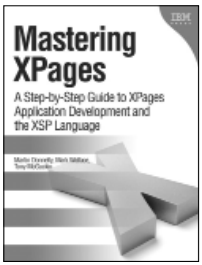
IBM PRESS

XPages Extension Library

A Step-by-Step Guide to the Next Generation of XPages Components

Paul Hannan, Declan Sciolla-Lynch, Jeremy Hodge,
Paul Withers, and Tim Tripcony

Related Books of Interest



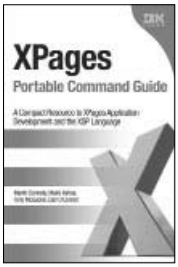
Mastering XPages

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

By Martin Donnelly, Mark Wallace, and Tony McGuckin

ISBN: 0-13-248631-8

The first complete, practical guide to XPages development—direct from members of the XPages development team at IBM Lotus. Martin Donnelly, Mark Wallace, and Tony McGuckin have written the definitive programmer's guide to utilizing this breakthrough technology. Packed with tips, tricks, and best practices from IBM's own XPages developers, *Mastering XPages* brings together all the information developers need to become experts—whether you're experienced with Notes/Domino development or not. The authors start from the very beginning, helping developers steadily build your expertise through practical code examples and clear, complete explanations. Readers will work through scores of real-world XPages examples, learning cutting-edge XPages and XSP language skills and gaining deep insight into the entire development process. Drawing on their own experience working directly with XPages users and customers, the authors illuminate both the technology and how it can be applied to solving real business problems.



XPages Portable Command Guide

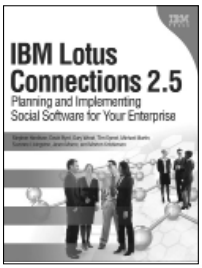
A Compact Resource to XPages Application Development and the XSP Language

By Martin Donnelly, Maire Kehoe, Tony McGuckin, and Dan O'Connor

ISBN: 0-13-294305-0

Now, there's a perfect portable XPages quick reference for every working developer. Straight from the experts at IBM, *XPages Portable Command Guide* offers fast access to working code, tested solutions, expert tips, and example-driven best practices. Drawing on their unsurpassed experience as IBM XPages lead developers and customer consultants, the authors explore many lesser known facets of the XPages runtime, illuminating these capabilities with dozens of examples that solve specific XPages development problems. Using their easy-to-adapt code examples, you can develop XPages solutions with outstanding performance, scalability, flexibility, efficiency, reliability, and value.

Related Books of Interest



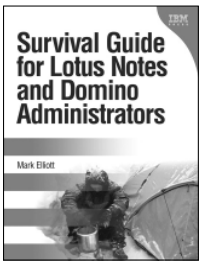
IBM Lotus Connections 2.5

Planning and Implementing Social Software for Your Enterprise

By Stephen Hardison, David M. 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.

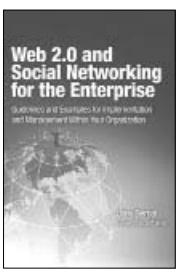


Survival Guide for Lotus Notes and Domino Administrators

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.



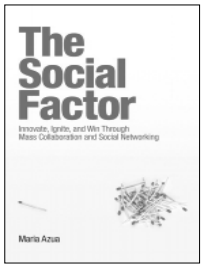
Web 2.0 and Social Networking for the Enterprise

Guidelines and Examples for Implementation and Management Within Your Organization

By Joey Bernal

ISBN: 0-13-700489-3

Related Books of Interest



The Social Factor

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

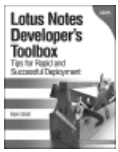
By Maria Azua

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.

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



Lotus Notes Developer's Toolbox

Elliott

ISBN: 0-13-221448-2

DB2 9 for Linux, UNIX, and Windows

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

Baklarz, Zikopoulos

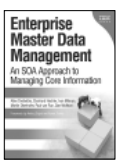
ISBN: 0-13-185514-X

The Art of Enterprise Information Architecture

A Systems-Based Approach for Unlocking Business Insight

Godinez, Hechler, Koening, Lockwood, Oberhofer, Schroeck

ISBN: 0-13-703571-3

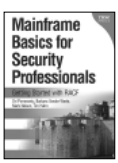


Enterprise Master Data Management

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 Weele, Nelson, Hahn

ISBN: 0-13-173856-9

This page intentionally left blank

XPages Extension Library

This page intentionally left blank

IBM PRESS

XPages Extension Library

A Step-by-Step Guide to the Next Generation of XPages Components

Paul Hannan, Declan Sciolla-Lynch, Jeremy Hodge, Paul Withers, and Tim Tripcony

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

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

Associate Publisher: Greg Wiegand
Marketing Manager: Stephane Nakib
Acquisitions Editor: Mary Beth Ray
Publicist: Heather Fox
Development Editor: Eleanor Bru
Editorial Assistant: Vanessa Evans
Technical Editors: Brian Benz, Chris Toohey
Managing Editor: Kristy Hart
Cover Designer: Alan Clements
Project Editor: Jovana Shirley
Copy Editor: Gill Editorial Services
Indexer: Lisa Stumpf
Compositor: Gloria Schurick
Proofreader: Mike Henry
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@pearsoned.com

The Library of Congress cataloging-in-publication data is on file.

ISBN-13: 978-0-13-290181-9

ISBN-10: 0-13-290181-1

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing May 2012

To Katie and Alec, my family—Paul H.
To my wonderful wife, Terri, for all her support—Declan
To the IBM Lotus XPages team for giving us all this
Open Source ExtLib goodness—Jeremy
To Mandy, my wife—Paul W.
To Paul Hannan: This book was your vision, and it owes its existence to your persistence, diligence, and enthusiasm.—Tim

Contents

Foreword

XPages is a truly groundbreaking technology. Its initial release in 2009 revolutionized web application development on Notes®/Domino® and brought new life and vibrancy to the developer community. As a runtime framework built on top of standards-based technologies and open source libraries, it greatly simplified the art of web development for the existing community and removed barriers to entry for non-Domino developers. Suddenly, it was a breeze to create a web page that pulled data from a Domino view or extracted a set of fields from a Notes document. The process of weaving these pages together to form compelling web applications became a no-brainer. In a nutshell, the advent of XPages meant that cranking out a half-decent Domino web application was easy and fast.

The good news is that after the 2009 revolution, XPages evolution continued apace. Within just nine months of XPages' official debut, we shipped a new release in Notes/Domino 8.5.1, which included lots of new features and, most notably, support for the Notes Client. This meant that users could take XPages web applications offline and run them locally in Notes! While we were working hard to push out more XPages technology, its adoption continued to grow. By Lotusphere® 2010, we were getting invaluable customer feedback on real-world XPages application development—the good, the bad, and the ugly. (It was mostly good!) A key theme emerged from the community at this time, one that really resonated with us. The message was simple: Yes, it was indeed easy and fast to write XPages web applications, but developing truly sleek and professional applications remained difficult and required expertise that was often beyond the core skill set of the typical Domino developer. Solving this would be our next big challenge.

One means of enabling the community to write better applications was through technical empowerment. Opening the XPages application programming interfaces (APIs) would allow developers to add their own XPages components to the framework and consume assets from other third parties. Thus, for Notes/Domino 8.5.2, we released the first public XPages APIs and

integrated the OSGi framework into the Domino server. As a means of illustrating how to use the APIs, we decided to provide a set of sample artifacts. The thinking was that if customers learned from these samples to build their own controls and shared them with each other across the community, developing top-drawer web applications would be easier to achieve. This led to the creation of a new XPages extension project, initially named Porus.

According to Plato, Porus was the personification of plenty, so this new library was intended to provide an abundance of new capabilities. True to its name, Porus quickly grew and soon boasted a large set of new controls, datasources, and other XPages assets. In fact, it was so effective that we wanted to build our next generation of XPages application templates on top of it, and that's where we ran into a problem: The library was simply too big to fit into the next Notes/Domino maintenance release. Moreover, we didn't want to wait for the next release. We wanted our customers to benefit from all the bountiful goodies of Porus as quickly as possible, and that meant being able to install it on top of the latest Notes/Domino release (8.5.2). What to do?

With the benefit of 20-20 hindsight, perhaps moving our internal Porus library to a public open source model out on OpenNTF.org was the obvious next move, but this was not so clear cut at the time. You must bear in mind that none of the core XPages runtime or Notes/Domino platform code is available as open source, so going down this road would be a new departure for us. The advantages of an open source model, however, were appealing. First, we could share our library with the development community more or less immediately and then update it when needed. This would allow us to deliver before the next Notes/Domino maintenance release and be independent of its constraints. It would also allow us to provide all the benefits of our Extension Library (ExtLib) while they are their most relevant to the community. The IT industry evolves at a rapid pace, so what's new and cool today can be old hat tomorrow; the timeliness of technology delivery can be a crucial factor in its success or failure. Being at the bleeding edge requires an agile delivery model, and we recognized that our traditional model simply could not adapt and respond quickly enough to the rapidly mutating demands of the market.

Of course, we had firsthand experience of the dynamic nature of open source systems by virtue of the fact that XPages depends on such components. The Dojo JavaScript library, which is at the core of XPages, is a perfect example. It typically provides two major releases per year, plus some maintenance updates. Not only do these releases constantly add new features and fixes, they target the latest browsers available in the market. With the most popular browsers piling through major release after major release in quick-fire succession and auto-updating themselves on end-user desktops, the Dojo project is well adapted to what is required to stay relevant in the modern IT world. The Notes/Domino product release cycle, on the other hand, is a heavyweight process. The last months in our release cycles are spent solidifying the products, with no new features being added, to minimize quality risks. On the one hand, this process helps to produce high-quality software, but on the other, it doesn't keep pace with the overall evolution rate of the modern industry.

Quite apart from speed and agility, however, is the critical element of transparency. Twenty-first century developers no longer want black boxes of code that they can use blindly. They expect to go further: They want to understand what the code does and how it works. They want to be able to debug it, to extend it. They want to share with a community. If you don't provide these capabilities, developers will find a way to get access to your code anyway. By nature, script languages are delivered in source form (if sometimes obfuscated), and even compiled languages such as Java™ or C# can be easily introspected.

September 1, 2010 was a landmark date for XPages, because it was when the XPages ExtLib was introduced as an open source project on OpenNTF.org. The response was amazing. The community latched on to this project from the get-go and ran with it. Today it proudly stands well clear of the field as the most active project on OpenNTF, with more than 26,000 downloads.

Despite the XPages ExtLib's runaway adoption success, other issues arose. Soon it became clear that although the open source model gave us many benefits, it was by no means perfect. Open source projects are often started by developers who put a greater emphasis on the code itself, leaving other pieces, such as documentation, test, accessibility, and support, behind. This is generally not acceptable for enterprise software intended for production. In fact, installing open source software in production environments is prevented by policy in many organizations. Perhaps even more significant is the fact that open source projects generally rely heavily on a small set of core developers. Open source repositories, like SourceForge and GitHub, are full of static projects that individuals started in their spare time and then left behind as the founders moved on to new pastures. For these projects to be successful, organizations that are prepared to stand behind the projects must endorse them. Without this endorsement, the use of open source software inevitably carries a certain amount of risk.

At this juncture, it was natural to wonder if we had gone full circle. To give customers the latest and greatest cutting-edge technology, we had to put a greater emphasis on code development. The open source model helped us achieve this. To give customers a system that IBM® fully supports and endorses, we needed to reinvest in all the aforementioned elements that we had sacrificed along the way for speed of innovation. Was it impossible to have both? We thought long and hard on this problem to come up with alternative distribution models that could satisfy the largest spectrum of users, from the early adopters to the more risk-averse conservative consumers. Our strategy can be summarized in three practices:

- We continue to deliver source code as early and frequently as possible to OpenNTF.org. Early adopters can continue to consume these offerings, which are supported not by IBM but by the ExtLib community. Thus, answers to questions and fixes to problems can be delivered promptly.
- Periodically, we package a subset of the ExtLib functionality available on OpenNTF.org and include this in an Upgrade Pack (UP) for Notes/Domino. Such UPs are fully supported by IBM and install on top of the latest shipping version of the Notes/Domino platform.

- The latest UP, plus any important subsequent features or fixes from OpenNTF, is always rolled into the next release of the product. Thus, between Notes/Domino release cycles, there is the potential for multiple UPs.

This three-tiered model has numerous advantages. It allows us to continue to get real feedback from the early adopters—the consumers of the OpenNTF project. By the time the code actually makes the official UP, or later into the core product, it has already been used in many projects, making it robust as we fix and deliver the open source project on a frequent basis. Also, regardless of the distribution mode, the source code is always provided. On December 14, 2011, we delivered on this proposed model by shipping our first UP: Notes/Domino 8.5.3 UP1. There are more to come!

In a long-standing software organization, like Notes/Domino, UP was a real revolution—2009 all over again! It was the first time IBM Collaboration Solutions (aka Lotus®) had delivered significant pieces of software in this way. It was a huge challenge, but we successfully achieved it because of the high level of commitment of the XPages team, the help of the broader Notes/Domino application development teams, and, most importantly, the great support of the community. Thanks to all of you, the Upgrade Pack has been a tremendous success.

Speaking of success, the release of the first XPages book, *Mastering XPages*, at Lotusphere 2011 exceeded our initial expectations. Despite having shipped three times the normal stock levels to the Lotusphere bookstore, because of the high number of online preorders, the book was completely sold out by Tuesday morning. That had never happened before. Coincidentally, this was also the first Lotusphere that discussed the ExtLib. So with the buzz of *Mastering XPages* in full flow, we floated the idea of another book, dedicated to the ExtLib. This proposal was a little different. By this time we were surfing the social wave; given the open source model on which the project rested, we wanted to get the community involved. Later that same Tuesday, the idea of a new ExtLib book was tweeted, proposing that a different author write each chapter. This social technique worked well. We rapidly got a list of volunteers from the community, which demonstrated both the great commitment of our community as well as the power of social media today. As a result, we ended up with a team of great experts, la crème de la crème, contributing to this book.

You'll note as you leaf through the chapters that the XPages ExtLib is moving to Social. We added numerous social-oriented features, which are certainly going to evolve rapidly over time. Take advantage of them, add social capabilities to your applications, and connect them to the world. There are fantastic opportunities opening up in this space. At the time *Mastering XPages* was published in 2011, we claimed we were at the beginning of a great XPages odyssey. Without a doubt, the success of the ExtLib has proven this. But we're not done; the story relentlessly continues. Further adventures in Social and Mobile will be our major themes going forward, and the XPages ExtLib will continue to be at the core of our innovation.

Enjoy the ExtLib as much as we do!

—Philippe Riand and Martin Donnelly, XPages Architects

Preface

Lotusphere 2011 was memorable in a lot of ways. It was another rip-roaring success for XPages as it continues to gain traction, make converts out of once-skeptics, and project a vision of what application development is going to look like in the years to come. The same event was also notable for the publication of the first real technical book on this technology, *Mastering XPages* by Martin Donnelly, Mark Wallace, and Tony McGuckin. Its approach was to document XPages in a way that hadn't been done before. It created a fantastic stir at Lotusphere 2011 that has reverberated throughout the coming year. Lotusphere, similar to other events, brings like-minded people together to meet face to face and talk. It was at Lotusphere 2011 that a group of XPagers (anyone who develops XPages applications) was talking about how wonderful the *Mastering XPages* book was and expressing how they couldn't wait until the next XPages book was written. This started the ball rolling.

We all have ideas. Some of these ideas never see the light of day, which is not necessarily a bad thing. Other ideas don't go away. The idea for another XPages book began to snowball. By the end of Lotusphere week, more than a few of us nearly swore in blood that we would write this book. And so we did.

The initial target for publication of this book was Lotusphere 2012. When we started to write this book in June 2011, that target was realistic. But as the long summer progressed, those busy bees in the XPages development team were deep into a process of reshaping the XPages ExtLib so IBM would fully support it. Add on the new support for relational databases and the new features to support social application development released to OpenNTF in the latter half of the year; the authors were effectively writing about a moving target. Each moving target stops occasionally to catch its breath.

A milestone was developing with the release of the Lotus Notes Domino 8.5.3 Upgrade Pack (UP) in December 2011. It was a significant release, because it was the first of its type in the

20-year history of Lotus Notes Domino. New features were being released to the market between major releases of the core project, which brought forth the fully IBM-supported version of the XPages Extension Library (ExtLib). What better event to base a book around?

This Book's Approach

The main desire for this book is to collate the knowledge of the XPages ExtLib and to communicate that knowledge to you, the reader. We seek to do this in a progressive way, starting with the basics and finishing with the more technical areas. And it's these advanced areas that we believe will take XPages application development to new heights.

Most chapters, apart from Chapter 13, "Get Social," use one or two applications for reference: the XPages ExtLib Demo application (**XPagesExt.nsf**) and the TeamRoom XL template (**teamrm8xl.ntf**). At the time of writing, both of these applications contain examples for 100% of the controls and components available from the XPages ExtLib. In these examples, we will take you through how to use these controls, describe what the various properties are for, and in some cases recommend how you can take advantage of such controls.

This book targets the December 2011 releases of the XPages ExtLib, be it in the form of the Lotus Notes Domino 8.5.3 UP 1 release or the release to the OpenNTF project. The feature set encapsulated in these releases represents a high point in the story of the technology. But this is not to say that this story is complete—far from it. There may be another book in the offing that will tell the story of how this technology will reach its next high point. Only time will tell.

We recommend that before picking up this book, you become familiar with XPages. One excellent shortcut for this is reading the *Mastering XPages* book, which will give you a firm grounding before you step into the XPages ExtLib. However, you don't have to be an expert in XPages. A basic knowledge of XPages is all you need to take advantage of the ExtLib and build better, more efficient applications more quickly.

Some Conventions

This book employs a few conventions of note that will make reading smooth.

User-interface elements, such as menus, buttons, links, file paths, folders, sample XPages, and Custom Control and so on in Domino Designer or in applications, are styled in the text as bold, for example, "Go to the **Download/Releases** section." Attributes and their options that are selectable from the All Properties view in Designer are also in bold.

Code, be it programming script, markup, or XSP keywords in the text, is typically styled in mono font size. For example, "Developers who have used the Dojo dialog in the past will know that it is opened via Client-Side JavaScript using the `show()` function and closed using the `hide()` function."

Also, in code, the XPages XML markup examples that typically form the listings throughout the book have split multiple attributes to a new line. This makes it easier to read the markup.

Those experienced with reading XPages markup will recognize the default prefix used for the core controls namespace: `xp,` as in `xp:viewPanel` or `xp:button`. They will also recognize

that Custom Controls have their own prefix: `xc` as in `xc:layout` from the Discussion XL template. The XPages ExtLib namespace has its own prefix, `xe`, which is used for the more than 150 ExtLib controls; for example, `xe:dataView`.

How This Book Is Organized

This book is divided into four parts, each a progression for you to navigate through various levels of XPages ExtLib knowledge.

Part I, "The Extension Library, Installation, Deployment, and an Application Tour": This part is aimed at getting you started with the XPages ExtLib. It explains what it is and how you install and deploy it, and it demonstrates in a production-ready application how and why it is used.

- **Chapter 1, "The Next Generation of XPages Controls":** This chapter introduces you to the XPages ExtLib, explains why the controls and components contained within will take XPages application development to the next level, and describes some of the areas that are likely to help grow the XPages technology even further.
- **Chapter 2, "Installation and Deployment of the XPages Extension Library":** This chapter describes the various ways to install and deploy versions of the ExtLib, be it IBM Lotus Notes Domino R8.5.2 or R8.5.3, or server, Domino Designer, or Notes Client.
- **Chapter 3, "TeamRoom Template Tour":** The purpose of this chapter is twofold. First, it is to gently introduce you to the XPages ExtLib. Second, it is to demonstrate how an existing template was modernized with this exciting new technology with features that are built entirely using the ExtLib in a production-ready application.

Part II, "The Basics: The Applications Infrastructure": This is the part of the book where each of more than 150 controls in the XPages ExtLib is described. These six chapters are laid out in a way that a typical Domino application developer might expect; start with a form, and then move on to views and to the overall navigation and layout. That is not to say that you have to read these chapters in that sequence to get a full understanding of the controls. An XPages app developer typically starts with the application layout and navigation before moving on to view and form controls. The sequence in how you read them is up to you. Each chapter can be taken in a standalone fashion.

- **Chapter 4, "Forms, Dynamic Content, and More!":** This chapter, along with Chapters 5 and 6, describes those controls that are typically used in the form of an XPage. With the use of Form Layout, Post, and Dynamic Content and Switch controls, you can quickly take advantage of these prebuilt and preformatted components to deploy complex layouts and design patterns.

- **Chapter 5, "Dojo Made Easy":** Whether you are familiar with Dojo or not, this chapter is aimed at how you can take advantage of this toolkit, which has been encapsulated into the Dojo controls for the XPages ExtLib. Without the ExtLib, configuring Dojo components can be tricky. The controls in the ExtLib make it easier.
- **Chapter 6, "Pop-Ups: Tooltips, Dialogs, and Pickers":** The ExtLib contributes tooltips for displaying additional content, dialogs for displaying or managing content, and pickers for facilitating selection of values. The XPages ExtLib makes this easier for developers, overcoming some of the challenges of integrating Dojo and XPages. This chapter describes all this.
- **Chapter 7, "Views":** Before the ExtLib, there were three available core container controls for displaying a collection of documents: the View Panel, the Data Table, and the Repeat Control. The ExtLib provides some new controls to help you take the display of a data collection to new levels. This chapter describes each one of these new view controls.
- **Chapter 8, "Outlines and Navigation":** For the end user to be able to switch between the different views in the application, you need to create an application layout and navigation. This chapter covers both the Dojo layout controls and navigation controls that have been added to the XPages ExtLib.
- **Chapter 9, "The Application's Layout":** In this chapter, you learn use of the Application Layout control, which helps you meet the challenge of creating an effective application interface that is not only pleasing, but intuitive and consistent, allowing users to predict what behaviors will produce the desired effect. All this is despite the difficulties presented when developing applications with the browser as your target platform.

Part III, "Bell and Whistles: Mobile, REST, RDBMS, and Social": In this part of the book, the big four deliverables to the XPages ExtLib in 2011 are described. If Part II of this book marks a step up in developing XPages applications, this part marks another. The next four chapters effectively describe the direction application development will progress in the coming years. Each of these chapters stands alone.

- **Chapter 10, "XPages Goes Mobile":** Mobile is the technology of the age. Owning a mobile device is no longer a luxury but a necessity. This fact is becoming increasingly important in business, as desktops and laptops are being superseded by tablets and smartphones. This transition has many challenges, ranging from the user interface (UI) design to security. XPages and the ExtLib are in place to meet these mobile challenges. This chapter shows how to meet and overcome these obstacles.

- **Chapter 11, "REST Services":** REpresentational State Transfer (REST) is important to the new Web 2.0 programming model. New technologies like OpenSocial and Android are embracing REST services to allow remote clients access to Server-Side data. The XPages ExtLib has RESTful services in place, so a whole range of exciting data-handling options open for the XPages developer.
- **Chapter 12, "XPages Gets Relational":** This chapter reviews concepts behind integrating relational data and the new relational database components that the ExtLib provides, including JDBC, the Connection Pool and Connection Manager, the datasources, and the Java and Server-Side JavaScript (SSJS) APIs included to integrate relational data into an XPages application.
- **Chapter 13, "Get Social":** Social and social business are the buzzwords of the age. This chapter uses a definition of social applications in the context of XPages, custom application development, and IBM Lotus Domino/IBM XWork Server. It describes the new requirements, maps them to technologies, and shows how the ExtLib helps implement these new requirements.

NOTE: At the time we were writing this manuscript, we were using the product called LotusLive™. This product has since been renamed IBM SmartCloud™ for Social Business.

Part IV, "Getting Under the Covers with Java": Gaining a fuller understanding of XPages Extensibility can be achieved with a little knowledge of Java. In this part of the book, the aim is to help you round out this knowledge and enable you to get the most out of the ExtLib.

- **Chapter 14, "Java Development in XPages":** With the addition of XPages to IBM Lotus Notes Domino, the capacity for inclusion of Java in applications has never been easier or more powerful. This chapter provides a glimpse into some of the many ways Java can take your applications to the next level, as well as a few ways that you can get even more use out of some of the XPages ExtLib controls already described in previous chapters.

This page intentionally left blank

Acknowledgments

Books aren't produced by one person. If they were, there would be very few of them. It takes a team of people to get a book to its rightful place on the shelf. That's stating the obvious, we know, but it's to make the point that we would like to thank a whole ream of people who have helped us get this book out the door.

First, we would like to thank the contributing authors for helping out on the book. Without Niklas Heidloff, Stephen Auriemma, Lorcan McDonald, and Simon McLoughlin, we wouldn't be where we are.

A sincere expression of gratitude has to go to the technical reviewers, Brian Benz and Chris Toohey. You guys rock! Your patience, insight, and expertise were a great help to us. Thanks for sticking with us through our adventure.

Thanks for all the leadership help of the Notes Domino Application Development team, especially Eamon Muldoon, Martin Donnelly, Philippe Riand, Pete Janzen, and Maureen Leland for supporting this book from the beginning to the end.

Still at IBM, we would like to thank the following people, who helped put the XPages ExtLib on the map: Andrejus Chaliapinas, Brian Gleeson, Darin Egan, Dan O'Connor, Dave Delay, Edel Gleeson, Elizabeth Sawyer, Graham O'Keeffe, Greg Grunwald, Jim Cooper, Jim Quill, Joseph J Veilleux, Kathy Howard, Kevin Smith, Lisa Henry, Maire Kehoe, Mark Vincenzes, Michael Blout, Mike Kerrigan, Padraic Edwards, Peter Rubinstein, Rama Annavajhala, Robert Harwood, Robert Perron, Teresa Monahan, Tony McGuckin, and Vin Manduca.

Going back to the beginning, we would like to thank Philippe Riand (yes, him again) for lighting the fire with that Twitter post (https://twitter.com/#!/philriand/status/32730855042457601) at Lotusphere 2011. This tweet reverberated, and the XPages community and the wider Lotus Community responded. It is safe to say that without this community, the idea for the book would never have gotten off the ground, so a great big thank-you to all. There aren't

enough pages available to thank everyone in the community, but we would like to mention Bruce Elgort, Darren Duke, David Leedy, John Foldager, John Roling, Matt White, Michael Bourak, Michael Falstrup, Nathan T. Freeman, Per Henrik Lausten, Phil Randolph, René Winkelmeyer, Tim Clark, Tim Malone, and Ulrich Krause for the help and inspiration in achieving liftoff and flight.

Still in the community, we would like to thank all those who have participated in the ExtLib project through OpenNTF who have been the early adopters of this technology. Without your feedback, this project likely wouldn't have gotten off the runway.

Finally, we would like to thank Mary Beth Ray, Chris Cleveland, Ellie Bru, Vanessa Evans, Jovana Shirley, Lori Lyons, Steven Stansel, Ellice Uffer, and Karen Gill at IBM Press and Pearson Education for being such wonderful partners in this project.

About the Authors

This book has many authors, all from the XPages community.

Paul Hannan is a senior software engineer in the IBM Ireland software lab in Dublin and a member of the XPages runtime team. He has worked on XPages since it was known as XFaces in Lotus Component Designer. Previous to this, he worked on JSF tooling for Rational® Application Developer, and before that on Notes Domino 6 back to Notes 3.3x and Lotus ScreenCam. A native of County Sligo, Paul now lives in Dublin with his wife Katie and son Alec. A recent convert (dragged kicking and screaming) to opera (not the web browser), Paul also enjoys thinking about stuff, taking pictures, commanding the remote control, and playing with his son and his Lego.

Declan Sciolla-Lynch was born in Dublin, Ireland and now lives in Pittsburgh, Pennsylvania. Declan has been working with IBM Lotus Notes/Domino for more than 15 years. He wrote one of the first XPages learning resources on his blog and is widely considered one of the community's XPages gurus. Declan has spoken at Lotusphere on a number of occasions and has contributed popular projects to OpenNTF, the community's open source hub. He is also an IBM Champion. He and his wife have three dogs and three cats and go to Disney theme parks whenever they get a chance.

Jeremy Hodge, from southern Michigan, is a software architect with ZetaOne Solutions Group and has more than 15 years' experience in the software design industry. He has designed and implemented applications in the vertical market application, custom application, Software as a Service (SaaS), and off-the-shelf product spaces in many platforms and languages, including IBM Lotus Notes/Domino, C/C++/Objective-C, Java, Object Pascal, and others. He has served as the subject matter expert for courses with IBM Lotus Education, including those on XPages applications. He blogs on XPages at XPagesBlog.com and his personal blog at hodgebloge.com.

Paul Withers is senior Domino developer and team leader at Intec Systems Ltd, an IBM Premier Business partner in the UK. He is an IBM Champion for collaboration solutions and the cohost of The XCast XPages podcast. Paul has presented at Lotusphere and various Lotus User Groups across Europe. He has written blogs, wiki articles, and a NotesIn9 episode. He has authored reusable XPages controls and an application, XPages Help Application, on OpenNTF. Outside of work, Paul is a Reading FC supporter and netball umpire in the England Netball National Premier League.

Tim Tripcony leads the Transformer ExtLib development team at GBS, creating XPage components and other JSF artifacts that extend the native capabilities of the Domino platform. He maintains a popular technical blog, Tip of the Iceberg (TimTripcony.com), offering tips on cutting-edge Domino development techniques. He frequently speaks at user group meetings and technical conferences, including Lotusphere. Tim is a globally recognized expert on advanced XPage and JSF development and has been designated an IBM Champion.

Contributing Authors

Niklas Heidloff is a software architect working for the software group in IBM. He is focused on invigorating the application development community and promoting XPages as IBM's web and mobile application development platform for collaborative and social applications. In this role, he is the technical committee chair and a director of the Board of Directors of the open source site OpenNTF.org. Previously, Niklas was responsible for other application development areas in the IBM Lotus Domino space, including composite applications. Before this, he worked on IBM Lotus Notes, IBM WebSphere® Process Choreographer, and IBM Workplace Client Technology. In 1999, he joined IBM as part of the Lotus Workflow team. Niklas studied at the university in Paderborn, Germany, and has a degree in Business Computing (Diplom Wirtschaftsinformatiker).

Stephen Auriemma is an advisory software engineer currently working in the IBM Littleton software lab on an XPages and Domino Access (REST). Stephen has a master's degree in computer science from Boston University. In the past, he worked as a developer on various projects, including Composite Applications for Notes 8.0, the open source project on Apache called Xalan for IBM Research, and Domino Offline Services for Lotus. Stephen started his career with IBM in 1996, providing development technical support for Notes programmability. He lives in Chelmsford, Massachusetts, with his wife and two daughters, Jessica and Amanda.

Simon McLoughlin is a graduate software developer in the IBM Ireland software lab in Dublin working for the XPages mobile team. A graduate of the Institute of Technology, Tallaght, he was responsible for reworking and adding the mobile front end to the Discussion and TeamRoom templates delivered with the XPages ExtLib. In college, he studied computer science. In his last year there, he joined with IBM on a research project; the result was a smartphone push alert system to alert native iPhone/Android users that a server undergoing a long run test was running low on resources or approaching some critical state. This project finished in the top 3 for the

Irish software awards for the student category of most commercially viable/innovative. Living in Dublin, Simon enjoys experimenting with new mobile technology and suffers greatly from an addiction to computer games.

Lorcan McDonald is a senior software engineer on the XPages team in the Dublin office of the IBM Ireland software lab. He is the tech lead on the XPages Mobile controls project and has worked on the Domino platform for three years, split between the XPages Runtime team and Quickr® Domino. Before coming to IBM, Lorcan worked on financial web applications for the credit card and trading industries. Born and raised in Sligo, he has been living in Dublin for more than a decade. He never stops thinking about computing problems. He has been known to perform and record music as 7800 beats, presumably via some sort of web interface.

CHAPTER 5

Dojo Made Easy

Ever since IBM Lotus Domino Release 8.5.0, the Dojo toolkit has been IBM's JavaScript framework of choice. It comes preinstalled with the Domino server and is intrinsically linked with the XPages runtime. Much of the standard XPages functionality extends the standard Dojo toolkit. Developers have been integrating Dojo with XPages since its introduction into Domino, taking advantage of the prebuilt code libraries to enhance their XPages applications. Subsequent releases have specifically targeted making it easier to combine XPages and Dojo. To this end, the Extension Library Dojo controls are designed to make it easier still to implement some of the more frequently used modules, whether for novice developers or seasoned developers making extensive use of the Dojo modules and attributes available.

Developers already familiar with Dojo might want to jump to the section "Dojo Modules and Dojo in the Extension Library." For those who have never or rarely used Dojo, the following sections will give some background and walk through a couple of examples of Dojo modules in XPages.

What Is Dojo?

Dojo is an open source JavaScript framework, a free collection of cross-browser-compatible functions and widgets, first released in 2006. Each JavaScript file is an object with various attributes and functions, referred to as a Dojo module. For example, `dijit.form.TextBox` is a Dojo module that converts an HTML input tag to a Dojo-styled text box. Modules can also extend other Dojo modules, so `dijit.form.ValidationTextBox` and `dijit.form.NumberTextBox` both extend `dijit.form.TextBox`. This allows developers to add functionality by creating their own extensions without needing to modify the preinstalled files. One of the strengths of these Dojo modules is that they are specifically designed to support developers in addressing accessibility requirements.

All the XPages Client-Side JavaScript functionality can be found in script libraries in the Dojo root folders; most either extend or mimic standard Dojo modules. For example, any partial refresh calls `dojo.xhrGet()` or `dojo.xhrPost()`, the standard Dojo AJAX requests to the server. The XPages DateTimeHelper extends a number of Dojo modules, including `dijit.form.Button`, `dojo.date`, and `dijit._widget`. Client-Side validation also mimics the format of Dojo functions. Consequently, the core Dojo libraries are loaded in an XPage by default, so even a blank XPage in which you are not explicitly including Client-Side JavaScript libraries will include the following Dojo JavaScript libraries, as shown in Figure 5.1:

/xsp/.ibmxspres/dojoroot-1.6.1/dojo/dojo.js

/xsp/.ibmxspres/.mini/dojo/.en-gb/@Iq.js (for English)

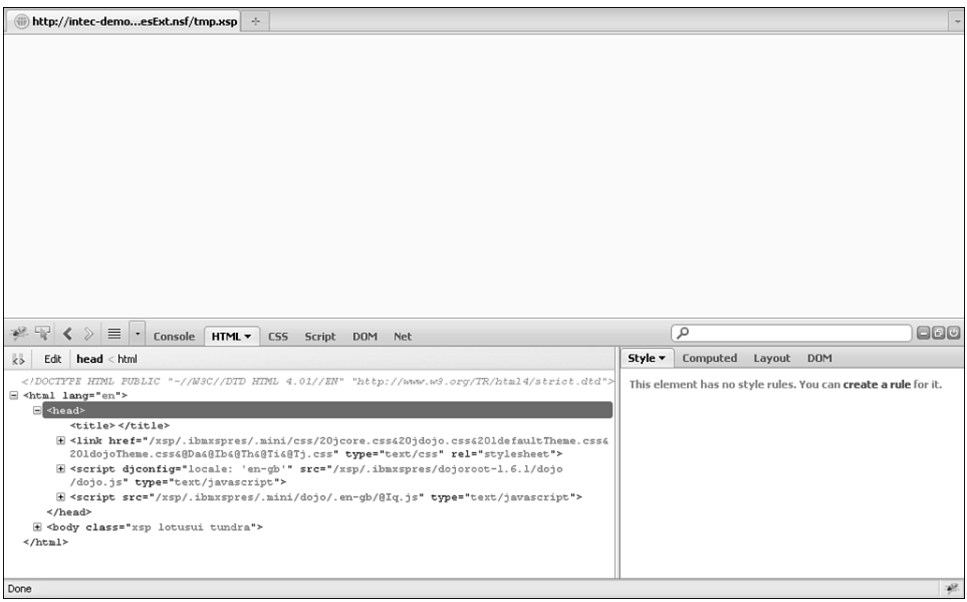


Figure 5.1 Dojo libraries loaded.

Default Dojo Libraries Using Dojo Modules in XPages

Before Domino 8.5.2, incorporating Dojo modules into XPages was challenging because many controls did not have a dojoType attribute. The only way to implement Dojo on an EditBox, for example, was to apply it programmatically. So in addition to the core control client side,

JavaScript was required to trigger on load. Listing 5.1 demonstrates this programmatic implementation of the `dijit.form.ValidationTextBox`. Lines 1 to 4 show the core Edit Box control. Line 6 then begins an Output Script control, triggering `XSP.addOnLoad()` in line 16. The `addOnLoad()` calls a function that generates a new `dijit.form.ValidationTextBox` on line 9 adding various attributes. Line 13 adds the parameter to the new function, which applies the Dojo module to the Edit Box control.

Listing 5.1 Programmatic Implementation of `dijit.form.ValidationTextBox`

```
1  <xp:inputText
2     id="response"
3     value="#{ansDoc.response}">
4  </xp:inputText>
5
6  <xp:scriptBlock
7     id="scriptBlock1">
8     <xp:this.value><![CDATA[var convertInput = function() {
9     new dijit.form.ValidationTextBox(
10          {name:"#{id:response}",
11                required: true,
12                promptMessage: "Please complete the field"},
13          XSP.getElementById("#{id:response}")
14          );
15    };
16    XSP.addOnLoad(convertInput);
17    ]]></xp:this.value>
18  </xp:scriptBlock>
```

There is no reason you cannot use programmatic conversion of a core control to a Dojo module, if applicable. But with Domino 8.5.2, it became possible to declaratively convert the control thanks to the addition of the dojoType attribute to a variety of core controls. So for the Edit Box control, for example, in Domino 8.5.2 a Dojo panel was added and **dojoType** and **dojoAttributes** properties appeared on the All Properties panel, as shown in Figure 5.2. Not only is this easier to implement, but text strings entered as Dojo attribute values are picked up if localization is required and turned on for an application.

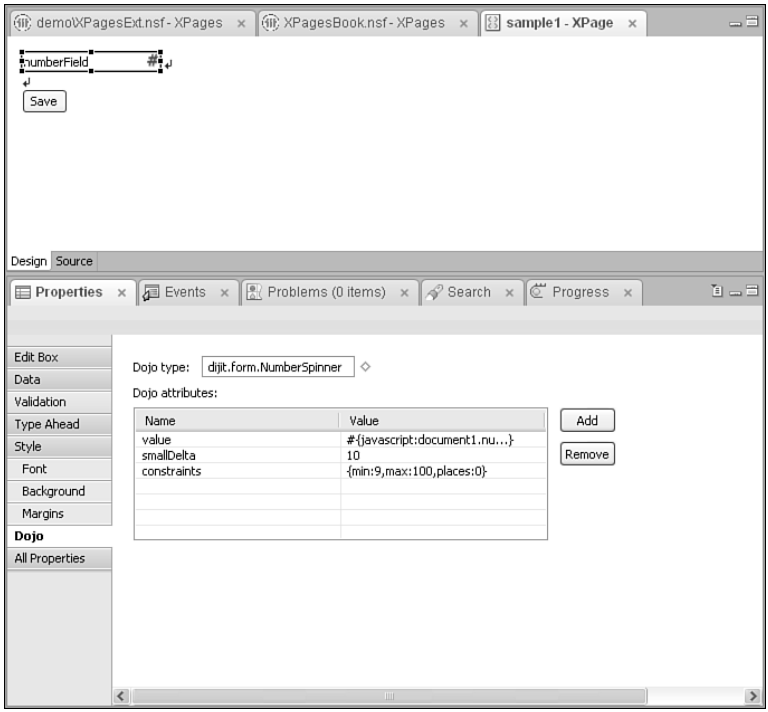


Figure 5.2 Dojo panel on Edit Box control.

Before digging into the Extension Library, let's review several examples of implementing Dojo in XPages. Any developer who has used Dojo modules in XPages is aware of the steps required, ingrained quite probably by forgetting one of the steps at one time or another. The first critical step is to set `dojoParseOnLoad` and `dojoTheme` attributes to `"true"`, as shown in lines 4 and 5 of Listing 5.2. The former tells the browser that after loading it needs to convert all content with a **dojoType** property; the latter tells the browser to load the relevant theme for styling all Dojo widgets (or *dijits*). The final step is to add as resources on the XPage any Dojo modules referenced on the page in a **dojoType** property.

Listing 5.2 `dojoParseOnLoad` and `dojoTheme`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xp:view
3     xmlns:xp="http://www.ibm.com/xsp/core"
4     dojoParseOnLoad="true"
5     dojoTheme="true">
6
7  </xp:view>
```

Of course, you can perform all this on either an XPage or a Custom Control, but for simplicity, the reference will only be made to XPages. To provide a more appropriate comparison with the Extension Library controls, the examples in the sections that follow focus on declarative implementations of Dojo modules.

Simple Dojo Example: `dijit.form.ValidationTextBox`

The Dojo modules applied to an Edit Box are among the simplest implementations of Dojo. The `dijit.form.ValidationTextBox` is a simple extension to the Edit Box, which adds Client-Side validation with a styling consistent with other dijits to offer immediate validation and a prompt message. It has a number of Dojo attributes, some of which you can see in Listing 5.3. Figure 5.3 shows the resulting output. There is a host of printed and online documentation of Dojo (for examples, see the Dojo Toolkit website http://dojotoolkit.org/reference-guide/index.html). This book will not seek to exhaustively reproduce a glossary of the Dojo attributes and what they do.

Listing 5.3 `dijit.form.ValidationTextBox`

```
<xp:this.resources>
   <xp:dojoModule
      name="dijit.form.ValidationTextBox">
   </xp:dojoModule>
</xp:this.resources>
<xp:inputText
   id="inputText1"
   value="#{viewScope.validationBox}"
   dojoType="dijit.form.ValidationTextBox">
   <xp:this.dojoAttributes>
      <xp:dojoAttribute
         name="required"
         value="true">
      </xp:dojoAttribute>
      <xp:dojoAttribute
         name="promptMessage"
         value="Please complete this field">
      </xp:dojoAttribute>
   </xp:this.dojoAttributes>
</xp:inputText>
```

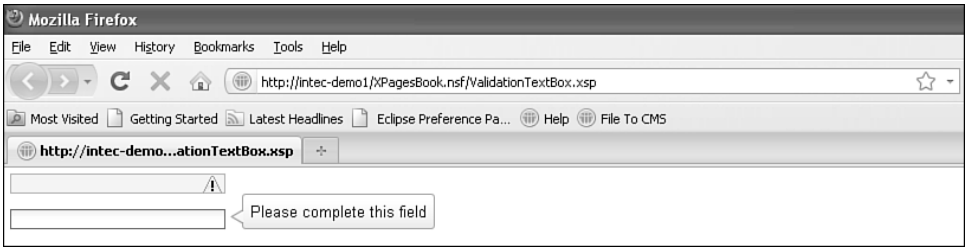


Figure 5.3 `dijit.form.ValidationTextBox`.

Defining Dojo modules and attributes is made a little challenging because there is no type-ahead or other context-sensitive help to advise on the Dojo modules available for use. There is also no validation of the correct naming conventions for the modules or validation of additional resources that need to be included. But this is to provide developers with the flexibility to take advantage of new releases of Dojo at the earliest opportunity and develop their own Dojo modules. For developers who are comfortable with the attributes available, this is not a problem; however, novice developers might find the size of the Dojo toolkit daunting.

Dojo Example for Slider

Some dijits are more involved than just setting a Dojo type and attributes to a Core control. A good example of this is the slider. There are actually two types of sliders: `dijit.form.HorizontalSlider` and `dijit.form.VerticalSlider`. The implementations are similar, so we shall just cover the `HorizontalSlider`.

As with `dijit.form.ValidationTextBox`, the slider is an input control, so you need to store the value in an Edit Box control (or, in most implementations, a Hidden Input control). However, you cannot directly attach the slider to the Edit Box. Instead, you apply the Dojo styling to a div and add an onchange event to pass the value to the Edit Box. Although the XPages Div control has **dojoType** and **dojoAttributes** properties, it does not have an onchange event, so it is easier to use an HTML div.

Further code is required to apply labels to the horizontal slider. You must apply an additional Dojo module to an HTML ordered list, `dijit.form.HorizontalRuleLabels`. Listing 5.4 shows the combination of XPage and HTML markup used to create a horizontal slider, which allows the user to select a value (in multiples of 10) within a range of 0 and 100, showing labels at increments of 20. The code required is rather extensive for a simple slider. Figure 5.4 shows the resulting output.

Listing 5.4 dijit.form.HorizontalSlider

```
<xp:this.resources>
   <xp:dojoModule
      name="dijit.form.HorizontalSlider">
   </xp:dojoModule>
   <xp:dojoModule
      name="dijit.form.HorizontalRuleLabels">   </xp:dojoModule>
</xp:this.resources>

<div
   id="horizontalSlider"
   dojoType="dijit.form.HorizontalSlider"
   value="50"
   minimum="0"
   maximum="100"
   discreteValues="11"   style="width:500px"
   showButtons="false"
   onChange="dojo.byId('#{id:horizontalHolder}').value =
dijit.byId('horizontalSlider').value">
   <ol
      dojoType="dijit.form.HorizontalRuleLabels"
      container="bottomDecoration">
      <li>0</li>
      <li>20</li>
      <li>40</li>
      <li>60</li>
      <li>80</li>
      <li>100</li>
   </ol>
</div>
<br />
<xp:inputText
   id="horizontalHolder"
   value="#{viewScope.horizontalSlider}"
   defaultValue="50">
</xp:inputText>
```

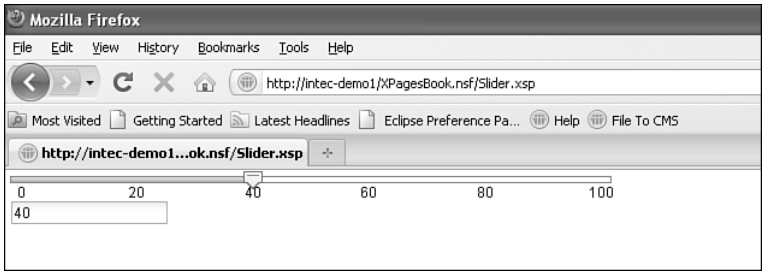


Figure 5.4 `dijit.form.HorizontalSlider`.

Dojo Themes

All the dijits are styled according to a theme. The theme is defined on the **XPages** tab in the **Application Properties**, accessed from **designer**, using the **Application Theme** dialog list, as in Figure 5.5. The OneUI and Server Default themes use tundra by default. If the property **Use runtime optimized JavaScript and CSS resources** at the bottom of this tab is checked, a single aggregated stylesheet is delivered to the browser. This includes the following stylesheet:

/xsp/.ibmxspres/dojoroot-1.6.1/dijit/themes/tundra/tundra.css

In addition, the tundra theme is applied to the body tag, so the output HTML is `<body class="xsp lotusui tundra">`.

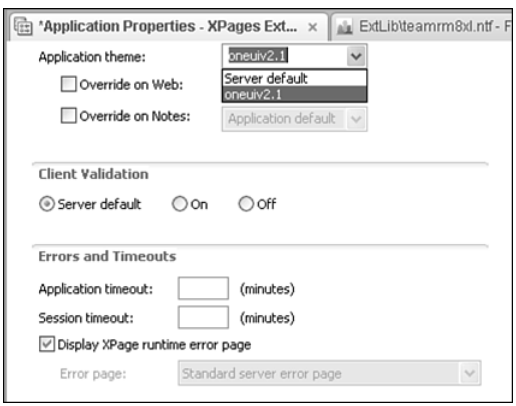


Figure 5.5 XPages tab of Application Properties in Domino Designer.

Dojo provides three other themes: nihilo, soria and, since Dojo 1.5.0, claro. Implementing these themes is just a matter of including the relevant stylesheets and applying the style to the body tag. The former is straightforward in XPages, the latter a little more involved. Within an XPage, you are limited on the attributes you can manipulate. However, via a custom theme, you can apply the Dojo theme to the body tag and reference the relevant stylesheets. If an application

is not currently using a theme, just create a new Theme design element, found under the **Resources** category in the **Application** pane.

You can insert the code in Listing 5.5 between the theme tags. Lines 1 through 5 include the Dojo-themed stylesheet. Lines 8 through 14 apply the Dojo theme to the `ViewRoot` control, which becomes the body tag when the web page is loaded. Note in particular the inclusion in lines 2 and 8 of `dojoTheme="true"`. By adding this, the logic checks whether the developer has set dojoTheme to `"true"` on the XPage or CustomControl. If the developer has set `dojoTheme` to `"true"`, the stylesheet is loaded and the class is applied. If not, the stylesheet is not loaded and the class is not applied. To use soria or claro, just replace the three instances of `nihilo` with the relevant theme name.

Listing 5.5 Applying a Dojo Theme

```
1  <!-- Include Dojo stylesheet -->
2  <resource dojoTheme="true">
3     <content-type>text/css</content-type>
4     <href>/.ibmxspres/dojoroot/dijit/themes/nihilo/nihilo.css</href>
5  </resource>
6
7  <!-- Add style to body element -->
8  <control dojoTheme="true">
9     <name>ViewRoot</name>
10    <property mode="concat">
11       <name>styleClass</name>
12       <value>nihilo</value>
13   </property>
14 </control>
```

Dojo Modules and Dojo in the Extension Library

As the examples in the preceding sections demonstrate, some Dojo modules are easy to implement into XPages, but others are more convoluted. Even for a confident developer already accustomed to using dijits in applications, it could get annoying to have to keep adding dojoTypes and dojoAttributes to all core controls, which was one of the driving forces behind implementing the Dojo controls in the Extension Library. Using native controls offered several other benefits:

- Easier to implement drag-and-drop functionality
- Promoting some of the more common Dojo modules available for use within XPages
- Validating and manipulating values
- Limiting the number of controls that need to be dropped onto the XPage or Custom Control

That is not to say that the Extension Library precludes the need to implement Dojo manually within XPages. It does not, nor is it intended to. Some Dojo modules, such as the `dojox.image.Lightbox` control, are not available in the Extension Library controls. Equally, there might be instances in which developers have created their own Dojo extensions that they still intend to use but do not have the skills or are not ready to componentize.

Benefits and Differences of Dojo Extension Library Components

By componentizing the Dojo modules as extended controls, the Extension Library offers several benefits. Performance is one aspect. Another is that if a Dojo control from the Extension Library is used, `dojoParseOnLoad` or `dojoTheme` does not need to be set and the relevant Dojo module(s) does not need to be added to an XPage. Whether accustomed or not to adding the gamut of dojo attributes to Dojo controls, the extended controls also avoid the need to remember (and indeed avoid mistyping!) dojo attributes. This also means that it is quicker to implement the extended controls than just setting a Dojo type and attributes, whether dragging and dropping and using the "pretty panels" or typing directly into the Source pane. And for developers who are integrating with Java beans, controls also allow options for integration with backend Java classes, whether with valueChangeListeners or for controlling return types of, for example, the Dojo Number Text Box or Dojo Number Spinner.

However, for dijits to use a Dojo theme other than tundra, the code outlined in Listing 5.5 for a Theme design element is still required to apply the relevant Dojo theme to the body tag. There is nothing within the Extension Library to short-circuit that requirement.

In the examples that follow, properties of the Extension Library are hard-coded, for ease of explanation. But remember that, as with any other property in XPages, the value of all the properties of the Extension Library controls can be programmatically calculated, either using on page load or dynamically.

Without further ado, let's start looking at the Dojo form controls from the Extension Library that add to the form controls we covered in the previous chapter. Other Dojo controls are covered in subsequent chapters. For example, the Dojo Data Grid control is covered in Chapter 7, "Views."

Dojo Extensions to the Edit Box Control

Many controls extend the Edit Box control, whether for storing text values, number values, or date/time values. These controls are not used in the TeamRoom database, so we will review the Extension Library demo database, which is available from OpenNTF. Specifically, we will review the **Core_DojoFormControls.xsp** XPage.

Dojo Text Box (`xe:djTextBox`)

The Dojo Text Box control is an excellent example of a control that appears to be simple but can provide functionality not available in the core Edit Box control. In most implementations, all that is required is to drag and drop it onto the XPage or custom control.

When you look at the properties available and compare them to the core Edit Box control, some differences become apparent. Table 5.1 describes the main properties that are standard across the Dojo widgets.

Table 5.1 Dojo Widget Properties

Property	Description
alt	Holds alternate text if the browser cannot display the control; uncommon for form controls.
waiRole	Defines the WAI-ARIA role for the control. For more information on WAI-ARIA, see http://www.w3.org/WAI/.
waiState	Defines the WAI-ARIA state of the control. For more information on WAI-ARIA, see http://www.w3.org/WAI/.
trim	Removes leading or trailing spaces, but not duplicate spaces within the field's value.
dragRestriction	If true, prevents the field from being draggable.
intermediateChanges	If true, triggers the onChange event for each value change.
tooltip	For most controls, such as Dojo Text Box, the title property is used to add hover text. Some controls, such as the Dojo Tab Pane, use the title property for the tab label. For those controls, this tooltip property is used instead to add hover text.

Table 5.2 describes the properties specific for the Dojo Text Box controls. On the All Properties panel of the Dojo Text Box, the data category contains the same properties as the Edit Box (`xp:inputText`) control. But a smaller subset of properties is listed under the basics category. Some of the options, including **autocomplete**, **password**, **htmlFilterIn,** and **htmlFilter**—visible on an Edit Box control—are not available for this control. Note that some properties like **readonly** and **maxlength** are camel case for the Dojo controls and become **readOnly** and **maxLength** on the Dojo Text Box control.

Table 5.2 `xe:djTextBox` Properties

Property	Description
lowercase	If true, the field's value is converted to lowercase when the user exits the field.
propercase	If true, the field's value is converted to propercase when the user exits the field.
uppercase	If true, the field's value is converted to uppercase when the user exits the field.

The Dojo Text Box also offers some additional properties. Some properties, such as **alt**, **tabIndex**, **title**, **waiRole**, and **waiState**, are standard for the Dojo extended controls, always appearing under the accessibility category. WAI might be unfamiliar to some Domino developers who are not used to web development. WAI is an initiative by the World Wide Web Consortium (W3C) to ensure that websites follow accessibility guidelines. This has been extended for applications by Web Accessibility Initiative—Accessible Rich Internet Applications (WAI-ARIA), which differentiates applications from static web pages. It is not yet standard, but it is good practice. A full taxonomy of roles (http://www.w3.org/WAI/PF/GUI/roleTaxonomy-20060508.html) and states (http://www.w3.org/WAI/PF/adaptable/StatesAndProperties-20051106.html) is available on the W3C site. The good news is that even if you do not define the **waiRole** and **waiState** properties on the Dojo extended controls, default roles and states are added. But, if required, the properties are exposed to allow you to override the defaults.

Other properties are exposed that offer additional functionality over the Edit Box control or even the standard TextBox control in the Dojo toolkit. In the basics category, the **maxLength** property enables developers to ensure that users are restricted to a certain number of characters. This is triggered on key press, so rather than alerting users after they have left the field, the user physically cannot type more characters than you allow. However, bear in mind that if the field should include punctuation, decimal separators, and so on, each counts as one character. You can use the **trim** property to remove any leading or trailing spaces. It does not remove duplicate spaces within the string.

The dojo category is expanded from the Edit Box control with some additional Dojo properties: **dragRestriction**, **intermediateChanges,** and **tooltip**. These properties are standard for the Dojo widgets and may not be appropriate for all controls. For example, the **tooltip** property is used only for controls such as the Dojo Tab Pane, where the **title** property has a different function than applying hover text. The format category provides boolean properties **lowercase**, **uppercase,** and **propercase** to force case conversion. The formatting takes effect as soon as the user exits the field.

Some of the differences in the events category between the Edit Box control and the Dojo Text Box control are just minor. Properties like **onfocus**, **onblur**, **onchange**, and **onclick** become **onFocus**, **onBlur**, **onChange,** and **onClick**. It's not a major difference, and indeed there is no difference in implementation. But there are a few additions. The mousing events are supplemented by **onMouseEnter** and **onMouseLeave**, ostensibly no different from **onMouseOver** and **onMouseOut**. A simple alert statement will show that the onMouseOver event is triggered before the onMouseEnter event. Likewise, onMouseOut is triggered before onMouseLeave.

Dojo Validation Text Box (`xe:djValidationTextBox`)

There are no prizes for guessing that the Dojo Validation Text Box control is similar to the Dojo Text Box control, except that it adds validation. All the properties we outlined on the Dojo Text Box control are available, including those for dynamically setting the value to lowercase, uppercase, or propercase and trimming the value.

However, the Dojo Validation Text Box is not, by default, mandatory. Initially, this sounds incomprehensible. What's the point of the Dojo Validation Text Box if it's not validated? But if we investigate a little further, we will come across the **promptMessage** property. This enables the developer to add a message for the user. At runtime, this is delivered to the user by default as a tooltip, as in Figure 5.6.

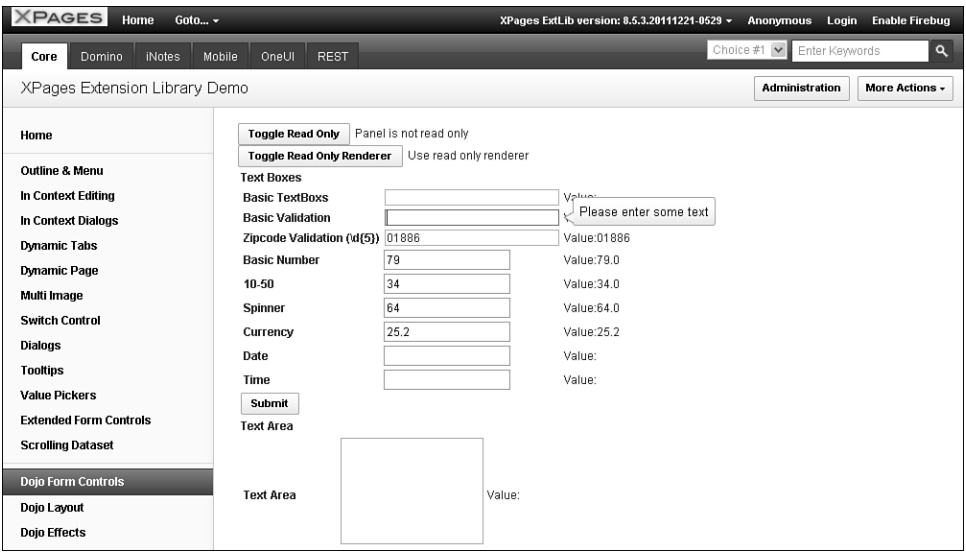


Figure 5.6 Dojo Validation Text Box promptMessage.

Basic validation is managed in the same way as for any other input control: by using the **required** property. But validation for the traditional Edit Box control is handled on the client or the server, as determined by the developer in the **Application Properties** or the administrator in the **Server Settings**. In the Dojo Validation Text Box, validation is always handled Client-Side, even if client validation is switched off in the **Application Properties**. That is because the Dojo Validation Text Box is a Dojo control, and Dojo validation runs Client-Side (because Dojo is a set of Client-Side JavaScript libraries). So as soon as the user tabs out of the field, the validation is triggered and the field is highlighted, as in Figure 5.7. As with the `dijit.form.ValidationTextBox` Dojo module, an error message in the **invalidMessage** property has no effect if the control just has the **required** property set to `"true"` but no other validation applied.

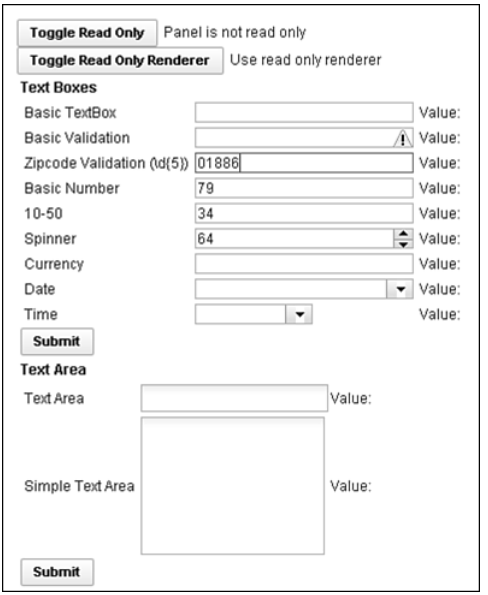


Figure 5.7 Dojo Validation Text Box error message.

But the Dojo Validation Text Box doesn't just validate that a value has been entered. In the dojo-widget category, the **regExp** property takes as its value a regular expression, a standard web development validation notation that is designed to be agnostic of programming language. The **regExpGen** property can generate a regular expression using Client-Side JavaScript. Rather than researching and typing a regular expression, Dojo provides some prebuilt objects for validating standard regular expressions, such as **dojo.regexp.realNumber** and **dojo.regexp.ipAddress**. These can be found in files like **dojo.number** and **dojox.validate**, all of which extend **dojo.regexp**, the object that defines the function to validate against regular expressions. For example, Listing 5.6 takes the ipAddress function in **dojox.validate.regexp.js**, amending it only to expect no parameters. As a function in the **regExpGen** property, this code will validate that the user enters a valid IP address, without the need to work out or type in the relevant regular expression. As with traditional XPages validation, there is a default, but developers can also provide their own message, using the **invalidMessage** property.

Listing 5.6 Validating an IP Address

```
<xe:djValidationTextBox
   value="#{sessionScope.djValidationTextBox1}"
   invalidMessage="Please enter a valid ip address">
   <xe:this.regExpGen><![CDATA[// summary: Builds an RE that matches an
IP address
```

```
//
// description:
//  Supports five formats for IPv4: dotted decimal, dotted hex, dotted
octal, decimal, and hexadecimal.
//  Supports two formats for Ipv6.
//
// flags  An object. All flags are boolean with default = true.
//    flags.allowDottedDecimal  Example, 207.142.131.235. No zero
padding.
//    flags.allowDottedHex  Example, 0x18.0x11.0x9b.0x28. Case
insensitive. Zero padding allowed.
//    flags.allowDottedOctal  Example, 0030.0021.0233.0050. Zero
padding allowed.
//    flags.allowDecimal  Example, 3482223595. A decimal number between
0-4294967295.
//    flags.allowHex  Example, 0xCF8E83EB. Hexadecimal number between
0x0-0xFFFFFFFF.
//      Case insensitive. Zero padding allowed.
//    flags.allowIPv6   IPv6 address written as eight groups of four
hexadecimal digits.
//    FIXME: ipv6 can be written multiple ways IIRC
//    flags.allowHybrid   IPv6 address written as six groups of four
hexadecimal digits
//      followed by the usual four dotted decimal digit notation of
IPv4. x:x:x:x:x:x:d.d.d.d

// assign default values to missing parameters
flags = {};
if(typeof flags.allowDottedDecimal != "boolean"){
flags.allowDottedDecimal = true; }
if(typeof flags.allowDottedHex != "boolean"){ flags.allowDottedHex =
true; }
if(typeof flags.allowDottedOctal != "boolean"){ flags.allowDottedOctal
= true; }
if(typeof flags.allowDecimal != "boolean"){ flags.allowDecimal = true;
}
if(typeof flags.allowHex != "boolean"){ flags.allowHex = true; }
if(typeof flags.allowIPv6 != "boolean"){ flags.allowIPv6 = true; }
if(typeof flags.allowHybrid != "boolean"){ flags.allowHybrid = true; }
// decimal-dotted IP address RE.
var dottedDecimalRE =
// Each number is between 0-255. Zero padding is not allowed.
```

Listing 5.6 (Continued)

```
    "((\\d|[1-9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])\\.){3}(\\d|[1-
9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])";

// dotted hex IP address RE. Each number is between 0x0-0xff. Zero
padding is allowed, e.g. 0x00.
var dottedHexRE = "(0[xX]0*[\\da-fA-F]?[\\da-fA-F]\\.){3}0[xX]0*[\\da-
fA-F]?[\\da-fA-F]";

// dotted octal IP address RE. Each number is between 0000-0377.
// Zero padding is allowed, but each number must have at least four
characters.
var dottedOctalRE = "(0+[0-3][0-7][0-7]\\.){3}0+[0-3][0-7][0-7]";

// decimal IP address RE. A decimal number between 0-4294967295.
var decimalRE =  "(0|[1-9]\\d{0,8}|[1-3]\\d{9}|4[01]\\d{8}|42[0-
8]\\d{7}|429[0-3]\\d{6}|" +
   "4294[0-8]\\d{5}|42949[0-5]\\d{4}|429496[0-
6]\\d{3}|4294967[01]\\d{2}|42949672[0-8]\\d|429496729[0-5])";

// hexadecimal IP address RE.
// A hexadecimal number between 0x0-0xFFFFFFFF. Case insensitive. Zero
padding is allowed.
var hexRE = "0[xX]0*[\\da-fA-F]{1,8}";

// IPv6 address RE.
// The format is written as eight groups of four hexadecimal digits,
x:x:x:x:x:x:x:x,
// where x is between 0000-ffff. Zero padding is optional. Case
insensitive.
var ipv6RE = "([\\da-fA-F]{1,4}\\:){7}[\\da-fA-F]{1,4}";

// IPv6/IPv4 Hybrid address RE.
// The format is written as six groups of four hexadecimal digits,
// followed by the 4 dotted decimal IPv4 format. x:x:x:x:x:x:d.d.d.d
var hybridRE = "([\\da-fA-F]{1,4}\\:){6}" +
    "((\\d|[1-9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])\\.){3}(\\d|[1-
9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])";

// Build IP Address RE
var a = [];
if(flags.allowDottedDecimal){ a.push(dottedDecimalRE); }
```

```
if(flags.allowDottedHex){ a.push(dottedHexRE); }
if(flags.allowDottedOctal){ a.push(dottedOctalRE); }
if(flags.allowDecimal){ a.push(decimalRE); }
if(flags.allowHex){ a.push(hexRE); }
if(flags.allowIPv6){ a.push(ipv6RE); }
if(flags.allowHybrid){ a.push(hybridRE); }
var ipAddressRE = "";
if(a.length > 0){
    ipAddressRE = "(" + a.join("|") + ")";
}
return ipAddressRE; // String]]></xe:this.regExpGen>
</xe:djValidationTextBox>
```

Alternatively, if developers already have a prexisting Client-Side JavaScript function to validate the value entered, the **validatorExt** property in the dojo-widget category provides an extension point to call the function. The beauty of this is that developers only need to enter a Client-Side JavaScript object that is a function; the XPage runs the validation in all the events that are appropriate. This speeds up development and minimizes the effort of refactoring.

By default, your validation triggers only when the user has finished editing the field. To trigger validation or other events with each key press, you can set **intermediateChanges** to true. (By default, it is false.)

On top of all this, the **validator** and **validators** properties still exist for core XPages validation. Overall, the Dojo Validation Text Box provides an extremely flexible mechanism for validating the control while maintaining the Dojo look and feel.

Two additional formatting properties are available: **displayMessageExt** and **tooltipPosition**. The **tooltipPosition** property defines the position relative to the field in which any tooltip messages will appear. With the **displayMessageExt** property, a developer can write a Client-Side JavaScript function to override the appearance of the prompts and validation error messages.

What Are Regular Expressions?

For those who are not familiar with the notation, there are websites that can provide standard regular expressions and help you build and test your own. A good starting point is http://www.regular-expressions.info. The zipcode field is a good example of a regular expression in action. `\d{5}` means the field must consist of five characters, all of which are digits. Regular expressions can be simple, as in this example, or extremely complex. The UK postcode is a good example of a particularly complex regular expression, where specific combinations of letters and numbers are allowed:

```
(GIR 0AA)|(((A[BL]|B[ABDHLNRSTX]?|C[ABFHMORTVW]|D[ADEGHLNTY]
|E[HNX]?|F[KY]|G[LUY]?|H[ADGPRSUX]|I[GMPV]|JE|K[ATWY]
|L[ADELNSU]?|M[EKL]?|N[EGNPRW]?|O[LX]|P[AEHLOR]|R[GHM]|S[AEGKL
MNOPRSTY]?|T[ADFNQRSW]|UB|W[ADFNRSV]|YO|ZE)[1-
9]?[0-9]|((E|N|NW|SE|SW|W)1|EC[1-4]|WC[12])[A-HJKMNPR-
Y]|(SW|W)([2-9]|[1-9][0-9])|EC[1-9][0-9]) [0-9][ABD-
HJLNP-UW-Z]{2})
```

If you have a specific format of entry, there's usually a regular expression to validate it.

Table 5.3 summarizes the additional properties of the Dojo Validation Text Box, extending those already covered under the Dojo Text Box.

Table 5.3 `xe:djValidationTextBox` Properties

Property	Description
promptMessage	Enables developers to add a field hint to users when they enter the field.
invalidMessage	Enables a developer to add an error message if any field validation fails. The message will not appear if the only validation applied is required="true".
validatorExt	Holds a Client-Side JavaScript function to extend validation.
regExp	Holds a regular expression with which to validate the value the user entered.
regExpGen	Holds Client-Side JavaScript, which returns a regular expression with which to validate the value the user entered.
displayMessageExt	Holds Client-Side JavaScript to customize the display of Dojo prompt or validation messages.
tooltipPosition	The position relative to the field with which to display any prompt or validation messages.

Dojo Number Text Box, Dojo Currency Text Box (`xe:djNumberTextBox` and `xe:djCurrencyTextBox`)

The Dojo Number Text Box and Dojo Currency Text Box controls extend the Dojo Validation Text Box still further in relation to validating numeric values. All the validation methods we have covered are already available, although the **required** property is virtually redundant, because a blank value is translated to 0 on save. But the power of the Dojo Number Text Box lies in the `xe:djNumberConstraints` extension. It is a complex property comprising a variety of child properties, as can be seen in Figure 5.8. The significant property, as shown, is **type**. This determines the output format from the control, but because of an issue with Dojo, **scientific** is not yet supported. Similarly, the value **currency** and the related properties **currency** and **symbol** are only applicable for the Dojo Currency Text Box.

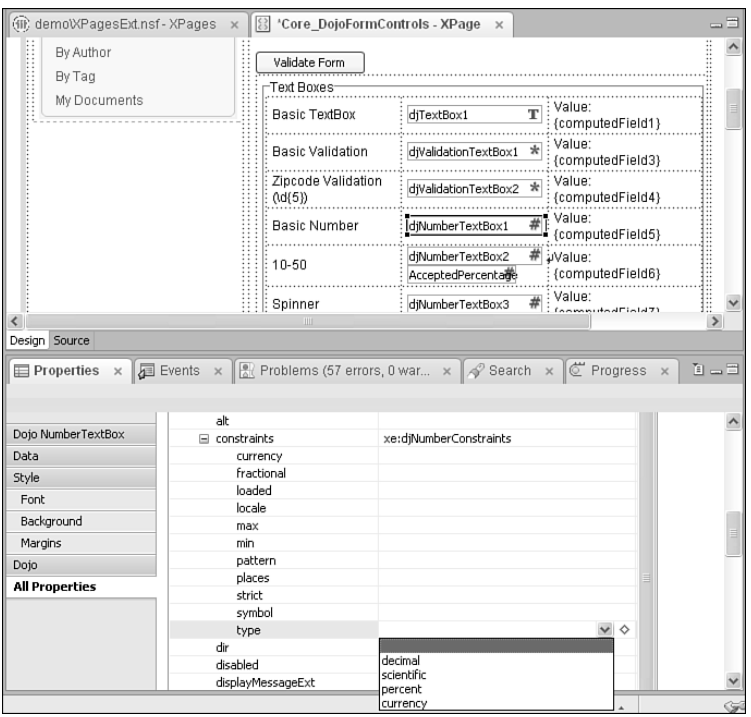


Figure 5.8 `xe:djNumberConstraints`.

The main strength of the `xe:djNumberConstraints` extension, whose properties are shown in Table 5.4, is enforcing appropriate data entry by the user. Percentages can be messy to enforce, handling the percentage sign if users do or do not enter it, manipulating the value for subsequent calculations, and so on. Setting type to **percent** gets around this by ensuring the user

enters a number followed by the percentage sign, such as "50%", which the control then converts to the decimal value "0.5". Likewise, specifying a pattern or places can translate the value entered by the user into an expected format, such as with a certain number of leading zeros or decimal places. With use of **min** and **max,** the entered value can be validated against a range, with an appropriate message defined in the **rangeMessage** property, specific for these controls. See Figure 5.9.

Table 5.4 `xe:djNumberConstraints` Properties

Property	Description
currency	Defines the relevant currency symbol to be applied to the field. The value should be a three-character ISO 4217 currency code, such as GBP. This property relates only to the Dojo Currency Text Box.
fractional	Defines whether to include the fractional portion, for Dojo Currency Text Box only.
locale	The locale to be applied to determine formatting rules for the field's value, one of the extraLocale values loaded in the Dojo config.
max	Defines the maximum value allowed for the field.
min	Defines the minimum value allowed for the field.
pattern	Defines the formatting rule for the field's value, to override any locale-specific formatting.
places	The number of digits to force entry of after the decimal place.
strict	Defines the degree of tolerance allowed to user input; it is false by default. This is more applicable to date/time constraints.
symbol	Defines the currency symbol to be applied to the field, overriding the default currency symbol for the ISO 4217 currency code defined in the currency property. This property relates only to the Dojo Currency Text Box.
type	Defines the type applied to the field: decimal, scientific (not supported), percent, currency (Dojo Currency Text Box only).

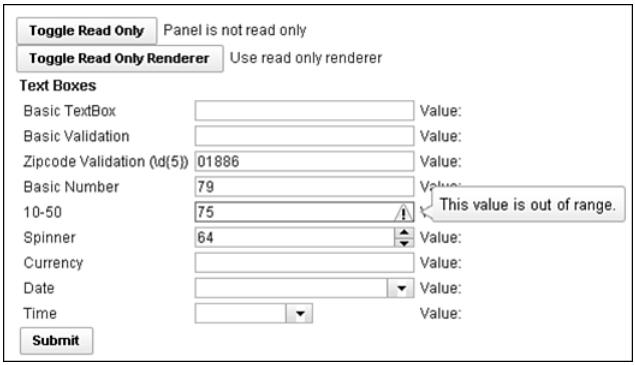


Figure 5.9 Dojo Number Text Box, Dojo Number Spinner, and Dojo Currency Text Box.

The Dojo Number Text Box has one further property that is of particular benefit if the entered value is passed to a managed bean or another Java object. This is the **javaType** property. Anyone who has worked with managed beans will be aware that the value is sometimes handled as a java.util.Long, sometimes as a java.util.Double, but never consistently. It all depends on the value the user enters, which can be annoying. The **javaType** property enables developers to override the type of the value passed to your underlying Java object and ensure it is always an int, always a double, always a float, and so on. Table 5.5 summarizes these additional properties available for the Dojo Number Text Box and Dojo Currency Text Box.

Table 5.5 `xe:djNumberTextBox` and `xe:djCurrencyTextBox` Properties

Property	Description
javaType	Defines the Java number type of the Server-Side value; by default, it is double.
rangeMessage	Defines the validation message to show if the value entered is outside the minimum and maximum bounds.

Dojo Number Spinner (`xe:djNumberSpinner`)

The Dojo Number Spinner allows the user to either type in a number or scroll up and down through the range with the keyboard or the buttons provided on the right edge of the control. This control is an implementation of `dijit.form.NumberSpinner` and an extension of the Dojo Number Text Box with all the properties applicable to that control (so currency-related properties of the `xe:djNumberConstraints` extension are not applicable). The control provides two properties for managing the incremental steps of the spinner: **smallDelta** and **largeDelta**. By default, the implicit increments are 1 and 10 respectively, but this can be overridden as required. The **smallDelta** increment is used when the user clicks the buttons provided or uses the cursor up

and down keys. To take advantage of the **largeDelta** increment, users need to click the **Page Up** or **Page Down** keys.

If you hold down one of the buttons or keys, the increments are repeated after half a second and subsequently applied quicker and quicker. The **defaultTimeout** property, expecting an integer in milliseconds, determines how long the user needs to hold down the key before the increment is repeated; by default, it is 500 milliseconds. You configure the degree to which the increments are sped up using the **timeoutChangeRate** property. Because this is 0.9, the increments are applied progressively quicker the longer the key or button is held down, until the maximum speed is reached. If you set it at 1.0, the increments are always applied at the same time interval, never increasing. A value of greater than 1.0 has no effect.

Table 5.6 summarizes the properties of the Dojo Number Spinner control.

Table 5.6 `xe:djNumberSpinner` Properties

Property	Description
defaultTimeout	Allows the developer to control the number of milliseconds the user needs to hold down the key before it becomes typematic, or auto-incrementing.
timeoutChangeRate	Defines how much quicker each typematic event occurs.
largeDelta	Defines the increment when the Page Up and Page Down buttons are pressed.
smallDelta	Defines the increment when the cursor Up and Down buttons are pressed.

Dojo Date Text Box and Dojo Time Text Box (`xe:djDateTextBox` and `xe:djTimeTextBox`)

The Dojo Date Text Box and Dojo Time Text Box controls extend the Dojo Validation Text Box control. However, like the Dojo Number Text Box, Dojo Currency Text Box, and Dojo Number Spinner, they have their own **constraints complex** property. For the Dojo Date Text Box and Dojo Time Text Box, the **constraints complex** property implements the `xe:djDateTimeConstraints` extension, as detailed in Table 5.7 and illustrated in Figure 5.10.

Table 5.7 `xe:djDateTimeConstraints` Properties

Property	Description
am	Allows the developer to override the "am" abbreviation for A.M. times. This is only applicable to the Dojo Time Text Box and only where **timePattern** is specified and uses the AM/PM portion (for example, **timePattern** is "h:mm a").
clickableIncrement	Defines the clickable increment of the Time Picker and is applicable only to the Dojo Time Text Box. The value is entered in the format Thh:mm:ss.
datePattern	Defines the date pattern and overrides any setting in the **formatLength** property. Date patterns are in accordance with Unicode Technical Standard 35 Date Format Patterns, such as dd-MM-yy.
formatLength	Defines the date or time format. Available options are long, short, medium, and full.
locale	The locale to be applied to determine formatting rules for the field's value, one of the **extraLocale** values loaded in the Dojo config.
pm	Allows the developer to override the "pm" abbreviation for P.M. times. This is only applicable to the Dojo Time Text Box and only where **timePattern** is specified and uses the AM/PM portion (for example, **timePattern** is "h:mm a").
selector	Defines the selector, either date or time.
strict	Defines the degree of tolerance allowed to user input; it is false by default.
timePattern	Defines the time pattern and overrides any setting in the **formatLength** property. Time patterns are in accordance with Unicode Technical Standard 35 Date Format Patterns, such as hh:mm a.
visibleIncrement	Defines the visible increment of the Time Picker and is applicable only to the Dojo Time Text Box. The value is entered in format Thh:mm:ss.
visibleRange	Defines the visible range of the Time Picker and is applicable only to the Dojo Time Text Box. The value is entered in the format Thh:mm:ss.

The main one for the Dojo Date Text Box is the **datePattern** property, which allows developers to define the format of the date presented to the user in the Dojo Date Text Box. For example, dd-MM-yyyy overrides the locale format to show 16th June 2011 as 16-06-2011, and dd MMM yyyy shows as 16 Jun 2011. Alternatively, the **formatLength** property can be used to choose one of four predefined date or time formats. If both are used, the **datePattern** property takes precedence.

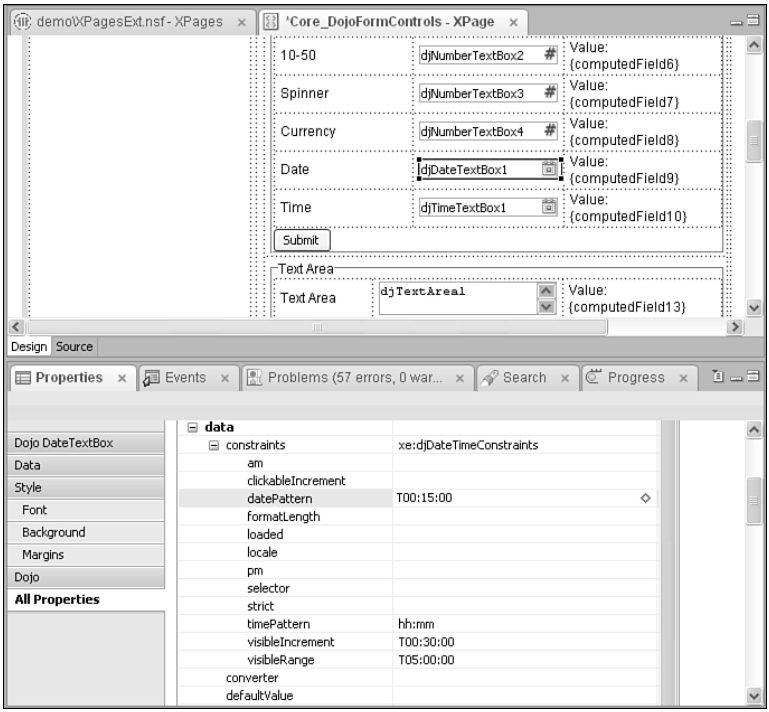


Figure 5.10 `xe:djDateTimeConstraints`.

The Dojo Time Text Box control also uses the **xe:djDateTimeConstraints** property. But unlike the Dojo Date Text Box, properties are surfaced to allow the developer to manage the display of the control. To control how many hours are shown, you can define the **visibleRange** property. The **visibleIncrement** property defines the labels presented to the user, and the **clickableIncrement** property defines the increment for each value the user can select. You define each property using the format THH:mm:ss, so a **visibleIncrement** of 30 minutes is T00:30:00, as in Figure 5.11. With **datePattern** for the Dojo Date Text Box, the **timePattern** property defines the format for the times displayed to the user and presented in the field. Therefore, a format of h:mm presents, for example, 9:00, 9:30, and so on.

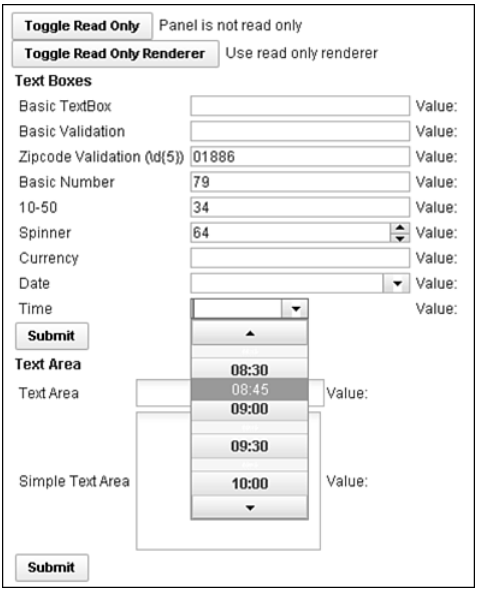


Figure 5.11 Time Picker.

Dojo Extensions to the Multiline Edit Box Control

There are two Dojo controls in the Extension Library that extend the Multiline Edit Box: the Dojo Text Area (`xe:djTextarea`) and the Dojo Simple Text Area (`xe:djSimpleTextarea`). One of the advantages of these controls is that they also have some of the string manipulation properties familiar from the Dojo extensions that are based on the Edit Box controls. So **trim**, **propercase**, **lowercase,** and **uppercase** are implemented, which makes it easy to manipulate the content as soon as the user leaves the field. There is no built-in Dojo functionality to validate the Dojo Text Area control, but you can utilize all the core XPages validation techniques.

One of the strengths of XPages is that you can present and edit a collection of documents in the same web page. However, the challenge for a developer is that, unless the user is editing a small document such as a Comments document, the editable form can take up a large amount of real estate. If that includes the Multiline Edit Box as well, it takes up even more real estate when **rows** and **cols** properties are defined. But the beauty of the Dojo Text Area control is that it is auto-expanding. This means it takes up less screen real estate while still expanding as much as is required to show the user all the content. The Dojo Simple Text Area control, however, is fixed size. Of course, size attributes can be computed using Server-Side JavaScript, just as they can for any other XPages properties.

As with the Multiline Edit Box, you can define the width of the field using the **rows** property or using CSS to specify the width. Of course, because the Dojo Text Area is auto-expanding, the **rows** property has no effect for that control, only for the Dojo Simple Text Area control.

Table 5.8 details two additional properties of the Dojo Text Area and Dojo Simple Text Area.

Table 5.8 `xe:djTextArea` and `xe:djSimpleTextArea` Properties

Property	Description
rows	Defines the number of rows the text area will show. This property is applicable only to the Dojo Simple Text Area control.
cols	Defines the number of columns the text area will show.

Dojo Extensions to the Select Control

As with the other input controls, the Dojo modules for selecting values have been included in the Extension Library. Besides the Dojo Radio Button (`xe:djRadioButton`) and Dojo Check Box (`xe:djCheckBox`) controls, there are two Dojo versions of the core Combo Box control: the Dojo Combo Box (`xe:djComboBox`) and Dojo Filtering Select (`xe:djFilteringSelect`).

The core Combo Box control is good for ensuring that users select from a restricted list of options, but it does not allow type-ahead. The Edit Box control offers this kind of type-ahead functionality, but it does not force the user to select one of the options provided. The benefit of the Dojo Combo Box and Dojo Filtering Select controls in the Extension Library is that they combine the type-ahead and restrict the user to just the options available. The sole difference between the two is that the Dojo Combo Box control holds a list only of values, whereas the Dojo Filtering Select control holds a list of label/value pairs.

Dojo Combo Box and Dojo Filtering Select (`xe:djComboBox` and `xe:djFilteringSelect`)

Developers who are more familior with dojo.data stores such as the ItemFileReadStore can take advantage of the **store** property and reference a JavaScript store. This is just a JSON object returning a collection of items that could be returned by an XAgent or some other API to return a JSON object. However, if the source data has been provided by a third party, it might not return a name attribute for the Dojo Combo Box to search. In this situation, the **searchAttr** property can be used to specify a different attribute in the JSON object on which to search. By default, any search, whether against defined items or against a dojo.data store, is case insensitive, but you can enforce case sensitivity by setting the **ignoreCase** property to true.

By default, whether querying a coded list of options or a dojo.data store, a **starts with** query will be performed. That is, the only results returned will be those that start with the letter or letters. Sometimes developers might prefer to query the store differently; Dojo provides this functionality. There are three expressions to be used for **starts with** searches, **contains** searches,

and **exact match** searches. However, the expressions use the phrase `"${"`, which has a specific meaning to the XSP Command Manager, so the easiest method of entering the expressions is using Server-Side JavaScript. The three variants are included in Listing 5.7, Listing 5.8, and Listing 5.9.

Listing 5.7 Contains Search Expression

```
<xe:djComboBox
   id="djComboBox2"
   value="#{sessionScope.djComboBox1}"
   tooltipPosition="before"
   title="This is a comboBox" pageSize="2">

<xe:this.queryExpr><![CDATA[${javascript:"*${0}*"}]]></xe:this.queryEx
pr>
   <xp:selectItem
      itemLabel="Ford"
      itemValue="ford">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Toyota"
      itemValue="toyota">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Renault"
      itemValue="renault">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Mercedes"
      itemValue="mercedes">
   </xp:selectItem>
</xe:djComboBox>
```

Listing 5.8 Exact Match Search Expression

```
<xe:djComboBox
   id="djComboBox2"
   value="#{sessionScope.djComboBox1}"
   tooltipPosition="before"
   title="This is a comboBox"
   pageSize="2">
```

Listing 5.8 (Continued)

```
<xe:this.queryExpr><![CDATA[${javascript:"$\{0}"}]]></xe:this.queryExpr
>
   <xp:selectItem
      itemLabel="Ford"
      itemValue="ford">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Toyota"
      itemValue="toyota">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Renault"
      itemValue="renault">
</xp:selectItem>
   <xp:selectItem
      itemLabel="Mercedes"
      itemValue="mercedes">
   </xp:selectItem>
</xe:djComboBox>
```

Listing 5.9 Starts with Search Expression

```
<xe:djComboBox
   id="djComboBox2"
   value="#{sessionScope.djComboBox1}"
   tooltipPosition="before"
   title="This is a comboBox" pageSize="2">

<xe:this.queryExpr><![CDATA[${javascript:"*$\{0}"}]]></xe:this.queryExp
r>
   <xp:selectItem
      itemLabel="Ford"
      itemValue="ford">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Toyota"
      itemValue="toyota">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Renault"
```

```
      itemValue="renault">
   </xp:selectItem>
   <xp:selectItem
      itemLabel="Mercedes"
      itemValue="mercedes">
   </xp:selectItem>
</xe:djComboBox>
```

To ease selection, a number of properties are available. The **pageSize** property allows you to define some entries that the drop-down box should show. If the query returns more entries, a link is added to allow the user to page down and page up through the available options, as shown in Figure 5.12 and Figure 5.13. This property doesn't enhance performance by minimizing the number of options delivered to the browser, but you can use it to enhance presentation. As with the Dojo Number Spinner control, it is also possible to manage the response to the selection. In this case, the **searchDelay** property allows you to set the number of milliseconds delay before matching results are returned.

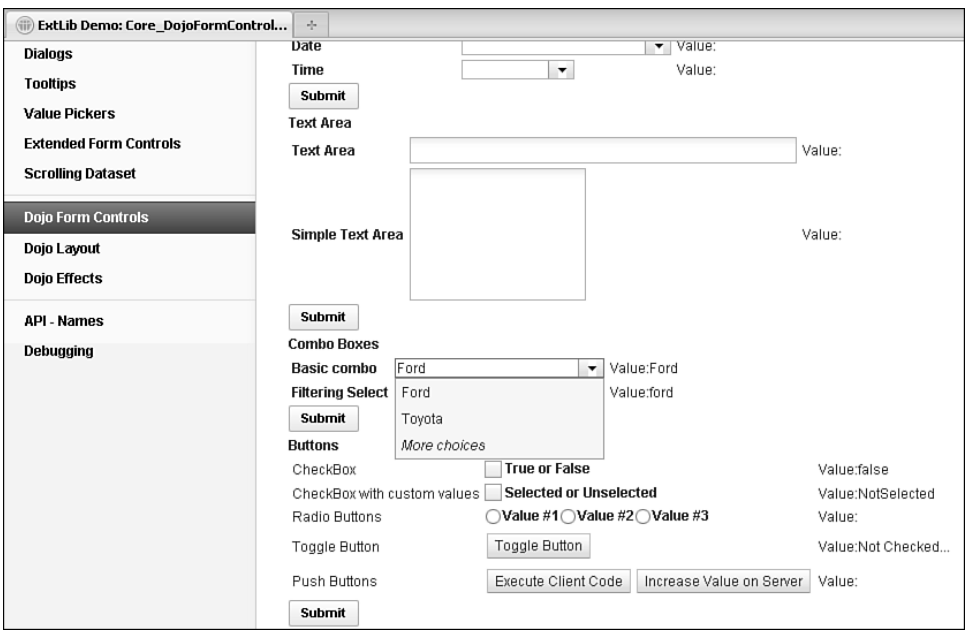


Figure 5.12 More choices on Dojo Combo Box.

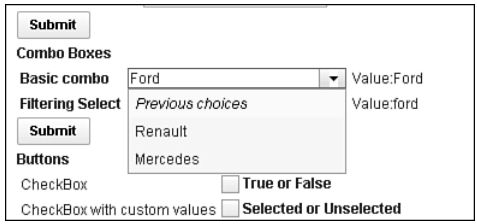


Figure 5.13 Previous choices on Dojo Combo Box.

Because the Dojo Filtering Select uses label/value pairs and the Dojo Combo Box uses just a list of values, Dojo Filtering Select takes advantage of two additional properties and an event to handle the labels displayed. The first is **labelType**. By default, the labels are treated as plain text, but by setting this property to **html,** the labels are treated as HTML. The second is **labelAttr**, applicable for developers using a datastore. As with the **searchAttr** property, you can use this with a Dojo datastore to tell the Dojo Filtering Select to display labels from the store based on an attribute other than `name`. This does not affect the attribute from the store that is used to search on as the user types. To do that, you need to define the **searchAttr** property as well. An additional event is available on the Dojo Filtering Select called **labelFunc**. This triggers on selection of a valid entry and can trigger either Client-Side or Server-Side JavaScript.

Chapter 11, "REST Services," covers REST services and other data integration, so at this point only a brief example of this functionality is shown in Listing 5.10. Lines 1 to 22 cover the REST service. Note that the jsId defined for the service in line 3 is allocated to the djFiltering-Select in line 26. In line 27, the FilteringSelect shows a list of U.S. states using the **labelAttr** property, but searches on the two-character abbreviation using the **searchAttr** property. The results are limited to 10 per page using the **pageSize** property in line 29.

Listing 5.10 Dojo Filtering Select Using DataStore

```
1  <xe:restService
2  id="restService1"
3  jsId="stateStore">
4     <xe:this.service>
5        <xe:viewItemFileService
6           viewName="AllStates"
7           defaultColumns="true"
8           dojoType="dojo.data.ItemFileReadStore"
9           count="400">
10          <xe:this.columns>
11             <xe:restViewColumn
12                columnName="Name"
13                name="Name">
14             </xe:restViewColumn>
15             <xe:restViewColumn
```

```
16                columnName="Key"
17                name="Key">
18             </xe:restViewColumn>
19          </xe:this.columns>
20       </xe:viewItemFileService>
21    </xe:this.service>
22 </xe:restService>
23 <xe:djFilteringSelect
24    id="djComboBox3"
25    value="#{sessionScope.djComboBox2}"
26    store="stateStore"
27    labelAttr="Name"
28    searchAttr="Key"
29    pageSize="10">
30 </xe:djFilteringSelect>
```

Table 5.9 details the noteworthy properties of the Dojo Combo Box and Dojo Filtering Select.

Table 5.9 `xe:djComboBox` and `xe:djFilteringSelect` Properties

Property	Description
hasArrow	Defines whether a drop-down arrow appears beside the field, to show selections.
ignoreCase	Defines whether the search of the store is case-sensitive.
queryExpr	Defines a query for the way the store is searched, as a "starts with", "contains", or "exact match". For terminology, see Listing 5.7, Listing 5.8, and Listing 5.9.
searchAttr	Defines the attribute in the Dojo datastore to search on; by default, it is `name`.
searchDelay	Defines the number of milliseconds to delay before beginning the search.
pageSize	Allows the developer to specify the number of entries to show on each page of the search results.
store	Allows the developer to define a Dojo datastore from which to take the options for the Dojo Combo Box or Dojo Filtering Select.
labelAttr	Defines the attribute in the Dojo datastore from which to retrieve the label. If no property is defined, the attribute in the **searchAttr** property is used. This property is available only for the Dojo Filtering Select.
labelFunc	Defines an event handler to be called when the label changes, returning the label to be displayed. This property is available only for the Dojo Filtering Select.
labelType	Defines whether the label is plain text or HTML. This property is available only for the Dojo Filtering Select.

Dojo Check Box and Dojo Radio Button

The primary intention of the Dojo Check Box and Dojo Radio Button controls is to style the controls appropriate for other Dojo controls. Both controls support the same functionality as the core control versions, so you can assign them to a group with custom values defined. The main difference with the Radio Button Group or Check Box Group is that the core controls for groups display their options in a table within a fieldset. The Dojo Check Box and Dojo Radio Button controls display options inline. In addition to this standard functionality and similarity to the other Dojo controls, the Dojo Check Box and Dojo Radio Button are enabled for accessibility. So the **title** property and the WAI-related properties can be defined, as can any of the other Dojo controls.

Dojo Extensions to Buttons

There are two Dojo Extensions to Buttons: the Dojo Button control and the Dojo Toggle Button control. Like the Dojo Check Box and Dojo Radio Button controls, the Dojo Button is not appreciably different from the core control version. Again, the main differences are the Dojo styling and the inclusion of properties for accessibility, the same ones covered earlier. Just like the core Button control, the Dojo Button control can have a label, show an icon, or both. The **label** property allows the developer to control the text to show, but the **showLabel** property can suppress the label from appearing. However, showing an icon is not merely a case of selecting an image. CSS handles the icon, with the relevant class defined as a string in the **iconClass** property. Dojo has some built-in icons for various editing functions, defined in the **<dojoroot>\dijit\themes** folder and shown in Listing 5.11. Line 4 shows the Dojo theme classes `dijitEditorIcon` and `dijitEditorIconCut` applied to the button. The former loads a sprite (a collection of images, held in a single file to minimize calls to the server), and the latter positions the sprite to show a specific image—in this case, the **Cut** icon. Line 15 applies an icon to a second button, this time using a CSS class. Listing 5.12 shows the stylesheet that loads an image from the **icons** folder on the server. Note that because this is a stylesheet, it is loaded using the HTTP server, not the XSP Command Manager, so standard Domino web URL syntax applies rather than ***/.ibmxspres/.…***. You can see the buttons produced in Figure 5.14. If multiple images from the **icons** folder are to be included in the application, using a sprite would be the recommended approach.

Listing 5.11 Dojo Button Icons

```
1  <xe:djButton
2  id="djButton2"
3  label="Execute Client Code"
4     iconClass="dijitEditorIcon dijitEditorIconCut">
```

```
5     <xp:eventHandler
6        event="onClick"
7        submit="false">
8        <xp:this.script><![CDATA[alert("You clicked me,
#{javascript:@UserName()}!")]]></xp:this.script>
9     </xp:eventHandler>
10 </xe:djButton>
11 <xe:djButton
12    id="djButton3"
13    showLabel="false"
14    label="Increase Value on Server"
15    iconClass="testIcon">
16    <xp:eventHandler
17       event="onClick"
18       submit="true"
19       refreshMode="partial"
20       refreshId="computedField19">
21       <xp:this.action><![CDATA[#{javascript:if
(sessionScope.djButton4) {
22    sessionScope.djButton4+=1
23 } else {
24    sessionScope.djButton4 = 1
25 }}]]></xp:this.action>
26    </xp:eventHandler>
27 </xe:djButton>
```

Listing 5.12 `testIcon` Class

```
.testIcon {
   background-image: url(/icons/actn010.gif); /* editor icons sprite
image */
   background-repeat: no-repeat;
   width: 18px;
   height: 18px;
   text-align: center;
}
```

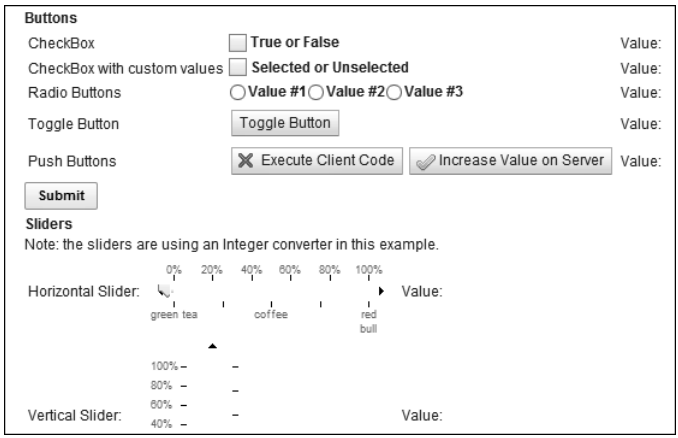


Figure 5.14 Dojo buttons.

Dojo Toggle Button Control

The Dojo Toggle Button is a control that is new to developers who are not familiar with Dojo. The control is similar to the Dojo Check Box control but is styled like the Button control. Like the Dojo Check Box, it can be bound to a datasource, with a value set when the button is unclicked and a different value set when the button is clicked. From inspecting the source HTML produced for the Dojo Toggle Button control, it becomes apparent that the Dojo Toggle Button consists of a button with a dojoType and a hidden input field, as shown in Figure 5.15—a similar technique to the way developers have built the kind of functionality the Dojo Toggle Button provides. Not surprisingly, when the user clicks the Dojo Toggle Button, a value is set into the hidden field. The toggle effect runs Client-Side, although Server-Side events can also be triggered. The hidden field has the same ID as the button, except that it is suffixed with **_field**. The value of the hidden field is not the **checkedValue** or **uncheckedValue** properties, but an empty string if unchecked or **on** if checked.

Figure 5.15 Dojo Button HTML.

By default, as with the Dojo Check Box, the values are false when unclicked and true when clicked. But you can override these values by defining the **checkedValue** and **uncheckedValue** properties, the property names highlighting that this is an extension of the Dojo Check Box control. The only downside is that the styling of the toggle button does not change depending on whether the button is clicked or unclicked. But with the understanding of the HTML produced by the control, it is a simple matter to add that functionality as in Listing 5.13. Lines 8 to 20 add an onChange `xp:eventHandler`_to the control. Note that this has to be defined as an `xp:eventHandler` rather than the default `xe:eventHandler`, which does not exist. Line 11 loads the Client-Side ID of the button into a variable. Line 12 gets the button itself using `dojo.byId()` because of the classneeds setting, not a dojoAttribute. Lines 13 and 14 get the field and test whether the value is **on**. Lines 15 and 17 then set the class of the button.

Listing 5.13 Styling the ToggleButton Control

```
1  <xe:djToggleButton
2      id="djToggleButton1"
3      title="Toggle Button"
4      value="#{sessionScope.djButton3}"
```

Listing 5.13 (Continued)

```
5     label="Toggle Button"
6     checkedValue="Checked..."
7     uncheckedValue="Not Checked...">
8     <xp:eventHandler
9        event="onChange"
10       submit="false">
11       <xe:this.script><![CDATA[var id="#{id:djToggleButton1}";
12 var btn=dojo.byId(id);
13 var field = dojo.byId(id+"_field");
14 if (field.value == "on") {
15    btn.setAttribute("class","btnRed");
16 } else {
17    btn.setAttribute("class","btnGreen");
18 }
19 ]]></xe:this.script>
20    </xp:eventHandler>
21 </xe:djToggleButton>
```

Listing 5.14 shows the CSS for the classes.

Listing 5.14 `btnRed` and `btnGreen` Classes

```
.btnRed {
   color: rgb(255,0,0);
}

.btnGreen {
   color: rgb(0,255,0);
}
```

Composite Dojo Extensions

Some extension controls are available under the Dojo category that do not fit into the previous categories. Rather than extending core controls available, these controls add new functionality not previously available as controls in XPages.

As Listing 5.3 shows, the `dijit.form.HorizontalSlider` requires multiple HTML elements. In the same way, some of the Dojo controls are more complex. Sliders comprise multiple components for their implementation, whereas the Dojo Link Select and Dojo Image Select controls have complex properties to define the values.

Sliders

The beginning of this chapter covered adding a slider with traditional Dojo. The code was covered in Listing 5.4, where the slider comprised a div with an ordered list of labels and an onchange event passing the value to a hidden field via Client-Side JavaScript. The sliders in the Extension Library remove the necessity to use a div with an onChange event to store the value. Rather, the sliders themselves are bound directly to the field.

There are two types of sliders, the **Dojo Horizontal Slider** (`xe:djHorizontalSlider`) and the **Dojo Vertical Slider** (`xe:djVerticalSlider`), as Figure 5.16 shows. Although the properties for both are identical and shown in Table 5.10, you need to choose the relevant slider at development time.

Table 5.10 `xe:djHorizontalSlider` and `xe:djVerticalSlider` Properties

Property	Description
clickSelect	Defines whether the user can change the value by clicking on a position on the bar in addition to dragging the slider.
discreteValues	Defines the number of discrete values between the minimum and maximum values.
maximum	Defines the maximum value for the slider.
minimum	Defines the minimum value for the slider.
pageIncrement	Defines the number of increments applied to the slider when the user clicks the Page Up or Page Down button.
showButtons	Defines whether buttons are shown to move the slider.
slideDuration	Defines the number of milliseconds it takes to move the slider from 0% to 100%; it is 1000 milliseconds by default.

The values of the slider are controlled by four properties: **defaultValue** defines the initial starting value (if the field the control is bound to does not already have a value), whereas **minimum** and **maximum** define the bounds of the slider, and **discreteValues** defines the number of steps between the minimum and maximum. By default, whenever the user clicks on a part of the slider, that value is selected, and this is controlled by the **clickSelect** property. If set to `false`, this functionality is suppressed. Also, by default, there are buttons on either end of the slider for moving the current position. Again, these can be suppressed by setting the **showButtons** property to `false`.

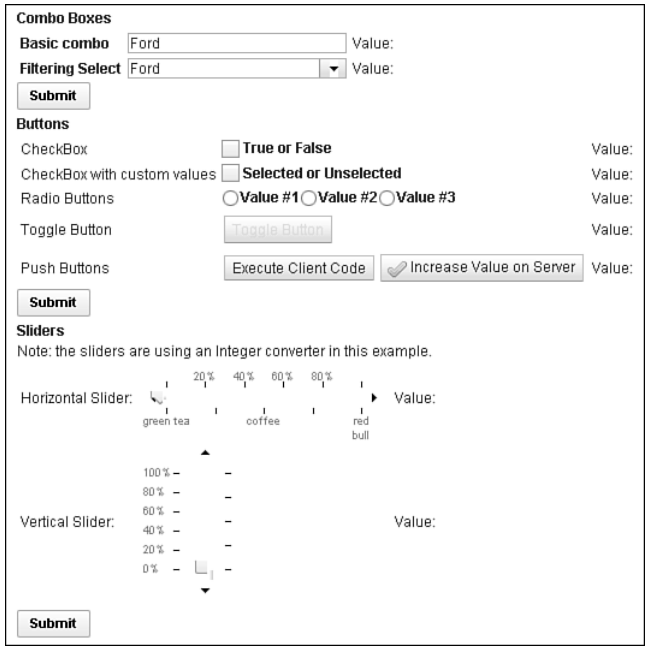


Figure 5.16 Sliders.

Besides clicking on a position of the slider or using the buttons, you can use keyboard shortcuts to control the movement, like you did for the spinner controls. All four cursor keys can be used for both sliders: left (←) and down (↓) moving in one direction, right (→) and up (↑) moving in the other direction. Although the cursor keys can be used to increment in small amounts, **Page Up** and **Page Down** increment in larger amounts. The smaller increment is always one step on the slider, but the developer can override the larger increment—by default 2 steps—using the **pageIncrement** property. Furthermore, because the speed of increment could be controlled for the spinners, it can also be controlled for the sliders, by means of the **slideDuration** property. This is a value in milliseconds that the slider will take to move from one end of the slider to the other; by default, it is one second.

As with the traditional Dojo implementation, you can add labels. This comprises two further controls: the **Dojo Slider Rule** (`xe:djSliderRule`) for the markers and the **Dojo Slider Rule Labels** (`xe:djSliderRuleLabels`) for the actual labels. For both controls, two properties determine how many and where the rules appear: **count** and **container**. The **container** provides a ComboBox list of options, with all four options available regardless: `topDecoration`, `leftDecoration`, `bottomDecoration`, and `rightDecoration`. Obviously, you must choose the relevant container for the relevant slider; `rightDecoration` and `leftDecoration` are not applicable for the Dojo Horizontal Slider.

You can map styling to CSS classes for both controls. You can style the markers by using the **ruleStyle** property on the Dojo Slider Rule, whereas you can style the labels by using the **labelStyle** property on the Dojo Slider Rule Labels.

You can set a number of additional properties for the Dojo Slider Rule Labels. The **minimum** and **maximum** properties set the top and bottom level for the labels, and **numericMargin** can define how many labels to omit at either end of the label list. So setting the value to 1 omits 0% and 100% from a default Dojo Slider Rule Labels control. As this suggests, the default labels are percentages, running from 0% to 100%. But you can override this in two ways. You can pass an array of labels into the **labels** property or use the **labelList** property, as shown in Listing 5.15. This method is recommended over <li> tags because it supports localization.

Listing 5.15 Dojo Horizontal Slider

```
<xe:djHorizontalSlider
   id="djHorizontalSlider2"
   value="#{sessionScope.djSlider1}"
   maximum="100"
   minimum="0"
   style="margin: 5px;width:200px; height: 20px;"
   discreteValues="10"
   pageIncrement="3">
   <xp:this.converter>
      <xp:convertNumber
         integerOnly="true">
      </xp:convertNumber>
   </xp:this.converter>
   <xe:djSliderRuleLabels
      id="djSliderRuleLabels2"
      container="topDecoration"
      style="height:10px;font-size:75%;color:gray;"
      count="6"
      numericMargin="1">
   </xe:djSliderRuleLabels>
   <xe:djSliderRule
      id="djSliderRule5"
      container="topDecoration"
      style="height:5px;" count="6">
   </xe:djSliderRule>
   <xe:djSliderRule
      id="djSliderRule6"
      style="height:5px;"
      count="5"
      container="bottomDecoration">
```

Listing 5.15 (Continued)

```
   </xe:djSliderRule>
   <xe:djSliderRuleLabels
      id="djSliderRuleLabels5"
      container="bottomDecoration"
      style="height:10px;font-size:75%;color:gray;">
<xe:this.labelsList>
         <xe:djSliderRuleLabel
            label="green tea">
         </xe:djSliderRuleLabel>
         <xe:djSliderRuleLabel
            label="coffee">
         </xe:djSliderRuleLabel>
         <xe:djSliderRuleLabel
            label="red bull">
         </xe:djSliderRuleLabel>
      </xe:this.labelsList>   </xe:djSliderRuleLabels>
</xe:djHorizontalSlider>
```

Table 5.11 shows the properties for the Dojo Slider Rule and Dojo Slider Rule Labels.

Table 5.11 `xe:djSliderRule` and `xe:djSliderRuleLabels` Properties

Property	Description
count	Defines how many markers or labels should appear.
labels	Allows the developer to write a Client-Side JavaScript expression to define the labels. This property is available only for the Dojo Slider Rule Labels.
labelsList	Allows the developer to define a localizable set of labels. This property is available only for the Dojo Slider Rule Labels.
maximum	Defines the maximum position for the labels. This property is available only for the Dojo Slider Rule Labels.
minimum	Defines the minimum position for the labels. This property is available only for the Dojo Slider Rule Labels.
numericMargin	Defines the number of labels to omit from either end of the label list. This property is available only for the Dojo Slider Rule Labels.
container	Defines where in relation to the slider line the markers or labels should appear.
ruleStyle	Defines the styling for the markers.
labelStyle	Defines the styling for the labels and is available only for Dojo Slider Rule Labels.

Dojo Link Select (`xe:djLinkSelect`)

The Dojo Link Select control allows developers to group link options so that when one link is selected, the others are deselected. You can see this in action with the filter area of the All Documents page on the TeamRoom database. Here, for example, selecting All by Date not only selects that entry but deselects the default All link. Unlike the traditional link functionality, you can bind the Link Select to a field or scoped variable. In addition, you can trigger a wealth of events from the Link Select.

Despite having properties **multipleTrim** and **multipleSeparator**, the control allows only one value to be selected at any one time. You can define the available options in a number of ways. The All Documents page (**allDocumentsFilter.xsp** custom control) uses **selectItem** controls, but you can also use a selectItems control. As with the ComboBox and FilteringSelect controls covered earlier, there is currently no mechanism to add an `xp:selectItem` or `xp:selectItems` control from the palette. So you can use the core ComboBox or ListBox control to define the values; then you can cut and paste the code across from the core control to the Dojo control.

Alternatively, there are three dataProviders available. Those who are comfortable with Java may choose to use the beanValuePicker. The other options are the simpleValuePicker and the dominoViewValuePicker. The simpleValuePicker allows a developer to define a list of options as a string of label value pairs. The label values themselves are defined in the **valueList** property. You can define the separator between the label and the value using the **labelSeparator** property, and you can define the separator between values using the **valueListSeparator** property. The dominoViewValuePicker allows you to select the options from a view, by defining the **databaseName** and **viewName** properties. The **labelColumn** property defines the column from which the values will be picked. The value set when the label is clicked is pulled from the first column in the view. So Listing 5.16 shows a Dojo Link Select where the options are pulled from the AllStates view, showing the Names column. Figure 5.17 shows the resulting output. As you can see, the onChange event refreshes the computed field with the value whenever you select a new link.

Listing 5.16 Link Select Control with `dominoViewValuePicker`

```
<xe:djextLinkSelect
   id="djextLinkSelect2"
   defaultValue="MA"
   value="#{viewScope.link3}">
   <xe:this.dataProvider>
      <xe:dominoViewValuePicker
         viewName="AllStates"
         labelColumn="Name">
      </xe:dominoViewValuePicker>
   </xe:this.dataProvider>
   <xp:eventHandler
```

Listing 5.16 (Continued)

```
        event="onChange"
        submit="true"
        refreshMode="partial"
        refreshId="computedField3">
    </xp:eventHandler>
</xe:djextLinkSelect>
```

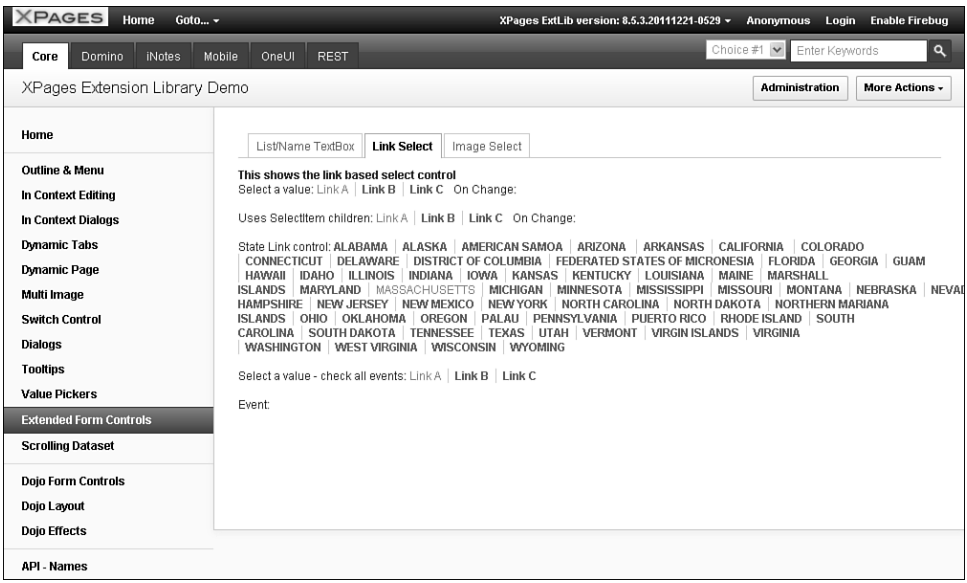


Figure 5.17 Link Select with `dominoViewValuePicker`.

Table 5.12 shows the pertinent properties for the Dojo Link Select control.

Table 5.12 `xe:djLinkSelect` Properties

Property	Description
dataProvider	Provides the options for the Dojo Link Select as an `xe:simpleValuePicker`, `xe:dominoViewValuePicker`, or `xe:beanValuePicker`.
firstItemStyle	Defines styling for the first link.
firstItemStyleClass	Defines the class to be applied to the first link.
itemStyle	Defines styling for the intermediate links.
itemStyleClass	Defines the class to be applied to the intermediate links.
lastItemStyle	Defines styling for the last link.
lastItemStyleClass	Defines the class to be applied to the last link.

Dojo Image Select

The Dojo Image Select control is similar to the Link Select in that it provides a group of links, or in this case images, only one of which can be selected. Again, it is bound to a field or scoped variable, with a default value that can be set. The images are defined using **selectImage** child controls of the **imageValues** property. Each **selectImage** has **image** and **selectedImage** properties, to define the images that appear when the link is deselected or selected. The **selectedValue** property defines the value that will be set when the image is clicked. In addition, properties are available for styling each image, both in its deselected state and its selected state. The example on the **Core_FormControl.xsp** XPage in the Extension Library Demo database, reproduced in Listing 5.17 and shown in Figure 5.18, shows buttons appropriate for a Calendar View control, although, as will be shown in Chapter 7, a slightly different method is used for the calendar view in the TeamRoom database.

Listing 5.17 Dojo Image Select for Calendar Picker

```
<xe:djextImageSelect
   id="djextImageSelect1"
   title="Select a value default is two days"
   value="#{viewScope.image1}"
   defaultValue="T">
   <xe:this.imageValues>
      <xe:selectImage
         selectedValue="D"

selectedImage="/.ibmxspres/.extlib/icons/calendar/1_Day_selected_24.
gif"

image="/.ibmxspres/.extlib/icons/calendar/1_Day_deselected_24.gif"
```

Listing 5.17 (Continued)

```
         imageAlt="One Day">
      </xe:selectImage>
      <xe:selectImage
         selectedValue="T"

selectedImage="/.ibmxspres/.extlib/icons/calendar/2_Days_selected_24.
gif"

image="/.ibmxspres/.extlib/icons/calendar/2_Days_deselected_24.gif"
         imageAlt="Two Days">
      </xe:selectImage>
      <xe:selectImage
         selectedValue="F"

selectedImage="/.ibmxspres/.extlib/icons/calendar/1_Work_Week_selected_
24.gif"

image="/.ibmxspres/.extlib/icons/calendar/1_Work_Week_deselected_24.gif"
         imageAlt="One Work Week">
      </xe:selectImage>
      <xe:selectImage
         selectedValue="W"

selectedImage="/.ibmxspres/.extlib/icons/calendar/1_Week_selected_24.
gif"

image="/.ibmxspres/.extlib/icons/calendar/1_Week_deselected_24.gif"
         imageAlt="One Week">
      </xe:selectImage>
      <xe:selectImage
         selectedValue="2"

selectedImage="/.ibmxspres/.extlib/icons/calendar/2_Weeks_selected_24.
gif"

image="/.ibmxspres/.extlib/icons/calendar/2_Weeks_deselected_24.gif"
         imageAlt="Two Weeks">
      </xe:selectImage>
      <xe:selectImage
         selectedValue="M"

selectedImage="/.ibmxspres/.extlib/icons/calendar/Month_selected_24.
gif"
```

```
image="/.ibmxspres/.extlib/icons/calendar/Month_deselected_24.gif"
         imageAlt="One Month">
      </xe:selectImage>
      <xe:selectImage
         selectedValue="Y"

selectedImage="/.ibmxspres/.extlib/icons/calendar/All_Entries_selected_
24.gif"

image="/.ibmxspres/.extlib/icons/calendar/All_Entries_deselected_24.gif
"
         imageAlt="All Entries">
      </xe:selectImage>
   </xe:this.imageValues>
   <xp:eventHandler
      event="onClick"
      submit="true"
      refreshMode="partial"
      refreshId="computedField3">
   </xp:eventHandler>
</xe:djextImageSelect>
```

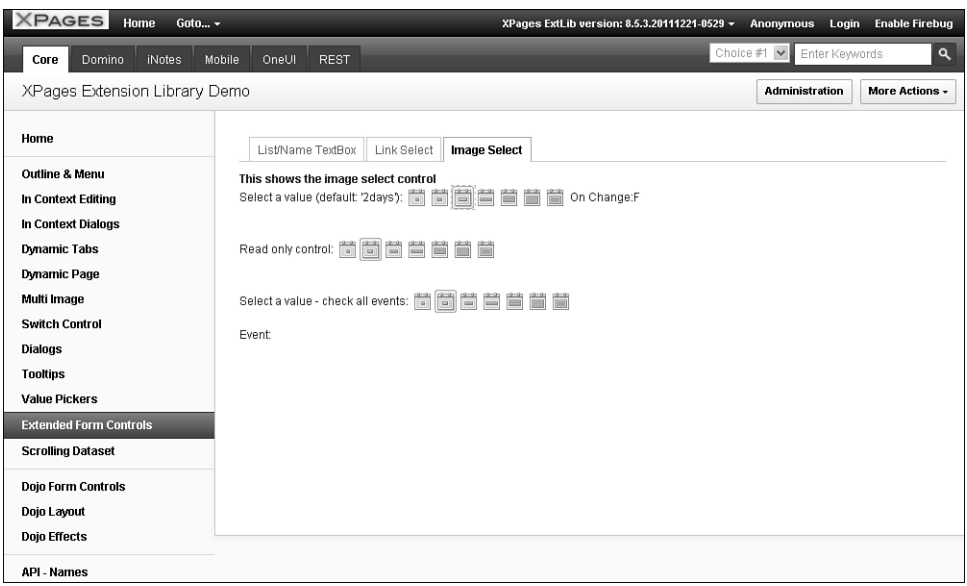


Figure 5.18 Dojo Link Select for Calendar Picker.

Table 5.13 details the additional properties available for the Dojo Image Select control.

Table 5.13 `xe:djImageSelect` Properties

Property	Description
image	Defines the image shown when this image is not selected.
imageAlt	Defines the alt text to appear when the user hovers over the image.
selectedImage	Defines the image shown when this image is selected.
selectedStyle	Defines styling to be applied when this image is selected.
selectedStyleClass	Defines the class to be applied when this image is selected.
selectedValue	Defines the value to pass when this image is selected.
style	Defines styling to be applied when this image is not selected.
styleClass	Defines the class to be applied when this image is not selected.

Dojo Effects Simple Actions

The inclusion of Dojo within the Extension Library extends beyond controls for storing user-entered content. Some commonly used Dojo effects have also been added, implemented as Simple Actions. So you can easily add them to buttons, links, or anything else that has an event. These simple actions add animations to a form, to enhance the user experience.

So, for example, you can use a Dojo effect to fade in or wipe in helper text beside a field when the user clicks into it, and fade out or wipe out when the user exits the field. And because all the Dojo effects run Client-Side, there is no performance hit of round-tripping to the server.

Dojo Fade and Wipe Effects

The fade or wipe effects—either in or out—have additional properties that can be set. The **node** property is the component to be faded/wiped, a Server-Side component ID, as can be seen from Figure 5.19. The **var** property, as elsewhere, is a variable name the function uses to play the Dojo effect. You cannot reference it elsewhere on the XPage via Client-Side JavaScript, because it is scoped only to the eventHandler.

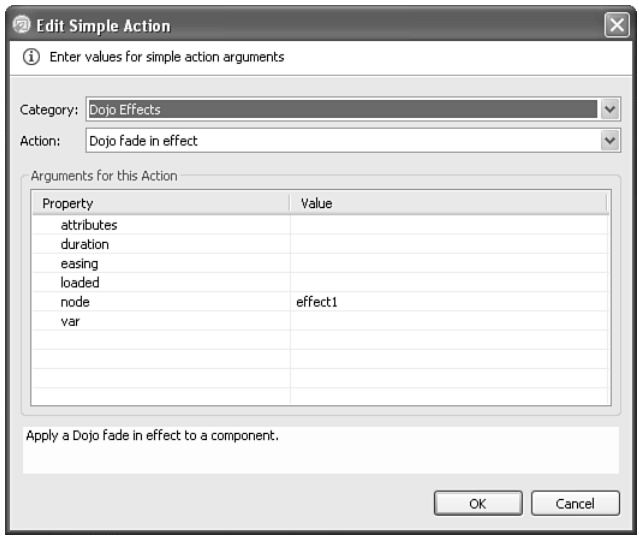


Figure 5.19 Dojo Fade In Effect.

The **duration** property defines how long in milliseconds the effect takes to run, whereas the **easing** property takes a function that will handle how the effect runs, such as accelerating the rate with which the node fades in. You can write this function from scratch, as on the **Core_DojoEffects.xsp** XPages Extension Library Demo database, or as a predefined function, such as those in the dojo.fx.easing object (see Listing 5.18).

Listing 5.18 Dojo Fade Out with dojo.fx.easing

```
<xp:this.resources>
   <xp:dojoModule
      name="dojo.fx.easing">
   </xp:dojoModule>
</xp:this.resources>

<xp:button
   value="Fade Out - Duration 2s"
   id="button3">
   <xp:eventHandler
      event="onclick"
      submit="false">
```

Listing 5.18 (Continued)

```
      <xp:this.script>
         <xe:dojoFadeOut
         node="effect1"
         duration="200"
         easing="dojo.fx.easing.expoInOut">
      </xe:dojoFadeOut>
   </xp:this.script>
</xp:eventHandler>
```

Table 5.14 shows the main properties for the Dojo Fade and Wipe simple actions.

Table 5.14 `xe:dojoFadeIn`, `xe:dojoFadeOut`, `xe:dojofxWipeIn`, and `xe:dojofxWipeOut` Properties

Property	Description
duration	Defines the duration the animation should take.
easing	Requires a Client-Side JavaScript function to define the rate of acceleration of the animation.
node	Defines the node to which the animation should be applied.
var	Defines a variable name under which the animation runs.

Dojo Slide To Effect

The slide effect has all the properties of the fade and wipe effects but also two additional properties, **top** and **left**, for defining how far relative to the top and left of the screen the relevant node should be slid. You can set all the properties available with a specific value or calculate them via Server-Side JavaScript. The slide effect in Listing 5.19 shows how or why to use the **attributes** property: namely, to enable the developer to set any of the effects via Client-Side JavaScript. Why not just type `dojo.coords(_id).t` directly into the **top** property? First, because `_id` has a specific meaning to the XSP Command Manager, so it throws an error. Second, because the **top** property must be a number, not a string. So you must use the **attributes** property to pass the function, which sets top to the node's current **top** property, to the browser. This function also shows how to retrieve a node's current position to slide a node relative to that current position.

Listing 5.19 Slide Effect with attributes Property

```
<xp:button
   value="Slide left"
   id="button8">
   <xp:eventHandler
      event="onclick"
      submit="false">
      <xp:this.script>
         <xe:dojofxSlideTo
            node="effect1"
            left="0">
            <xp:this.attributes>
               <xp:parameter
                   name="top"
                   value="dojo.coords(_id).t">
                </xp:parameter>
            </xp:this.attributes>
         </xe:dojofxSlideTo>
      </xp:this.script>
   </xp:eventHandler>
</xp:button>
```

Table 5.15 shows the significant properties of the Dojo Slide To Effect.

Table 5.15 `xe:dojofxSlideTo` Properties

Property	Description
left	Defines how far relative to the left of the screen the node should be slid.
top	Defines how far relative to the top of the screen the node should be slid.

Dojo Animation

The Dojo animation effect implements the dojo.animateProperty object within a simple action. The effect has all the properties already covered in the other Dojo effect simple actions. In addition, there are some specific properties. You can use the **delay** property to add a delay in milliseconds before the effect should start. You can use the **rate** property to change the number of frames per second at which the animation runs; by default, it is 100 frames per second, which is rather quick. The value of the **rate** property is a number in milliseconds, so to change it to 5 frames per

second, the value would be `200` (200 × 5 = 1000 milliseconds = 1 second). You can use the **repeat** property to repeat the animation a certain number of times. But the most important property is the **properties** property, allowing one or more `xe:dojoAnimationProps` objects to be added. These handle what animation runs and its varying settings.

Table 5.16 shows the main properties for the Dojo animation effect.

Table 5.16 `xe:dojoDojoAnimateProperty` Properties

Property	Description
delay	Defines the delay before the animation begins.
duration	Defines the duration of the animation.
easing	Requires a Client-Side JavaScript function to define the rate of acceleration of the animation.
node	Defines the node to which the animation should be applied.
properties	Defines the animation properties.
rate	Defines the rate per second, taking a value in milliseconds.
repeat	Defines the number of times the animation should repeat.
var	Defines a variable name under which the animation runs.

In addition to the **loaded** property, the `xe:dojoAnimationProps` object has four properties shown in Table 5.17. The Extension Library demo database has an example of this on the **Core_DojoEffects.xsp** XPage, for increasing the size of a box, shown in Listing 5.20. Line 9 sets the animation to run on the bluebox component. Lines 14 and 15 define the starting and ending width and height of the box.

Table 5.17 `xe:dojoDojoAnimationProps` Properties

Property	Description
end	Defines the ending value of the attribute this animation applies to.
name	Defines the attribute this animation applies to, such as “width” or “height”.
start	Defines the starting value for the attribute this animation applies to.
unit	Defines the unit for the values in start and end.

Listing 5.20 `Core_DojoEffect.xsp` Dojo Animation Simple Action

```
1  <xp:button
2     value="Grow the box"
3     id="button5">
4     <xp:eventHandler
5        event="onclick"
6        submit="false">
7        <xp:this.script>
8              <xe:dojoAnimateProperty
9              node="bluebox"
10             duration="3000">
11             <xp:this.properties>
12                <xe:dojoAnimationProps
13                   name="width"
14                   start="200"
15                   end="400">
16                </xe:dojoAnimationProps>
17                <xe:dojoAnimationProps
18                   name="height"
19                   start="200"
20                   end="400">
21                </xe:dojoAnimationProps>
22             </xp:this.properties>
23          </xe:dojoAnimateProperty>
24       </xp:this.script>
25    </xp:eventHandler>
26 </xp:button>
```

Earlier in this chapter, code was provided to style the ToggleButton control. At this point, it is appropriate to revisit that code, shown in Listing 5.13. Listing 5.21 shows alternate code for the ToggleButton using a Dojo animation simple action, with the output shown in Figure 5.20. To revisit the functionality, the animation should change the font color of the ToggleButton, alternating between red and green. However, the properties of the `xe:dojoAnimationProps` object can only accept literal values or Server-Side JavaScript returning a literal value. It is not possible to add Client-Side JavaScript code to ensure the end color alternates. As a result, you must use the **attributes** property to compute the properties object in Client-Side JavaScript, in lines 16 to 29. Line 18 creates the color object (the **name** property of an `xe:dojoAnimationProps` object). Line 19 sets the start attribute of the color object, although `_id.style.color` is not set when the page is loaded. Lines 20 to 26 set the end attribute to a function that sets the color to red if it is initially green, otherwise red.

Listing 5.21 Using Dojo Animation Simple Action to Style the ToggleButton

```
<xe:djToggleButton
id="djToggleButton2"
   value="#{sessionScope.djButton3}"
   label="Toggle Button"
   checkedValue="Checked..."
   uncheckedValue="Not Checked..."
   style="color:rgb(255,0,0)">
   <xp:eventHandler
      event="onclick"
      submit="false">
      <xp:this.script>
         <xe:dojoAnimateProperty
            node="djToggleButton2"
            duration="500">
            <xe:this.attributes>
               <xp:parameter
                  name="properties">
                  <xp:this.value><![CDATA[{"color":
{"start":_id.style.color,
"end":function() {
   if (_id.style.color=="rgb(0, 255, 0)") {
         return "rgb(255,0,0)";
   } else {
         return "rgb(0,255,0)";
   }
}
}
}]]></xp:this.value>
               </xp:parameter>
            </xe:this.attributes>
         </xe:dojoAnimateProperty>
      </xp:this.script>
   </xp:eventHandler>
</xe:djToggleButton>
```

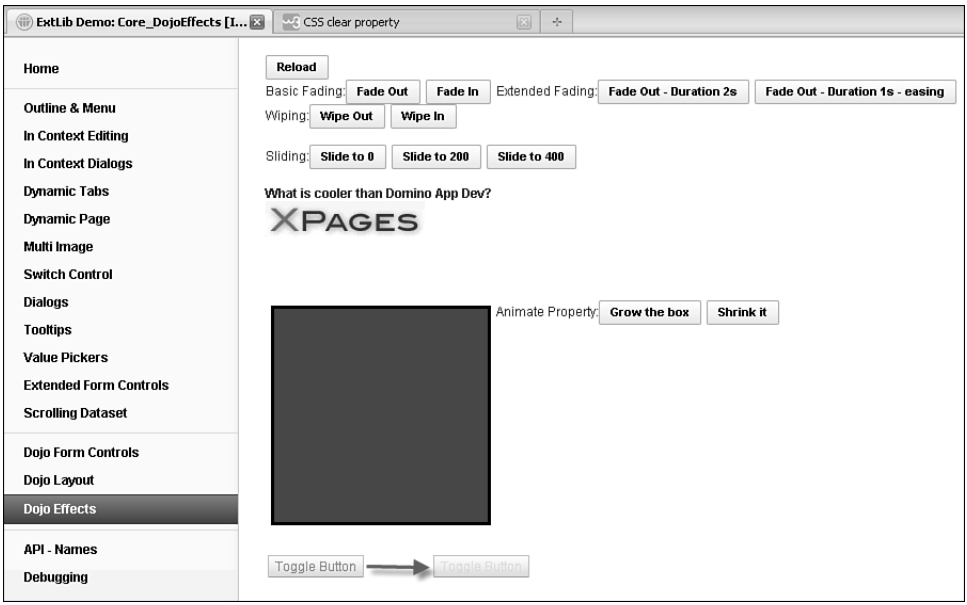


Figure 5.20 Dojo Fade In Effect.

Conclusion

This chapter covered many of the Dojo controls provided by the Extension Library to add to the content controls covered in the previous chapter. These Dojo controls offer little additional functionality to the traditional Dojo controls, but they do make it easier to implement the controls and minimize the risk of mistyping or misremembering Dojo attributes.

This page intentionally left blank

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P–Q

R

S

T

U

V

W

X–Y–Z