Panel to do a default rendering of this view, you notice a problem (refer to **subCatsA.xsp** in **Chapter9a.nsf** for convenience). As shown in Figure 9.17, XPages recognizes the entries as category columns, but the subcategories are not indented. The next section describes how to address this.

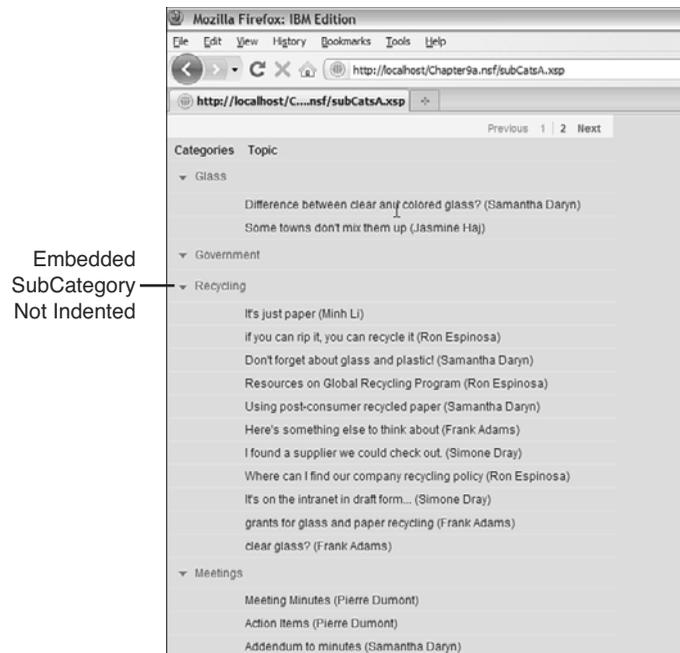


Figure 9.17 XPages View Panel default rendering of embedded subcategories

Making It Look Like Notes!

Building an XPage to emulate the Notes client rendering can be achieved in the following eight steps:

1. Create a new XPage called **subCatsB.xsp** and add a View Panel from the palette.
2. Bind to the **By Category** view but only include the **Topic** column.
3. As shown earlier, insert a new column before the **Topic** column and give it a title of “**Categories**” by updating the view column header.
4. In the **Display** panel set the **Content type** to HTML.
5. Add `var="rowData"` to the `<xp:viewPanel>` tag to gain access to the current row via server-side JavaScript while the View Panel is being populated.
6. Add the following server-side JavaScript snippet to compute the column’s value:

```
if (rowData.isCategory()) {
    // Use the standard twistie icons
    var src =
        "/xsp/.ibmxxspres/global/theme/common/images/expand.gif";
```

```

    // Get the value of the Categories column
    var colValue = rowData.getColumnValue("Categories");
    // Return "Not Categorized" for null or undefined data
    if (typeof colValue == 'undefined' ||
        colValue == null) {
        colValue = "Not Categorized";
    }
    // Invert the twistie depending on row state
    if (rowData.isExpanded()) {
        src =
        "/xsp/.ibmxspress/global/theme/common/images/collapse.gif";
    }
    // return the <span> tag including the twistie & value
    return "<span style='cursor:pointer'><img src='" +
        src + "' alt='' class='xspImageViewColumn'/>" +
        colValue + "</span>";
}

```

7. Add the following server-side JavaScript snippet to compute the column's style property, i.e. **All Properties > Styling > Style > Compute value**:

```

if (rowData.isCategory()) {
    // This API tells us if a category column is indented
    var indent = rowData.getColumnIndentLevel();
    // Insert padding for each indent level
    if (indent == null || indent == 0) {
        return "padding-left:0px";
    } else {
        return "padding-left:10px";
    } // continue if deeper category levels exist ...
};

```

8. Move to the **Events** tab for this column and for the only defined event, `onclick`, add another server-side JavaScript snippet:

```

rowData.toggleExpanded();

```

The `subCatsB.xsp` XPage has already been created for you in **Chapter9a.nsf**, so you can load this or preview your own creation if you have worked through the steps above. In either case the results you see should match those shown in Figure 9.18.

Indented Embedded SubCategory

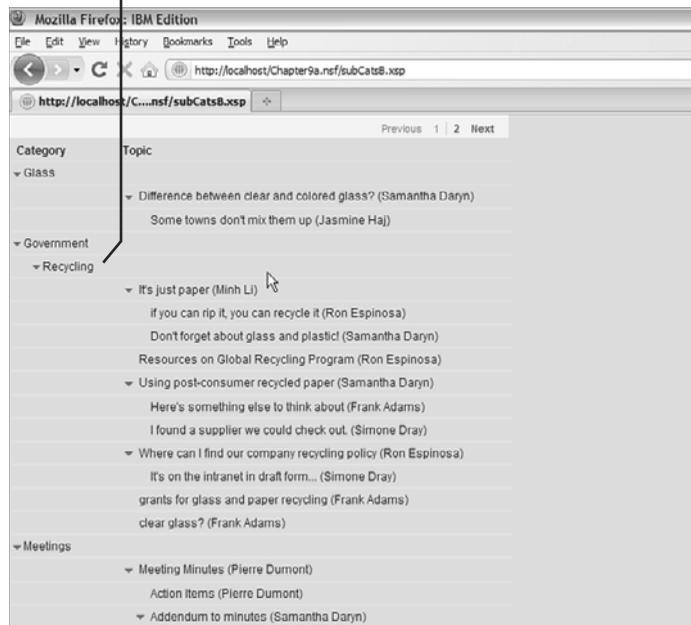


Figure 9.18 XPages View Panel displaying inline subcategories

The key pieces to the customized category column shown in Figure 9.18 are achieved using server-side JavaScript. Obviously, the `NotesXspViewEntry` class exposed via the `rowData` object is critical when working on view customizations as it gives full programmatic access to each view row as it is rendered. This JavaScript class is a pseudo class for the `DominoViewEntry` Java class defined in the XPages runtime, which, in turn, wraps the `ViewEntry` class defined in Notes Java API. JavaScript pseudo classes such as this one allow you to access the associated Java class without having to enter the entire package name, and have an automatic built-in type-ahead facility for method names when used in the JavaScript editor. In this example, for each row it allows you to

- Check if the row is a category: `rowData.isCategory()`
- Get the column value: `rowData.getColumnValue("Categories")`
- Check the expand/collapse state of the row: `rowData.isExpanded()`
- Check for embedded categories: `rowData.getColumnIndentLevel()`
- Toggle the expand/collapse state of the row: `rowData.toggleExpanded()`

Appendix A, “XSP Programming Reference,” includes documentation resources that provide a full outline of the `DominoViewEntry XPages` class, which `NotesXspViewEntry` uses under the covers. It is worthwhile to study this class in more detail to get to know the full set of tools you have at your disposal when working on view customizations. You can also resolve the mappings for any JavaScript/Java classes using a handy tool on the Domino Designer wiki:

www-10.lotus.com/ldd/ddwiki.nsf/dx/XPages_Domino_Object_Map_8.5.2

The other interesting tidbit from this example is that it exposes the internal URLs used to locate embedded runtime resources like images, style sheets, and so on. The following URL, for example, points to the standard row expansion twistie that is part of the XPages runtime:

```
"/xsp/.ibmjspxres/global/theme/common/images/expand.gif"
```

You see URLs just like this one whenever you view the source of a rendered XPage in a browser, and you can use these URLs as has been done in this example as part of your own customizations.

TIP Prior to the Notes/Domino 8.5.2 release, it was not possible to dynamically compute the column style property, as is done here. This issue has been addressed; however, if you are using an older version, you can still achieve the same result by computing the `styleClass` property. It just means that you must return class names instead of inline CSS, and you need a style rule defined in a CSS resource for each name returned. A tad more awkward, but it’s no big deal...although it’s another good reason to move to 8.5.2 if you have not already upgraded!

Incidentally, a similar technique can be used to render category view columns inline like this, even when they are managed as separate category columns, i.e. as was the case with the **subCats** view used in **Chapter9.nsf**, shown in Figure 9.14. A **subCats2.xsp** XPage has been included in that sample application to illustrate how to reformat the column category display. In essence, however, it is only the server-side JavaScript code outlined previously in steps 6 and 7 that has been modified. Listing 9.3 shows the revised code that computes the column value and the style property.

Listing 9.3 Server-Side JavaScript for View Column value and style Properties

```
<xp:this.value>
  <![CDATA[#{javascript:if (rowData.isCategory()) {
    // Use the standard twistie icons
    var src = "/xsp/.ibmjspxres/global/theme/common/images/expand.gif";
    // Look for the deepest subcategory first
    var colValue = rowData.getColumnValue("SubCategories")
    // If not found, keep looking back until back to top level cat
    if (colValue == null) {
      colValue = rowData.getColumnValue("Categories");
```

```

    }
    // Return "Not Categorized" for null or undefined data
    if (typeof colValue == 'undefined' || colValue == null) {
        colValue = "Not Categorized";
    }
    // Invert the twistie depending on row state
    if (rowData.isExpanded()) {
        src = "/xsp/.ibmjspxres/global/theme/common/images/collapse.gif";
    }
    // return the <span> tag including the twistie & value
    return "<span style='cursor:pointer'><img src='" + src +
        "' alt='' class='xspImageViewColumn'>" + colValue +
        "</span>";
    }]]]>
</xp:this.value>
<xp:this.style>
    <![CDATA[#{javascript:
    if (rowData.isCategory()) {
        // Start at the deepest subcategory and work back to root
        var colValue = rowData.getColumnValue("SubCategories");
        // Insert padding for 10 pixel padding for 2nd column
        if (colValue != null && colValue != "") {
            return "padding-left:10px";
        }
        // Insert more padding if needed back to the top level
        } else {
            return "padding-left:0px";
        }
    }
    }]]]>
</xp:this.style>

```

As you can see from the code, the principle is exactly the same as previously, but the means of detecting the category columns has changed. No longer are the column values embedded in the `Category\Subcategory` fashion, so the `rowData.getColumnIndentLevel()` API is of no use here. Instead, the indentation is determined based on the structure of the backend view—the deepest subcategory columns are sought first, rewinding to the top level if no value is found. Load the `subCats2.xsp` page and compare the results to Figure 9.15.

This tucked-in form of category styling seems popular in the community based on various Notes app dev forum postings and other customer feedback, so hopefully this section clarified how to achieve the Notes client look and feel in XPages. It might become a standard View Panel property in a future release.

View Properties and View Panel Properties

When working with views, any features to do with data structure and content are defined at the backend in the view design element itself—you have just seen with this with the sorting and categorization examples, insofar as these capabilities needed to be enabled in the view. The view design element also contains properties that are purely related to presentation within the Notes client or classic web engine and, as such, do not apply to the XPages view controls. For example, the **Type** option in Figure 9.10 defines whether a categorization data is maintained for a particular column in the view, but the twistie options contained in the adjacent tab (see Figure 9.19) only apply to native Notes rendering and not to XPages.

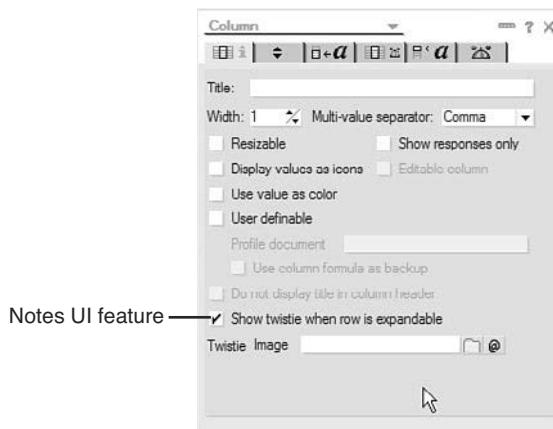


Figure 9.19 View Column Presentation properties

It is important to be able to distinguish the native view rendering features from the XPages View control presentation properties. In **Chapter9.nsf** a new version of the (**\$xpByAuthor**) view, namely (**\$xpByAuthorExt**), has been provided for use in an example that helps clarify this area. The extended view contains an extra column that totals the byte size of the documents for each category. These totals are shown in the Notes client for each category only, but can be displayed for each individual row entry if so desired. The hide/show nature of this data is determined using the **Hide Detail Rows** checkbox shown in Figure 9.20.

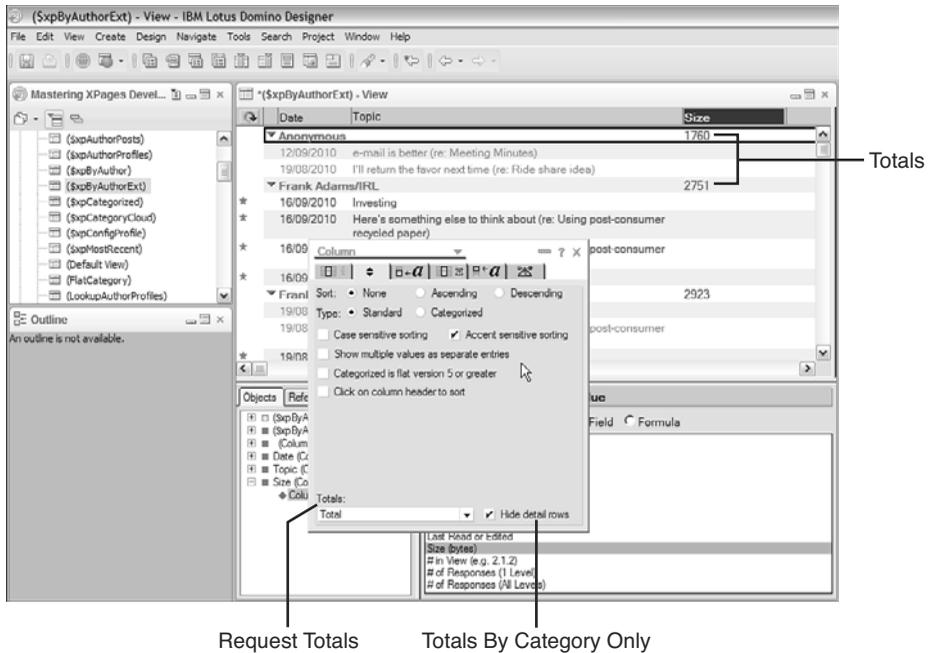


Figure 9.20 (\$xpByAuthorExt) with document size totals for each category

If you toggle the **Hide Detail Rows** checkbox value and refresh the view data from within Designer, you see the document byte size displayed for each entry. An agent has also been supplied in the sample application, which prints the column values for each view row entry using the Java API. The agent (`getViewEntryData`) details are shown in Listing 9.4.

Listing 9.4 Java Agent to Print View Column Data

```
import lotus.domino.*;

public class JavaAgent extends AgentBase {
    public void NotesMain() {
        try {
            // Standard agent code to get session & context objects
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();
            // get the current db and the new ($xpByAuthorExt) view
            Database db = session.getCurrentDatabase();
            View view = db.getView("$xpByAuthorExt");
            // iterate over each view entry and print the Topic & Size
            ViewEntryCollection vec = view.getAllEntries();
            if (vec != null) {
```

```

        for (int i = 0; i < vec.getCount(); i++) {
            ViewEntry ve = vec.getNthEntry(i);
            if (ve != null)
                // just get the 3rd & 4th column values
                // ViewEntry index is zero-based!
                System.out.println(
                    ve.getColumnValues().get(2)
                    + " " +
                    ve.getColumnValues().get(3) );
        }
    }
} catch(Exception e) {
    e.printStackTrace();
}
}
}
}

```

Listing 9.5 shows some sample output generated when the (**\$xpByAuthorExt**) view is configured to hide detail rows. To run the agent yourself in Designer, you first launch the Java debug console (**Tools > Show Java Debug Console**), right-click `getViewEntryData` in the agent view, and select the **Run** menu. All the `println` output then appears in the Java console. As you can see, the detail totals rows are all included in the data returned by the `getColumnValues()` API call *regardless* of **Hide Details Rows** property setting.

Listing 9.5 Snippet of Java Agent Output

```

...
if you can rip it, you can recycle it (re: It's just paper) 573.0
It's just paper 618.0
Using post-consumer recycled paper 1045.0
who't this? (re: Meeting Minutes) 629.0
phone number inside (re: Meeting Minutes) 631.0
Difference between clear and colored glass? 927.0
...

```

Because XPages depends on the Java API to populate its View control, the detail rows appear in *any* XPages View control that includes the **Size** column. The **Hide Detail Rows** property is really just used in the core view rendering code and not honored in the programmability layer. Given the view customization tips and tricks you have learned thus far, you are now be in a position to figure out how to emulate Notes **Hide Detail Rows** view display property in XPages! All you really need to do is not show the **Size** column value when the row is not a category. This

is done for you in `hideDetails.xsp` page in `Chapter9.nsf`, which contains a View Panel with four standard columns (**Name**, **Date**, **Topic**, **Size**) plus a computed column. The server-side JavaScript used to compute the column value is trivial, as demonstrated in Listing 9.6.

Listing 9.6 Server-Side JavaScript Snippet to Emulate Hide Detail Rows in a View Panel

```
<xp:this.value>
<![CDATA[#{javascript:
    // Only show the Total column value for category rows
    if (rowData.isCategory()) {
        return rowData.getColumnValue("Size");
    }]}></xp:this.value>
<!-- Also include a converter to display whole numbers only -->
<xp:this.converter>
    <xp:convertNumber type="number"
        integerOnly="true">
    </xp:convertNumber>
</xp:this.converter>
```

The converter just used was added via the same **Data** property panel used to add the JavaScript code in Designer. Simply set the **Display type** to `Number` and check the **Integer only** control to eliminate the decimal points you see printed in the raw data in Listing 9.5. When loaded or previewed, the `hideDetails` XPage looks like Figure 9.21.

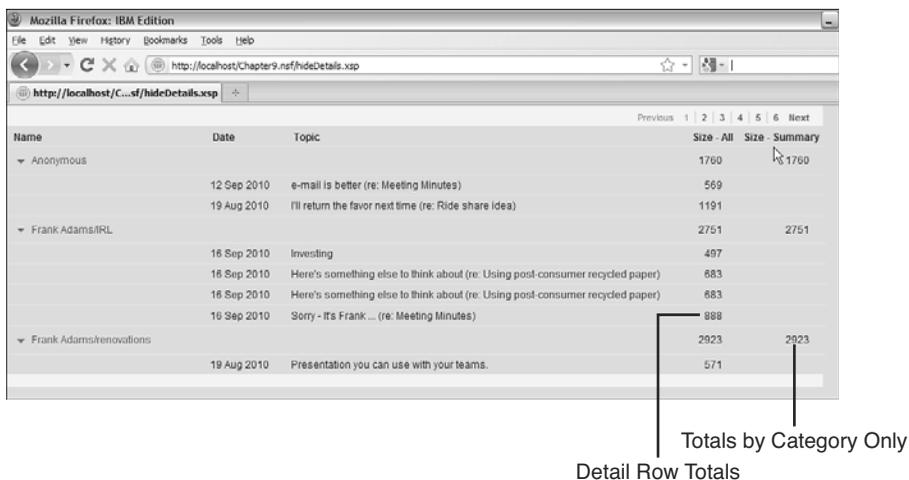


Figure 9.21 XPage with totals for detail and category-only rows

The discussion thus far covered all the main View Panel properties and dived into examples of how to customize View Panels using server-side JavaScript and other tools. The next most logical focus area for the View Panel would be styling. No doubt, as you have examined the View Panel properties, you noticed a slew of specialized style class properties (`rowClass`, `columnClass`, `viewClass`, and so on), which can modify its appearance. Rather than do that here in this chapter, it is covered in the section, “Working with Extended styleClass and Style Properties,” in Chapter 14, “XPages Theming.” The discussion here instead shifts to the Data Table container control.

Data Table

The Data Table uses a simple table structure to display content. The table is configured to contain three row elements, such as a header, a content row, and a footer. The header and footer typically contain static elements, such as column titles, pagers, or just arbitrary one-off control instances. The content row usually contain a collection of individual controls that are bound to elements of a data source, and this row is then rendered repeatedly for each entry in the data source (once for every row in a view) when the Data Table is invoked as part of a live application.

Unlike a View Panel, however, all the controls contained in the Data Table must be added and bound manually, and certain other capabilities are simply not available, e.g. categorization. In essence, it is like a dumbed-down View Panel control, but it can be useful if you need to display simple nonhierarchical data in a customized fashion. You see an example of a good use case in this section.

To start with, try to present a regular view using a Data Table to get familiar with its features and behaviors. You should create a new XPage, say **myDataTable.xsp**, and drag-and-drop a Data Table control from the palette. Compared to the View Panel drag-and-drop experience, you might be underwhelmed with results. Basically, a shell of a table is created, and it’s pretty much up to you to populate it with controls and bind these in a meaningful way.

Designer prompts you that a data source needs to be created if one does not already exist on the page, so for the purposes of this example, you should create a view data source targeting the **xpAllDocuments** view. This can be done in a number of ways, such as from the **Data** property panel on the XPage itself or using the **Define Data Source** combo box entry on the **Data** palette data source picker. Whatever your preferred route might be, simply pick the aforementioned view as the data source. Even though you now have a page containing a Data Table and a view data source, they are not connected and know nothing about each other. You can wire these together using the main **Data Table** property panel, as shown in Figure 9.22.

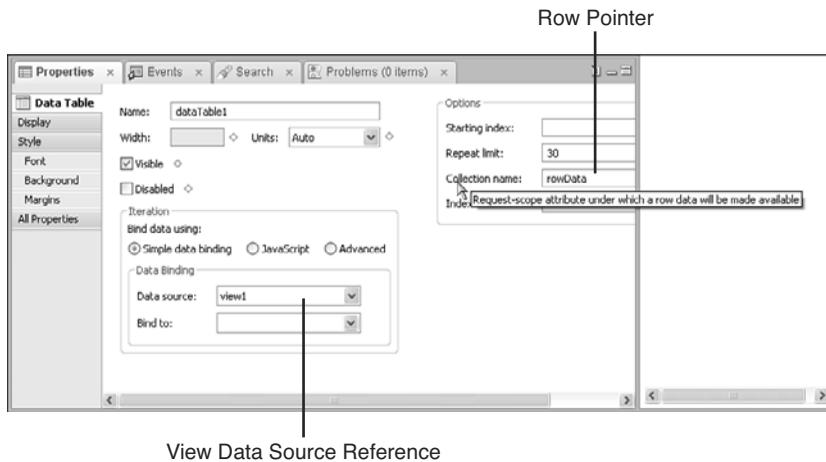


Figure 9.22 Connecting a Data Table to a view data source in Designer

With the Data Table entry selected in the **Outline** view, pick the newly created view data source instance (“viewAll”) using the **Data source** combo box, and you also need to enter a **Collection name**. The collection name, “rowData” in this example, is used as the object to gain programmatic access to each row entry as it is being rendered—just as it was in the View Panel examples earlier. Rather than use server-side JavaScript in this case, however, you could just use simple Expression Language (EL) bindings. First, however, you need some controls to display the row data, so drag-and-drop a Computed Field from the **Core Controls** palette to the first cell in the middle row and then repeat the process for the adjacent table cell. These Computed Field instances can be selected and bound using EL expressions—or **Simple data binding**, as it is described in Designer’s **Value** property panel and displayed in Figure 9.23. Bind the first field to the `_MainTopicsDate` column and the second field to the `_Topics` column.

The EL data binding markup generated by Designer has the following form. The name of the column is provided as a key to the row data entry:

```
{rowData['_MainTopicsDate']}
```

TIP You can use EL expressions or server-side JavaScript for data binding. The EL expression `rowData['_MainTopicsDate']` produces the same result as `rowData.getColumnValue("_MainTopicsDate")` in JavaScript. Some column names, however, are incompatible with the EL expression language and thus cannot be used at all. For example, many column names in the standard Domino templates begin with a dollar symbol, such as **\$126**, **\$150**, and so on. An EL expression like `rowData['$126']` would be expanded to a Java bean expression like `rowData.get$126()`, which is illegal in the Java language. It was precisely for this reason that this example uses the **xpAllDocuments** view rather than the **(\$All)** view. The former is essentially the same view as the latter, but with column names that are EL friendly. In this sense, JavaScript binding can be less problematical than EL binding, especially if you happen to have no control over the names of the data source elements.

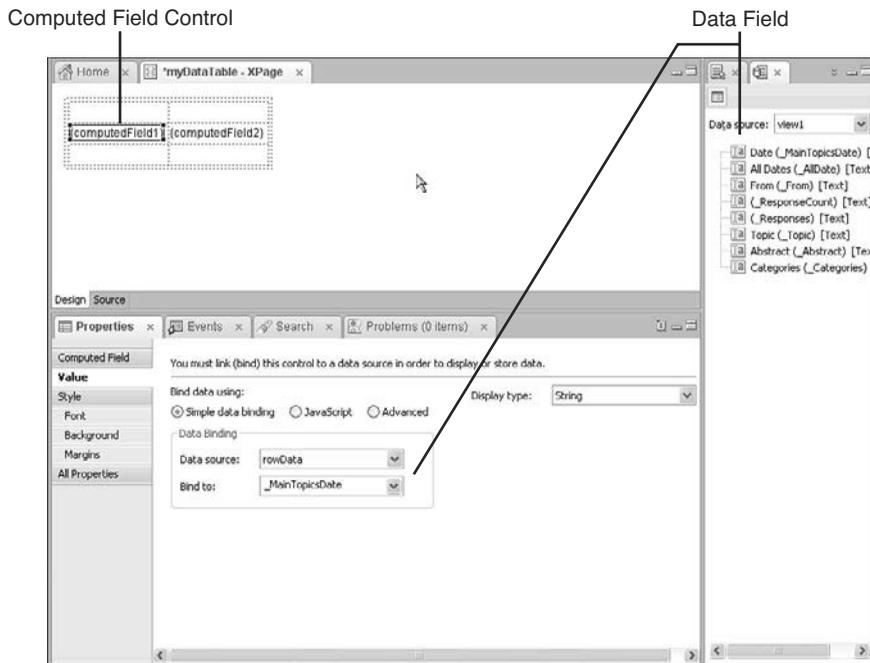


Figure 9.23 Binding a Computed Field to a view data source element in Designer

You should also drop two Label controls from the palette directly into the two cells in the top row of the Data Table and change their values to Date and Topic, respectively. You can also assign the Data Table a width of 600 pixels for quick aesthetics using the **Width** and **Units** controls shown in Figure 9.22. After you complete this step, you are ready to preview or load this Data Table. The results should be just like the page you see displayed in Figure 9.24.

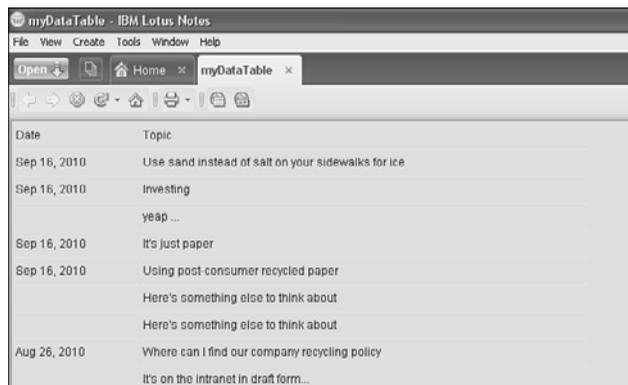


Figure 9.24 Data Table displaying data from xpAllDocuments view

The Data Table could do with a pager to split the rows into manageable chunks. The first step is to set the `rows` property of the Data Table to smaller number than its default value of 30 (for example, 10). Interestingly, the pager you have worked with up to now in the View Panel is not an intrinsic part of that control, but an independent entity that can be used with any of the view controls. The View Panel just happens to include a pager instance by default. To add a pager to the Data Table, look for the Pager control in the **Core Controls** palette and drag it into one of the footer cells. Then, activate the **Pager** property panel and attach it to the Data Table by picking the ID of the Data Table from the **Attach to** combo box—where Designer kindly enumerates a list of eligible candidate controls for you! At the same time, turn on partial refresh so that paging updates are performed using AJAX. The various property panel selections are shown in Figure 9.25.

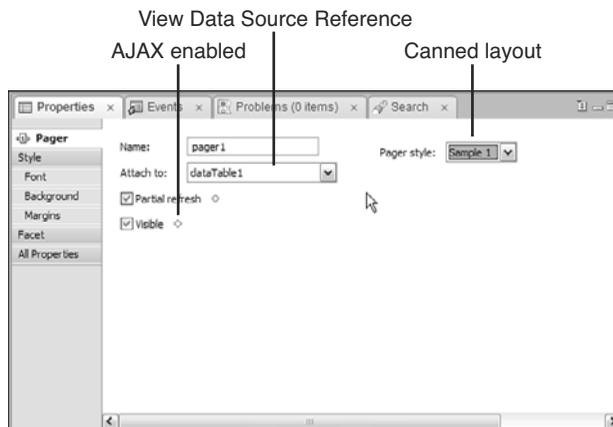


Figure 9.25 Pager property panel

Because the Pager is capable of working with any view control, you must nominate a target container. The **Partial refresh** checkbox selection instructs XPages to update just the targeted view control via an AJAX request when a pager action is executed. This means that only the view data in the Data Table is refreshed when the end user navigates from one page to the next, which is obviously more efficient than refreshing the entire page every time.

The only problem with the pager right now is that it resides in the wrong place. It has been dropped into the footer cell of a column when it really needs to be in the footer of the Data Table itself. Unfortunately, the footer of the Data Table is not an identifiable drag-and-drop target in Designer, so you must go to the **Source** pane move the markup manually. Simply cut and paste the entire `<xp:pager>` tag from its current location so that it is a direct child of the Data Table. It should also be wrapped in a `<xp:this.facets>` tag—see the final markup in Listing 9.7.

To best illustrate the effect of the AJAX partial refresh, however, it is worthwhile adding two more Computed Fields to the XPage. Place the first Computed Field in one of the Data Table

footer cells and then the second control can be dropped anywhere else on the page as long as it is outside the Data Table. Then, add the following server-side JavaScript as the computed value for both fields:

```
@Now().getMilliseconds();
```

Domino developers no doubt are familiar with the `@Now()` function, which returns the current data and time. The `getMilliseconds()` call expresses the time in milliseconds when the page is loaded. When you load or preview the page, both fields should display the same number. If you start navigating through the view data using the navigator, you notice that the Computed Field within the Data Table is updated with the current time milliseconds value while the field external to the Data Table is not. This demonstrates the efficient behavior of the partial refresh feature.

Figure 9.26 shows the updated XPage in action. The full markup is done for you in the `dataTable.xsp` XPage in `Chapter9.nsf` and is printed in Listing 9.7.

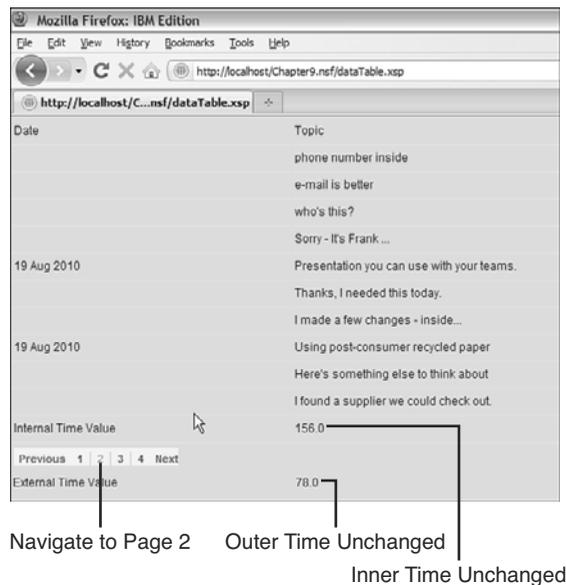


Figure 9.26 Data Table with partial refresh paging enabled

Listing 9.7 XSP Markup for SampleData Table

```
<?xml version="1.0" encoding="UTF-8" ?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core">
  <!-- The data source defined at root level -->
  <xp:this.data>
```

```

    <xp:dominoView var="viewAll"
        viewName="xpAllDocuments"></xp:dominoView>
</xp:this.data>
<!-- The data table finds the data source using value prop -->
<xp:dataTable id="dataTable1" rows="10" var="rowData"
    value="#{viewAll}" style="width:600px">
    <xp:column id="column1">
        <!-- column header and footer entries -->
        <xp:this.facets>
            <xp:label value="Date" id="label1"
                xp:key="header"></xp:label>
            <xp:label value="Internal Time Value"
                id="label3" xp:key="footer"></xp:label>
        </xp:this.facets>
        <!-- Bound to the date field using EL -->
        <xp:text escape="true" id="computedField1"
            value="#{rowData['_MainTopicsDate']}">
        </xp:text>
    </xp:column>
    <xp:column id="column2" style="width:300px">
        <xp:this.facets>
            <!-- column header and footer entries -->
            <xp:text escape="true" id="computedField3"
                xp:key="footer"
                value="#{javascript:@Now().getMilliseconds();}">
            </xp:text>
            <xp:label value="Topic" id="label2"
                xp:key="header"></xp:label>
        </xp:this.facets>
        <!-- Bound to the Topic field using EL -->
        <xp:text escape="true" id="computedField2"
            value="#{rowData._Topic}">
        </xp:text>
    </xp:column>
    <xp:this.facets>
        <xp:pager layout="Previous Group Next" id="pager1"
            for="dataTable1"
            xp:key="footer"
            panelPosition="left"
            partialRefresh="true">

```

```

        </xp:pager>
    </xp:this.facets>
</xp:dataTable>
<!-- Table only used for layout alignment -->
<xp:table style="width:600px;text-align:left">
    <xp:tr><xp:td>
        <xp:label value="External Time Value"
            id="label4">
        </xp:label></xp:td>
        <!-- external computed field -->
        <xp:td style="width:300px; text-align:left">
            <xp:text escape="true" id="computedField4"
                value="#{javascript:@Now().getMilliseconds();}"
                style="text-align:left"></xp:text>
        </xp:td>
    </xp:tr>
</xp:table>
</xp:view>

```

Although working with the Data Table may be vaguely interesting, it must occur to you that what you have just built could be achieved using a View Panel control in a fraction of the time with just a few point-and-click operations. So, why bother with the Data Panel at all? The answer is that the Data Panel can be useful when you want to build a small bare bones tabular view with a highly customized user interface. Perhaps these use cases are not commonplace but they do occur. The next exercise serves as a good example.

Building a Mini Embedded Profile View using a Data Table

Carry out the following steps, drawing on what you learned in the current section up to this point:

1. Create a new XPage called **dtProfile.xsp** and add a Data Table from the palette.
2. Create a view data source targeting the **xpAuthorProfiles** view.
3. Connect the Data Table to the data source and set its **Collection name** to "rowData" in the Data Table property sheet. This should result in a `var="rowData"` attribute being created in the underlying `<xp:dataTable>` tag.
4. Append two new columns to the Data Table using the right mouse menu.
5. Add a Computed Field to the 1st content cell; that is, first column, middle row.
6. Bind this field to the **From** column in the data source using JavaScript:

```
rowData.getColumnValue("From")
```

7. Add a link control for the palette to both the 2nd and 3rd cells in the content row.

8. For the first link, activate the **Link** property panel and set the **Label** and **Link type** fields. For the label, enter “email” in the edit box, and then for the latter, add some server-side JavaScript to compute a URL. This is a `mailto` URL, created by simply concatenating a “mailto:” to the **Email** column value, as follows:

```
"mailto:" + rowData.getColumnValue("Email")
```

9. Set the label for the second link to “Download” and compute its type in the same way as before, this time building a Domino resource image URL like this:

```
"/" + rowData.getUniversalID() + "/$FILE/" +  
rowData.getColumnValue("FileUpFilename")
```

10. Drag-and-drop an image control to the fourth and final content row cell, using the **Use an image placeholder** radio button for now so that you can compute the image reference.
11. In the **Image** property panel, compute the **Image source** using *exactly the same* server-side JavaScript as previously shown.
12. For presentation purposes, select the **All > Style** cell in the property panel for each Data Table column and set this CSS rule:

```
text-align:center; vertical-align:middle
```

13. In the same way, set the **All > Style** property for the Data Table itself to this:

```
width:400px;
```

You already practiced most of the 13 steps in one way or another when working through View Panel or Data Table examples, so only a few steps need any further explanation.

Step 6 simply returns the name of the author of the document. This is in Notes canonical form, so it would be more natural to present the common user name in this column instead. Experienced Domino developers instinctively know to do this using the `@Name @Function`, which can reformat Notes names in a number of ways. Although `@Functions` and other traditional building blocks are covered in more detail in Chapter 11, “Advanced Scripting,” in the section, “Working with `@Functions`, `@Commands`, and Formula Language,” it is no harm to start dabbling with some simple use cases at this stage according as the need arises. To do this, simply wrap the JavaScript binding command in with an `@Name()` call:

```
@Name(" [CN]", rowData.getColumnValue("From"));
```

Step 9 uses JavaScript to build a Domino resource URL. The generic form of this URL is

```
/UNID/$FILE/filename
```

where the first part is an ID to identify the document to use, the second part indicates that the URL represents a file attachment resource, and the third part is the name of the attachment. This form of URL has been used in classic Domino web development for a long time. Back in Chapter 3, you learned about special IDs that Notes maintains to manage its databases and documents. The universal ID (UNID) is a 32-character hexadecimal representation that uniquely

identifies a document. The profile documents in the Discussion template each contain a single image (or placeholder image) of the author and the name of this image file can be obtained from the **FileUpFilename** column in the **xpAuthorProfiles** view. Thus, a resource URL can be dynamically constructed for all registered users and this URL resolves the image and retrieves it from the profile documents for display in the Data Table. An example of a real live resource URL is highlighted in the status bar of the browser in Figure 9.27.

You are now ready to preview or load the new XPage. **Chapter 9.nsf** contains some sample profile documents, so you see these listed in the Data Table. The actual intention, however, is to display this Data Table as an embedded view in the **My Profile** page. To do this, you need to open the **authorProfileForm** custom control and copy/paste the markup from **dtProfile.xsp** to the bottom of the XPage, just before the final `</xp:view>` tag. Naturally, you do not copy the `<xp:view>` tag from **dtProfile.xsp** but just the Data Table and data source markup—everything you see in Listing 9.8. Figure 9.27 shows a snapshot of a **My Profile** page from **Chapter9.nsf**.

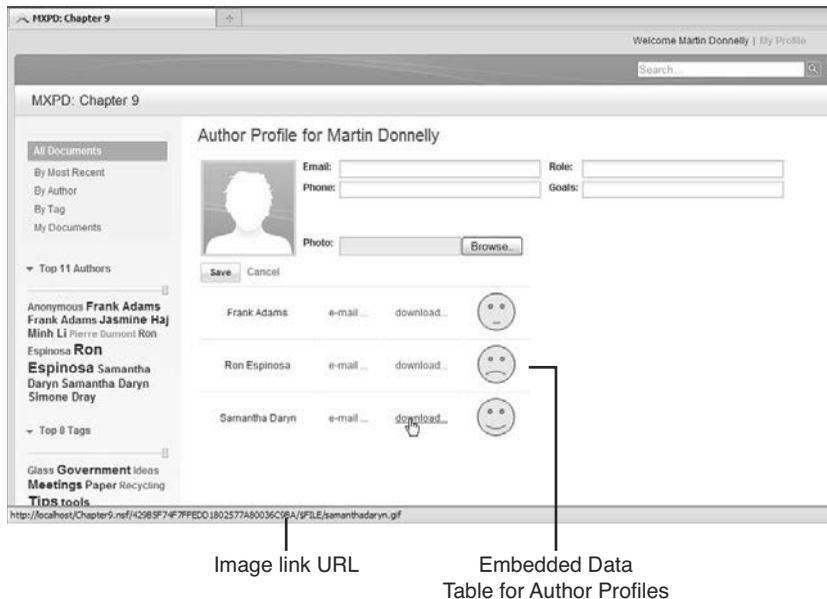


Figure 9.27 My Profile Page with Embedded Data Table

TIP The next chapter introduces the XPage custom control and discusses all of its features in great detail. Suffice to say, at this stage that, it would have been a better design approach to create **dtProfile.xsp** as a custom control and drop it into **authorProfileForm.xsp** rather than copying and pasting the actual code. If you are already familiar with custom controls, it is trivial to rework this example accordingly. If not, perhaps it is worth revising this example to use a custom control after you read Chapter 10.

Listing 9.8 Data Table Displaying Profile Data

```

<xp:this.data>
  <xp:dominoView var="view1" viewName="xpAuthorProfiles">
  </xp:dominoView>
</xp:this.data>

<xp:dataTable id="dataTable1" rows="30" value="#{view1}"
  var="rowData" style="width:400px">
  <!-- style each column like this -->
  <xp:column id="column1"
    style="text-align:center; vertical-align:middle">
    <!-- get the common user name -->
    <xp:text escape="true" id="computedField1">
      <xp:this.value><![CDATA[#{javascript:
        @Name(" [CN]", rowData.getColumnValue("From"));
      }]]></xp:this.value>
    </xp:text>
  </xp:column>
  <xp:column id="column2"
    style="text-align:center;vertical-align:middle">
    <!-- return a mailto link -->
    <xp:link escape="true" text="e-mail ..." id="link2">
      <xp:this.value><![CDATA[#{javascript:"mailto:" +
        rowData.getColumnValue("Email");}]]></xp:this.value>
    </xp:link>
  </xp:column>
  <xp:column id="column3"
    style="text-align:center; vertical-align:middle">
    <!-- return Domino resource URL -->
    <xp:link escape="true" text="download..." id="link1">
      <xp:this.value><![CDATA[#{javascript:
        "/" + rowData.getUniversalID() + "/$FILE/" +
        rowData.getColumnValue("FileUpFilename")}]]>
      </xp:this.value>
    </xp:link>
  </xp:column>
  <xp:column id="column4"
    style="text-align:center; vertical-align:middle">
    <!-- use the same Domino resource URL for the image -->
    <xp:image id="image2" style="height:50px;width:50.0px">

```

```

<xp:this.url><![CDATA[#{javascript:"/" +
rowData.getUniversalID() + "/$FILE/" +
rowData.getColumnValue("FileUpFilename")}]]>
</xp:this.url>
</xp:image>
</xp:column>
</xp:dataTable>

```

Had you used a View Panel for this particular use case, you would have had to undo a lot of the features it gives you for free, such as pagers, column headers, and so on. You would also have had to customize the columns to display HTML and then return link and image HTML elements for three of the four columns. The Data Table actually simplifies the process by allowing you to drag-and-drop and arbitrary control into any content row cell and then just compute its value.

Another good example of Data Table usage is the File Download control. This out-of-the-box control is really a Data Table that has been adapted by the XPages runtime to display a simple table of any attachments contained in a nominated rich text field. Figure 9.28 shows the File Download control displaying some attachments in the Discussion application—it should be easy to see how this was built, given what you have just done to implement the embedded profile Data Table.

That is the Data Table, all done and dusted!

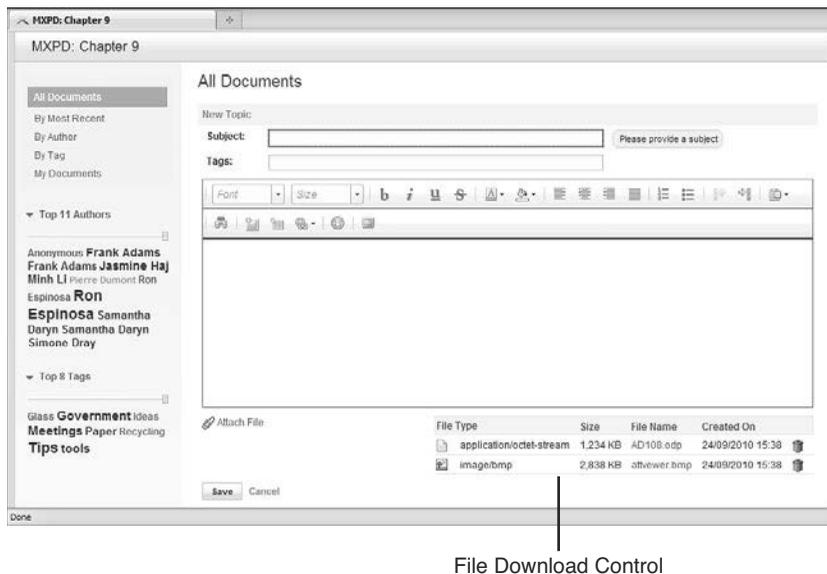


Figure 9.28 Example of the File Download control in the Discussion application

Repeat Control

The Repeat control is similar to the Data Table. The Repeat control does not have a table structure, but just like the Data Table, it can contain arbitrary controls that can be bound to elements of a collection object (like a Domino view or Java array). When the Repeat control is rendered, all child controls are repeated for each entry in the data source.

In fact, to prove just how similar the two controls are, do a quick exercise that involves rebuilding the previous Data Table as a Repeat. The steps are

1. In the Designer Navigator, copy and paste the **dtProfile.xsp** XPage.
2. Rename the new copy from **dtProfile_1** to **repeatProfile** and open it in Designer (the Designer right-mouse menu has a **Rename** option).
3. Use the Find/Replace dialog (Ctrl-F) to replace all occurrences of `dataTable` with `repeat`.
4. In the **Source** pane, delete all the `<xp:column ...>` and `</xp:column>` tags from **repeatProfile.xsp**.
5. Just before the closing `repeat` tag, `</xp:repeat>`, insert a line break using these tags `<xp:br></xp:br>`.
6. Move to the WYSIWYG editor and manually insert some spaces between the child controls so they are not touching each other.

Reload or preview the page and presto! Your new page is now working just as the Data Table page did, although the individual elements do not align as neatly as they would when placed in a table. If you executed the six steps correctly, your **repeatProfile.xsp** should contain the same markup as Listing 9.9.

Listing 9.9 Displaying Profile Data Using a Repeat Control

```
<!-- data source has not changed. -->
<xp:this.data>
  <xp:dominoView var="view1" viewName="xpAuthorProfiles">
  </xp:dominoView>
</xp:this.data>
<!-- dataTable tag changed to repeat -->
<xp:repeat id="repeat1" rows="30"
  var="rowData" style="width:400px" value="{view1}">&#160;
  <!-- removed columns but kept controls exactly as they were -->
  <xp:text escape="true" id="computedField1">
    <xp:this.value><![CDATA[#{javascript:
      @Name("[CN]", rowData.getColumnValue("From"));}]>
    </xp:this.value>
  </xp:text>
```

```

<!-- spaces represented as HTML entities in markup: &#160 -->
  &#160;&#160;
<xp:link escape="true" text="e-mail ..." id="link1">
  <xp:this.value><![CDATA[#{javascript:"mailto:" +
    rowData.getColumnValue("Email");}]]></xp:this.value>
</xp:link>
&#160;&#160;
<xp:link escape="true" text="download ..." id="link2">
  <xp:this.value>
    <![CDATA[#{javascript:"/" +
      rowData.getUniversalID() + "/$FILE/" +
      rowData.getColumnValue("FileUpFilename")}]]>
  </xp:this.value>
</xp:link>
&#160;&#160;
<xp:image id="image1" style="height:50px;width:50.0px">
  <xp:this.url>
  <![CDATA[#{javascript:"/" +
    rowData.getUniversalID() + "/$FILE/" +
    rowData.getColumnValue("FileUpFilename")}]]>
  </xp:this.url>
</xp:image>
<xp:br></xp:br>
</xp:repeat>

```

This exercise shows that the bulk of the properties are shared across both controls and that the containment relationships are compatible—otherwise, your page would not build in Designer, let alone actually work at runtime.

A Repeat Control Design Pattern

Just because the Repeat control is not contained within a table does not mean it cannot use a tabular layout scheme. The **All Documents** page in the Discussion template provides a great pattern for Repeat usage. If you go back to Figure 9.1, which illustrates all the fancy features of the Repeat control, you see the page does have a tabular structure. The top of the view has a set of **Collapse All** | **Expand All** links and a pager—effectively, this is a header. The bottom of the view has a page size picker on the left side and a pager on the other—effectively, this is a footer. The data rows are repeated in between the header and footer using a Repeat control and make use of many other advanced features to generate dynamic content. Figure 9.29 features an outline view of the relevant parts of the page, tagged with pointers identifying various recognizable landmarks.

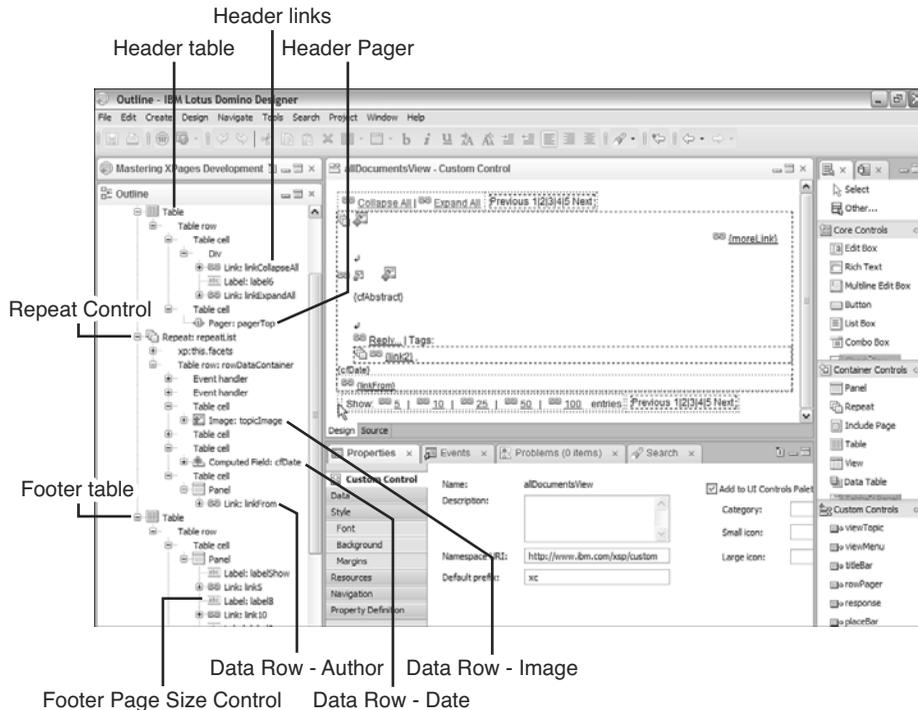


Figure 9.29 Outline structure of the all documents view

As you can see, the header and footer are encapsulated as HTML tables. This content is static, so an HTML table works fine for containment and layout. The middle section, which comprises all the data rows, is also contained in a HTML table, although this may not be immediately obvious. Note that the Repeat has a header facet, which emits an HTML `<table ... >` tag, and a footer facet, which closes the table tag with `</table >`. Again, header and footer facets are not repeated but just rendered once, so this sets up a middle table for the data rows. A table row is then repeated for each entry in the data source (`xpAllDocuments`) and the various table cells are populated with controls, and then bound, formatted, and scripted as required. The only element to be iterated over and repeated, therefore, is the HTML table row tags (`<tr >`), which makes the entire process efficient but, at the same time, well structured. This `Table | Repeat | Table` pattern is a recommended as a best practice for complex views of this nature.

Nested Repeats

Some of the tricks used in the data rows are definitely worth exploring. For example, when it was stated earlier that the Repeat control can contain arbitrary child controls, this does not exclude other Repeat control instances. There is a good example in the `allDocumentsView` custom control of a nested Repeat being put to smart use. The particular snippet of XSP markup is displayed in Listing 9.10, with some comments added in bold script.

Listing 9.10 Nested Repeat Control Bound to a JavaScript Array

```

<!-- Nested Repeat control - note removeRepeat="true" -->
<xp:repeat id="repeatTags" rows="30" var="tagData"
    first="0" indexVar="tagIndex" repeatControls="false"
    removeRepeat="true"
    themeId="Repeat.Tags">
    <!-- Repeat is not bound to a View but to a Java array! -->
    <xp:this.value><![CDATA[#{javascript:
        // Category can be a single string or multi-text item
        var obj = rowData.getColumnValue("_Categories");
        var size = 0;
        var array = null;
        // must return an array regardless!
        if(typeof obj == "string"){
            var str = obj.toString();
            if(str != null){
                array = new Array();
                array[0] = str;
                size = 1;
            }
        }else if(typeof obj == "java.util.Vector"){
            array = obj.toArray();
            size = array.length;
        }
        return array;}}]>
    </xp:this.value>
    <!-- create a link for each item in the tagData array! -->
    <xp:link escape="true" id="link2" themeId="Link.person"
        text="#{javascript:tagData}" value="/byTag.xsp">
        <!-- set the ?categoryFilter param to the array item -->
        <xp:this.parameters>
            <xp:parameter value="#{javascript:tagData;}"
                name="categoryFilter">
            </xp:parameter>
        </xp:this.parameters>
    </xp:link>
    <!-- only include a comma if multiple array items exist -->
    <xp:label value="," id="label5"
        themeId="Text.commaSeparator">
    <xp:this.rendered><![CDATA[#{javascript:
        size > 1 && tagIndex < size - 1}]]>

```

```
        </xp:this.rendered>
    </xp:label>
</xp:repeat>
```

This nested Repeat control is created on the fly, along with some other sibling controls, whenever the end-user expands a top level row using the **More** link. The Repeat control's value property does not in fact point to a view data source, as has been the norm up to now, but to a Java array that contains one or more tags, i.e. tags are the contents of the `_Categories` multivalue field. Within this nested Repeat, a Link control is created for each category found in the tag array. The link text is set to the tag text and the link value (URL) is set to the `byTag.xsp` XPage plus a `categoryFilter` parameter, which is also set to the tag text (for example, `/byTag.xsp?categoryFilter=Government`). After all the links are generated, the Repeat removes itself from the component tree (`removeRepeat="true"`), because it is no longer required. Play with the sample application and see this feature in action. You can probably think of use cases for your own applications that would be well served using dynamic nested Repeats in this way.

The Rich Get Richer

One little amendment you could make to further enhance the rich nature of the Repeat control content is to insert the actual rich text into the dynamic row when the **More** link is clicked. Right now, it is the plain text stored in the **Abstract** column of the `xpAllDocuments` view that is displayed, but if you locate that value binding in the custom control (search all `DocumentsView.xsp` for `"cfAbstract"`), you could replace it, as shown in Listing 9.11.

Listing 9.11 Server-Side JavaScript Code to Extract HTML from Rich Text Fields Saved in MIME Format

```
// search for "Abstract" and comment out this next line of code
// return rowData.getColumnValue("Abstract");
// get the Notes document and body rich text field
var nd:NotesDocument = rowData.getDocument();
var mime = nd.getMIMEEntity("body");
// if it is MIME then you can passthrough as HTML
if (mime != null) {
    return mime.getContentAsText();
}
// Otherwise just return the plain text
else {
    return nd.getItemValueString("body");
}
```

You need to configure the **cfAbstract** Computed Field to have a content type of HTML. This has been done for you in the **allDocumentsView** custom control, but the code is commented out. If you would like to see this feature in action, simply enable the code in Designer. Figure 9.30 shows some sample rich content expanded in the Repeated rows.

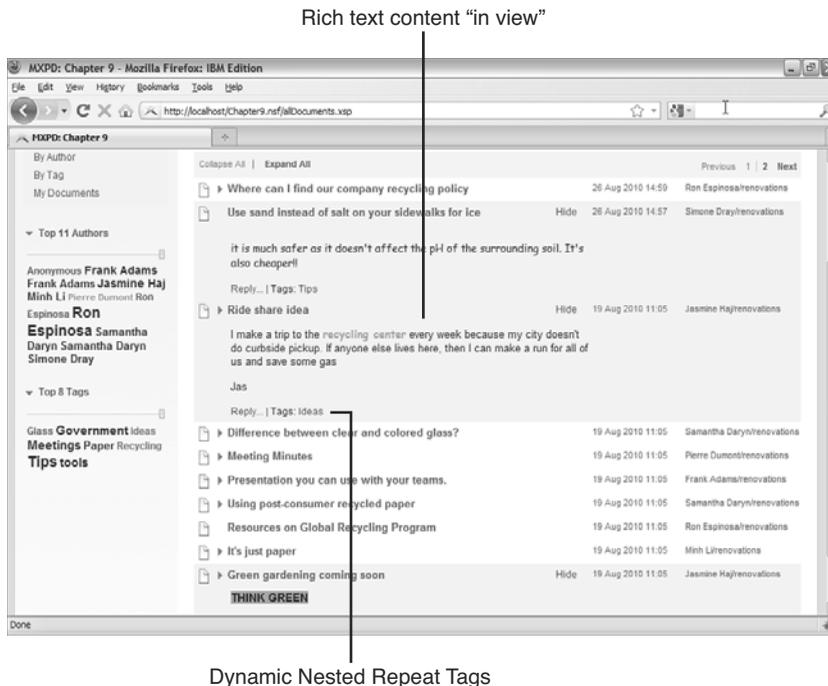


Figure 9.30 Expanded Rich Text Content in Repeat Control

Obviously, it is not efficient to open documents when building views, although this *only* occurs when the user clicks the **More** link, so the expense is only incurred on request and not for every repeated item. This example concludes our discussion of the Repeat Control.

Some Fun with the Pager

After all the hard work done in this chapter, you might as well finish on a light note. The common view pager that you have worked with in various examples is actually a highly configurable control, even though it has only been used in its default state thus far. The next exercise shows how to transform the look and feel of your pager.

You should start by revisiting the **dataTable.xsp** XPage and making a new copy of this, called **dataTableExt.xsp**. In the new XPage, activate the **Source** pane and find the facets tag for the Data

Table—careful not to accidentally pick the facets tag for one of the columns! Copy and paste the existing `<xp:pager>` tag that’s already defined in the Data Table facets and then set `xp:key="header"` and `panelPosition="right"` on one of them. After completing this task, the Data Table should have two pagers: one on the right hand side of the header and one on the left hand side of the footer. Select the header pager in the **Outline** view and activate the WYSIWYG editor and **Pager** property panel.

The first thing you can do is apply different pager styles to the header pager (for example, Sample 1 through Sample 7), and preview or reload the XPage to see what features are exposed in the different canned styles. What’s more interesting, however, is to play around with a custom layout. For this example, select the footer pager in the **Outline** view and change the **Pager style** combo box style to Custom. This causes a new list of controls to be displayed in the Property panel—select the ones shown in Figure 9.31.

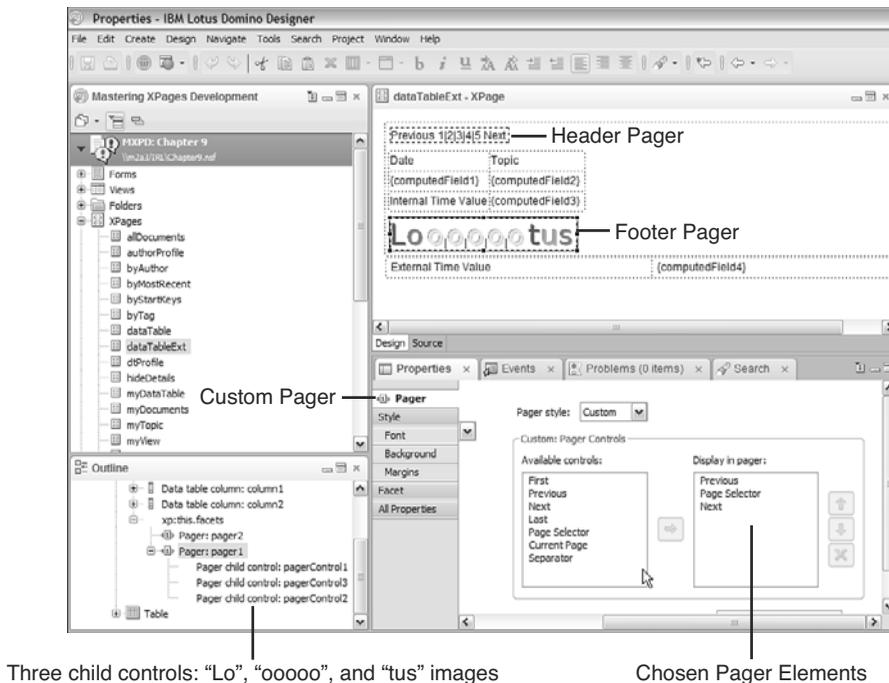


Figure 9.31 Working with a custom pager in Designer

In the **Outline** view, select each of the newly created three child controls in turn and assign images to them. The **Previous** control should be assigned “/Lo.gif”, the **Group** control (Page Selector) should be assigned “/ooooo.gif”, and the **Next** control should be assigned “/tus.gif”. These image resources have been already added to **Chapter9.nsf** for your convenience. In fact, a

dataTableExt.xsp XPage is also included if you do not feel like building this example—it's been a long chapter! The updated markup for the Data Table facets tag should now look like Listing 9.12.

Listing 9.12 Custom Pager Definitions

```
<xp:this.facets>
  <xp:pager id="pager2" for="dataTable1" xp:key="header"
    panelPosition="right" partialRefresh="true">
  </xp:pager>
  <xp:pager xp:key="footer" id="pager1" for="dataTable1"
    partialRefresh="true" disableTheme="true">
    <xp:pagerControl id="pagerControl1" type="Previous"
      image="/Lo.gif">
    </xp:pagerControl>
    <xp:pagerControl id="pagerControl3" type="Group"
      image="/ooooo.gif">
    </xp:pagerControl>
    <xp:pagerControl id="pagerControl2" type="Next"
      image="/tus.gif">
    </xp:pagerControl>
  </xp:pager>
</xp:this.facets>
```

With this markup in place, preview the page. In Figure 9.32, observe that navigating on the footer pager updates the header pager state—as you would expect! So, even though the header and footer pagers no longer bear any visual resemblance to each other, their behaviors are identical.

TIP A new pager property was introduced in 8.5.2 called `alwaysCalculateLast`. Calculating the entry count in large categorized and/or hierarchical views can be expensive because the code has to navigate each view path to figure out the total count. Thus, the **Last** pager control was not always enabled in the Pager due to the cost associated with the calculation. If having a **Last** pager option is more important to you than any performance hits incurred as a result of calculating it, you should set `alwaysCalculateLast="true"` on the Pager control; you can find this property in the **basics** category of the **All Properties** sheet. This means that you always can jump to the end of the view no matter what!

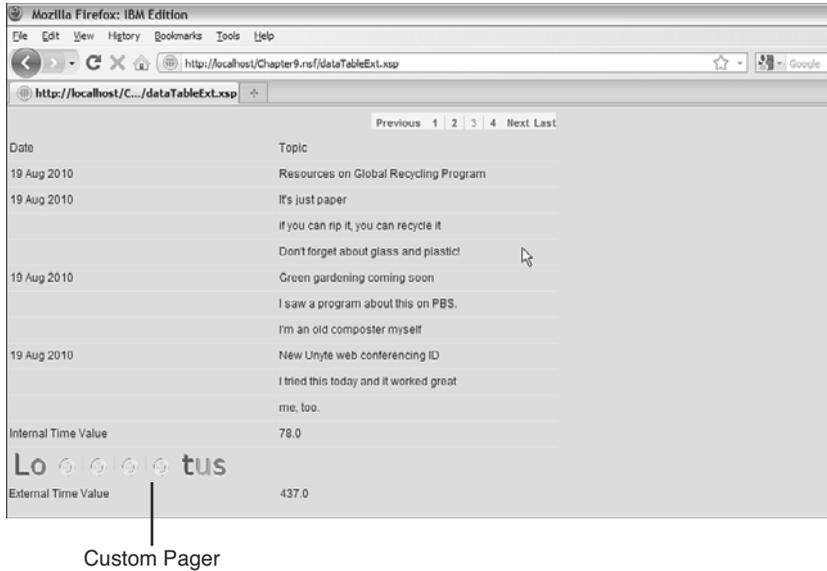


Figure 9.32 Custom Looooootus Pager

Conclusion

This chapter extensively covered the three view container controls: the View Panel, Data Table, and Repeat control. You learned how to apply the lesser-used control properties, when to use one control over another, and how to customize the look and behavior of all three. Hopefully, this material will help you build cool, slick, and efficient views that satisfy your own unique use cases. Go forth and view!

Index

Symbols

{ character sequence, 57

A

Access Control command
(Application menu), 19

access control lists (ACLs), 19
access levels, 675-676
implementing, 689-690

access control. *See* security, 675

Access Key Validator, 469

access levels (ACL), 675-676

ACF (Active Content Filtering),
699-702

acl property, 680

ACLs (access control lists), 19
access levels, 675-676
implementing, 689-690

action group simple action,
184-186

action property (Domino
document data source), 218

action property (xp:eventHandler
tag), 166

actions

Cancel, 38

client-side actions, refreshing
with, 162-163

Delete Selected Documents,
40

document actions, 235-236
executing multiple, 184-186

server-side actions, refreshing
with, 160-161

simple actions, 118-125, 167
action group, 184-186

change document mode,
168-169

confirm, 169-170

create response document,
170-171

delete document, 171

delete selected documents,
172

execute client script, 173

execute script, 173-174

modify field, 174-175

open page, 175-176

publish component
property, 176-177

publish view column,
177-178

save data sources, 179-180

save document, 180-182

set component mode,
182-183

set value, 183-184

Submit, 37

Active Content Filtering (ACF),
699-702

Add Bookmarks dialog, 501

Add Simple Action dialog, 39-41

addon link event, 410

agents, 405-412

Aggregate Container pattern,
357-358

aggregating XPages Discussion
component and Notes Google
widget, 533-536

AJAX, partial refresh. *See* Partial
Refresh option, 369-376

alert() method, 211, 509

allDocuments XPage, 418-419

allDocumentsView control, 418

allowDeletedDocs property
(Domino document data
source), 218, 234

Anonymous users, 690

APIs (application programming
interfaces), JSF API, 137-138

Append Column command
(View menu), 285

Application command (New
menu), 15

application development
and performance, 654-655
creating applications,
5, 24-26

- CRUD operations,
 - supporting, 36-42
 - explained, 23-24
 - forms, 26-31
 - views, 26-31
 - creating, 31-36
 - XSP markup, 33-34
 - application frameworks, 367-368
 - application layer (security), 675-677
 - Application Level themes, 569-570
 - Application menu commands,
 - Access Control, 19
 - application performance
 - and application development, 654-655
 - reducing CPU utilization, 658
 - GET- versus POST-based requests, 658-659
 - immediate property, 661-663
 - partial execution mode, 665-668
 - partial refresh, 663-664
 - readonly property, 660-661
 - reducing memory utilization, 668-669
 - dataCache property, 670-672
 - HTTPJVMMaxHeapSize parameter, 669
 - HTTPJVMMaxHeapSize Set parameter, 669
 - xsp.persistance.* properties, 669-670
 - request processing lifecycle, 655-656
 - GET-based HTTP requests, 656-659
 - POST-based HTTP requests, 656-659
 - Application Properties editor,
 - configuring themes with, 580-583
 - applications
 - application frameworks, 367-368
 - application layer of security, 675-677
 - composite applications, 528
 - aggregating XPages
 - Discussion component and Notes Google widget, 533-536
 - creating components, 529-531
 - listening components, 531-532
 - online video about, 540
 - receiving and publishing events, 536-539
 - creating, 687-688
 - JSF-based applications. *See* JSF (JavaServer Faces)
 - Notes Discussion
 - banner area, 507-508
 - bookmarks, 501-503
 - client versus web, 508-511
 - disableModified flag, 513-516
 - enableModified flag, 513-516
 - launching, 498-500
 - Save dialog for dirty documents, 511-513
 - tab management, 516-519
 - working offline, 503-506
 - applicationScope variable, 138, 193-196
 - Apply Request View phase (JSF request processing lifecycle), 134
 - architecture of themes, 569
 - inheritance levels, 585-587
 - Platform Level versus
 - Application Level themes, 569-570
 - theme configurations
 - supported by XPages, 570-576
 - Array class, 201
 - authentication, 675
 - AUTHOR access level, 676
 - Authors field, 685
- B**
- Background tab (Style properties panel), 545-546
 - backing beans, 483-486
 - banner area (Discussion application), 507-508
 - base.xsp-config file, creating, 446-449
 - basic authentication, 675
 - beans. *See* backing beans, 483-486
 - behavioral interfaces, 143-145
 - bidirectional resources, 605-606
 - binding data, 306
 - binding expressions, 136, 152-153
 - body tag, 49
 - Bookmark command (Create menu), 501
 - bookmarks, 501-503
 - Boolean Check Box, 469
 - Boolean class, 201
 - Boolean Value, 469
 - browsers, previewing XPage
 - design elements in, 18-21
 - bundle resource element, 591-592
 - business logic
 - JavaScript. *See* JavaScript overview, 157-160
 - simple actions, 167
 - action group, 184-186
 - change document mode, 168-169
 - confirm, 169-170
 - create response document, 170-171
 - delete document, 171
 - delete selected documents, 172
 - execute client script, 173
 - execute script, 173-174
 - modify field, 174-175

- open page, 175-176
 - publish component property, 176-177
 - publish view column, 177-178
 - save data sources, 179-180
 - save document, 180-182
 - set component mode, 182-183
 - set value, 183-184
 - xp:eventHandler tag
 - example to display current date/time, 160
 - properties, 164-167
 - refreshing with client-side JavaScript, 164
 - refreshing with client-side simple action, 162-163
 - refreshing with server-side JavaScript, 161
 - refreshing with server-side simple action, 160-161
 - buttonNewTopic control, 662
 - buttons
 - Cancel, 38
 - Submit, 37
 - xp:button tag, 71-72
 - XSP markup, 38
 - buttonSave button, 666
- C**
- caching view data, 265-269
 - CAE (Composite Application Editor), 533-536
 - category tag, 468
 - Cancel buttons, 38
 - captionStyleClass property, 566
 - categorized columns, 293-300
 - category tags, 443
 - categoryFilter property (xp:dominoView tag), 246-249
 - ccTagCloud control, 561, 660-661
 - CDATA (character data), 55
 - Change Document Mode action, 236
 - change document mode simple action, 168-169
 - changing
 - document mode, 168-169
 - pass-through text, 191
 - character data (CDATA), 55
 - Character Set Type Picker, 469
 - checkbox groups, 81
 - checkboxes, 79
 - checking for Public Access, 703
 - CKEditor, 238-242
 - classes
 - classes available in XPages, 7
 - Notes/Domino Java API classes, online resources, 714
 - online documentation, 492
 - style classes
 - advantages of, 553-554
 - computed values, 561-562
 - defined, 552
 - extended styleClass properties, 563-566
 - stylingWithClasses XPage, 554-558
 - use by browser or client, 559-561
 - XSP Java classes, online resources, 712-714
 - XSP JavaScript pseudo classes, online resources, 715-716
 - classloader bridge (NSF), 695
 - Clean dialog, 624
 - Clear Private Data button, 525
 - client fix packs, 11
 - client IDs, 206-208
 - client scripts, executing, 173
 - Client Side Event Editor, 469
 - Client Side Script Editor, 470
 - client-side actions, refreshing with, 162-163
 - client-side JavaScript
 - adding client and server logic to same event, 209-210
 - control IDs versus client IDs, 206-208
 - including server data in client JavaScript, 208-209
 - localization, 639
 - XSP client JavaScript library, 210-211
 - client-side script libraries, localization, 641-643
 - client-side scripting, 125-127
 - clients
 - client fix packs, installing, 11
 - client user experience, 8
 - configuring, 11-12
 - XPiNC (XPages in the Notes client), feature scope, 7
 - ClientSideValidator, 146
 - columnClasses property, 566
 - columns (View Panel)
 - categorized columns, 293-300
 - custom pager, 321-323
 - decorating with images, 284-287
 - displaying column data, 277-279
 - displaying document hierarchy, 281
 - emulating Notes client rendering, 296-300
 - publishing, 177-178
 - reordering, 279-280
 - sorting, 270, 287, 290-292
 - View Title components, 288-292
 - Combo Box, 470
 - combo boxes, 76-79
 - command controls
 - xp:button tag, 71-72
 - xp:eventHandler tag, 70-71
 - xp:link tag, 72-73
 - complex properties (XSP), 54
 - complex types, 439
 - complex types, specifying, 453-463
 - complex values (XSP), 54-55
 - component mode, setting, 182-183
 - component tag, 433

- component tree, scripting, 187-192
- component-class tag, 433
- component-extension tag, 433
- component-family tag, 433
- component-type tag, 433
- ComponentBindingObject, 462
- components. *See also* specific components
 - creating for composite applications, 529-531
 - extensions. *See* UI component extensions, creating
 - JSF standard user-interface components, 148-151
 - JSF user interface component model, 136, 143
 - listening components, 531-532
- Composite Application Editor (CAE), 533-536
- composite applications, 528
 - aggregating XPages Discussion component and Notes Google widget, 533-536
 - creating components, 529-531
 - listening components, 531-532
- compositeData object, 346-352
- compound documents, 49
- computed expressions, localization, 636-639
- computed fields, 83-84, 308
- computed properties (XSP), 55-59
- computed values
 - control property values, 616
 - style property, 552
 - styleClass property, 561-562
- computeDocument property (Domino document data source), 218
- computeWithForm property, 685
- computeWithForm property (Domino document data source), 218, 225
- concurrencyMode property (Domino document data source), 218, 227
- concurrent document updates, managing, 227
- configuration, variable resolvers, 140
- configuring
 - clients, 11-12
 - event parameters, 384-386
 - localization options, 624-626
 - Public Access, 703
 - themes
 - theme configurations supported by XPages, 570-576
 - with Application Properties editor, 580-583
- confirm simple action, 169-170
- confirm() method, 211, 509
- confirming actions, 169-170
- ConstraintValidator, 147
- containers
 - Aggregate Container pattern, 357-358
 - Layout Container pattern, 358-365
 - tags. *See* individual tag name
- content modifiers (view)
 - expandLevel property, 257-259
 - startKeys property, 256-257
- Content Type Picker, 470
- content-type element, 589
- context global object, 196
- context variable, 155
- control declaration snippets, 190
- control definitions, 613-614
- control element
 - control definitions, 613-614
 - control properties. *See* controls, properties
- control IDs, 206-208, 633-634
- Control Picker, 470
- controls. *See also* specific controls
 - adding to XPages, 21-22
 - Core Controls, setting properties on, 616, 619
 - custom control properties, 635
 - Custom Controls. *See* Custom Controls, 327
 - data binding, 59-60
 - explained, 64-65
 - properties
 - computing control property values, 616
 - control property types, 619-621
 - explained, 614-616
 - setting properties on XPages Core Controls, 616, 619
 - xp:button tag, 71-72
 - xp:checkBox tag, 79
 - xp:checkBoxGroup tag, 81
 - xp:comboBox tag, 76-79
 - xp:dataTable tag, 94-95
 - xp:dataTimeHelper tag, 68-69
 - xp:eventHandler tag, 70-71
 - xp:fileDownload tag, 86-87
 - xp:fileUpload tag, 84-85
 - xp:image tag, 84
 - xp:include tag, 99
 - xp:inputRichText tag, 67
 - xp:inputText tag, 65-66
 - xp:label tag, 83
 - xp:link tag, 72-73
 - xp:listBox tag, 74-76
 - xp:panel tag, 87-90
 - xp:radio tag, 80
 - xp:radioGroup tag, 81-82
 - xp:repeat tag, 95-98
 - xp:section tag, 100
 - xp:tabbedPanel tag, 99-100
 - xp:table tag, 90-91
 - xp:text tag, 83-84
 - xp:view tag, 91-93
- Controls Palette, 17
- Converter interface, 145

- converters, 107-109, 145-146
 - Cookie variable, 138
 - Core Controls, setting properties on, 616
 - CPU utilization, reducing, 658
 - GET- versus POST-based requests, 658-659
 - immediate property, 661-663
 - partial execution mode, 665-668
 - partial refresh, 663-664
 - readonly property, 660-661
 - Create Control dialog, 433-434
 - Create menu commands,
 - Bookmark, 501
 - Create New Custom Control dialog, 329
 - Create Response Document
 - action, 236
 - Create Response Document dialog, 221
 - create response document simple
 - action, 170-171
 - createViewNavFromCategory()
 - method, 248
 - createViewNavFromDescendants()
 - method, 252
 - CRUD operations, supporting, 36-42
 - CSS (Cascading Style Sheets)
 - files, table of, 719-720
 - inline styling, 545
 - online resources, 545
 - styles
 - classes. *See* styles (CSS), style classes
 - computed values, 552
 - extended style properties, 563-566
 - setting manually, 550-551
 - setting with Style properties panel, 545-547
 - Styling XPage, 548-550
 - use by browser or client, 551-552
 - current date/time
 - displaying, 160
 - refreshing
 - with client-side
 - JavaScript, 164
 - with client-side simple
 - action, 162-163
 - with server-side
 - JavaScript, 161
 - with server-side simple
 - action, 160-161
 - Custom Controls, 5, 635
 - compositeData object, 346-352
 - creating, 329-337
 - design patterns
 - Aggregate Container pattern, 357-358
 - Layout Container pattern, 358-365
 - explained, 327-329
 - Property Definitions
 - explained, 337-340
 - Property tab, 340-343
 - summary, 346
 - Validation tab, 343-345
 - Visible tab, 345
 - replyButton control, 352
 - multiple instances and property groups, 355-357
 - onClick event, 353-355
 - custom Dojo widgets,
 - integrating, 393-398
 - custom pager, 321-323
 - custom responses, generating
 - with XPages, 399-401
- D**
- data binding, 59-60, 306
 - data contexts, 63
 - Data Source Picker, 470
 - data source tags (XSP)
 - xp:dataContext, 63
 - xp:dominoDocument, 61-62
 - xp:dominoView, 62-63
 - data sources
 - connecting Data Tables to, 305-307
 - Domino document data
 - sources, 216
 - basic data source
 - markup, 217
 - events, 231-233
 - multiple data sources, 228-230
 - properties, 217, 233-234
 - filters, 246
 - categoryFilter property, 246-249
 - ignoreRequestParams
 - property, 252
 - keys property, 253-256
 - keysExactMatch property, 255-256
 - parentId property, 251-252
 - search property, 249-251
 - searchMaxDocs
 - property, 251
 - saving, 179-180
 - data table control, 94-95
 - Data Tables, 305
 - building embedded profile
 - view with, 311-315
 - Computed Fields, 308
 - connecting to data source, 305-307
 - Pager property panel, 308
 - sample XSP markup, 309-311
 - Data tool, 17
 - database global object, 196
 - database variable, 155
 - databaseName property (Domino document data source), 218, 234
 - databaseName property
 - (xp:dominoView tag), 245-246
 - dataCache property
 - (xp:dominoView tag), 265-269, 670-672
 - dataTableStyle property, 566
 - dataTableStyleClass property, 566
 - Date class, 201

- date/time, displaying, 160
 - with client-side JavaScript, 164
 - with client-side simple action, 162-163
 - with server-side JavaScript, 161
 - with server-side simple action, 160-161
- date/time picker control, 68-69
- DateTimeConverter, 146
- DateTimeRangeValidator, 147
- debugging XPages in Notes client, 525-528
- decode() method, 482
- default variables
 - JSF (JavaServer Faces), 138-139
 - XPages, 154-156
- default-prefix tag, 433
- Delete Document action, 236
- delete document simple action, 171
- Delete Selected Documents action, 40
- delete selected documents simple action, 172
- Delete Selected Documents action, 236
- deleting documents, 171-172
- DEPOSITOR access level, 676
- deprecated locale codes, 648-650
- description tag, 433, 443
- design element layer (security), 677
 - form access control options, 678-679
 - view access control options, 679-680
 - XPage access control, 680-684
- Design menu commands, 18-20
- design patterns
 - Aggregate Container pattern, 357-358
 - Layout Container pattern, 358-365
 - Repeat control design pattern, 317-318
- DESIGNER access level, 675
- designer-extension tag, 443
- designer-extension tags, 468-469
- development
 - and performance, 654-655
 - of XPages, xvii, 4-7
- dialogs. *See* specific dialogs
- directories
 - Dojo directory, 599-600
 - HTML directory, 597-598
 - XPages Global directory, 598-599
- DirectoryUser class, 201-203
- dirty documents, saving in Notes client, 511-513
- disableClientSideValidation property, 117
- disableModified flag, 513-516
- Discussion application. *See* Notes Discussion application, 498
- display controls
 - xp:fileDownload tag, 86-87
 - xp:fileUpload tag, 84-85
 - xp:image tag, 84
 - xp:label tag, 83
 - xp:text tag, 83-84
- Display XPage instead property, 283
- display-name tag, 433, 443
- @DocDescendants function, 278
- document collection for folders/views, retrieving, 262-264
- document hierarchy, displaying, 281
- document layer (security), 684-686
- document mode, changing, 168-169
- document signing, 686
- Document Type Definitions (DTDs), 48
- documentation, XPages classes, 492
- documentId property (Domino document data source), 218
- documents, 215
 - actions, 235-236
 - controlling URL parameter usage, 220
 - creating, 219-220
 - data sources, 216
 - basic data source markup, 217
 - events, 231-233
 - multiple data sources, 228-230
 - properties, 217, 233-234
 - deleting, 171-172
 - document hierarchy, displaying, 281
 - document mode, changing, 168-169
 - editing, 219-220
 - executing form logic, 224-227
 - in-memory documents, 405-412
 - JavaScript, 236-238
 - linking View Panel to, 281-284
 - managing concurrent document updates, 227
 - profile documents, 197-198, 405-412
 - response documents, 170-171, 220-224
 - rich text, 238-242
 - saving, 180-182
- Dojo directory, 599-600
- Dojo integration, 386-387
 - dojoAttributes property, 389
 - dojoModule resource, 388-389
 - dojoParseOnLoad property, 387-388
 - dojoTheme property, 387-388
 - dojoType property, 389
 - extending Dojo class path, 390-391

- integrating Dojo widgets, 390-391
 - custom Dojo widgets, 393-398
 - generating custom responses with XPages, 399-401
 - standard Dojo widgets, 391-393
 - Dojo modules, 105
 - Dojo Toolkit, 648
 - dojo.require() statement, 388
 - dojoAttributes property, 389
 - dojoModule resource, 388-389
 - dojoModule resource element, 592
 - dojoParseOnLoad property, 387-388
 - dojoTheme property, 387-388, 600
 - dojoType property, 389
 - DOM library, 205-206
 - Domino, 5
 - documents. *See* documents
 - Domino links versus Notes links, 520-524
 - history and development, xiii-xvi
 - views. *See* views
 - Domino Designer, 5
 - adding Package Explorer to, 424-426
 - applications
 - creating, 15, 24-26
 - CRUD operations, supporting, 36-42
 - forms, 26-31
 - client configuration, 11-12
 - client fix packs, installing, 11
 - Controls Palette, 17
 - Data tool, 17
 - documents, 61-62
 - downloading, 9-10
 - installing, 10-11
 - library, 197-198
 - Outline tool, 17
 - perspective, 14-15
 - property sheets, 17
 - views, 26-31, 62-63
 - creating, 31-36
 - XSP markup, 33-34
 - Welcome screen, 13-14
 - XPage design elements
 - adding controls to, 21-22
 - creating, 16-18
 - previewing, 18-21
 - tool, 16
 - XPages Editor, 16
 - DoubleRangeValidatorEx2, 147
 - DoubleValue, 470
 - downloading
 - Domino Designer, 9-10
 - files, xp:fileDownload tag, 86-87
 - DTDs (Document Type Definitions), 48
 - _dump() method, 526
- E**
- ECLs (Execution Control Lists), 697-699
 - edit box control, 65-66
 - editing documents, 219-220
 - editing controls, 64-65
 - xp:dataTimeHelper tag, 68-69
 - xp:inputRichText tag, 67
 - xp:inputText tag, 65-66
 - EDITOR access level, 676
 - editor tag, 469-472
 - EL (Expression Language), 136
 - elements. *See* specific elements
 - ELResolver class, 141
 - embedded profile view, building with Data Tables, 311-315
 - embedding Java in JavaScript, 190
 - empty theme, 583-585
 - enableModified flag, 513-516
 - encodeBegin() method, 482
 - encodeEnd() method, 482
 - encryption, 686
 - endsWith() method, 211
 - error() method, 211
 - escape property, 57
 - event handlers, 70-71, 164-167
 - event property (xp:eventHandler tag), 164
 - eventParametersTable control, 386
 - events
 - adding client and server logic to same event, 209-210
 - document data source events, 231-233
 - event parameters, 384-386
 - receiving and publishing, 536-539
 - exceptions, NoAccessSignal, 703
 - execId property (xp:eventHandler tag), 142, 164
 - execMode property (xp:eventHandler tag), 142, 164
 - execute script simple action, 173-174
 - executing client scripts, 173
 - Execution Control Lists (ECLs), 697-699
 - expandLevel property (xp:dominoView tag), 257-259
 - exporting resource bundle files, 628-629
 - Expression Language (EL), 136
 - expressions
 - computed expressions, localization, 636-639
 - formula language expressions, 404
 - ExpressionValidator, 147
 - extended style properties, 563-566
 - extending Dojo class path, 390-391
 - extensibility. *See* UI component extensions, creating
 - Extensible Hypertext Markup Language (XHTML), 48-50

F

- faces-config tag, 432
- faces-config-extension tag, 432
- faces-config.xml file, 139, 413
- FacesAjaxComponent, 143
- FacesAutoForm, 143
- FacesComponent, 143
- facesContext variable, 138
- FacesDataIterator, 144
- FacesDataProvider, 144
- FacesDojoComponent, 145
- FacesDojoComponentDelegate, 145
- FacesInputComponent, 144
- FacesInputFiltering, 144
- FacesNestedDataTable, 144
- FacesOutputFiltering, 144
- FacesPageIncluder, 144
- FacesPageProvider, 144
- FacesParentReliantComponent, 144
- FacesPropertyProvider, 145
- FacesRefreshableComponent, 145
- FacesRequiredValidator, 147
- FacesRowIndex, 145
- FacesSaveBehavior, 145
- FacesThemeHandler, 145
- facets, 92
- Favorite Bookmarks command (Open menu), 502
- field encryption, 686
- fields
 - Computed Fields, adding, 308
 - modifying, 174-175
- file download control, 86-87
- File menu commands,
 - Replication, 504-506
- file upload control, 84-85
- files. *See also* specific files
 - resource bundle files
 - adding, 637-638
 - adding strings, 632
 - changing strings, 631-632
 - exporting, 628-629
 - importing, 630
 - localization within, 623
 - removing strings, 632-633
- XSP CSS (Cascading Style Sheets) files
 - style class reference, 720-726
 - table of, 719-720
- filters
 - view content modifiers
 - expandLevel property, 257-259
 - startKeys property, 256-257
 - view data source filters
 - categoryFilter property, 246-249
 - ignoreRequestParams property, 252
 - keys property, 253-256
 - keysExactMatch property, 255-256
 - parentId property, 251-252
 - search property, 249-251
 - searchMaxDocs property, 251
- findForm() method, 211
- findParentByTag() method, 211
- fix packs, 11
- folders
 - compared to views, 261
 - Java source code folders, 426-427
 - retrieving document collection for, 262-264
- Font tab (Style properties panel), 545
- form logic, executing, 224-227
- formName property (Domino document data source), 218
- forms, 26-31
 - access control options, 678-679
 - executing form logic, 224-227
- formula language, 404-405
- formula language expressions, 404
- functions. *See* specific functions
 - @Functions, 402-405
 - @Functions library, 205

G

- generating custom responses with XPages, 399-401
- Generic File Picker, 470
- generic head resources, 106
- GET-based HTTP requests, 656-659
- getAttributes() method, 439
- getBrowser() method, 601
- getBrowserVersion() method, 601
- getBrowserVersionNumber() method, 602
- getClientId() method, 192, 197
- getColumnIndentLevel() method, 298
- getColumnValue() method, 298, 306, 311-312
- getColumnValues() method, 278
- getComponent() method, 196, 353-355
- getComponentAsString() method, 189
- getComponentsAsString() method, 189
- getDatabasePath() function, 395
- getDocument() method, 236
- getElementById() method, 211
- getFacetsAndChildren() method, 192
- getFamily() method, 428
- getForm() method, 196
- getLabelFor() method, 196
- getMilliseconds() method, 309
- getParameterDocID() method, 405
- getSubmittedValue() function, 539
- getUserAgent() method, 602
- getVersion() method, 602
- getVersionNumber() method, 602
- getView() method, 196

- getViewAsString() method, 189
 - getViewEntryData agent, 302-303
 - global objects (JavaScript), 193
 - @Functions library, 205
 - context global object, 196
 - database global object, 196
 - DOM library, 205-206
 - Domino library, 197-198
 - global object maps, 193-196
 - runtime script library, 198-200
 - session global object, 196
 - standard library, 200-201
 - view global object, 196-197
 - XSP script library, 201-204
 - group tag, 443
 - group-type tag, 443
 - group-type-ref tag, 443
- H**
- handlers property
 - (xp:eventHandler tag), 165
 - hasEntry() method, 602
 - head tag, 49
 - Header variable, 138
 - headerValues variable, 138
 - Hello World XPage, 187
 - help
 - Notes/Domino Java API classes, 714
 - XPages websites, 727-728
 - XSP Java classes, 712-714
 - XSP JavaScript pseudo classes, 715-716
 - XSP tag reference, 711-712
 - hiding sections, paragraphs, and layout regions, 685-686
 - history
 - of Eclipse, 12-13
 - of XPages, xvii, 4-5
 - href element, 589
 - HTML (Hypertext Markup Language), 47-48
 - directory, 597-598
 - tags (XSP), 127-128
 - html tag, 49
 - HTML htmlFilter property, 699-700
 - htmlFilterIn property, 699
 - HTTP
 - GET-based HTTP requests, 656-659
 - POST-based HTTP requests, 656-659
 - sample HTTP servlet, 132-133
 - HTTPJVMMaxHeapSize parameter, 669
 - HTTPJVMMaxHeapSizeSet parameter, 669
 - Hypertext Markup Language (HTML), 47-48
- I**
- I18n class, 199
 - IBM developerWorks, 492
 - IDs
 - control IDs, 206-208, 633-634
 - themeID, 611-613
 - ignoreRequestParams property (Domino document data source), 218-220
 - ignoreRequestParams property (xp:dominoView tag), 252
 - Image File Picker, 470
 - images
 - adding to columns, 284-287
 - xp:image tag, 84
 - immediate property, 166, 661-663, 667-668
 - implementing ACLs (access control lists), access levels, 689-690
 - importing resource bundle files, 630
 - in-memory documents, 405-412
 - in-palette tag, 468
 - include page control, 99
 - Indent Responses control, 281
 - infoboxes, 27
 - inheritance, themes, 569
 - inheritance levels, 585-587
 - Platform Level versus Application Level themes, 569-570
 - theme configurations supported by XPages, 570-576
 - inheriting xsp-config properties, 441-446
 - initParam variable, 138
 - inline styling, 545
 - Insert Column command (View menu), 284
 - installing
 - client fix packs, 11
 - Domino Designer, 10-11
 - Integer Value, 470
 - interfaces. *See* specific interfaces
 - international enablement, built-in functionality, 643-644
 - internationalization, 621
 - international enablement, built-in functionality, 643-644
 - localization
 - computed expressions, 636-639
 - control IDs, 633-634
 - custom control properties, 635
 - deprecated locale codes, 648-650
 - explained, 622
 - JavaScript, 636-639
 - locales in XPages, 644-647
 - merging XPage changes, 631-633
 - need for, 621
 - script libraries, 640-643
 - setting localization options, 624-626
 - testing localized applications, 627-628
 - within resource bundle files, 623
 - working with translators, 628-630

- Invoke Application phase (JSF request processing lifecycle), 135
 - isCategory() method, 298
 - isChrome() method, 602
 - isDirectionLTR() method, 605
 - isDirectionRTL() method, 605
 - isExpanded() method, 298
 - isFireFox() method, 602
 - isIE() method, 602
 - isOpera() method, 603
 - isRunningContext() method, 604
 - isSafari() method, 603
- J**
- jAgent agent, 408-409
 - Java
 - embedding in JavaScript, 190
 - getViewEntryData agent, 302-303
 - Notes/Domino Java API classes, online resources, 714
 - security exceptions, troubleshooting, 706-707
 - source code folders, 426-427
 - XSP Java classes, online resources, 712-714
 - Java Build Path editor, 414
 - Java Community Process (JCP), 4
 - Java Specifications Request (JSR), 4
 - JavaScript, 186
 - adding client and server logic to same event, 209-210
 - control IDs versus client IDs, 206-208
 - documents, 236-238
 - embedding Java in, 190
 - global objects, 193
 - @Functions library, 205
 - context global object, 196
 - database global object, 196
 - DOM library, 205-206
 - Domino library, 197-198
 - global object maps, 193-196
 - runtime script library, 198-200
 - session global object, 196
 - standard library, 200-201
 - view global object, 196-197
 - XSP script library, 201-204
 - including server data in client JavaScript, 208-209
 - localization, 636-639
 - refreshing with client-side JavaScript, 164
 - refreshing with server-side JavaScript, 161
 - scripting component tree, 187-192
 - XPages object model, 186-187
 - XSP client JavaScript library, 210-211
 - XSP JavaScript pseudo classes, online resources, 715-716
- JavaServer Faces. *See* JSF
 - JavaServer Pages. *See* JSP
 - JCP (Java Community Process), 4
 - JSF (JavaServer Faces), 3-4, 130-131
 - APIs, 137-138
 - application integration, 137
 - benefits, 129
 - binding expressions, 136, 152-153
 - integration with JSP (JavaServer Pages), 136
 - JSF default variables, 138-139
 - per-request state model, 137
 - presentation tier, 133, 141-142
 - rendering model, 137
 - request processing lifecycle, 134-135, 142
 - explained, 655-656
 - GET-based HTTP requests, 656-659
 - POST-based HTTP requests, 656-659
 - resources, 131
 - sample HTTP servlet, 132-133
 - sample JSP with JSF tags, 133
 - standard user-interface components, 148-151
 - user interface component model, 136, 143
 - variable resolvers, 139-141
 - XPages
 - behavioral interfaces, 143-145
 - converters, 145-146
 - default variables, 154-156
 - validators, 146-148
- JSP (JavaServer Pages), 5
 - integration with JSF (JavaServer Faces), 136
 - sample JSP with JSF tags, 133
 - JSR (Java Specifications Request), 4
- K-L**
- keys property (xp:dominoView tag), 253-256
 - keysExactMatch property (xp:dominoView tag), 255-256
 - keyView, 270
 - labels, 83
 - Language Direction Picker, 470
 - Language Picker, 470
 - LargeSmallStepImpl.java, 458-461
 - LargeSmallStepInterface.java, 455
 - lastSubmit property, 211
 - launching
 - Domino Designer perspective, 14
 - Notes Discussion, 498-500

Layout Container pattern, 358-365

layout regions, hiding, 685-686

LCD (Lotus Component Designer), xiv, 4

LengthValidatorEx, 147

libraries

- @Functions library, 205
- DOM library, 205-206
- Domino, 197-198
- runtime script library, 198-200
- script libraries
 - creating, 101-103
 - localization, 640-643
 - xp:script tag, 102-103
- standard library, 200-201
- ViewUtils script library, 188-189
- XPages Extension Library, 492
- XSP client JavaScript library, 210-211
- XSP script library, 201-204

linking View Panel to documents, 281-284

linkResource resource element, 594

links

- Notes links versus Domino links, 520-524
- xp:link tag, 72-73

linkSubject link control, 659

listboxes, 74-76

loaded property, 683

loaded property (Domino document data source), 218, 234

loaded property (xp:dominoView tag), 263

loaded property (xp:eventHandler tag), 165

Locale class, 199-200

locales

- deprecated locale codes, 648-650
- in XPages, 644-647

localization

- computed expressions, 636-639
- control IDs, 633-634
- custom control properties, 635
- deprecated locale codes, 648-650
- explained, 622
- JavaScript, 636-639
- locales in XPages, 644-647
- merging XPage changes, 631-633
- need for, 621
- resource bundle files, importing/exporting, 628-630
- script libraries
 - client-side script libraries, 641-643
 - server-side script libraries, 640-641
- setting localization options, 624-626
- testing localized applications, 627-628
- within resource bundle files, 623
- working with translators
 - exporting resource bundle files, 628-629
 - importing resource bundle files, 630

log() method, 211

LongRangeValidatorEx2, 147

Lotus Component Designer (LCD), xiv, 4

Lotus Expeditor (XPD), 496

Lotus Notes Template Development ID file, 689

Lotus Notes. *See* Notes

M

managed beans, 412-419

managed-bean-class tag, 413

managed-bean-name tag, 413

managed-bean-scope tag, 413

MANAGER access level, 675

managing concurrent document updates, 227

Margins tab (Style properties panel), 546

mask characters, 146

MaskConverter, 146

Math class, 201

media element, 589

memory utilization, reducing, 668-669

- dataCache property, 670-672
- HTTPJVMMaxHeapSize parameter, 669
- HTTPJVMMaxHeapSizeSet parameter, 669
- xsp.persistence.* properties, 669-670

merging XPage changes, 631-633

metaData resource element, 594-597

metadata resources, 106-107

Method Binding Editor, 471

methods. *See* specific methods

milliSecsParameter, 385-386

MIME Image Type Picker, 471

MinMaxPair interface, 450

MinMaxUIInput, 444-446

modifying fields, 174-175

Modify Field action, 236

modify field simple action, 174-175

ModulusSelfCheckValidator, 148

moreLink link, 417

multiline edit boxes, 66

Multiline Text, 471

multiple actions, executing, 184-186

multiple document data sources, 228-230

multiple views, 259-260

mxdp.data.ViewReadStore custom widget, 397

mxdp.ui.ViewTree widget, 399-400

mxdp1 theme, 613-614

mxdp2 theme, 614

N

namespace-uri tag, 433
 namespaces, XML, 49
 nanoTimeParameter, 385-386
 Native and Custom Control
 Custom Visualization Best Practices' article, 469
 navigate property
 (xp:eventHandler tag), 166
 nested Repeat controls, 318-320
 New Application dialog, 15-16, 424-425, 687-688
 New File dialog, 431
 New Java Class dialog, 429, 435, 456-457
 New Java Interface dialog, 454
 New menu commands
 Application, 15
 Theme, 577
 XPage, 16
 New NSF Component dialog, 534
 New Replica dialog, 504
 New Script Library dialog, 102
 New Source Folder dialog, 426-427
 New Style Sheet dialog, 103
 New Theme button, 577
 New Theme dialog, 578-579
 New XPage dialog, 16, 35, 216-217, 225, 433
 NO ACCESS access level, 676
 NoAccessSignal exception, 703
 Notes, history and development, xiii-xvi
 Notes client, XPages in
 composite applications, 528
 aggregating XPages
 Discussion component and Notes Google widget, 533-536
 creating components, 529-531
 listening components, 531-532
 online video about, 540
 receiving and publishing events, 536-539

debugging, 525-528
 emulating Notes client rendering, 296-300
 explained, 495-497
 Notes Discussion application
 banner area, 507-508
 bookmarks, 501-503
 client versus web, 508-511
 disableModified flag, 513-516
 enableModified flag, 513-516
 launching, 498-500
 Save dialog for dirty documents, 511-513
 tab management, 516-519
 working offline, 503-506
 Notes links versus Domino links, 520-524
 previewing design elements, 18
 security, 696
 ACF (Active Content Filtering), 699-702
 Execution Control Lists (ECLs), 697-699
 Notes Discussion application
 banner area, 507-508
 bookmarks, 501-503
 client versus web, 508-511
 disableModified flag, 513-516
 enableModified flag, 513-516
 launching, 498-500
 Save dialog for dirty documents, 511-513
 tab management, 516-519
 working offline, 503-506
 NotesViewEntry class, 715
 NotesXspDocument class, 201, 204
 NotesXspViewEntry class, 201
 NSF classloader bridge, 695
 Number class, 201
 Number Format Editor, 471
 NumberConverter, 146

O

Object class, 201
 object model (XPages), 186-187
 Object Technology International (OTI), 12
 objects, JavaScript global
 objects, 193. *See also* specific objects
 @Functions library, 205
 DOM library, 205-206
 Domino library, 197-198
 global object maps, 193-196
 runtime script library, 198-200
 standard library, 200-201
 XSP script library, 201-204
 offline, working offline, 503-506
 onComplete property
 (xp:eventHandler tag), 167
 onError property
 (xp:eventHandler tag), 167
 oneuiv2 theme, 605
 onStart property
 (xp:eventHandler tag), 167
 Open menu commands
 Favorite Bookmarks, 502
 Replication and Sync, 505
 Open Page action, 236
 open page simple action, 175-176
 opening pages, 175-176
 OpenNTF, xv, 492
 ?OpenXPage command, 503
 OTI (Object Technology International), 12
 outerStyleClass property, 563-565
 Outline tool, 17

P

Package Explorer, adding to
 Domino Designer perspective, 424-426
 Pager property panel, 308, 321-323
 pages, opening, 175-176
 panels, 87-90

- paragraphs, hiding, 685-686
- Param variable, 138
- parameters, event parameters, 384-386
- parameters property
 - (xp:eventHandler tag), 166
- paramValues variable, 138
- parentId property (Domino document data source), 218
- parentId property
 - (xp:dominoView tag), 251-252
- parseVersion() method, 603
- partial execution mode, 369, 654-668
- partial refresh, 663-664
 - online resources, 369
 - performing with Partial Refresh option, 369-376
 - scripting, 376-377
 - partialRefreshGet() function, 377-381
 - partialRefreshPost() function, 381-382
- Partial Refresh option, 369-376
- PartialRefreshField control, 373-375
- partialRefreshGet() function, 377-381
- partialRefreshGet() method, 211
- partialRefreshPost() function, 381-382
- partialRefreshPost() method, 211
- pass-through text, changing, 191
- Password Value, 471
- paths, resource paths, 597
 - bidirectional resources, 605-606
 - Dojo directory, 599-600
 - dojoTheme property, 600
 - HTML directory, 597-598
 - user agent resources, 600-605
 - XPages Global directory, 598-599
- patterns
 - Aggregate Container pattern, 357-358
 - Layout Container pattern, 358-365
 - Repeat control design pattern, 317-318
- per-request state model (JSF), 137
- performance
 - and application development, 654-655
 - reducing CPU utilization, 658
 - GET- versus POST-based requests, 658-659
 - immediate property, 661-663
 - partial execution mode, 665-668
 - partial refresh, 663-664
 - readonly property, 660-661
 - reducing memory utilization, 668-672
 - request processing lifecycle, 655-659
- perspective (Domino Designer), 14-15
- Platform Level themes, 569-570
- POST-based HTTP requests, 656-659
- postNewDocument property (Domino document data source), 218, 232-233
- postOpenDocument property (Domino document data source), 218
- postOpenView property (xp:dominoView tag), 263-264
- postSaveDocument property (Domino document data source), 218
- presentation tier, 133, 141-142
- Preview in Browser option, 168
- Preview in Notes command (Design menu), 18
- Preview in Web Browser command (Design menu), 20
- PreviewBean class, 415-417
- previewHandler XPage, 400
- previewing XPage design elements, 18-21
- print() method, 526
- print-to-console debugging
 - example, 526
- printing view column data, 302-303
- Process Validations phase (JSF request processing lifecycle), 134
- profile data, displaying with Repeat control, 316-317
- profile documents, 197-198, 405-412
- Programmability Restrictions, 691-693
- prompt() method, 211, 509
- properties. *See also* specific properties
 - custom control properties, 635
 - event handler properties, 164-167
 - Property Definitions, 337-339
 - Property tab, 340-343
 - summary, 346
 - Validation tab, 343-345
 - Visible tab, 345
 - theme properties, 607-611
 - UI component extension properties. *See* UI component extensions, creating
 - View Panel properties, 301-305
 - view properties, 301-305
- XSP
 - complex properties, 54
 - complex values, 54-55
 - computed properties, 55-59
 - data binding, 59-60
 - simple properties, 52
- Property Definitions, 337-340
 - Property tab, 340-343
 - summary, 346
 - Validation tab, 343-345
 - Visible tab, 345
- property element, 607-610

- property sheets, 17
 - Property tab (Property Definitions), 340-343
 - property tag, 443
 - property-class tag, 443
 - property-extension tag, 443
 - property-name tag, 443
 - Public Access, 702-703
 - publish component property simple action, 176-177
 - publish view column simple action, 177-178
 - publishEvent() method, 211, 510
 - publishing
 - component properties, 176-177
 - events, 536-539
 - view columns, 177-178
- Q-R**
- queryNewDocument property (Domino document data source), 218
 - queryOpenDocument property (Domino document data source), 219, 231
 - queryOpenView property (xp:dominoView tag), 263-264
 - querySaveDocument property (Domino document data source), 219
 - radio button groups, 81-82
 - radio buttons, 80
 - @Random() function, 404
 - RCP (Rich Client Platform), 12, 497
 - READER access level, 676
 - reader access lists, 685
 - Readers field, 685
 - readMarksClass property, 566
 - readonly property, 660-661, 683
 - reducing
 - CPU utilization, 658
 - GET- versus POST-based requests, 658-659
 - immediate property, 661-663
 - partial execution mode, 665-668
 - partial refresh, 663-664
 - readonly property, 660-661
 - memory utilization, 668-670
 - refresh, partial refresh, 663-664
 - refreshId property (xp:eventHandler tag), 166
 - refreshing
 - with client-side JavaScript, 164
 - with client-side simple action, 162-163
 - with server-side JavaScript, 161
 - with server-side simple action, 160-161
 - refreshMode property (xp:eventHandler tag), 166
 - RegExp class, 201
 - registering backing beans, 486
 - registerModulePath() function, 396-398
 - Regular Expression Editor, 471
 - Release Line Picker, 471
 - reloadPage() method, 644
 - removing strings, 632-633
 - Render Response phase (JSF request processing lifecycle), 135
 - render-markup tag, 468
 - rendered property (xp:eventHandler tag), 166, 683
 - renderers, 423
 - creating, 434-437
 - UISpinnerRenderer, 477-483
 - rendering model (JSF), 137
 - RenderKit-specific client script handlers, 165
 - renderkits, 137
 - reordering columns, 279-280
 - repeat control, 95-98, 274-276
 - Repeat control
 - design pattern, 317-318
 - displaying profile data with, 316-317
 - nesting, 318-320
 - rich text content in, 320-321
 - replaceItemValue() method, 405
 - Replication and Sync command (Open menu), 505
 - Replication command (File menu), 504-506
 - replyButton control, 352
 - multiple instances and property groups, 355-357
 - onClick event, 353-355
 - request processing lifecycle (JSF). *See* JSF (JavaServer Faces), request processing lifecycle
 - requestParamPrefix property (Domino document data source), 219, 229-230
 - requestParamPrefix property (xp:dominoView tag), 260
 - requests, 656-659
 - requestScope, 193-196
 - requestScope variable, 138
 - RequiredValidator, 148
 - resetting Domino Designer perspective, 14
 - resource bundle files
 - adding, 637-638
 - exporting, 628-629
 - importing, 630
 - localization within, 623
 - strings, 631-633
 - resource bundles, 104-105
 - resources
 - Dojo modules, 105
 - generic head resources, 106
 - metadata resources, 106-107
 - Notes/Domino Java API classes, 714
 - resource bundles, 104-105
 - resource paths, 597
 - bidirectional resources, 605-606
 - Dojo directory, 599-600
 - dojoTheme property, 600
 - HTML directory, 597-598

- user agent resources, 600-605
- XPages Global directory, 598-599
- script libraries
 - creating, 101-103
 - xp:script tag, 102-103
- security, 687
- style sheets
 - creating, 103-104
 - xp:styleSheet tag, 104
- theme resources. *See* themes, resources
- XSP. *See* XSP, resources
- Resources XPage, 595-596
- response documents, 170-171, 220-224
- Restore View phase (JSF request processing lifecycle), 134
- restoreState() method, 440
- restricted operation, 693-694
- Rich Client Platform (RCP), 12, 497
- rich text, 67, 238-242, 320-321
- rowClasses property, 566
- rowData expression, 306
- Run as web user option, 405
- runOnServer() method, 412
- runtime script library, 198-200
- runWithDocumentContext() method, 406-407, 410-412

S

- Save Data Sources action, 236
- save data sources simple action, 179-180
- Save dialog for dirty documents, 511-513
- Save Document action, 236
- save document simple action, 180-182
- save property (xp:eventHandler tag), 166
- save() method, 197
- saveLinksAs property (Domino document data source), 219, 234

- saveState() method, 440
- saving
 - data sources, 179-180
 - documents, 180-182
 - state between requests, 440
- scope property
 - Domino document data source, 219, 234
 - xp:dominoView tag, 263
- script libraries
 - creating, 101-103
 - localization, 640-643
 - xp:script tag, 102-103
- script property (xp:eventHandler tag), 167
- script resource element, 592-593
- scripting
 - @Functions, 402-405
 - agents, 405-412
 - client-side scripting, 125-127
 - client scripts, executing, 173
 - component tree, 187-192
 - Dojo integration. *See* Dojo integration
 - in-memory documents, 405-412
 - JavaScript. *See* JavaScript managed beans, 412-419
 - partial refresh scripting, 376
 - partialRefreshGet() function, 377-381
 - partialRefreshPost() function, 381-382
 - profile documents, 405-412
 - runtime script library, 198-200
 - scripts, executing, 173-174
 - ViewUtils script library, 188-189
 - XSP script library, 201-204
- search property (xp:dominoView tag), 249-251
- searchMaxDocs property (xp:dominoView tag), 251
- section control, 100
- sections, hiding, 685-687
- security, 673-674
 - ACF (Active Content Filtering), 699-702

- ACLs (access control lists), 675-676, 689-690
- application layer, 675-677
- design element layer, 677
 - form access control options, 678-679
 - view access control options, 679-680
- XPage access control, 680-684
- document layer, 684
 - Authors and Readers fields, 685
 - computeWithForm property, 685
 - reader access list, 685
 - sections, paragraphs, and layout regions, 685-686
- Java security exceptions, troubleshooting, 706-707
- Notes client, 696-699
- online resources, 687
- Programmability
 - Restrictions, 691-693
- Public Access, 702-703
- restricted operation, 693-694
- server layer, 674-675
- sessionAsSigner sessions, 704-705
- signatures, 689-691
- workstation ECL layer, 686-687
- XPages security checking, 695-696
- security checking, 695-696
- Select Element to Update dialog, 371
- selection controls
 - xp:checkBox tag, 79
 - xp:checkBoxGroup tag, 81
 - xp:comboBox tag, 76-79
 - xp:listBox tag, 74-76
 - xp:radio tag, 80
 - xp:radioGroup tag, 81-82
- server data, including in client JavaScript, 208-209
- server layer (security), 674-675
- Server Options, 369-371

- server-side actions, refreshing with, 160-161
- server-side JavaScript
 - global objects. *See* global objects (JavaScript)
 - scripting component tree, 187-192
 - XPages object model, 186-187
- server-side script libraries, localization, 640-641
- servers
 - Domino, xiii-xvi, 5
 - server layer of security, 674-675
- servlets, sample HTTP servlet, 132-133
- session authentication, 675
- session global object, 196
- session variable, 155
- sessionAsSigner sessions, 704-705
- sessionAsSigner variable, 155
- sessionAsSignerWithFullAccess sessions, 704
- sessionAsSignerWithFullAccess variable, 156
- sessionScope, 193-196
- sessionScope variable, 139
- set component mode simple action, 182-183
- set value simple action, 183-184
- setLocaleString() method, 644
- setRendererType() method, 429
- Shape Type Picker, 471
- Show View dialog, 424-425
- showSection() method, 211
- Sign Agents or XPages to Run on Behalf of the Invoker field, 693
- Sign Agents to Run on Behalf of Someone Else field, 692-693
- Sign or Run Unrestricted Methods and Operations field, 692
- Sign Script Libraries to Run on Behalf of Someone Else field, 693
- signatures, 689-691
- Simple Actions, 39-40, 118-125, 167
- simple actions
 - action group, 184-186
 - change document mode, 168-169
 - confirm, 169-170
 - create response document, 170-171
 - delete document, 171
 - delete selected documents, 172
 - execute client script, 173
 - execute script, 173-174
 - modify field, 174-175
 - open page, 175-176
 - publish component property, 176-177
 - publish view column, 177-178
 - refreshing with, 160-163
 - save data sources, 179-180
 - save document, 180-182
 - set component mode, 182-183
 - set value, 183-184
- simple properties (XSP), 52
- sorting columns, 270, 287, 290-292
- SpinnerBean
 - creating, 485
 - registering, 486
 - xpSpinnerTest .xsp, 486-491
- standard Dojo widgets, integrating, 391-393
- standard library, 200-201
- standard user-interface components (JSF), 148-151
- Standard Widget Toolkit (SWT), 131
- Start Configuring Widgets wizard, 531-532
- startKeys property (xp:dominoView tag), 256-257
- startsWith() method, 211
- state, saving between requests, 440
- stateful runtime environment, 367
- StateHolder, 440, 462
- stateless runtime environment, 367
- String class, 201
- String Value, 471
- strings
 - adding, 632
 - changing, 631-632
 - removing, 632-633
- Style Class Editor, 471
- style classes. *See* styles (CSS), style classes
- Style Editor, 471
- Style properties panel, 545-547
- style property
 - computed values, 552
 - extended style properties, 563-566
 - setting manually, 550-551
 - setting with Style properties panel, 545-546
 - Styling XPage, 548-550
 - use by browser or client, 551-552
- style sheets, 103-104
- styleClass attribute, 472-473
- styleClass property
 - advantages of, 553-554
 - computed values, 561-562
 - extended styleClass properties, 563-566
 - stylingWithClasses XPage, 554-558
 - use by browser or client, 559-561
- styles (CSS)
 - computed values, 552
 - extended style properties, 563-566
 - inline styling, 545
 - online resources, 545
 - setting manually, 550-551
 - setting with Style properties panel, 545-547

- style classes
 - advantages of, 553-554
 - computed values, 561-562
 - defined, 552
 - extended styleClass
 - properties, 563-566
 - stylingWithClasses
 - XPage, 554-558
 - table of, 720-726
 - use by browser or client, 559-561
 - Styling XPage, 548-550
 - use by browser or client, 551-552
 - styleSheet resource element, 593-594
 - Styling XPage, 548-550
 - stylingWithClasses XPage, 554-558
 - Submit buttons, 37
 - submit property
 - (xp:eventHandler tag), 166
 - submitLatency property, 211
 - supporting CRUD operations, 36-42
 - SWT (Standard Widget Toolkit), 131
- T**
- tab management in Notes client, 516-519
 - tabbed panel control, 99-100
 - table containers, 90-91
 - tables, Data Tables. *See* Data Tables
 - tag-name tag, 433
 - tagField input control, 403
 - tags. *See* specific tags, 51, 597
 - testing
 - localized applications, 627-630
 - UI component extensions, 437-438, 483
 - creating backing bean, 483-485
 - creating final test application, 486-491
 - look and feel, 491
 - registering backing bean, 486
 - text
 - pass-through text, changing, 191
 - rich text in Repeat controls, 320-321
 - Theme command (New menu), 577
 - ThemeControl, 145
 - themeId property, 611-613
 - themes
 - architecture and inheritance
 - inheritance levels, 585-587
 - Platform Level versus Application Level
 - themes, 569-570
 - theme configurations
 - supported by XPages, 570-576
 - benefits of, 568-569
 - control definitions, 613-614
 - control properties
 - computing control
 - property values, 616
 - control property types, 619-621
 - explained, 614-616
 - setting properties on XPages Core Controls, 616, 619
 - creating, 577-580
 - empty theme, 583-585
 - explained, 567-568
 - properties, 607-611
 - resources
 - bundle resource element, 591-592
 - dojoModule resource element, 592
 - explained, 587-591
 - linkResource resource element, 594
 - metaData resource element, 594-597
 - script resource element, 592-593
 - styleSheet resource element, 593-594
 - setting, 580-583
 - themeId, 611-613
 - Time Zone Picker, 471
 - time/date, displaying, 160
 - with client-side JavaScript, 164
 - with client-side simple action, 162-163
 - with server-side JavaScript, 161
 - with server-side simple action, 160-161
 - TimeZone class, 199-200
 - toggleExpanded() method, 298
 - translators, working with, 628-630
 - trim() method, 211
 - troubleshooting XPages Java security exceptions, 706-707
- U**
- UI component extensions, creating, 421-422
 - completing implementation, 473-477
 - extension class, 428-431
 - initial application, 424
 - Java source code folder, 426-427
 - Package Explorer, adding to Domino Designer
 - perspective, 424-426
 - process overview, 422-424
 - properties, 438, 452
 - adding to components, 439-440
 - complex types, 439
 - inheriting xsp-config properties, 441-446
 - specifying complex-type properties, 453-463

- specifying simple
 - properties, 440-441
 - StateHolder, 440
- renderer implementation, 434-437, 477-483
- test application, 483
- testing, 437-438
 - creating backing bean, 483-485
 - creating final test application, 486-491
 - look and feel, 491
 - registering backing bean, 486
- XPages Extensibility API
 - Developers Guide, 492
- XPages Extension Library, 492
- xsp-config file
 - base.xsp-config, creating, 446-449
 - completing, 464-467
 - creating, 431-432
 - designer-extension tags, 468-469
 - editor tag, 469-472
 - inheriting xsp-config properties, 441-446
 - interface, creating, 450-452
 - styleClass attribute, 472-473
 - tags, 432-434
- UICallback, 149
- UIColumnEx, 149
- UICommandButton, 149
- UICommandEx2, 149
- UIComponentBase class, 428
- UIComponentTag, 149
- UIDataColumn, 149
- UIDataEx, 149
- UIDataIterator, 149
- UIDataPanelBase, 149
- UIDateTimeHelper, 149
- UIEventHandler, 149
- UIFileuploadEx, 149
- UIFormEx, 149
- UIGraphicEx, 150
- UIInclude, 150
- UIIncludeComposite, 150
- UIInputCheckbox, 150
- UIInputEx, 150
- UIInputRadio, 150
- UIInputRichText, 150
- UIInputText, 150
- UIMessageEx, 150
- UIMessagesEx, 150
- UIOutputEx, 150
- UIOutputLink, 150
- UIOutputText, 150
- UIPager, 150
- UIPagerControl, 150
- UIPanelEx, 150
- UIPassThroughTag, 150
- UIPassThroughText, 150
- UIPlatformEvent, 150
- UIRepeat, 150
- UIRepeatContainer, 151
- UIScriptCollector, 151
- UISection, 151
- UISelectItemEx, 151
- UISelectItemsEx, 151
- UISelectListbox, 151
- UISelectManyEx, 151
- UISelectOneEx, 151
- UISpinner component extension, 423-424
 - initial application, 424
 - Java source code folder, 426-427
 - LargeSmallStepImpl.java, 458-461
 - LargeSmallStepInterface.java, 455
 - MinMaxUIInput, 444-446
- Package Explorer, adding to
 - Domino Designer perspective, 424-426
- properties, 438, 452-453
 - adding to components, 439-440
 - complex types, 439, 453-463
 - inheriting xsp-config properties, 441-446
 - simple types, 440-441
- StateHolder, 440
- test application, 483
- testing, 437-438
 - creating backing bean, 483-485
 - look and feel, 491
 - registering backing bean, 486
 - xpSpinnerTest .xsp, 486-491
- UISpinner.java, 473-477
- UISpinner extension class, 428-431
- UISpinnerRenderer, 434-437, 477-483
 - xsp-config file. *See* xsp-config file
- uispinner.xsp-config, 451-452, 464-467
- UISpinnerRenderer, 434-437, 477-483
- UITabbedPanel, 151
- UITabPanel, 151
- UITypeAhead, 151
- UIViewColumn, 151
- UIViewColumnHeader, 151
- UIViewPager, 151
- UIViewPanel, 151
- UIViewRootEx2, 151
- UITitle, 151
- unreadMarksClass property, 566
- Update Model Values phase (JSF request processing lifecycle), 135
- updates, managing concurrent document updates, 227
- uploading files, xp:fileUpload tag, 84-85
- URL parameter usage, controlling, 220
- user agent resources, 600-605
- user interface component model (JSF), 136, 143
- user-interface development, 543-545
- users
 - Anonymous, 690
 - client user experience, 8

V

validateAllFields property, 210
 Validation tab (Property Definitions), 343-345
 Validator interface, 146
 validators, 110-118, 146-148
 ValueBindingObject, 462
 ValueBindingObjectImpl, 462
 ValueHolder interface, 483
 values, setting, 183-184
 var property (Domino document data source), 219
 variable resolvers (JSF), 139-141
 variables
 JSF (JavaServer Faces)
 default variables, 138-139
 variable resolvers, 139-141
 XPages default variables, 154-156
 View Browser Configuration
 button, 526
 view control, 91-93
 view global object, 196-197
 view inspector outline, 192
 View menu commands
 Append Column, 285
 Insert Column, 284
 View Panel
 categorized columns, 293-300
 custom pagers, 321-323
 decorating columns with images, 284-287
 displaying column data, 277-279
 displaying document hierarchy, 281
 emulating Notes client rendering, 296-300
 features, 276-277
 linking to documents, 281-284
 properties, 301-305
 reordering columns, 279-280
 sorting columns, 287, 290-292
 View Title components, 288-292

View variable, 139
 Viewcontrol. *See* View Panel
 ViewReadStore custom widget, 397
 views, 26-31, 243-244
 access control options, 679-680
 caching view data, 265-269
 columns, publishing, 177-178
 compared to folders, 261
 content modifiers, 256-259
 creating, 31-36
 data source filters. *See* data sources, filters
 Data Tables. *See* Data Tables
 databaseName property, 245-246
 Domino views, 62-63
 examples, 273
 multiple views, 259-260
 properties, 301-305
 Repeat control, 274-276
 design pattern, 317-318
 displaying profile data with, 316-317
 nesting, 318-320
 rich text content in, 320-321
 retrieving document collection for, 262-264
 sorting columns, 270
 View Panel. *See* View Panel
 XSP markup, 33-34
 viewScope, 193-196
 viewScope variable, 155
 viewStyleClass property, 566
 ViewTree widget, 399-400
 ViewUtils script library, 188-189
 Visible tab (Property Definitions), 345
 Vulcan, xv

W

WAS (WebSphere Application Server), 496
 web browsers, previewing XPage design elements in, 18-21

websites, XPages resources, 727-728
 WebSphere Application Server (WAS), 496
 Welcome screen (Domino Designer), 13-14
 widgets, integrating Dojo widgets, 390
 custom Dojo widgets, 393-398
 generating custom responses with XPages, 399-401
 standard Dojo widgets, 391-393
 wizards, Start Configuring Widgets, 531-532
 working offline, 503-506
 workstation ECL layer (security), 686-687

X

XFaces, 4, 129-130
 xhrGet() function, 400
 XHTML (Extensible Hypertext Markup Language), 48-50
 XML
 comparing
 to HTML, 47-48
 to XHTML, 48-50
 compound documents, 49
 namespaces, 49
 xmlns attribute, 49
 XSP. *See* XSP
 XML User Interface Language (XUL), 496
 xmlns attribute, 49
 xp:acl tag, 680-683
 xp:aclEntry tag, 681-682
 xp:actionGroup tag, 120, 184-186
 xp:br tag, 127
 xp:button tag, 71-72
 xp:changeDocumentMode tag, 118, 168-169
 xp:checkBox tag, 79
 xp:checkBoxGroup tag, 81

- xp:comboBox tag, 76-79
- xp:confirm tag, 119, 169-170
- xp:convertDateTime tag, 107
- xp:convertList tag, 107
- xp:convertMask tag, 107
- xp:convertNumber tag, 107
- xp:createResponse tag, 119, 170-171
- xp:customConverter tag, 107
- xp:customValidator tag, 110
- xp:dataContext tag, 63
- xp:dataTable tag, 94-95
- xp:dataTimeHelper tag, 68-69
- xp:deleteDocument tag, 119, 171
- xp:deleteSelectedDocuments tag, 119, 172
- xp:dojoModule tag, 105, 388-389
- xp:dominoDocument tag, 61-62, 216
- xp:dominoDocument tag. *See also* documents, 216
- xp:dominoView tag. *See* views
- xp:eventHandler tag, 70-71
 - example to display current date/time, 160
 - properties, 164-167
 - refreshing, 160-164
- xp:executeClientScript tag, 119, 163, 173
- xp:executeScript tag, 119, 173-174
- xp:fileDownload tag, 86-87
- xp:fileUpload tag, 84-85
- xp:handler tag, 126
- xp:image tag, 84
- xp:include tag, 99
- xp:inputRichText tag, 67
- xp:inputText tag, 65-66
- xp:label tag, 83
- xp:link tag, 72-73
- xp:listBox tag, 74-76
- xp:metaData tag, 106
- xp:modifyField tag, 119, 174-175
- xp:openPage tag, 119, 175-176
- xp:pager tag, 308-311, 321-323
- xp:panel tag, 87-90
- xp:paragraph tag, 127
- xp:publishValue tag, 119, 176-177
- xp:publishViewColumn tag, 119, 177-178
- xp:radio tag, 80
- xp:radioGroup tag, 81-82
- xp:repeat tag, 95-98
- xp:save tag, 120, 179-180
- xp:saveDocument tag, 120, 180-182
- xp:script tag, 102-103
- xp:scriptBlock tag, 125
- xp:section tag, 100
- xp:setComponentMode tag, 120, 182-183
- xp:setValue tag, 120, 183-184
- xp:span tag, 127
- xp:styleSheet tag, 104
- xp:tabbedPanel tag, 99-100
- xp:table tag, 90-91
- xp:text tag, 83-84
- xp:this.facets tag, 308
- xp:validateConstraint tag, 110
- xp:validateDateTimeRange tag, 110
- xp:validateDoubleRange tag, 110
- xp:validateExpression tag, 110
- xp:validateLength tag, 110
- xp:validateLongRange tag, 110
- xp:validateModulusSelfCheck tag, 110
- xp:validateRequired tag, 110
- xp:view tag, 51, 91-93
- xp:viewPanel tag. *See* View Panel, 284
- XPage command (New menu), 16
- XPages
 - access control, 680-684
 - design elements, 46-47
 - adding controls to, 21-22
 - creating, 16-18
 - previewing, 18-21
 - XML. *See* XML, 47-50
 - XSP. *See* XSP
- extensibility. *See* UI
 - component extensions, creating
 - history and development, xiv-xv
 - locales in, 644-647
 - in Notes client. *See* Notes client, XPages in object model, 186-187
 - security checking, 695-696
 - website resources, 727-728
- XPages application development. *See* application development
- XPages Design Elements tool, 16
- XPages development paradigm, 5-7
- XPages Editor, 16
- XPages Extensibility API Developers Guide, 492
- XPages Extension Library, 492
- XPages Global directory, 598-599
- XPages in the Notes client (XPiNC), 7
- XPages Resource Servlet, accessing resource paths with
 - bidirectional resources, 605-606
 - Dojo directory, 599-600
 - dojoTheme property, 600
 - HTML directory, 597-598
 - user agent resources, 600-605
- XPages Global directory, 598-599
- XPD (Lotus Expeditor), 496
- XPiNC (XPages in the Notes client), feature scope, 7
- xpQuickTest, 438
- xpSpinnerTest .xsp, 486-491
- XSP
 - CDATA (character data), 55
 - client-side scripting, 125-127
 - command control tags, 71-73
 - complex properties, 54
 - complex values, 54-55
 - computed properties, 55-59

- container tags
 - xp:dataTable tag, 94-95
 - xp:include tag, 99
 - xp:panel tag, 87-90
 - xp:repeat tag, 95-98
 - xp:section tag, 100
 - xp:tabbedPanel tag, 99-100
 - xp:table tag, 90-91
 - xp:view tag, 91-93
- control tags
 - explained, 64-65
 - xp:dataTimeHelper tag, 68-69
 - xp:inputRichText tag, 67
 - xp:inputText tag, 65-66
- converters, 107-109
- CSS (Cascading Style Sheets)
 - CSS files, 719-720
 - style class reference, 720-726
- data binding, 59-60
- data source tags
 - xp:dataContext tag, 63
 - xp:dominoDocument tag, 61-62
 - xp:dominoView tag, 62-63
- Data Table markup, 309-311
- display control tags
 - xp:fileDownload tag, 86-87
 - xp:fileUpload tag, 84-85
 - xp:image tag, 84
 - xp:label tag, 83
 - xp:text tag, 83-84
- explained, 50-51
- HTML tags, 127-128
- markup, 33-34, 38
- resources
 - Dojo modules, 105
 - generic head resources, 106
 - metadata resources, 106-107
 - Notes/Domino Java API classes, 714
 - resource bundles, 104-105
 - script libraries, 101-103
 - style sheets, 103-104
 - tag reference guide, 711-712
 - XSP Java classes, 712-714
 - XSP JavaScript pseudo classes, 715-716
- selection control tags
 - xp:checkBox tag, 79
 - xp:checkBoxGroup tag, 81
 - xp:comboBox tag, 76-79
 - xp:listBox tag, 74-76
 - xp:radio tag, 80
 - xp:radioGroup tag, 81-82
- simple actions, 118-125
- simple properties, 52
- tags. *See* individual tag name
- validators, 110-118
- XSP client JavaScript library, 210-211
- XSP Document Action Picker, 472
- XSP Page Picker, 472
- XSP script library, 201-204
- xsp-config file
 - completing, 464-467
 - creating, 431-432
 - creating base.xsp-config, 446-449
 - creating interface, 450-452
 - defined, 422
 - designer-extension tags, 468-469
 - editor tag, 469-472
 - inheriting xsp-config properties, 441-446
 - styleClass attribute, 472-473
 - tags, 432-434
- xsp.css file, 720
- xsp.persistence.* properties, 669-670
- xsp.properties file, 581-583
- XSPContext class, 201
- xspFF.css file, 720
- xspIE.css file, 720
- xspIE06.css file, 720
- xspIE78.css file, 720
- xspIERTL.css file, 720
- xspLTR.css file, 720
- xspRCP.css file, 720
- xspRTL.css file, 720
- xspSF.css file, 720
- XSPUrl class, 201-203
- XSPUserAgent class, 201-203, 601-603
- XUL (XML User Interface Language), 496
- XULRunner, 496-497