



/THEORY/IN/PRACTICE

SharePoint 2010 at Work

Tricks, Traps, and Bold Opinions:
The Best of EndUserSharePoint.com

O'REILLY®

Mark Miller

www.it-ebooks.info

SharePoint 2010 at Work

“What a fantastic resource. No other book in the SharePoint community hits this ‘power user’ or ‘end user’ audience like this book does. There are powerful examples in here that will really solve business problems and set the reader on the right track.”

–Joel Oleson, Global SharePoint Evangelist, the first SharePoint architect

If you work with SharePoint, you may have discovered that there are countless tricks for using this platform to solve real-world problems—and you certainly wouldn’t mind learning some of them. That’s the purpose behind EndUserSharePoint, a community site that lets end users share ingenious new ways for putting SharePoint to work.

This insightful and entertaining book presents a compilation of popular, well-written articles from the site, published by contributors for people who use SharePoint at their companies but don’t have access to its technical server side. Each engaging story puts you into the narrative as a participant, rather than a passive observer, so you can easily visualize the situation and share the “aha!” solution with the author. Learn some tricks, gain some insight—and have fun doing it.

These articles will help you:

- Build a documented framework for evaluating whether your company is getting the most value from SharePoint
- Create documentation and script management with OneNote and a SharePoint library
- Use the Data View web part to create hyperlinks from existing SharePoint data
- Implement data visualization in SharePoint without access to the server
- Create document libraries with mixed content sources in any SharePoint version
- Pull information from disparate site collections into a single navigation system

Mark Miller, Founder and Editor of EndUserSharePoint.com, is currently Director of Global Strategy and Senior Storyteller at Fpweb.net.

Contributors

Sadalit Van Buren
Kerri Abraham

Jim Bob Howard
Marc D. Anderson
Laura Rogers

Waldek Mastykarz
Alexander Bautz
Dessie Lunsford

Eric Alexander
Peter Allen

Twitter: @oreillymedia
facebook.com/oreilly

US \$39.99

CAN \$41.99

ISBN: 978-1-449-32100-0



O'REILLY[®]
oreilly.com

SharePoint 2010 at Work

Mark Miller, Kerri Abraham, Eric Alexander, Peter Allen, Marc Anderson, Alexander Bautz, Sadalit Van Buren, Jim Bob Howard, Dessie Lunsford, Waldek Mastykarz, and Laura Rogers

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

www.it-ebooks.info

SharePoint 2010 at Work

by Mark Miller, Kerri Abraham, Eric Alexander, Peter Allen, Marc Anderson, Alexander Bautz, Sadalit Van Buren, Jim Bob Howard, Dessie Lunsford, Waldek Mastyskarz, and Laura Rogers

Copyright © 2012 O'Reilly Media. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor: Rachel Roumeliotis
Production Editor: Melanie Yarbrough
Copyeditor: Amy Thomson
Proofreader: Linley Dolby

Indexer: Fred Brown
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

February 2012: First Edition.

Revision History for the First Edition:
2012-02-02 First release

See <http://oreilly.com/catalog/errata.csp?isbn=9781449321000> for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *SharePoint 2010 at Work* and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-32100-0

[LSI]

1328299811

CONTENTS

	PREFACE	v
1	THE SHAREPOINT MATURITY MODEL	1
	<i>Evolution of the Model</i>	2
	<i>Structure of the Model</i>	3
	<i>Applying the Model</i>	6
	<i>Summary</i>	11
2	EMPOWER THE POWER USER	13
	<i>SharePoint Designer: To Allow or Not To Allow? That Is the Question</i>	13
	<i>Middle Ground: Configuration Management</i>	16
	<i>Solution: The SharePoint Rudder</i>	17
	<i>Connect Up OneNote</i>	27
	<i>The Five “W”s of Documentation</i>	34
	<i>OneNote Templates</i>	35
	<i>Empowered Utopia in 10 Steps</i>	41
	<i>Configuration Management Is Thoughtful Maintenance</i>	41
	<i>Empowerment Without Responsibility Is Chaos</i>	42
	<i>Summary</i>	43
3	JQUERY TO THE RESCUE	45
	<i>Automate an All-Day Event</i>	46
	<i>Requesting a Review Only Once Per User</i>	48
	<i>Default Text Based on Radio Button Click</i>	52
	<i>Writing a Survey ID to a List on Response Creation (without Workflow)</i>	55
	<i>Labeled Sections on Default Forms</i>	63
	<i>Where To from Here?</i>	67
	<i>Summary</i>	67
4	UNLOCKING THE MYSTERIES OF THE SHAREPOINT DATA VIEW WEB PART XSL TAGS	69
	<i>More About Data View Web Parts</i>	71
	<i>Summary</i>	83
5	HYPERLINKS IN THE DATA VIEW WEB PART	85
	<i>Setup for Walkthroughs</i>	86
	<i>URLs in SharePoint</i>	88
	<i>XSLT List View Web Part Hyperlinks</i>	90
	<i>DVWP Hyperlinks</i>	95
	<i>Modal Dialog Box</i>	113
	<i>Summary</i>	115

6	BUILDING A QUOTE OF THE DAY WEB PART IN SHAREPOINT 2010	117
	<i>Part I: Building the Quote of the Day Web Part</i>	118
	<i>Part II: Preparing Quote of the Day Web Part for Redistribution</i>	130
	<i>Summary</i>	152
7	SPJS CHARTS FOR SHAREPOINT	155
	<i>Technical Overview</i>	155
	<i>Version History</i>	156
	<i>Initial Setup</i>	157
	<i>The Edit Chart GUI</i>	159
	<i>How to Make Web Part Templates</i>	167
	<i>Multiple Charts in One Page</i>	168
	<i>Summary</i>	169
8	TAMING THE ELUSIVE CALCULATED COLUMN—LOGIC FUNCTIONS	171
	<i>The Functions</i>	171
	<i>The IFs</i>	172
	<i>The Cousins: OR and AND</i>	200
	<i>Summary</i>	211
9	CREATING DOCUMENT LIBRARIES WITH MIXED CONTENT SOURCES	213
	<i>Background</i>	213
	<i>Configuring a Document Library in SharePoint Server 2010/SharePoint Foundation 2010</i>	214
	<i>Configuring a Document Library in MOSS 2007/WSS 3.0</i>	217
	<i>What the Content Type Does</i>	220
	<i>Extending the Link to a Document Content Type</i>	221
	<i>Summary</i>	224
10	SHAREPOINT 2010 TAB PAGE	225
	<i>Implementation</i>	226
	<i>jQuery Implementation</i>	230
	<i>Tab Page Layout Code</i>	230
	<i>Summary</i>	230
11	A GLOBAL NAVIGATION SOLUTION ACROSS SITE COLLECTIONS	231
	<i>Implementation</i>	232
	<i>Summary</i>	241
	INDEX	243

P R E F A C E

Most people have heard the timeless parable of the six blind men trying to describe an elephant. Each man touches the elephant and, based upon the part of the body he touches, proceeds to give a definitive description of “an elephant.” None of the descriptions are correct, and yet they are all correct. Talking about SharePoint, even for those of us who have worked with it for years, is like a blind man describing an elephant. It is impossible, and yet we do it every day.

The NothingButSharePoint/EndUserSharePoint community site that I run has hundreds of contributors trying to describe the SharePoint elephant on a daily basis. They examine the beast from every angle, as a Developer, as an IT Pro, or as an End User, each with a different perspective and for different reasons. This book is a compilation of stories from the end user perspective, for those who must use and support SharePoint at their companies but don’t have access to the technical server side of the platform.

Background

My official title in the SharePoint Community is *Mark Miller, Senior Storyteller*. As such, I like to think of the articles provided to EndUserSharePoint (EUSP) as stories, not blog posts. We “publish” content each day, as opposed to “posting” content. There’s a subtle distinction. Publishing insinuates something more than a simple thought dashed off and posted in a matter of minutes. To publish means the item is meant to be read as a thoughtful narrative. It is implied that there is an idea behind the content that will be useful at multiple levels. When we publish

an article on EUSP, we think of it as a story that will resonate with the SharePoint End User. Underlying the process is the premise that a story will engage readers by putting them into the narrative as participants, not as passive observers.

Each of the authors in this collection was selected because of his or her ability to tell a good story. They take the technical aspects of the narrative and weave a tale around the daily life of a SharePoint Site Administrator or the constant struggles and frustrations of a typical End User. Each author has his or her own voice and perspective, but the stories are tied together with a consistent theme: SharePoint is flexible enough to help solve real-world business problems, if you can determine the right part of the elephant to examine.

This book will be most effective for those who are looking to solve business problems through the use of SharePoint but don't know which end of the elephant to tackle. For those who have never encountered an elephant, or SharePoint, the stories here might not be very helpful. Those who will benefit the most are those who have worked with SharePoint for a while and need to know which end of the beast to examine for the specific problem they are trying to solve. For them, the stories will become the catalyst for further investigation and discovery.

The Authors and Their Stories

The stories in this collection were chosen because of their popularity on EndUserSharePoint. Like a blind man describing an elephant, each tale takes a different view of the platform and shows how you can use SharePoint to solve real-world business problems. The solutions and concepts have been viewed hundreds of thousands of times on the EndUserSharePoint site. The authors have used the feedback on those articles to refine their ideas, making them useful for the broadest spectrum of the SharePoint Community. The technical aspects of each of the stories have been updated to the SharePoint 2010 environment, but the concepts remain timeless and can be applied to any version of SharePoint 2007 or 2010.

The stories can be read in any order, but I suggest that everyone at least review [Chapter 1](#) just to get an idea of where your SharePoint implementation sits on the maturity scale. From there, glance through the rest of the stories and see what you'd like to tackle first.

The SharePoint Maturity Model—Sadie Van Buren

If there is a single chapter in the book that will be useful for everyone, [Chapter 1](#) is it. It helps you examine the entire elephant. Sadie has experience with over 50 SharePoint implementations and uses the knowledge she has gained to create a documented framework for evaluating where your company stands when it comes to getting the most value from SharePoint.

Empower the Power User—Kerri Abraham

For some reason, OneNote has never really received the recognition it deserves, nor has the internal SharePoint Power User. I use OneNote every day and know people like Kerri who can't even imagine getting work done without it. Kerri's story in this collection is one of the longer ones, but when you see the power of what she has done to create documentation and script management within OneNote and a SharePoint library, you might consider opening up your environment a little more to give real power to your internal SharePoint heroes.

jQuery to the Rescue—Jim Bob Howard

jQuery is that special sauce that makes everything go better with the presentation layer. Jim Bob gives us five solutions you can implement immediately without recourse to the server. Some of the solutions are jaw-dropping to people who didn't think it was possible to do cool stuff in SharePoint. As Jim Bob says, "It's just the tip of the iceberg."

Unlocking the Mysteries of the SharePoint Data View Web Part XSL Tags— Marc Anderson

In this story, Marc takes a core piece of what is needed to implement presentation-layer solutions with the Data View web part (DVWP) and the XSL that drives it. It's one of those things that hardly ever gets touched, because it seems so mysterious. With the DVWP as the main character and XSL as its sidekick, this little adventure story is the beginning of a much longer tale.

Hyperlinks in the Data View Web Part—Laura Rogers

As a professional storyteller, I like to engage the audience immediately when I'm giving a talk. One of the things I can always count on is the audience knowing the answer to the question, "Who is Queen of the Data View web part?" Laura owns that space in the mind of the SharePoint community. In this update to one of her most popular articles, she demonstrates how to create hyperlinks from existing data in SharePoint.

Building a Quote of the Day Web Part in SharePoint 2010—Waldek Mastykarz

Without exception, the Quote of the Day web part is one of the most popular downloads at EndUserSharePoint. I created it in a half hour after hearing Lori Garcia tell a story about manually updating her site each day with a new quote. Waldek saw the solution and extended it to pull the quotes from a SharePoint list instead of having them embedded in the Content Editor web part.

SPJS Charts for SharePoint—Alexander Bautz

“A picture is worth a thousand words” is a cliché for a reason. Visualization of data within SharePoint is one of the most powerful and useful aspects of the platform. Unfortunately, it’s not all that easy to do. In this story, Alexander shows us a solution that any site manager or site collection administrator can implement, even without access to the SharePoint server.

Taming the Elusive Calculated Column—Logic Functions—Dessie Lunsford

Dessie’s a funny kind of guy. I met him on the SharePointU forums when I first started working with SharePoint. He likes to go four-wheeling when he’s not cranking out stories for EUSP. The calculated column is one of the most underutilized features in SharePoint, useful for displaying inline visualization within any list or library. With his series of over 40 articles on EUSP, I think I can easily crown Dessie “King of the Calculated Column.” This story is a comprehensive step-through of the logic functions available within the calculated column.

Creating Document Libraries with Mixed Content Sources—Eric Alexander

Eric is my “go-to guy” when there’s a SharePoint issue I don’t know how to handle. As a matter of fact, Eric is the go-to guy for the thousands of people who have asked questions on our Stump the Panel Forum (STP) at EUSP, since he is the lead moderator. He has taken an interesting question from the forum, how to provide mixed content in a library, and created a solution that can be used in any version of SharePoint.

SharePoint 2010 Tab Page—Peter Allen

I first met Peter when he redid a solution I had created for formatting pages in a SharePoint wiki. In the updated solution he provides here for a tab-based interface, the fun part of the story is that he actually uses the solution to describe the solution.

A Global Navigation Solution Across Site Collections—Peter Allen

In this solution, Peter utilizes the SharePoint Web Services library created by Marc Anderson to pull information from disparate locations into a single navigation system. It is one of the most requested solutions when people have expanded beyond their first site collection and realize there is no visibility between data across domains.

Summary

There you have it. Eleven stories, each with a moral that clarifies a different piece of SharePoint. There is a second parable that is apropos for SharePoint that we have used at EndUserSharePoint. It’s the one on how to eat an elephant, but we’ll leave that to another time and place.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This icon signifies a tip, suggestion, or general note.

CAUTION

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*SharePoint 2010 at Work* by Mark Miller (O'Reilly). Copyright 2012 O'Reilly Media, Inc., 978-1-4493-2100-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online



Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at <http://my.safaribooksonline.com>.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://shop.oreilly.com/product/0636920024262.do>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

It has been exciting working with the authors on this book. We are in contact daily, but it's still a thrill and surprise to see what they come up with each morning. In addition, there are

hundreds of other authors who have written for EUSP who deserve recognition for their contributions. Thank you to every author who has contributed to the site.

Marsee Henon, Ken Brown, and Rachel Roumeliotis at O'Reilly have been some of my biggest supporters over the past couple years after a chance meeting at SPTEchCon in Boston. Thank you from me, and the authors, for the opportunity to publish our stories.

As Marc Anderson says in his chapter, Natasha Felshman is the engine that keeps EUSP running. It would not be in existence if not for her. Dr. Susan Zolla-Pazner allowed me to work with her team to coordinate AIDS vaccine research data for my first SharePoint project in 2006. Barbara Straw and Pat Iovanella believed in me enough to allow me to act as a SharePoint teacher and mentor in their companies. EUSP is a much more viable resource because of all of them.

My family—Rosemary, Orion, and Aurora—encourages me every day to do what I love, which is to write and tell stories. I look forward to many more years of tall tales and outright lies that keep us laughing and growing together as a family.

Mark Miller, @eusp Founder and Editor, EndUserSharePoint.com cofounder, NothingButSharePoint.com Director of Global Strategy and Senior Storyteller, Fpweb.net

The SharePoint Maturity Model

Sadalit Van Buren

How well does your organization use SharePoint? You probably can't answer that in any quantifiable terms, much less speak about how the various components of SharePoint are working for you.

With SharePoint's explosive popularity and adoption worldwide, a community of SharePoint experts has formed with the goal of sharing knowledge about this product. The authors in this book, as well as hundreds of others, dedicate their time and energy to help organizations understand and use the product, and have made a galaxy of resources available for different areas of functionality.

What's been missing is a cohesive way to analyze and understand the platform as a whole. Organizations don't know what they have, and they may be focusing too much on projects that yield little return, missing the quick wins, or declining to invest in areas that could truly transform their businesses.

I created the SharePoint Maturity Model to apply a holistic view to a SharePoint implementation and to bring standardization to the conversation around functionality, best practices, and improvement. My goal is to allow organizations to reach the full potential of their investments in SharePoint, and the SharePoint Maturity Model is the framework that enables this.

Evolution of the Model

My work as a SharePoint consultant, starting in 2006, gave me a perspective over many different companies and implementation types. I noticed there was a typical progression of SharePoint projects and the issues surrounding them. Most companies were working toward the same initiatives and struggling with the same challenges, though most saw their efforts as unique to their environment. Many clients hired us because we helped “other companies” solve the same problem. Other than anecdotal evidence, there was no way to truly compare organizations’ use of SharePoint.

In 2009, when Microsoft Office SharePoint Server (MOSS) 2007 made SharePoint a viable platform for a broad range of companies, I expressed the trend of implementation as a quadrant of Complexity and Risk vs. Buy-in (see [Figure 1-1](#)), where companies typically started with the “low-hanging fruit” projects and saved the more culture-changing and resource-intensive projects for later.

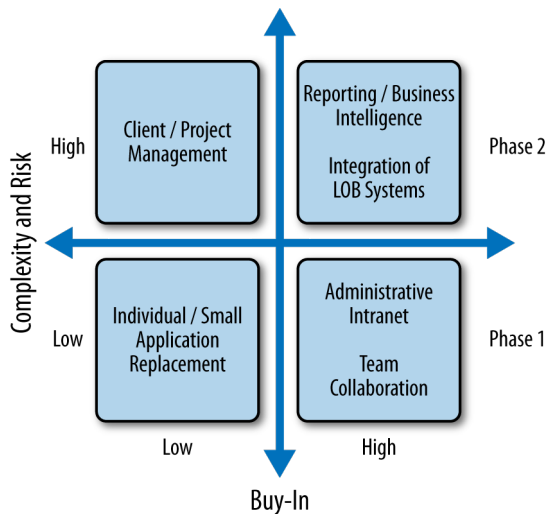


FIGURE 1-1. Early attempt to characterize SharePoint maturity (courtesy of Knowledge Management Associates LLC)

With the release of SharePoint 2010, it was obvious that the picture was much more complex than this. A few other SharePoint experts had attempted to set out a model of SharePoint maturity, but these were limited to specific segments of the technology, such as [deployment](#) and [collaboration](#). At this time, I was also seeing an evolution in the community’s thinking, from largely technical to more business-focused concerns. I was frustrated with the mostly tactical, technology-oriented conversations I kept hearing about SharePoint, and I had a vision for a standardized way for people to talk about their implementations and a means for them

to benchmark against others and show progress over time, which is critical to justifying investments in IT.

In the Fall of 2010, I created Version 1 of the SharePoint Maturity Model and, with the support of Mark Miller, published it on [EndUserSharePoint](#) for community evaluation and feedback. The enthusiastic response it received showed there had truly been a need for this kind of tool, and suggestions from many community members have led to continuous improvements since its release. Since then, organizations of all sizes, from 20 users to 40,000 users, have assessed their progress against the Model. A selection of the current data from these assessments is available on [SharePointMaturity.com](#).

Structure of the Model

The Model has 5 maturity levels and 11 competencies, which are divided into three groups: core solution competencies, advanced solution competencies (both of which are oriented toward the technological, tool-based side), and readiness competencies (oriented toward the environmental, human-based side).

The maturity levels follow the five-level standard set out in the [Capability Maturity Model](#) (see [Table 1-1](#)) and many other models, and describe the processes around implementation of the platform. These levels and competency definitions are not specific to SharePoint 2010, but can be used for 2010, and will be updated when the features of Wave 15 are released from Non-Disclosure.

TABLE 1-1. SharePoint maturity levels

Level	Definition
500	The area is functioning optimally and continuous improvement occurs based on defined and monitored metrics. Return on investment (ROI) is demonstrable.
400	The area is centrally supported, standardized, and implemented across the entire organization. Governance is defined and understood and followed.
300	The way the area is implemented is defined and/or standardized, but not in use across the entire organization. Governance is defined, but may not be widely understood or followed. ROI is considered.
200	The area is managed by a central group (often IT), but the focus and definition varies by functional area or is limited to a single area.
100	The starting point of SharePoint use.

The core competencies are where organizations typically focus first, because they tend to yield greater results with lower investment and often serve as an update to systems or functionality with which the business is already familiar. [Table 1-2](#) describes the core SharePoint competencies.

TABLE 1-2. Core SharePoint competencies

Competency name	Definition
Publication	Presentation of content in SharePoint for consumption by a varied audience of authenticated users. Areas of focus include navigation, presentation of content (static vs. personalized), content organization and storage, customizations to the template, and approvals and workflow.
Collaboration	Multiple individuals working jointly within SharePoint. Areas of focus include provisioning and deprovisioning, templates, organization (finding a site), archiving, and using SharePoint's capabilities (for example, versioning and document management, task management, calendar management, discussion thread, surveys, and workflow).
Business Process	Linked business activities with a defined trigger and outcome, standardized by SharePoint and/or custom automated workflow processes. Areas of focus include data (unstructured and structured), workflow, user security and roles, reporting and analytics, tracking and auditing, process modeling and simulation, and process optimization.
Search	The ability to query indexed content and return results that are ranked in order of relevance to the search query. Areas of focus include scopes, display of results, optimization, integration and connectors, and performance.

The advanced competencies (described in Table 1-3) are so named because they tend to be simultaneously more culture-changing and more resource-intensive. They may introduce concepts or functionality that are new to the End Users.

TABLE 1-3. Advanced SharePoint competencies

Competency name	Definition
People and Communities	The human capital of the organization as represented in SharePoint by profiles, MySites, and community spaces (the virtual spaces that support particular areas of interest that may span or fall outside the organizational structure).
Composites and Applications	Custom solutions specific to the needs of the business (traditionally served by paper forms, Excel spreadsheets, and/or Access databases) that may be accomplished by multiple technologies working together.
Integration	Line of business data and/or content from a separate Content Management System (CMS) integrated with the system, allowing users to self-serve in a controlled yet flexible manner. Maturity proceeds through integration with a single system, multiple systems, data warehouse, and external (partner, supplier, or industry) data.
Insight	The means of viewing business data in the system. Maturity proceeds through aggregation of views, drill-down and charting, actionability, and analytics and trending.

The Readiness competencies (described in Table 1-4) are common to most technology systems, and are critical to a successful SharePoint implementation.

TABLE 1-4. Readiness competencies

Competency name	Definition
Infrastructure and Administration	The hardware and processes that support the system. Areas of focus include farm planning, server configuration, storage, backup and restore, monitoring, and updates.
Staffing and Training	The human resources that support the system and the level of training with which they are provided.
Customizations	Custom development and/or third-party products that extend the out-of-box functionality of the system. Areas of focus include development environment, management of source code, method of build and deployment, testing, and development tier.

Together, the 11 competencies and the 5 levels create a matrix, shown in Figure 1-2, that describes the best practices and indicators for each competency level.

Level	Publication	Collaboration	Business Process	Search	People and Communities	Composites and Applications	Integration	Insight	Infrastructure	Staffing & Training	Customizations
500 Optimizing	Customs personalized for users. Content is shared across multiple devices and systems. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Collaboration across the cloud. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Process is rich. Features are available in the cloud for any device. Automated tagging and process.	Users understand underlying of tags. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.
400 Predictable	Customs are available. Content is shared across multiple devices and systems. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Collaboration across the cloud. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Process is rich. Features are available in the cloud for any device. Automated tagging and process.	Users understand underlying of tags. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.
300 Defined	Customs are available. Content is shared across multiple devices and systems. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Collaboration across the cloud. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Process is rich. Features are available in the cloud for any device. Automated tagging and process.	Users understand underlying of tags. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.
200 Managed	Customs are available. Content is shared across multiple devices and systems. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Collaboration across the cloud. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Process is rich. Features are available in the cloud for any device. Automated tagging and process.	Users understand underlying of tags. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.
100 Initial	Customs are available. Content is shared across multiple devices and systems. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Collaboration across the cloud. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Process is rich. Features are available in the cloud for any device. Automated tagging and process.	Users understand underlying of tags. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.	Users are able to create, edit, and share content. Content is rich. Features are available in the cloud for any device. Automated tagging and process.

FIGURE 1-2. The SharePoint Maturity Model's matrix structure; this image is for reference only—to examine the model in depth, see www.spmaturity.com/Resources/Sharepoint_Maturity_Model_Overview.pdf

Applying the Model

The SharePoint Maturity Model can benefit you at three levels:

As an individual implementation owner

If you are responsible for your organization’s SharePoint implementation and you have been struggling with adoption, business alignment, and making the case that there really is a return on your investment in SharePoint, the model can help you define your strategic roadmap and give you a quantitative sense of your progress when you reevaluate periodically. This can demonstrate the need for increased resources and will put clearer definition around which projects should take priority.

As part of your organization

Taking the time to evaluate your implementation and making continuous improvements based on your roadmap will ultimately lead to greater business process efficiency, happier and more empowered users, and a more stable SharePoint environment. The assessment can also help define ROI for upgrades or additional feature implementation.

As a member of the wider SharePoint community

By assigning a number value to your current state in the competencies, you are helping to build a data model that will help answer larger questions about where organizations are in their SP maturity—by industry, number of years of use, the number of IT staff supporting the implementation, etc. (Figure 1-3).

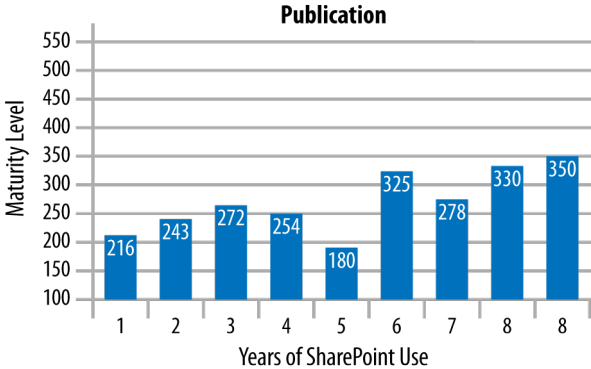


FIGURE 1-3. Example of data from SharePoint Maturity Model Assessments: Publication Maturity per Years of Use

The Model also has practical applications within your business, such as:

Project triage

Evaluate new project requests against the Model to determine whether your implementation can support them.

Staffing considerations

Know where and when specific resources will be needed as you move upward in maturity.

Risk assessments

Understand where the risks are in undertaking projects that are at a higher maturity level or that depend on related competencies where your maturity is not as high.

Training

Develop a curriculum for individuals and functional units based on their implementation's maturity level.

Products and services

When you've determined the competencies on which you'll focus, you can quickly get a sense of which companies and tools can help you get to the next level.

These examples should give some idea of the potential of the Maturity Model as a powerful tool for expanding and demonstrating SharePoint's influence on your business. Now we'll go through one of the competencies, publication, to see how an organization might progress through the levels and realize some of these benefits.

Focus on Publication

As with all the competencies in the SharePoint Maturity Model, the five levels of publication are identified by distinct characteristics (see [Figure 1-4](#)). This is not intended to be a comprehensive list of all possibilities, but is a framework for an organization to recognize the markers that define its level.

Let's say we're looking at a product company with several global locations and just over 1,000 employees. The company has a homegrown intranet that is nine years old and receives a lot of complaints from the employees—it's difficult to update, the home page is confusing due to a large number of links and buttons, and there is no overall search functionality. It is integrated with a content management system that is no longer supported. Everyone agrees it needs to be replaced.

The CIO does some informal benchmarking with her colleagues at similar companies and finds that SharePoint is the current platform of choice. She has her senior business analyst do some research and product evaluation of several content management systems to confirm the decision to go with SharePoint.

Level	Maturity Level Definition	Competency
500 Optimizing	The particular area is functioning optimally and continuous improvement occurs based on defined and monitored metrics.	Content is personalized to the user. Content is shared across multiple functions and systems without duplication. Feedback mechanism is in place for pages and taxonomy. Automated tagging may be present.
400 Predictable	The particular area is centrally supported, standardized, and in use across the entire organization. Governance is defined and followed.	Content is monitored, maintained, targeted to specific groups. Usage is analyzed. Digital assets are managed appropriately. If more than one doc mgmt system is present, governance is defined. Mobile access considered.
300 Defined	The way the particular area is leveraged is defined and/or standardized, but not in use across the entire organization.	Site Columns/Managed Metadata standardized the taxonomy. Custom content Types are created. Custom page layout & site templates are configured. Approval process is implemented. Incoming email activated for some lists/libs. Site Map is present. Some content targeted to groups.
200 Managed	The particular area is managed by a central group (often IT), but the focus and definition varies by functional area.	Custom metadata is applied to content. Templates standardized across sites. Lists used rather than static HTML Multiple document mgmt systems may be present w/out governance around purpose.
100 Initial	The starting point of SharePoint use.	Navigation & taxonomy not formally considered. Little to no checks content. Folder structure re-created from shared drives. Content that could be in lists is posted in Content Editor WP. Out of box templates / layouts are used.

FIGURE 1-4. The publication competency

Level 100—Goals and First Steps

The company’s main goals for the launch of the new SharePoint-based intranet are:

- Provide a streamlined home page with restructured navigation for easy browsing to all sections.
- Build out fully functional sites for human resources, finance, and information technology with emphasis on publication of current policies and procedures. The rest of the departments will follow in a later phase.
- Provide full-text search of all content across all sites.
- Introduce “People search” based on the company’s Active Directory.

In the first six months of the project, the SharePoint environment is built and an IT project manager and business analyst work with the three key departments to move their content from the old intranet to the new one.

At the end of this effort, IT does a soft launch by redirecting the three old departmental intranet pages to the new sites and putting a basic welcome message (which includes a list of Hot Links) and pictures of the company's locations on the home page. The global navigation has five tabs: Home, News, HR, Finance, and IT. The current navigation on each page reflects the SharePoint defaults with headings for Libraries and Lists.

Level 200—Lists and Navigation Links

After the soft launch, the marketing and communications department is invited to review the home page. They see it as a way to publish news and company events, including the all-important holiday schedule, and the department wants to brand it with the company colors. The IT business analyst creates lists for Announcements and Events, and an additional list for Hot Links, to replace the text-based list that was included in the Welcome message. Permissions are changed on these lists to give members of the marketing and communications department the right to contribute to them. Although the senior visual designer has many years of CSS and web design experience, branding SharePoint is complex and different enough that the team decides to hire an outside consulting firm to do this work.

During this phase, IT is also reaching out to other departments and functional areas to migrate their content from the old intranet to the new one. The IT business analyst creates a standard department template and uses this for each new site. Although most of the departments would like to have a link directly from the global navigation bar, the business analyst knows it's a best practice not to have too many tabs, so he creates a Departments heading and rolls HR, Finance, and IT under that tab, as well as any new department sites that are created. For any departments that are not ready to move their content, he creates a navigation link to their old intranet sites.

At this level of maturity (200), the employees have instant access to their holiday schedules—no extra clicks to a PDF document, hunting through their email, or calling HR for the most recent list. They can also access any site from any page in the intranet, and they're starting to rely on the SharePoint intranet as a place to find information from all departments, with a standard user experience across the sites.

Level 300—Defining and Standardizing

As the department sites are rolled out, the IT business analyst hears a need for location-specific information to be shared and posted. As employees travel between offices, they want quick insight into the local weather and news headlines for the different locations, plus the company-specific details for each location, such as building access policies, the approved car services, and other information. In the past, there had been no dedicated owner for such information, but now the business analyst works with HR to designate a content owner in each office who will

post his location's specific information. The business analyst converts the location pages into individual sites, each based on a standard Location template. Behind the scenes, he sets up site columns to standardize the company's much-used lists: Departments, Office Locations, and Product Lines. He uses the Location metadata to deliver filtered views to the local holidays on each location site.

Changes are happening on the home page, too. The consultants have delivered a new master page and stylesheet that incorporate the company's primary and secondary branding colors and give the web parts and navigation elements a more rounded look. The Hot Links on the home page are now targeted to specific audiences. On the Departments home page, the IT business analyst uses the Table of Contents web part to create a Site Map to help employees browse all the new sites that are being created.

In a parallel effort, as the company's Collaboration maturity improves, employees are able to provision their own Project sites, and a navigation link is added to this new area.

As the company moves into the 300 level of maturity, the site has more visual punch as it streamlines the way content is delivered. Employees are really seeing the benefit now. The most critical information is targeted to them on the home page, and they are able to drill down quickly to find what they need by department, location, and product line. New employees are trained in using the intranet on their first day, and they use it to learn about the company. Within any list or library, content owners can use standard metadata for location, product line, and department. From site to site, there's a familiar look and feel.

Level 400—Monitoring and Maintaining

The marketing and communications department has gotten some great feedback on the new home page, and it actively monitors page hits to see if this translates into real numbers. The IT department wants to maintain the positive momentum for the new site, so the business analyst is tasked with keeping an eye on what employees are searching for, and with developing a process for creating new content for any queries that yield no results.

Publication on the intranet is getting a bit more social, too. The CIO has started writing a blog on the IT department site, and the IT business analyst has added the NoteBoard web part to the Project site template so project team members can give quick updates. The business analyst is also given the responsibility of reading and responding to issues and comments that are submitted via the new Feedback link that appears in the header of all pages.

As more and more employees use the intranet during the work day, they feel supported by the Feedback link, because they receive prompt responses to their comments. The IT business analyst had expected to receive more bugs and issues than suggestions, but he soon needs to create a new category in the feedback list: Great Ideas. So many people in the company are thinking about the intranet now that the IT business analyst feels the truth of the saying, "All of us are smarter than any of us."

Level 500—Planning for Personalization

The next change the CIO is planning is a big one: personalizing the content in all the top-level landing pages of the site. To achieve this, the IT business analyst will need to work with all the content owners to change how they deliver content, from basic storage and display to a directed, well-thought-out plan for publication and dispersal. They are also planning to bring the increasing social activity into the light by adding web parts that display most-viewed content for a site, most recent searches, and top-rated content.

We've followed this fictional company through the stages of publication maturity at a rapid pace, rather like a time-lapse film of a flower blooming. The reality is that this kind of progression represents a major investment of time and resources, not just by IT staff, but by all the content owners in the company. It requires changing culture and work habits, learning new tools, and leaving the comfort zone. It's helped along by executive support and by a commitment to making the time for this new system, instead of just piling the tasks onto an already overflowing workload. In this story, there were no loud naysayers, no silent saboteurs, no talk of inadequate budget or of how IT's other responsibilities got done. Many companies are already quite good at being resistant to change and not investing enough time and resources in transformative projects, so I don't see a need to discuss this. I'd rather create a vision of what the ideal could be, what SharePoint publication could be at its full potential.

Evaluating Your Own Publication Maturity

As you consider the publication maturity of your intranet, ask yourself these questions: How well does it succeed at being a one-stop shop for information? When employees browse for specific content, are they successful? Does the taxonomy, layout, and structure of the intranet—its information architecture—make sense, and can it evolve as the company evolves? Is the content static or dynamic? Is it directed to everyone equally, or targeted to specific groups, or personalized to the user? Do employees trust that what they see there is fresh and well maintained? Do they have a consistent and intuitive user experience as they travel from page to page? All these considerations define your publication strategy and together will influence its maturity and success.

Summary

If you are trying to get your arms around your SharePoint implementation, decide where to invest, and demonstrate progress, the SharePoint Maturity Model framework can help you. I developed it as a veteran of over 50 SharePoint implementations, and it has been vetted and refined by experts around the globe. It will evolve as the product evolves, and it will guide your implementation through its own evolution, providing direction to the iterations and phases that lead to a mature solution.

If you'd like to learn more, and find the tools and resources to do a full self-assessment, visit SharePointMaturity.com.

Empower the Power User

Kerri Abraham

Many companies have a blanket “No SharePoint Designer” policy due to the risk the tool poses to the SharePoint environment. However, the real risk to the platform is not Designer, but a lack of development discipline. More than just opinion, this chapter introduces a model for empowering users that does not compromise the IT department’s need for strict oversight. By combining basic configuration and project management theory with the features of OneNote, a SharePoint library, and a few content types, it is possible to provide users with the opportunity to build powerful solutions and preserve judicious control for IT.

SharePoint Designer: To Allow or Not To Allow? That Is the Question

The SharePoint Designer lockdown debate is a heated one to be sure. What is perhaps most interesting is that this topic rarely falls into that standard SharePoint response, “It depends.” There seem to be two distinct camps: those in favor, typically SharePoint professionals who understand the tool and praise its ability to add another layer of sophistication to business solutions, and those against, mostly in IT departments where great effort is made to lock down and prohibit its use.

Does a SharePoint Designer lockdown in a workplace where strategic goals of engaging employees and enhancing the work environment make sense? What message is sent to users who strive to learn every aspect of the platform’s capabilities but are continually blocked from

the tools that would help them further support their team and build better solutions? The ramifications of enforcing such restrictive policy extend beyond stunting an organization's collaborative culture; ultimately, these lockdowns are oppressing future SharePoint talent.

Do we eliminate risk to the platform at the cost of crippling our most enthusiastic SharePoint evangelists? A study published in *The Journal of Applied Psychology* in September 2011 by the University of Iowa confirmed that workers who feel empowered experienced a higher level of morale and are more productive, and those same attributes further enhance the overall team dynamic.

Empowerment is not something one can bestow upon another, it is merely the act of removing the barriers that control an individual's ability to take action and make decisions about how she works. Successful organizations know that empowered employees are engaged employees. Considering the demand for current SharePoint professionals and the potential for SharePoint maturity that can be achieved with the help of an engaged and capable Power User, why would any organization risk restricting the growth of an individual who shows proficiency with the platform? An empowered employee who knows the business and SharePoint is a very valuable asset!

The Anatomy of a Power User

In the same way that not all rectangles are a square, not all Site Owners are Power Users. The SharePoint learning curve can be a steep one, and depending upon the environment, the work demands, and the opportunities to problem solve, it can take some time for a user to get up to speed. True Power Users will set themselves apart from site owners by their drive to expand their knowledge.

Users with this kind of passion are self-starters, just as likely to point out every flaw with the current process as they are to come into that discussion with ideas for improvement. In addition to having a "lean" mentality toward identifying inefficiencies, a Power User is both curious and courageous, pushing the capabilities of the platform to the limits just for the sake of learning where those boundaries fall. These skills are essential to learn prior to using a more powerful tool like Designer, not just for a Power User, but for anyone developing the platform.

Power Users are those who embrace SharePoint and build incredible business solutions for their team. They do not pick up the phone six times a day to ask how to fix a broken hyperlink, delete a document, or add a column. Power Users know how to manipulate lists, sorting, filtering, and grouping metadata like a pro. A seasoned Power User can answer End User problems without ever seeing the site or the actual issue. Sticky topics like content types and permissions are second nature to these folks.

Anyone developing a skill set such as that has obviously been bitten by the SharePoint bug, and that is a hard bite to shake. These rare folks are passionate about learning everything they can, and blocking them from expanding those lessons to SharePoint Designer is a shame. The

only possible explanation for keeping someone with adequate knowledge of the platform from a tool like Designer is fear.

Fear

These lockdowns are the borne from fear of losing control. To be perfectly honest, that fear is completely valid; SharePoint Designer in the hands of the wrong user can result in some rather serious damage. However, a blanket policy for a platform that was designed with the intent to empower business users to build their own solutions is not a courageous method for managing that fear.

It is widely accepted that SharePoint is not a product, but a platform. As such, it requires some customization and development to be useful. It is inevitable that every Power User will eventually come into contact with the greater SharePoint community and have their eyes opened wide to the myriad solutions that can be created, many of which require more advanced tools like Designer. For SharePoint maturity to advance, both in the skill set of the user and for the organization, some forward thinking is required where the benefits of using Designer are seriously evaluated. After all, aren't the goals on both sides of this endeavor the same? Everyone wants to see business prosper, and absolutely no one *ever* wants to be the cause of a site restore from backup!

Defuse: Server/Site Restore

The argument that a knowledgeable Power User might possibly crash the server if allowed access to SharePoint Designer is a weak one. The likelihood of such an outcome is rather remote, especially for a business user (developers, however, might run a higher risk of hosing things). Most SharePoint experts agree that it takes an extreme set of circumstances coupled with a dangerous bit of ignorant abuse to actually bring down a server with SharePoint Designer. This crashing the server notion is little more than a deceptive rumor used as propaganda by those who misunderstand the tool.

Now, the reality of crashing a site is a bit more probable. It happens; it's actually easy to do and it is sometimes extremely challenging to fix. However, it should never happen in production. Power Users should learn to use Designer in a test environment, which means they need to be provided one that is kept updated with production.

The use of a staging or test environment is fundamental to the reliable maintenance of any platform, but it is nonetheless true that some IT departments do not provide such an area for their Power Users. For the bulk of the customizations discussed here, a test environment would not necessarily require a separate farm or even web application; a Sandbox site collection will suffice.

Platform Legacy

Take SharePoint out of this discussion and realize that building a business platform requires a team effort. Business users should be recognized as the primary content contributors, while IT's role is centered on creating a sustainable environment. The crux of this debate is entirely dependent upon each side respecting the other's role and finding the middle ground that will both support and sustain the development that results in a successful deployment.

Easier said than done! How is IT supposed to ensure the future viability of countless business-critical solutions and accommodate such a varying degree of platform development? Most will reference the need for governance, but SharePoint provides an array of powerful tools right out of the box that allow Power Users to build elegant and somewhat complex solutions without ever laying hands on tools like Designer. What happens when those users leave? Who supports those solutions when they are gone? Did they leave behind any kind of documentation? Were they ever asked to?

Ironically, what should send a shiver of fear up the spine of any IT support team is not a handful of disciplined Power Users granted access to tools like Designer, but rather one rogue Power User building business-critical solutions without a note of documentation. Each user is a silo and when that silo leaves the business, that knowledge is lost. When a group of power users work together to learn the platform and document the journey, the team learns and leaves behind resources for further knowledge sharing. The fear should not be of more powerful tools like Designer, but of creating silos of knowledge that potentially deprive the business of crucial skill and understanding when those users depart.

Middle Ground: Configuration Management

So now the foundation is laid and the argument has been made that Power Users should be allowed to meet their full potential with the aid of SharePoint Designer. The business benefits from this more engaged and empowered work force, and because the tool is only given to those showing an advanced understanding of the platform, the risks have been mitigated. However, as SharePoint maturity grows, there is an even more menacing threat to the business in the absence of configuration management.

Traditionally, configuration management is a process of recording changes in either software or hardware systems with the intent of providing a traceable map of development and/or assets. Most would argue that SharePoint does not fall within traditional software or product categories, but the same principles of configuration management apply perfectly to SharePoint development in that the successful evolution of the platform (in the form of future upgrades) is dependent upon a comprehensive inventory of the solutions it is supporting.

The concept of empowerment may not seem immediately tied to that of configuration management, but the elements of planning, approval, documentation, and change management bridge that gap between business and IT in a manner that is enormously

beneficial to both sides. Any time a business-critical solution is created without supporting documentation, the future sustainability of that solution is at risk. The larger the environment, the larger the risk, since the right hand cannot possibly always know what the left is doing, nor why it is doing it, but at some point it will have to be supported just the same.

Developing, Development, Developer

The concepts introduced here hinge on the fact that the platform is being developed and, as such, that development must be documented, regardless of titles. Some SharePoint developers are quite vocal about the notion of sharing the word “developer” with business users. Squabbling over a title completely negates the core issue of not capturing business and project requirements for solutions that users are dependent upon for their work. Power Users are not SharePoint developers, however they are developing the platform and should be taught the disciplines that conscientious developers know to be fundamental to their practice.

One reason SharePoint does not fit the model of traditional software or product development is because it allows so many to participate in the process; the very reason why a method for managing it is so important. Documenting SharePoint development should be a marriage of disciplines between project and configuration management that provides structure and understanding without being overwhelming to maintain. This central repository should meet the key needs of IT by providing an overview and auditing process, and integrating features of approval workflow and review cycles, while providing business users an educational resource for sharing and expanding their knowledge.

Luckily, SharePoint is perfectly suited to practically any form of content management. The real challenge is in developing the discipline needed to maintain the solution. This is where IT has the advantage in stating, “No discipline? No empowerment.”

Solution: The SharePoint Rudder

I liked the imagery of creating a steering mechanism or guide for SharePoint development, so I’ve named my solution the “Rudder.” The foundations for the Rudder are based on an ingenious Scripting Resource Center that Mark Miller first introduced on EndUserSharePoint.com in January 2010. With the addition of Microsoft’s OneNote and few other site assets, Mark’s resource center can easily be expanded into a configuration management system.

The addition of OneNote is the only significant change made to Mark’s suggestions, so an overview of “Building a SharePoint Scripting Resource Center” on EndUserSharePoint.com would be of benefit to anyone interested in creating this solution. Mark’s original post uses a wiki site to meet the documentation needs and certainly is the best option for those without a user license for OneNote.

NOTE

As with every SharePoint solution, there are hundreds of possible options. Those outlined here are meant to get the beginner started. These concepts can be expanded upon depending on the level of development and the needs of the organization.

Getting Started

Like Mark's Resource Center, the Rudder should sit at the top of the site collection. My solution is contained in one document library. The location of the library and/or the need for it to be a separate subsite is entirely dependent upon the architecture and permission structure of the site. The library/site permissions must be set to read-only for all staff, granting elevated permissions to those contributing to the documentation.

From Site Actions, choose New Document Library. I plan to use this library to get quick proof of concept ideas and to capture documentation notes, so with that in mind, upon creation of the library, I have chosen the default options of the template to be a web part page document type (see [Figure 2-1](#)). This facilitates creating web part pages that are handy for all kinds of quick mockups and provides another option for restructuring and filtering information captured in this library on dedicated pages as this resource grows.

The screenshot shows the 'Name and Description' section of a SharePoint library settings page. The 'Name' field is 'SharePointRudder' and the 'Description' is 'Resource area for all SharePoint assets and documentation related to configuration management of the site.' Below this are three sections: 'Navigation' with 'Display this document library on the Quick Launch?' set to 'No'; 'Document Version History' with 'Create a version each time you edit a file in this document library?' set to 'No'; and 'Document Template' with 'Document Template:' set to 'Web Part page'. Red boxes highlight the 'No' radio buttons and the 'Web Part page' dropdown selection.

Name and Description Type a new name as you want it to appear in headings and links throughout the site. Type descriptive text that will help site visitors use this document library.	Name: SharePointRudder Description: Resource area for all SharePoint assets and documentation related to configuration management of the site.
Navigation Specify whether a link to this document library appears in the Quick Launch.	Display this document library on the Quick Launch? <input type="radio"/> Yes <input checked="" type="radio"/> No
Document Version History Specify whether a version is created each time you edit a file in this document library.	Create a version each time you edit a file in this document library? <input type="radio"/> Yes <input checked="" type="radio"/> No
Document Template Select a document template to determine the default for all new files created in this document library.	Document Template: Web Part page

FIGURE 2-1. Settings for the SharePoint Rudder library

Name the library without spaces. After creation, use the Library Tools, Library tab to modify the Library Settings and edit the Title field to make it more readable, as shown in [Figure 2-2](#). Create all lists and library names without spaces and edit after creation to include the spaces for readability.

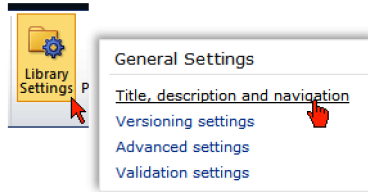


FIGURE 2-2. Create all lists and library names without spaces and edit after creation to include the spaces for readability

Navigational settings are dependent upon the location of the site, so set the Quick Launch settings to those that make the most sense to site structure. Creating a version history with each edit once OneNote was linked to the library caused a duplicating folder issue, so I have opted to turn versioning off.

Content Types

The concepts of using content types can be difficult to grasp, but the best way to learn them is to jump in and try it out. Content types provide structure for related types of content while allowing unique actions to take place based on specific properties. A content type also acts as a wrapper that gives items more presence in the database so they can be surfaced in other ways if the need arises.

Figure 2-3 represents the structure of the Rudder’s library content types.

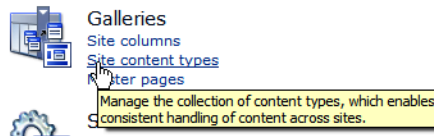


FIGURE 2-3. Site content types under the Galleries heading

Those familiar with the process can use the list as a quick guide. For anyone new to creating content types, the settings can be found under Site Actions, Site Settings under the Galleries subheading (see Figure 2-4). Click Create at the top of the Site Content Type page to get started (see Figure 2-5). If these settings are not available to you, check with your site administrator.



FIGURE 2-4. Choose Create to get started

Name: SharePoint Rudder

Description:

Parent Content Type:
 Select parent content type from: Document Content Types
 Parent Content Type: Document
 Description: Create a new document.

Put this site content type into:

Existing group:
 Custom Content Types

New group:
 SharePoint Rudder

FIGURE 2-5. Setting for the parent content type SharePoint Rudder

The content types described here are just suggestions. Depending upon the needs and level of development in the organization, the library can be expanded or even divided should the need arise. The structure of the library should make the process of documenting as simple as possible. Developing an engaging process or method of capturing documentation is really the key takeaway; the structure of the library should be based on the needs of the user.

The base or parent content type within this library structures the OneNote documentation, but when metadata is used in conjunction with OneNote, there is a tendency for the library to require check-in of each document even when metadata is not required. So I purposefully left out capturing metadata in this instance and added no additional columns. The parent content, then, is simply based on a document content type. Create a new group called “SharePoint Rudder” and put all related content types in this group as they are created (see [Figure 2-6](#)).

SharePoint Rudder	OneNote Documentation	
- Name		Parent: Folder
- Title		
Rudder Folders	Organization/putability	
- Title		Parent: Document
- Name		
- Keywords		
Assets	Templates - Tables - Buttons	
- Title		Parent: SharePoint Rudder
- Team Use		
- Choice: Team A, B, or C		
- Site URL		
- Multiple Lines of Rich Text, to allow for many URL/locations		
- Asset Type		
- Choice: Tables, InfoPath, Templates, etc.		
Pix	Site Images	
- Name		Parent: Picture
- Preview		
- Title		
- File Type		
- Picture Size		
- Comments		
- Keywords		
- Preview image URL		
Scripts Library	SPServices - jQuery - Scripts	
- Name		Parent: SharePoint Rudder
- Title		
- Comments		
- Keywords		
- Original Author		
- Original Location		
- Version		
- Script Type		
- Choice: Javascript, JQuery, JQuery library, SPServices, Other, etc.		

FIGURE 2-6. Quick guide to SharePoint Rudder content types

Folders

OneNote is the real star of the show and, since OneNote automatically creates folders via the native linking interface within a SharePoint library, I have opted to show the folder a little love and take advantage of the options that incorporating folders provides. Folders allow a level of comfort and *putability* (correct placement) of documentation that is particularly useful when using multiple content types. Folders may be lesser players, but when used in conjunction with (and not in lieu of) metadata, they do earn their spot in the SharePoint tool belt.

Creating folder content types is the same as creating any other content type. In this case, use the parent type of folders (Figure 2-7).

The screenshot shows the 'New Content Type' dialog box in SharePoint. The 'Name' field is filled with 'Rudder Folders'. The 'Description' field is empty. Under 'Parent Content Type', the 'Select parent content type from:' dropdown is set to 'Folder Content Types', and the 'Parent Content Type:' dropdown is set to 'Folder'. The 'Description:' for the parent type is 'Create a new folder.'. Under 'Put this site content type into:', the 'Existing group:' radio button is selected, and the dropdown is set to 'SharePoint Rudder'. The 'New group:' radio button is unselected, and its corresponding text box is empty.

FIGURE 2-7. Rudder Folders content type

Add from existing site columns to capture any additional keywords or consider adding a managed metadata field (Figure 2-8).

- Add from existing site columns
- Add from new site columns
- Column order

FIGURE 2-8. Keywords are added from existing site columns

Put It Together

Here is where it starts to come together. There are more content types to build, but if the library and these first content types are structured now, the rest of the build can be documented in the notes, getting this ball rolling and illustrating perfectly how slick this solution works.

Going back to the SharePoint Rudder library previously created (see Figure 2-1), use the Library Tools, Library tab to find Library Settings (see Figure 2-9).

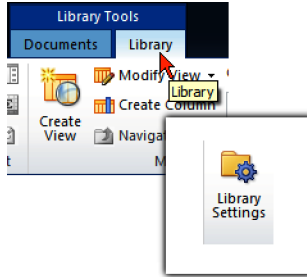


FIGURE 2-9. Library Settings must be modified to allow for additional content types

Within the Library Settings, under the General Setting grouping, choose Advanced Settings (Figure 2-10).

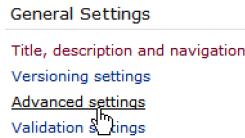


FIGURE 2-10. Configure the Advanced settings to optimize the library configuration and minimize confusion

There are three settings to change:

- Allow management of content types? Yes
- Make “New Folder” command available? No
- Should this document library be a site assets library? Yes

Clean Up

Even though the default of Web Part Page was picked for the document template, the Content Type is listed as Document, and that should be edited (see Figure 2-11).

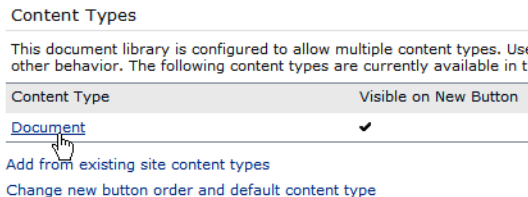


FIGURE 2-11. Click Document to change the title to web part Page

Renaming the content type WPP (short for Web Part Page) will eliminate confusion as further content types are added and the settings within the library are configured (see [Figure 2-12](#)).

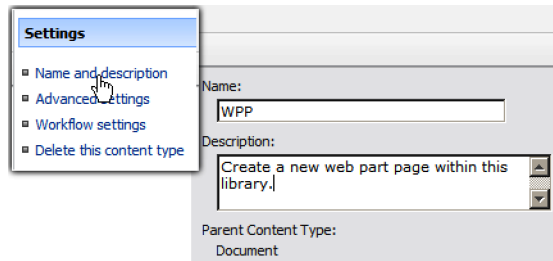


FIGURE 2-12. Rename the default content type from Document to WPP

Now add the two content types previously created (SharePoint Rudder and Rudder folders) to this library from the Document Library Settings page. Select Add from existing site content types. Select the SharePoint Rudder group to filter out the other site content types, and move them over to the Add column (see [Figure 2-13](#)).

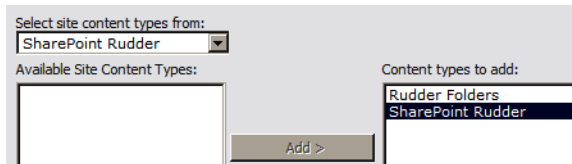


FIGURE 2-13. Add the first two SharePoint Rudder content types

One more setting to configure in this library is to show the toolbar, which is needed in order to structure this library with the appropriate content types and folders. From the All Documents default view of the Rudder library, click Site Actions, Edit Page (see [Figure 2-14](#)).

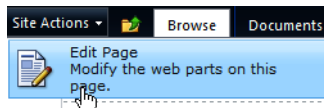


FIGURE 2-14. Select Edit Page to modify the toolbar settings for this library

Use the drop-down menu to edit the web part and make the toolbar visible (see [Figure 2-15](#)).

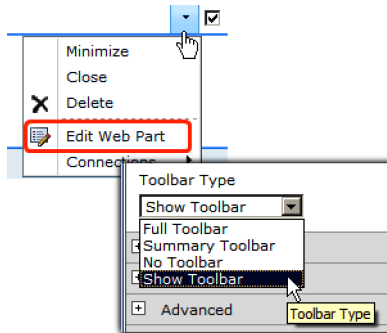


FIGURE 2-15. Select Edit Web Part to change the Toolbar Type to Show Toolbar

From New on the toolbar, there are new content type options, the two previously created and the default web part pages (see [Figure 2-16](#)).

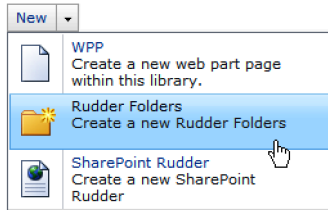


FIGURE 2-16. Three types of content options are now available in this library: the WPP default and the two additions

Create a Rudder folder for each content type listed in [Figure 2-3](#) and add one to house web part pages (see [Figure 2-17](#)).

<input type="checkbox"/>	Type	Name	Title
<input type="checkbox"/>	Folder	Assets	Assets
<input type="checkbox"/>	Folder	OneNote	OneNote Documentation
<input type="checkbox"/>	Folder	Pix	Pix
<input type="checkbox"/>	Folder	Scripts	Scripts
<input type="checkbox"/>	Folder	WPP	Web Part Pages

FIGURE 2-17. Create a Rudder folder for each type of content that will be stored in the Rudder library

With or without the addition of folders, the configuration of this library will integrate perfectly into SharePoint Documentation Sets.

If OneNote did not already default to folders, I might opt to leave folders out, but the addition of folders allows for more finite permissions control, provides a way to limit the type of content in each, and is handy for applying metadata to all items stored within. Start with the OneNote folder and select Change New Button Order in the Edit menu (Figure 2-18).

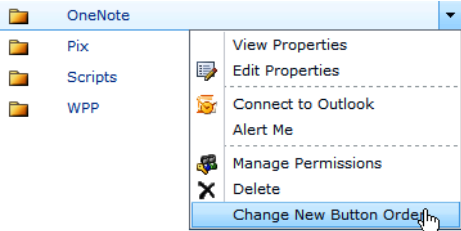


FIGURE 2-18. Use the menu options to control which content types can be created within each folder

Set the Visible content types on the OneNote folder to SharePoint Rudder, the parent content type created just for housing OneNote (Figure 2-19).

Visible	Content Type	Position from Top
<input type="checkbox"/>	WPP	2
<input type="checkbox"/>	Rudder Folders	3
<input checked="" type="checkbox"/>	SharePoint Rudder	1

FIGURE 2-19. Only the content type checked as Visible is the option available inside the folder

Use the same technique for the WPP folder and limit the content within to just web part page creation (the default template used to create the library, Figure 2-20).

Visible	Content Type	Position from Top
<input checked="" type="checkbox"/>	WPP	1
<input type="checkbox"/>	Rudder Folders	2
<input type="checkbox"/>	SharePoint Rudder	3

FIGURE 2-20. Repeat the process and limit the WPP folder to only WPP content type

Once this is set, the only option under New is WPP and the description is clearly displayed (Figure 2-21).

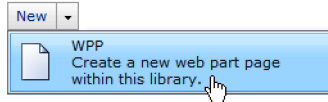


FIGURE 2-21. Example of how the folder can limit content type creation and placement within a folder

Connect Up OneNote

Now that the folder is created and ready to receive content, OneNote can be added and the documentation can commence. Open OneNote and, on the File tab, click New (Figure 2-22).

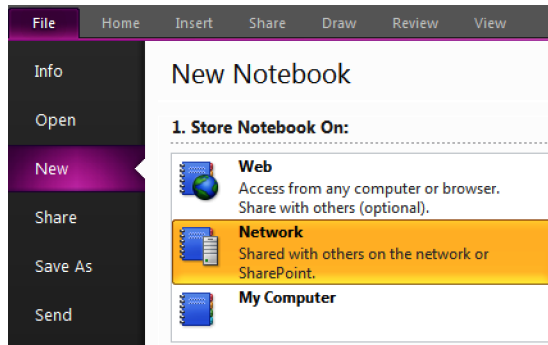


FIGURE 2-22. Store the new notebook on the network

Name the notebook and paste in the URL from the SharePoint Rudder library (since the library is new, it is unlikely to be listed under Recent Locations). Remove the `/Forms/AllItems.aspx` portion of the URL (see Figure 2-23).

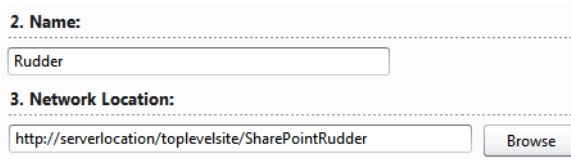


FIGURE 2-23. Use the URL from the newly created library to store the notebook in the correct network location

Browse to the OneNote folder and then create the notebook (see Figure 2-24).

Kerri Abraham SharePoint Rudder		
Type	Name	Modified By
Folder	Assets	Kerri Abraham
Folder	OneNote	Kerri Abraham
Folder	Pix	Kerri Abraham
Folder	Scripts	Kerri Abraham
Folder	WPP	Kerri Abraham

FIGURE 2-24. Pick the correct folder to store the OneNote notebook

A NOTE ON ONENOTE

OneNote has nearly seamless integration with the rest of the Microsoft Office suite, including, for example, the ability to send Outlook messages directly to any notebook page with a right-click. That same functionality carries over to the browser, too, enabling users to highlight and right-click to send any content to a page in OneNote. The notebooks stored in SharePoint follow the same security awareness as the rest of the site, but OneNote also has versioning options and can display the name of each author next to his contributions.

OneNote gives options for creating videos or audio recordings. It has image capture with the Screen Clippings options that, like all other items captured in the notebook, always records the URL of where the item came from. I suspect that Microsoft made OneNote just to document SharePoint development, the functionality is so perfectly suited!

Organizing the Notebook

If you are not yet familiar with how OneNote works, click Help in the upper left corner of the screen and watch the video to get started; it is quick and surprisingly educational. OneNote can be configured in hundreds of ways, and [Figure 2-25](#) represents the basics I am capturing as I document this build. The search functionality in OneNote is phenomenal, so organizing the notes can really be left up to what makes the most sense to the user. Since OneNote captures URLs with copy and paste, any time there is a question about related documentation from a site, a search on that URL should surface all related items from the OneNote library. Brilliant!

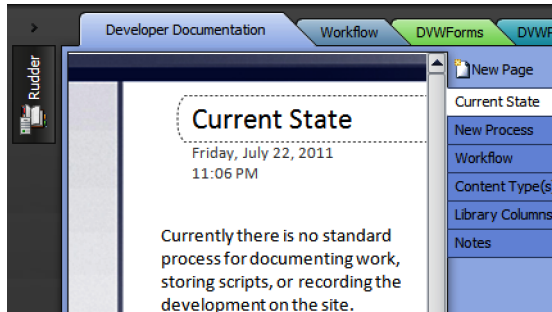


FIGURE 2-25. Tabs and pages offer great flexibility when documenting solutions

Content Types Continued

Okay, so now to finish up this library configuration and how that might look as OneNote documentation. The rest of the content types are built into the library in the same manner as the first two. All Rudder content types should go into the SharePoint Rudder group previously created, as it is a handy organizational method (but if you forget, the settings within the content type do allow for editing so that group placement can be corrected). There are three content types left to create: Assets, Scripts, and Site Pix (Figure 2-26).

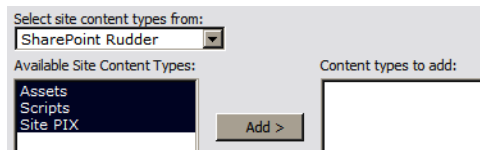


FIGURE 2-26. Remaining content types to add

Assets

In our environment, we use SharePoint wikis for eLearning and procedure manuals. To quickly create wiki content, we depend heavily on Microsoft Word's ability to publish to a SharePoint blog. We then copy the blog post to a wiki page and completely eliminate the need for uploading pictures in the process. This method depends on Word templates for a consistent look and feel. Templates like these are saved in the Assets library.

Additionally, our team site owners use a simple but consistent *button* approach (see Figure 2-27) by creating buttons in Word, converting them into images (you can use OneNote to capture them from the screen and then save them as images), and applying a hyperlink on the site. When these need to be edited or copied, it is great to have them stored as a site asset as well.



FIGURE 2-27. Example of a button created in Word

The Assets content type should be used for whatever assets you used to build the site, which could include everything from document templates, InfoPath forms, custom THMX themes, or PSD files for source code graphics.

Figure 2-28 represents the content type settings for Assets, a child of the SharePoint Rudder parent.

The screenshot shows the configuration interface for a content type named 'Assets'. The 'Name' field contains 'Assets'. The 'Description' field contains 'Templates and other assets used within the site collection'. Under 'Parent Content Type', the 'Select parent content type from:' dropdown is set to 'SharePoint Rudder', and the 'Parent Content Type:' dropdown is also set to 'SharePoint Rudder'. The 'Description:' field is empty. At the bottom, under 'Put this site content type into:', the 'Existing group:' radio button is selected, and the dropdown is set to 'SharePoint Rudder'.

FIGURE 2-28. Settings for the Assets content type

The quick guide in Figure 2-3 illustrates the suggested metadata needed. Once created, copy the columns from the Site Content Type Information page and paste them into the notebook, both for reference and to see how cleverly OneNote displays the information (Figure 2-29).

The copy and paste feature in OneNote is the absolute best reason to use it! OneNote automatically captures the “Pasted from” URL so the user never has to consider it. Additionally, when using highlight copy and paste directly from any settings page in SharePoint, OneNote formats it nicely and all the links stay active, so when questions arise or review is needed, OneNote provides a link (or many) directly to that location on the site.

Columns			
Name	Type	Status	Source
Name	File	Required	Document
Title	Single line of text	Optional	Item
Team Use	Choice	Optional	
Site URL	Multiple lines of text	Optional	
Asset Type	Choice	Optional	

Pasted from <http://serverlocation/toplevelsite/_layouts/ManageContentType.aspx?ctype=0x01010081b3412f11bf7c40bd99d302f7c98ae601&Source=http%3A%2F%2Fspdwiki%2Ecom%2Fkabraham%2F%5Flayouts%2Fmngctype%2Easpx>

FIGURE 2-29. The content type columns as they display in OneNote

Images

Site Pix is a Picture content type based on the parent Document and uses no additional columns from the default (see Figure 2-30). Some views are disabled when combining Picture with Document content types, so for more robust image needs, a dedicated picture library is a better choice. However, since the bulk of my images are automatically published to the Photos library on the blog site (with a consistent naming convention and reduction in file size), I use this Pix content type to store the pictures used across the entire site collection, images such as the buttons we create in Word. Some scripting solutions are also dependent upon images, so those will be stored here as well. This content type is just for the image—the source code/document behind their creation will be stored in Assets.

Columns			
Name	Type	Status	Source
Name	File	Required	Document
Preview	Computed	Optional	Picture
Title	Single line of text	Optional	Item
File Type	Computed	Optional	Picture
Picture Size	Computed	Optional	Picture
Comments	Multiple lines of text	Optional	Picture
Keywords	Multiple lines of text	Optional	Picture
Preview Image URL	Hyperlink or Picture	Optional	Picture

FIGURE 2-30. No changes are made to this default content type except to delete the Date Taken column

Scripts

Mark's original post separated the jQuery and SPServices libraries from scripts, but my scripting needs are rather limited, so I have opted to put them all in together. However, the columns employed here are the same as Mark's (see Figure 2-31). In fact, I use this content type for

everything code-related, so I will be storing CSS in here too, since I link absolutely everything through the Content Editor web part (CEWP) using the Content Link field, with references back to this library. That way, if I screw it up in some way, I have a backdoor of sorts to dismantle the mess.

Columns			
Name	Type	Status	Source
Name	File	Required	Document
Title	Single line of text	Optional	Item
Original Author	Single line of text	Optional	
Original Location	Hyperlink or Picture	Optional	
Version	Single line of text	Optional	
Script Type	Choice	Optional	
Keywords	Multiple lines of text	Optional	
Comments	Multiple lines of text	Optional	

FIGURE 2-31. Columns for Scripts content types

Now, speaking of screwing it up, something should be said here about not enabling versioning on this library. I have had the unfortunate experience of tweaking and using a script I found posted somewhere, then playing with it until it broke, but not realizing it until I had uploaded it over the working version. Of course, I did all that in production too (*don't do that!*).

So here is my “Combat Script Stupidity” vow:

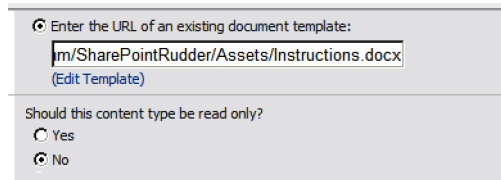
I will test all scripts I use on my site outside of the production environment. To the best of my ability, I will make every attempt to understand the code in the script and what it is doing before I test it. There will be documentation surrounding the use of the script, why it is needed, the origins of where I copied it from, and any and all edits I make will be included in this comprehensive documentation. I will educate myself on best practices, attempt to follow them at all times, and share that advice with those in my environment who may use this library. While my goal may be third-tier, I will fully recognize that I probably should stick to client-side manipulation until I have a better development background. I promise to follow the approval process for use of new scripts in my environment as outlined by my SharePoint administrator/architect.

The reality is that if I was doing more development, I would probably move OneNote out of this library and turn on versioning. My script use is pretty minimal, basic variations on the same script at this point, so this setup meets my current needs and probably the needs of most business users. For this reason, I am opting to keep it all in one place even though that means no version control, but clearly I understand those ramifications.

Tie Up Those Loose Ends

The last thing to do in structuring this library is to get all the content types added to the library and then tucked in under each of the related folders. Go back into the Library Settings and add those last few content types as shown in [Figure 2-26](#) and then restrict the creation of other content inside the designated content folder using the Change New Button Order menu options (see [Figure 2-18](#)).

Once all the folders are set, it is possible to further control the available content types under New by going back into the Library Settings with the Change New Button Order and Default Content Type options. As shown in [Figure 2-32](#), set the SharePoint Rudder as the only visible, and therefore default, option so that all new OneNote items are created on this default content type.



Enter the URL of an existing document template:

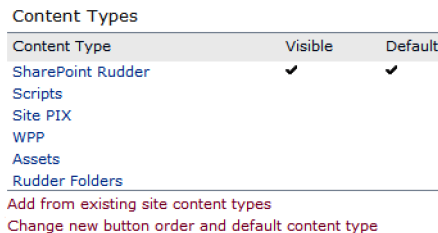
(Edit Template)

Should this content type be read only?
 Yes
 No

FIGURE 2-32. Adjust these settings only after the content types are aligned to the correct folders

The folders are not affected by the default changes made to the entire library; each folder will still allow creation of the specified content. I admit this may be overengineered for the intended advanced user of this library, but using folders provides a powerful guidance mechanism for the end user. Utilizing the ability to create unique document templates for each content type is an excellent way to restrict the creation of new documents to only those meant to reside in the designated folder.

With this set up, the default template of this content type can be used as an instruction guide. Just create a how-to in Word that describes the use of the Rudder and who to contact for more information, save it to the Asset content type in the library, and link it to the SharePoint Rudder's document template (see [Figure 2-33](#)).



Content Type	Visible	Default
SharePoint Rudder	✓	✓
Scripts		
Site PIX		
WPP		
Assets		
Rudder Folders		

Add from existing site content types
Change new button order and default content type

FIGURE 2-33. Settings to save content type templates are under Advanced Settings of the specific content type

Add a note of instruction at the bottom of the how-to to warn users not to save the document to the library; the content type in this situation cannot be set to read-only. This trick combined with the folder technique can be a powerful steering mechanism for End Users in libraries with multiple types of content where specific template needs are required.

The Five “W”s of Documentation

The real challenge to all of this is modifying old habits and developing the discipline of great note taking. OneNote should be the go-to resource for capturing requirements at the initiation of every project. Recording information in real time with project stakeholders is hugely beneficial in defining project expectations. Listening to End Users and integrating their ideas in the process builds confidence in the project success, which transforms into user adoption upon completion.

NOTE

Make it simple. Everyone knows that if documentation demands become too burdensome or tedious, it will not happen. Some may feign time restraints in resistance to use, but what they may not realize is that using that time wisely today in the form of effective note capture can be a real time saver tomorrow! At the very least, capture the absolute basics, most of which can be documented at that project initiation meeting.

Who

Record the *who* of the project. Which people, group, or team will benefit from this solution? Include titles along with names, because employees change, and those supporting this solution three years from now may not know Joe the same way you do.

What

Document the *what* with a simple description of the anticipated solution and any of its unique characteristics. This one may take some brainstorming, or it may be hammered out at that initial stakeholders meeting. Is this a multistep workflow, an InfoPath form, or even just a slick way of dealing with sensitive information using tricky filtering techniques and some simple disposition workflows? Consider which bits of information are most necessary for supporting this solution in the future.

Where

Where is this thing on the site, and what permissions are unique to that location? Simple enough, but imagine the unlikely scenario in which a solution is abandoned and years from

now someone would like to know more about it (can it be deleted, or does it actually serve an archiving role of some sort?). A simple search on the URL in OneNote should surface some notes.

Why

Similar to what, *why* pertains more to current process. Why is the current solution no good? What are people doing instead of using SharePoint? The why can also tie into ROI (return on investment); how much time is wasted doing it the current way? Take a moment to record ROI on those inefficient processes whenever they come up. Showing hard savings is sometimes a challenge, but can lead to justification for the creation of new SharePoint positions.

When

When is this thing going to be ready for use? A go-live date would be handy in the future to examine how long a list or library has been in existence, especially if disposition workflows are deleting all items that are 24 hours old.

OneNote Templates

Complex SharePoint projects may benefit from full project management sites, but even then, OneNote can play a major role in the documentation. Individual OneNote notebooks can be created in multiple locations, and when the project is completed, the notes can be easily moved, merged, or copied with a copy link provided. OneNote also provides a number of templates in the New Page drop-down menu that can be further customized to meet SharePoint needs, as shown in [Figure 2-34](#). Once changes are made, the page can be saved in the Templates Task pane as a template for all future projects.

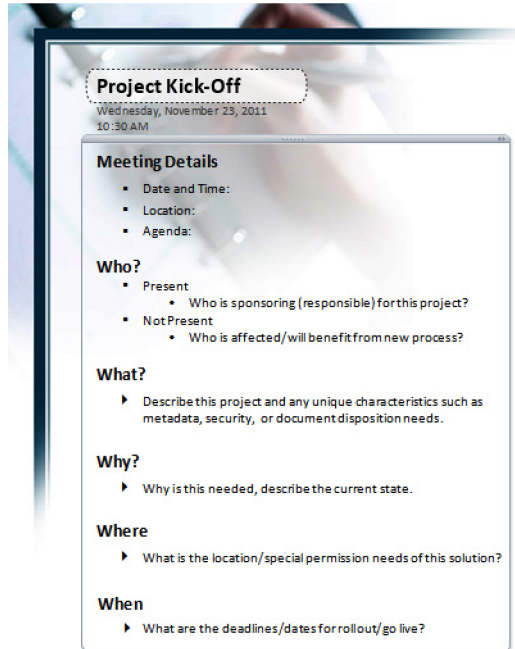


FIGURE 2-34. Custom OneNote template

Documenting Solutions

This should not be an exhausting process if the bulk of the work is done right in the room with stakeholders; the rest is simple image/screen capture with a few coherent notes thrown in for clarification. Just document with purpose. Consider this: if you have the unfortunate luck of being hit by a hog truck tomorrow on the way to work, your notes could easily make the difference between a sustainable or disposable solution.

Content types

A few things should always be documented, and content types are among them. Copy the configuration into OneNote, as shown in Figures 2-29 and 2-33. Document any reasoning behind the need to create a content type and also paste any parent/child relationships. These notes will be valuable should the content types need to be deleted or edited in the future.

Workflow

I have a tabbed section defined in the notebook for workflow, but I also have created a OneNote tag so that any project with workflow, even out of the box, can be tagged for search purposes. As shown in Figure 2-35, right-click applies tags, and you can use the Customize Tags option

to create new ones with labels that fit SharePoint development. OneNote even provides a Find Tags button in the ribbon that you can use to surface all tagged workflow on the site at the time of upgrades or patch testing.

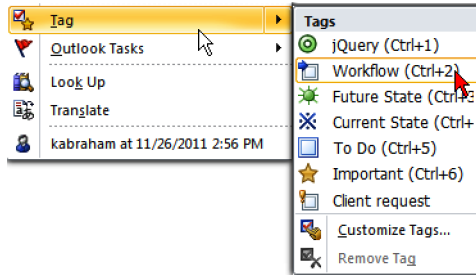


FIGURE 2-35. OneNote tags are a great way to apply metadata to notes

OneNote's Insert tab also has great options for attaching and inserting files into the notebook. The File Printout button will insert the file into the notebook in a readable printout fashion so Visio diagrams or flowcharts describing the workflow scenario can be copied directly onto any page, like the one shown in Figure 2-36.

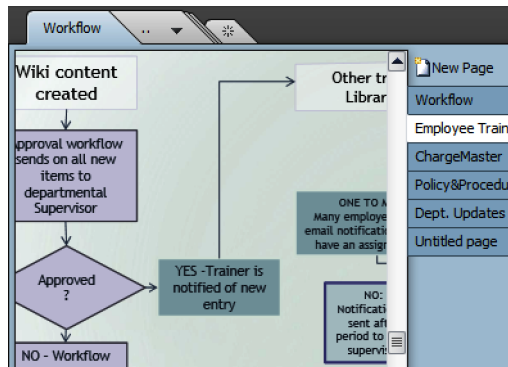


FIGURE 2-36. File printouts render as images in the notebook

Data View web parts/forms

Often, creating a New Item form requires working from an existing paper copy. Some of these forms can have 30 or more columns, which makes creating the lists and the associated forms a challenge. Scanning these forms to OneNote is handy for documenting requirements, plus you can use the drawing tools for highlighting and marking sections as the build progresses (see Figure 2-37).

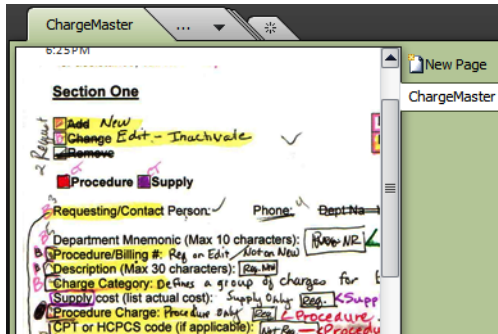


FIGURE 2-37. Scan handwritten notes to OneNote

Scripts

Use of client-side scripts is a privilege, and applying them in my environment means adopting a responsible attitude toward documentation. My notes regarding scripts are a real time-saver. Take Easy Tabs as an example (shown in Figure 2-38). Easy Tabs is Christophe Humbert's awesome JavaScript tabbing solution. You can find it at PathToSharePoint.com. I use Easy Tabs all over my sites to roll up content under other web parts to maximize page real estate. With the latest release of Easy Tabs, Christophe allows for multiple options in colors and functionality.

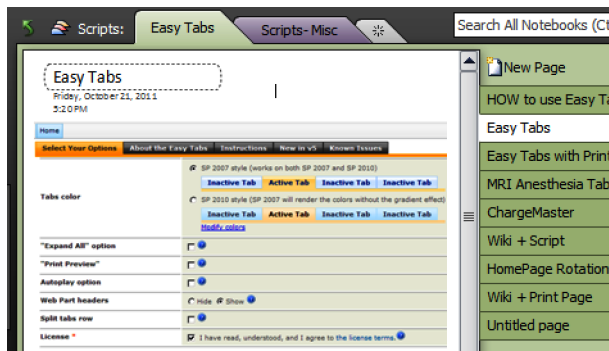


FIGURE 2-38. Documenting with Easy Tabs

I document the configuration of each script with screenshots of which options were picked from Christophe's site (see Figure 2-38), upload the TXT file to the library, then link it to the notebook page with the Attach File options. Since I use the relative URL of the script in the CEWP, I paste that to the page, too. Later, when Easy Tabs are needed on another site, the existing script functions are quick to review and, if they fit the need, I can grab the URL right

from the documentation. Easy Tabs has its own tab in my notebook, and each variation of the script has its own page. To the left of the screen clipping, I record the URL where that script resides across my site (see Figure 2-39).

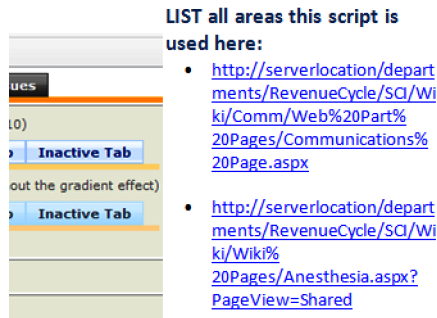


FIGURE 2-39. Capture the use of each instance of script on the site

This copy-and-paste business may seem tedious, but in actuality, OneNote is open so often on my desktop that it is a snap. While the page is open, it is easy to drop in the URL of the new location where this script is being used. A search on the page URL will surface these notes if troubleshooting is ever needed should a page go down and will provide a handy link should something need to be tested at the time of an upgrade.

OneNote provides such excellent linking options that any time resources such as articles or videos are used for reference, they can easily be linked in to the documentation. In this way, the Rudder becomes an incredibly valuable learning tool, one meant to be shared with other Power Users.

NOTE

Power Users should meet on a regular basis. Using the SharePoint community as a model, the goal of the meetings is both for education and support. The SharePoint team should provide regular updates on topics that affect the farm, such as upcoming patch application schedules or other project rollouts. Users should be encouraged to share their discoveries and present their solutions. OneNote should play a major part in these discussions, since exposure to the documentation will encourage a like-minded and more consistent approach toward organizing notes.

Rudder Workflow

Using SharePoint workflow for approval of more advanced development techniques such as those requiring Designer or jQuery is the backbone of this process and the key to providing IT with the oversight they need in order to be persuaded toward empowering users. OneNote

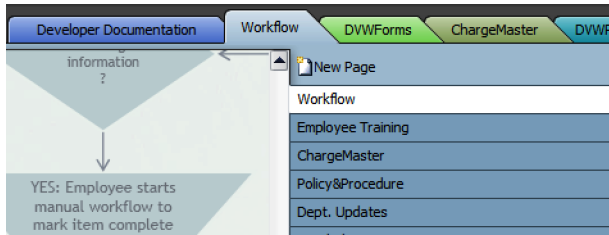


FIGURE 2-40. Tabs running across the top create a new item entry in the library; the pages, listed down the right, do not

Rudder Approval

Approval Needed:	<input type="text" value="11/30/2011"/>
	Date approval is needed.
Search Value/ project name:	<input type="text" value="ChargeMaster"/>
	Add the title of the project or page name to make finding the documentation easier. Please add tag to aid search.
Comments:	<input type="text" value="Please review the workflow solution I have created."/>
	Please add any additional information here.

FIGURE 2-41. Initiation form for a manually started workflow

provides an ideal canvas for documentation, however the structure of notes can be a bit problematic for workflow. Figure 2-40 clearly shows the tabs running horizontally across the top of the page, each of which creates a new item entry in the library. However, the pages within that tab (vertically listed) are not entries on their own. Any workflow employed here would initiate on the entire tabbed section, not a specific page. Of course, pages can be promoted as new tabs—restructuring notes in OneNote is easy enough, but there are a several ways to work around that problem.

You can use an Initiation Form workflow like the one shown in Figure 2-41 to gather more information and further define which pages need review and approval. The form can prompt for the search parameters upon manual initiation of the workflow and pass those values through in an email notification.

Additionally, the OneNote tags can be employed to help with search, too. The tags are easily customized, but even easier to search. A Tags Summary pane is available, as shown in Figure 2-42, that allows for searching on all tags used in the notebook.

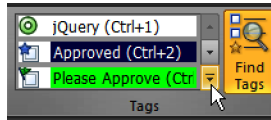


FIGURE 2-42. Use the drop-down options in the Tags section of the toolbar to customize

Empowered Utopia in 10 Steps

These 10 steps are more common sense than a plan; certainly, developers have some variation on this process they follow already. Why not apply the same principles to Power Users, too?

1. A governance policy is enforced that requires that all users with an interest in using SharePoint Designer be taught basic documentation techniques.
2. Power Users are either encouraged to create their own SharePoint Rudder libraries or are provided an organization-approved templated version by the SharePoint team.
3. Upon initiation of every SharePoint project, the designated OneNote template is used and the five Ws are captured.
4. Users are provided a test environment so that no production space is harmed in the expansion of SharePoint skill.
5. Power Users are granted access to SharePoint Designer, and creation of powerful business solutions commences only in the test environment.
6. The initiation meeting notes are combined with more documentation related to the solution being developed.
7. Once the working solution is complete, approval workflow is sent to the SharePoint team asking for review. This includes links to the test solution and the OneNote documentation.
8. Approval is granted, and the Power User builds the solution in the production environment to much acclaim of her fellow employees/stakeholders.
9. At the monthly SharePoint Power Userusers meeting, this solution is highlighted and demonstrated so the knowledge is shared and more people benefit from this example.
10. After 18 months, a workflow is triggered, asking the original creator of the solution for a review—this means making changes to ensure all development on the platform stays up to date.

Configuration Management Is Thoughtful Maintenance

Every organization will have its own notions of what documentation needs to be captured, but keep in mind that simplicity is the key to creating good habits and winning adoption. It should be understood that consistent documentation efforts are the means to empowerment.

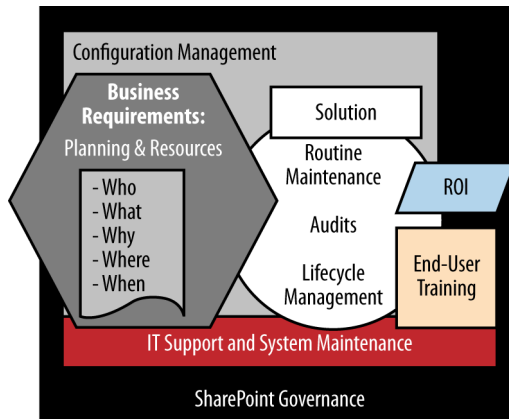


FIGURE 2-43. Configuration management is everyone's responsibility

Everyone developing the platform should be held to the same standard, and those who hold dedicated SharePoint roles should be leading by example.

Teaching users to gather basic business requirements should not be an option, but should be mandatory prior to beginning development. Every business-critical solution on the platform should be developed with future sustainability in mind. That effort, however, does not have to be exhaustive and cover every aspect of the build, but, rather, should be based on which facts must be present to support that solution in the future.

Both IT and business users share the responsibility of SharePoint support. [Figure 2-43](#) represents each solution on the platform. In an ideal configuration, every solution follows a governance policy where the rules and guidelines act as a mechanism for future sustainability. Within that governance, IT is committed to general support and maintenance of the platform and plays a role in both business/Power User and End User training efforts.

The business solution, the requirements gathered to build it, and all maintenance needed to sustain it for the future fall under configuration management restraints. These are the burden of the developer to supply, regardless of what title that person holds. Without documentation requirements and consideration for change management, any calculations for ROI are threatened should the sustainability of the solution come into question.

Empowerment Without Responsibility Is Chaos

This SharePoint Designer lockdown debate has long been centered on the potential chaos that can occur should it be allowed, but I say those days are over! Empowering workers does not mean simply opening the floodgate to a mess of hodge-podge development. Those who build solutions love the platform and do not want to jeopardize what they have created. Smart Power

Users recognize the opportunity to master Designer as a career opportunity, and the last thing they want to do is jeopardize that with rogue development. The exchange of configuration documentation as a concession for empowerment is an advantageous arrangement for both sides.

The platform is at risk when anyone developing solutions is missing fundamental SharePoint understanding. Those who jump directly into Visual Studio without first mastering out-of-the-box options can cause far more damage than any Power User with access to Designer. The outcome of such ignorance is often revealed years after a fool's departure, when all upgrade attempts are thwarted because some "reinvent the wheel," business-critical solution was created in Visual Studio instead of using two content types and an out-of-the-box workflow (all without a note of documentation).

Summary

Achieving SharePoint maturity is dependent upon an empowered workforce. The future promises cutting-edge techniques to engage employees through enhanced collaboration tools, and SharePoint is already a major player in that arena. Developing a skill set from within the organization that can meet those demands is a savvy strategy.

SharePoint takes vision! Employers who allow IT to block access to tools and handicap employees and the solutions they create are stuck in traditional software mindset. Innovators know that SharePoint skill is one of the most sought after in the IT industry and they are leading the charge by stealing great talent from those who have yet to recognize its value!

These restrictive lockdown policies only enforce the idea that business and IT are on opposite sides, but the truth is that they share the same goals of success. SharePoint is the bridge that facilitates success, and documenting the development is the support that keeps that structure strong (see [Figure 2-44](#)). I predict these lockdowns will soon be a thing of the past, but the ideas behind the need for configuration management in SharePoint? That is the stuff of the future.



FIGURE 2-44. SharePoint bridges the gap between IT and business

jQuery to the Rescue

Jim Bob Howard

It seems like every time I turn around, I have a project that reaches just beyond what SharePoint will do with out-of-the-box (OOB) functionality. And just about every time that happens, jQuery comes to the rescue to make the job easy, elegant, and robust.

Because SharePoint uses web technologies, we're blessed with being able to use JavaScript to manipulate the presentation layer programmatically. One of the most popular JavaScript libraries today is called jQuery. jQuery gives us an easy way to manipulate the document object model (DOM) of the web pages created by SharePoint. This all happens in the browser after the page has been served up and rendered. But, it's so fast, many of its features appear to happen on load. Others can be utilized with JavaScript to allow the page to change based on user interaction.

In this chapter, I've only touched the tip of the iceberg by showing some examples in which you can use jQuery for some great, easy-to-use functionality. When you're done with this chapter, you will know how to use jQuery to effect the following changes on your pages:

Automate all-day events

When you want all of your Calendar entries to be an all-day event, this little lesson shows you how to check the box automatically and then hide it so the user can't change it. The example here is a Mileage Report submission calendar, where only the date matters and not the time of day.

Requesting a review only once

Sometimes you want to solicit feedback on an article or give a quiz on the material covered, but you need each user to submit only one survey. Using jQuery and the jQuery Library for SharePoint Web Services (SPServices, for short), we can make sure they only get the option once.

Default text based on radio button click

Automatically fill in data based on the actions of the user. For an HR application, wherein a list of candidates is considered for a position, only one (at most) will be hired. For all the rest, a reason for *not* being hired is required. For the one happy candidate, jQuery helps us put “N/A” in the box.

Writing a survey ID to a list on response creation (without workflow)

Wow! That title was a mouthful, but I was hoping it would be found by folks having the same problem I was having: namely, workflows don’t fire from surveys. So, what do you do if you want something to happen when a survey response is created? As usual, jQuery comes to the rescue.

Labeled sections on default forms

Here’s one more trick that jQuery lets us do on out-of-the-box SharePoint forms: we can create sections to help group content, which can greatly improve the user’s experience.

So, come follow along and we’ll learn some tips, tricks, and bold opinions for employing jQuery in your SharePoint pages to enhance your users’ experience and make your applications really pop.

Automate an All-Day Event

I was working on a mileage reimbursement project that required a user to associate travel mileage with a specific date. To make things easy for the user, I wanted to use a Calendar list so the user could easily choose a travel date. That would also give me built-in date validation and makes things easier all around.

But there were some things I didn’t need: Recurrence, Workspace, End Time, or times in general. I wanted the functionality of an All-Day Event so that specific times were not necessary. And I only wanted one date associated with the mileage entry. So, I hid the End Time and set up a workflow to copy the Start Time into the End Time on create. Since both fields default to [Today], their “requiredness” is satisfied by default.

I also wanted Recurrence and Workspace hidden, as well as the time portion of Start Time. And I needed All-Day Event to be enabled automatically, and then hidden so it couldn’t be disabled. jQuery to the rescue.

1. Enter Edit mode on *NewForm.aspx*, by adding `?PageView=Shared&ToolPaneView=2` to your URL.
2. Add a hidden Content Editor web part (CEWP, pronounced *soup*) to the bottom of the page.

3. Click Source and enter the following text:

```
<script src="/scripts/jquery.min.js" type="text/javascript"></script>
// Download the current version of the jQuery library from jQuery.com,
// rename it as above and put it in a /scripts directory on your top-level site.
// Make sure the src above matches your path and filename.
<script type="text/javascript">
$(function() {
$("td.ms-datetimeinput").hide(); //hides the times on Start Time
$("span[title=All Day Event] >
input").attr("checked", "checked");
// checks All Day Event

//hide all of the unneeded checkboxes
$("tr:has(span[title='Recurrence'])").not("tr:has(tr)").hide();
$("tr:has(span[title='All Day Event'])").not("tr:has(tr)").hide();
$("tr:has(span[title='Workspace'])").not("tr:has(tr)").hide();
});
</script>
```

You'll need to do something similar in your *EditForm.aspx*:

1. Enter Edit mode the same way as above, leaving the ID=x and changing the ? before PageView to &. For example: `../EditForm.aspx?ID=1&PageView=Shared&ToolPaneView=2`.
2. Add a hidden CEWP to the bottom of the page.
3. Add the same jQuery code from above, but delete the first two lines in the function (since it's already an all-day event, there's no need to set it, and the times will already be hidden).

DispForm.aspx will be slightly different because the HTML is different.

1. Enter Edit mode the same way you did for *EditForm.aspx*.
2. Add a hidden CEWP to the bottom of the page
3. Add the following code to the source:

```
<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script type="text/javascript">
$(function() {
//hide all of the unneeded checkboxes
$('tr:has(td[id=SPFieldAllDayEvent])').not('tr:has(tr)').hide();
$('tr:has(td[id=SPFieldRecurrence])').not('tr:has(tr)').hide();
$('tr:has(td[id=SPFieldCrossProjectLink])').not('tr:has(tr)').hide();
});
</script>
```

Now every new list item added will automatically be an all-day event, and the times will be hidden, making for a richer user experience.

Requesting a Review Only Once Per User

Soliciting user-generated content (UGC) is all the rage these days. And one easy way to start collecting some of that is through a simple survey asking if the content was helpful, if it solved the problem, and if the person has any additional questions or comments. But, for any given piece of content, you probably want each person to only answer the usefulness survey once.

I recently worked on a project in which the users were doctors working toward earning continuing education credits. The doctors were asked to read through a case study and diagnose a fictional patient. When they finished the diagnosis, they could check their answers. To get credit, they had to answer a short three-question survey.

Easy enough: a Case Studies list was created to hold the case studies. In the appropriate place on the DispForm, we provided a link to Take the Survey (Figure 3-1), which also sends along a query string to prefill some column data using a calculated column to insert the CaseName and CaseNum (see Figures 3-1 and 3-2.)

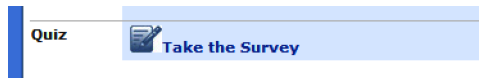


FIGURE 3-1. The Take the Survey link

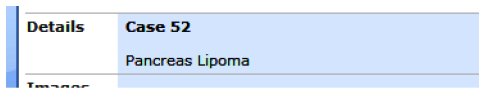


FIGURE 3-2. Prepopulated data in NewForm

The tricky part comes when we want to keep the doctors from taking the survey if they've already taken it. We want to give them a message telling them they can't take it again. In our case, we also want to provide a link to let them see what they answered before, as shown in Figure 3-3.

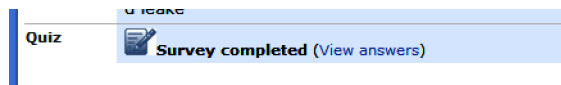


FIGURE 3-3. When the survey has been completed, the link changes to a statement

jQuery to the Rescue

(This section was written with the help of Marc Anderson’s jQuery Library for SharePoint Web Services on Codeplex: <http://spservices.codeplex.com>.)

If you’ve been using jQuery very long, you know that—like with the Automated All-Day Event—most solutions begin with, “Add a CEWP to the bottom of the page and hide it,” and this one is no different.

Here are the basics of what we want jQuery to do for us:

- Look in the survey list and find any records for the current user that match the case number (will be the same as the `CaseNum` parameter in the query string) of the case study we’re currently on.
- If it doesn’t find anything, leave the “Take the Survey” list as-is.
- If it does find something, change the message next to Quiz and provide a link to view the existing survey.

Step 1: Adding the libraries

First we’ll add the jQuery library and the SPServices library to our source so that we have access to them when the page loads.

```
<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script src="/scripts/jquery.SPServices.js" type="text/javascript"></script>
//This assumes you have the latest version installed in the /scripts directory of your top-level site,
//and have renamed it as above. See spservices.codeplex.com for the current versions you should be using.

<script type="text/javascript">
$(document).ready(function() {
//Here's where we're going to do stuff!
});
</script>
```

Step 2: Getting the case numbers

We need find out the case number for this case study. Right-click the page and click View Source or better, using Internet Explorer, Firefox, or Chrome, press F12 to open the developer tools and see the page’s hierarchical document object model (DOM). Search for “Case 52” (in my case) and build a selector to choose that text. My HTML source looks like this:

```
<span id="ctl00_m_g_73269c19_39a3_4cb6_8989_cb7155f6c0f1_ff2_1" style="font-weight:
bold;">Case 52</span>
```

I noticed the `ff2` in the ID and performed another search to see if it showed up again on the page. Since it didn’t, I knew I could always get the case number by using the following selector:

```
var CaseNum = $("span[id*='ff2']").text();
```

This is saying, “Find a `span` element that has an `id` that *contains* (*) the string `ff2`. Then, get the text inside that element.” For details on selectors, go to the documentation on [jQuery.com](http://docs.jquery.com>Selectors) (<http://docs.jquery.com>Selectors>).

Step 3: Getting the last survey item ID

Now that we have the case number, we can use `()`.`SPServices.SPGetLastItemId` to get the ID of the last survey item (if any) the current user created that has the current case number. (For documentation on `SPGetLastItemId`, go to [http://spservices.codeplex.com/wikipage?title=\\$\(\(\).SPServices.SPGetLastItemId.\)](http://spservices.codeplex.com/wikipage?title=$(().SPServices.SPGetLastItemId.)).)

```
var lastID = $(()).SPServices.SPGetLastItemId({
  listName: "Case Survey",
  CAMLQuery: "<Eq>" +
  "<FieldRef Name='Case_x0020_Number' />" +
  "<Value Type='Text'>" + CaseNum + "</Value>" +
  "</Eq>"
});
```

Enter the following information:

`listName`

Use the display name of the list you’re searching.

`FieldRef`

Use the static name of the column you want to search.

`CaseNum`

This is the name of the variable we created to look up the case number.

`lastID`

Contains zero (0) if the user hasn’t created a survey for this case study or, if a survey does exist, it will contain the ID of the last survey for this case study. If there are multiple entries before this is implemented—like in our case—it will return only the last one.

Step 4: Changing the survey link

We’ll only manipulate the Take the Survey link if `lastID` does not equal (`!=`) 0. The following script will yield the view in [Figure 3-4](#) when the survey has been completed.

```
if (lastID != '0') {
  var svyLink = $('td[id^=surveylink]'),
  svyHREF = "/site/Lists/Case%20Survey/DispForm.aspx?ID=" + lastID;
  svyLink.html("<a href='" + svyHREF + "'>" +
  "<img alt='Quiz Complete' src='/images/quiz.jpg' border='0' /></a>" +
  "<strong>Survey completed</strong>" +
  "<a href='" + svyHREF + "'>View answers</a>)"
);
};
```

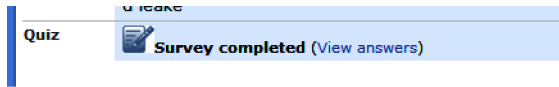


FIGURE 3-4. The Take the Survey link converted to text

Enter the following information:

svyLink

The selector we're using here helps us find the TD that contains our link to the survey. If you are working with a customized *DispForm.aspx*, you can add `id="surveylink"` to the TD. SharePoint, however, will change it to `surveylink{generate-id()}`, so when the page is actually created, it will look more like this:

```
<td id="surveylinkID0EAAA" width="400px" valign="top" ... >...</td>
```

That's why we use `^=`, which means "begins with."

svyHREF

Copy and paste the URL for your *DispForm.aspx* from your browser and paste it (without the number at the end) into the quotes. We'll supply the ID with our `lastID` variable.

svyLink.html

We're replacing everything inside the TD with our new text and link (and old image).

Of course, this doesn't stop the user from going to the survey list and creating another item. But, without the query string, it won't be tied to an actual case study and will be useless anyway. So, within the normal path of giving feedback on a given case study, this will simply restrict users to only create one survey item per case study.

Here's the final complete code:

```
<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script src="/scripts/jquery.SPServices.min.js" type="text/javascript"></script>

<script type="text/javascript">
$(document).ready(function() {
    var CaseNum = $('span[id*=ff2]').text(),
        lastID = $.SPServices.SPGetLastItemId({
listName: "CME Survey",
CAMLQuery: "<Eq>" +
"<FieldRef Name='Case_x0020_Number' />" +
"<Value Type='Text'>" + CaseNum + "</Value>" +
"</Eq>"
    });

    if (lastID != '0') {
        var svyLink = $('td[id^=surveylink]'),
            svyHREF = "/site/Lists/Case%20Survey/DispForm.aspx?ID=" + lastID;
        svyLink.html("<a href='" + svyHREF + "'>" +
"<img alt='Quiz Complete' src='/images/quiz.jpg' border='0' /></a>" +
"<strong>Survey completed</strong>" +
```

```

"(<a href='" + svyHREF + "'>View answers</a>)"
);
});
});
</script>

```

Now the doctors can quickly see if they've already taken this quiz, so they can move on to the next one.

Default Text Based on Radio Button Click

I had a user request to automatically fill in text in a required field based on the selection of a radio button on the page. The form that is being filled out is used by a hiring manager to document the decision process for a certain recruit. Since each position will only have one person hired, but typically more than one applying, the answer to the question, "Was this person hired?" will be "No" way more than it will be "Yes." Therefore, a reason for the "No" is required.

In the relatively rare event that the candidate is hired, the reason for not selecting him becomes moot. But rather than trying to manipulate the "requiredness" of the text box, I wanted to automatically fill in "N/A" on the rare and happy occasion "Yes" is selected.

First, I needed to find the specific radio buttons in question. For some reason, the name of the choice column that generates radio buttons does not include the name of the column anywhere in the input of the button itself. Instead, the first choice looks like this:

```

<span class="ms-RadioText" title="Yes"><input id="ctl00_PlaceHolderMain_TaskForm_ff2_1_
ctl00_ctl00" type="radio" name="ctl00$PlaceHolderMain$TaskForm$ff2_1$ctl00$RadioButtons"
value="ctl00" /><label for="ctl00_PlaceHolderMain_TaskForm_ff2_1_ctl00_ctl00">Yes</label>
</span>

```

and the second choice looks like this:

```

<span class="ms-RadioText" title="No"><input id="ctl00_PlaceHolderMain_TaskForm_ff2_1_
ctl00_ctl01" type="radio" name="ctl00$PlaceHolderMain$TaskForm$ff2_1$ctl00$RadioButtons"
value="ctl01" /><label for="ctl00_PlaceHolderMain_TaskForm_ff2_1_ctl00_ctl01">No</label>
</span>

```

If you're familiar with HTML, you'll quickly notice that the basic structure is an `<input>` tag followed by a `<label>` tag, both of which are wrapped by a `` tag.

Wading through the massive concatenated id, for, and name tags, notice the following structure to the two radio buttons:

```

<spanclass="ms-RadioText"title="choice 1">
<inputid="a long str w/ an ff# and ending with the value attribute of this tag"
type="radio"name="a long string with an ff# which will be the same for all choices in
this group"
value="a 0-based counter, therefore the last digit is 0 for this first choice" />
<labelfor="a long string matching the id of the preceding input tag">choice 1</label>
</span>

```

then:

```
<spanclass="ms-RadioText" title="choice 2">  
<inputid="a long string with an ff# and ending with the value attribute of this tag"  
  type="radio" name="a long string with an ff# which will be the same for all choices in  
  this group"  
  value="a 0-based counter, therefore the last digit is 1 for this second choice" />  
<labelfor="a long string matching the id of the preceding input tag">choice 2</label>  
</span>
```

The `ff#` in the `id`, `name`, and `for` attributes in the previous code will be the same as that found in the `.aspx` page (e.g., `NewForm.aspx`) if you are working from a custom page. For example:

```
<SharePoint:FormField runat="server" id="ff2{$Pos}" ControlMode="Edit"  
  FieldName="ChoiceColumn" __designer:bind="{ddwrt:DataBind('u',concat('ff2',$Pos),  
  'Value','ValueChanged','ID',ddwrt:EscapeDelims(string(@ID)), '@ChoiceColumn')}" />
```

If you are working from the default list form and don't want to customize it, you can learn the `ff#` for your radio buttons by viewing the source of the page (in IE, open up a `NewForm.aspx` file in the browser; right-click and select View Source) or by examining the DOM, using the Developer Tools (F12).

In the source of your page, search for the bold label that precedes your radio buttons (the display name of your column). You should then be able to move forward in the document until you find HTML similar to what is shown above (with an `<input>` and `<label>` tag pair—as shown in [Figure 3-5](#)—nested together inside a `` tag). Find the `ff#` inside the `id` or `name` attribute of the input field. This is what we'll use to find the right radio button.

Using jQuery, we can now identify the radio buttons we're looking for with the following statement:

```
$("input[name*='ff2']:radio")
```

This is saying, "Find the input tag with a type of radio that has a name attribute that contains `(*) ff2`."

Put your `ff#` in place of `myff2` in the example. Options for the name search are as follows:

```
name='ff2'
```

The name has to equal exactly `ff2` (`name="ff2"`).

```
name^='ff2'
```

The name must begin with `ff2` (`name="ff2_1"` or `name="ff2"`).

```
name$='ff2'
```

The name must end with `ff2` (`name="1_ff2"` or `name="ff2"`).

```
name*='ff2'
```

The name must contain `ff2` (`name="1_ff2_1"` or any of the previous examples).

Once we've found the right radio buttons, we can add an `onClick` event to each of them so that whenever either is clicked, a function will run to update our text field.


```

$("input[name*='ff2']:radio").click(function() {
    // do stuff to update our text field
})`

```

Since this same function will run no matter which choice is checked, we want to find the one that is checked, then look at the text value of the label that follows it, as shown in [Figure 3-5](#).

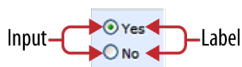


FIGURE 3-5. Each radio button has an input followed by a label

So, next we need to find the text of the label of the radio button that is checked. To do this, we can use the following:

```

$("input[name*='ff2']:checked + label").text()

```

This is saying, “Find the `input` tag that is checked that has a `name` attribute that contains `(*)ff2`. Once you’ve found that, look at the `label` tag that comes next `(+)` and get its text (whatever is wrapped in that tag, such as the `No` in `<label for="idOfInput">No</label>`.” We’ll assign this to a variable (`str`) so we can look at it more than once without having to copy and paste all of that.

```

var str = $("input[name*=' ff2']:checked + label").text();

```

Now we can look at the value of `str` and create an `if` statement that will update our targeted text box accordingly. So, let’s find the text column.

Fortunately for us, text columns include the column name in the `title` attribute of the `input` tag.

```

$("input[title='MyTextField']:text")

```

This should be getting familiar, but just to clarify, this is saying, “Find the `input` tag of type `text` that has a `title` attribute of `MyTextField`.” We want to set the `.val` of this column.

If our text column is multiple lines of text, we’ll use this instead:

```

$("textarea[title='MyTextBox']")

```

For a single line of text, our `if` statement looks like this:

```

if (str == "Yes") {
    $("input[title='MyTextField']:text").val("N/A");
} else if (str == "No") {
    $("input[title='MyTextField']:text").val("");
}

```

Of course, you’ll want to update `MyTextField` or `MyTextBox` with the actual title of your text column. You can find that in your `.aspx` page or in the source as described above.

Putting it all together for a single line of text field, add a hidden CEWP to the bottom of your page (in SharePoint or in SharePoint Designer). Then paste the following code into it (making adjustments where noted), and you're good to go:

```
<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script type="text/javascript">
$(function() {
  $("input[name*='ff2']:radio").click(function() {
    //Change ff2 to your ff#
    var str = $("input[name*='ff2']:checked + label").text();
    if (str == "Yes") {
      $("input[title='MyTextField']:text").val("N/A");
    } else if (str == "No") {
      $("input[title='MyTextField']:text").val("");
    }
    //Change MyTextField to the static name of your column
    //Obviously, you can set the .val to any text that works for you
  }
})
})
</script>
```

jQuery to the rescue again!

Writing a Survey ID to a List on Response Creation (without Workflow)

SharePoint surveys don't fire a workflow. You can create one, but it will never fire.

And the reason is fairly understandable. When should it fire? When is the survey item created? When is it changed? To understand how difficult it is to answer those questions, I need to take a moment to explain how surveys work.

Surveys can take two forms, *branched* or *unbranched*. An unbranched survey looks a lot like any other list and functions almost exactly like it (except for workflows).

But, a branched survey is a completely different proposition.

Unbranched Versus Branched Surveys

When you have a branched survey, the questions are presented a few at a time and depend on the answers from previous questions. This is great, especially when you have a requirement that looks like this:

Do you like ice cream? If yes, which flavor?

In other words, if the answer is *No*, I'm not going to ask you which flavor is your favorite.

SharePoint accomplishes this by starting with a *NewForm.aspx*, just like other lists, but it only shows the first branch of the survey. When the user clicks Next, the answer to the first branch is saved and an ID is assigned to the survey. The next page is displayed using *EditForm.aspx* and

passing in two parameters to the page: the ID of the survey, and the first question of the next branch. This happens each time the user clicks Next, until the last branch, when a Finish button is displayed rather than Next.

At any point, if the user clicks Cancel on an *EditForm.aspx* page, she is prompted with a question about whether the entire survey should be deleted (recycled, actually) or not. So, built into the creation of the survey item is also the possibility that it will be destroyed before all questions are answered, but after some data has already been saved.

Typically, with other lists, a create workflow is fired when *NewForm.aspx* is saved, that is, when the item ID is created. The problem with doing that with a branched survey is that the workflow may need required fields that come later in the survey and so are not available when *NewForm.aspx* is saved.

Likewise, with other lists, a change workflow is fired when something changes and the *EditForm.aspx* is saved. Again, with a survey, *EditForm.aspx* is used in the creation of the item, so a save from it doesn't necessarily mean something has changed; it may be the initial creation of the data for that column.

Why Would You Want a Workflow on a Survey?

It's often necessary or desirable to spawn a new task for a user to complete, which can be handled separately from the creation of the initial item. The challenge comes in when you want to easily link from the creator item to the created item and vice versa. In [Understanding SharePoint Journal's Bonus to Issue 4](#), I explained a method to use the Collect Date from a user workflow action to create that two-way link between a List item and a Task item using a workflow. In our case, we need the ability to ask certain questions based on the answers to other (branching), which a Task can't do, so a Survey is a better option.

In both cases, it's pretty useful to include a lookup column in both items (initiating item and task, or initiating item and survey response) that point to each other. With the former, you can fire a workflow on creation of the task to write its ID to the initiating item. With the latter, you can't.

Real-World Use Case

I was recently asked to create an intake list for work-related injuries and illnesses, which we are calling the Injury Report list. The simple setup allows employees to report when an injury or illness occurs at work and allows HR to see those reports for its legal documentation.

But, because the company is a healthcare provider, blood-borne pathogen exposure (BPE) is one of the possible types of injury/illness that can occur. And if it does occur, there's an entirely separate set of additional questions that need to be documented. Many of these are similar to our ice cream question above: if yes, then answer these additional clarifying questions; if no, move on to the next question.

These additional questions were perfectly suited for a survey. But, the answers have to be tied to the original injury intake report.

So, how can we get these two to talk to each other?

jQuery and SharePoint's Web Services to the Rescue

Using Marc Anderson's [jQuery Library for SharePoint Web Services](#) on Codeplex, we can get the information we need, when we need it, and write out where it needs to go.

The basic steps are:

1. If it's a BPE, email the user a survey response link that includes the Injury Report's ID in the querystring.
2. Add jQuery to the survey's *NewForm.aspx* to fill the lookup on the survey with the ID of the report from the query string.
3. Add jQuery to the survey's *EditForm.aspx* to update the Injury Report with the survey ID.

Step 1: Creating a link to the survey

Since our Injury Report is a list that can execute a workflow, it's easy to send an email message to the user that includes a link to the BPE survey *NewForm.aspx*. We also want to pass the ID of the Injury Report to the BPE survey, and we can do that through the query string. All we have to do is add "?ReportID=" and the Current Item's ID field to the anchor tag's HREF. If this isn't clear, see my separate post called "[Building a Hyperlink in an Email in SPD](#)".

Step 2: Adding jQuery to NewForm.aspx

We'll use Javascript and jQuery to capture the ReportID and update our report lookup column in the BPE survey with it. Of course, that means you'll need to add a lookup column as the first question (or at least in the first branch) of the survey so that it can link back to the Injury Report on creation. This will create a drop-down list as the first question. We're going to use this column to connect the two, but we're going to hide it. So, make it required and no one will be able to submit a survey without supplying a ReportID once we add a little jQuery to it.

1. Point your browser to your survey's *NewForm.aspx* (you don't need the ReportID in the query string yet).
2. Right-click on the page and choose View Source.
3. While viewing the source, press Ctrl+F and find the label text for your lookup field. It should look something similar to this:

```
<TR>
<TD valign="top" width="90%" class="ms-formlabel">
    Injury Report
</TD>
```

```

</TR>
<TR>
<TD valign="top" width="90%" class="ms-formbodiesurvey">
<!-- FieldName="Injury Report"
      FieldInternalName="ReportID"
      FieldType="SPFieldLookup"
      -->
<span dir="none"><select name="ctl00$m$g_a31f7024_8309_422f_9bba_84a452812e43$
      ctl00$ctl01$ctl00$ctl00$ctl00$ctl04$ctl00$Lookup"
      id="ctl00_m_g_a31f7024_8309_422f_9bba_84a452812e43_ctl00_ctl01_
      ctl00_ctl00_ctl00_ctl04_ctl00_Lookup" title="Injury Report">
<option selected="selected" value="0">(None)</option>
<option value="10">Migraine</option>
<option value="9">Needle stick</option>
<option value="15">Sprained ankle</option>
<option value="14">Threw back out</option>
</select><br/></span>
</TD>
</TR>

```

4. Notice that the select control has a title that matches your label (Injury Report). That's what we're going to be searching on with jQuery.
5. Add ?PageView=Shared&ToolPaneView=2 to the URL in your browser to put it in Edit mode.
6. Add a hidden CEWP to the bottom of the page.
7. Paste the following into the webpage dialog:

```

<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script type="text/javascript">
  $(document).ready(function() {
    $("input[value='Save']").hide(); // hides the Save buttons
    $("table.ms-formtoolbar tr:first").hide(); // hides the top toolbar
    var rptItem = $("select[title='Injury Report']"); // sets a variable to the select control
    //with our ID in it
    rptItem.hide(); //hides the select control
    fillDefaultValues("Injury Report", "ReportID"); // fills the select control with the data
    //from the querystring
    var rptText = $("select[title='Injury Report'] option:selected").text(); // sets
    //a variable to the text of the selected item
    rptItem.after(rptText); // prints the text of that variable after the select control
    //(which is now hidden)
  });

```

```

function fillDefaultValues(fieldName,value) {
    // Notice that you call fillDefaultValues, passing it: a) the Display Name of the
    //column, and b) the querystring key
    JSRequest.EnsureSetup();
    var qs = JSRequest.QueryString[value],
    x = $('select[title="'+fieldName+'"]');
    if (typeof(qs)!="undefined") {
        x.val(0)
    }else {
        x.val(qs)
    };
    x.change(function () {
        x.val(qs);
    }).change();
}
</script>

```

fillDefaultValues captures the query string value and puts it in the control matching the title supplied.

So far, what we've done is pass the Injury Report's ID to the BPE survey response via an email link. Steps 1 and 2 are complete.

Step 3: Connecting the CEWP and the jQuery shell

Here's where we really put Marc's library to work. We're going to use his web services library to update the item in the Injury Report list by writing the ID of this BPE survey response to the BPE survey lookup field (make sure you've created that column) in the Injury Report. We'll do this on the *EditForm.aspx*, since the ID of the survey response isn't available on *NewForm.aspx* (it hasn't been created yet).

Follow the steps above for adding a CEWP to the *EditForm.aspx* and getting the basic jQuery shell in place. From there, we'll start putting the pieces in place to make the final connection between these two items. Include the snippets for hiding the select control and displaying the text, as well as hiding the Save button and top toolbar (when we come back to the record in Edit mode, we'll want that same functionality to be in place). fillDefaultValues is not needed on the *EditForm.aspx*, however, because the connection will have already been made.

Looking through the library, we want to use the UpdateListItems operations of the Lists web service.

```

$.SPServices({
    operation: "UpdateListItems",
    async: false, // This ensures that processing waits for the response
    listName: "Injury Report", // Use the Display Name of your Source List
    updates: "<Batch OnError='Continue'>" +

```

```

"<Method ID='1' Cmd='Update'>" +
"<Field Name='SurveyID'>" + SurveyID + "</Field>" +
    // The first SurveyID is the StaticName of the lookup column in the Injury Report;
    // The second SurveyID is a variable we'll define in a minute
"<Field Name='ID'>" + ReportID + "</Field>" +
    // ID is the column name of the ID of Injury Report
    // ReportID is another variable we'll define shortly
"</Method>" +
"</Batch>",
completefunc: function(xData, Status) {
// We could put any post-processing or error handling here
}
});
});
});

```

So, we need to declare a couple of variables to make the above work.

SurveyID

Since we're on *EditForm.aspx*, the easiest place to get this is from the query string.

The following code snippets will do that for us:

We need to declare this variable (bold line below) before the SPServices call to `UpdateListItems`, so we'll put it right after we display the Report Title:

```

. . .
    rptItem.after(rptText);
    var SurveyID = getQuerystring('ID');
    // ID is found in the page URL (e.g. ../EditForm.aspx?ID=1)
});
. . .

//The code snippet for getQuerystring should go last before the <script> tag closes.

. . .
});

function getQuerystring(key, default_)
{
    if (default_ == null) default_ = "";
    key = key.replace(/[\/], "\\\\[\/].replace(/[\/]/, "\\");
    var regex = new RegExp("[\\?&]" + key + "=(^&#]*)");
    var qs = regex.exec(window.location.href);
    if(qs == null)
        return default_;
    else
        return qs[1];
}
</script>

```

ReportID

The `ReportID` is not accessible from this page unless we're on the first branch of the survey.

When the response is recorded, the first time we're on *EditForm.aspx* is on the second branch, so we need to get the `ReportID` some other way.

SPServices to the rescue again, in the form of the `GetListItem` operation. This helpful service allows us to get the XML for any list item by using common SQL syntax. First, we'll declare the variable, making it null. This gives it a scope outside the jQuery call to SPServices. For easy maintenance, we'll just declare it right after the `SurveyID` declaration:

```

. . .
    rptItem.after(rptText);
    var SurveyID = getQuerystring('ID');
    var ReportID = null;
});
. . .

```

Now, we'll call `GetListItem` before we call `UpdateListItems`, so that `ReportID` has data by the time we call `UpdateListItems`. Paste the following jQuery snippet (with edits) right before the `UpdateListItems` section:

```

$.SPServices({
    operation: "GetListItems",
    async: false,
    listName: "BPE Survey", //Again, use the Display Name here
    CAMLQuery: "<Query><Where><Eq>" +
                "<FieldRef Name='ID' />" +
                "<Value Type='Integer'>" + SurveyID + "</Value>" +
    // This is the variable we declared earlier
                "</Eq></Where></Query>",
    completefunc: function(xData, Status) {
        $(xData.responseXML).find("[nodeName= z:row]").each(function() {
            ReportId = $(this).attr("ows_ReportID");
            // Notice ows_ReportID; ReportID is StaticName of the Lookup column
            // in the Survey list;
            // Add the "ows_" to the front of that and you will have the XML
            // attribute you need.
        });
    }
});

```

Wrapping It Up

That's it. Here are the complete snippets we created (with notes left in to help you make the changes necessary for your implementation).

NewForm.aspx CEWP source:

```

<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script src="/scripts/jquery.SPServices.min.js" type="text/javascript"></script>
    // Download Marc's code here http://spservices.codeplex.com;
    // Save it in a SharePoint directory on your site and modify the previous line accordingly

<script type="text/javascript">
$(document).ready(function() {
    $("input[value='Save']").hide();
    $("table.ms-formtoolbar tr:first").hide();
    var rptItem = $("select[title='Injury Report']"); // Use title shown in source in step 3

```



```

    rptItem.hide();
    fillDefaultValues("Injury Report","ReportID"); // See function below for syntax
//on this one
    var rptText = $("select[title='Injury Report'] option:selected").text();
    // Use title shown in source in step 3
    rptItem.after(rptText);

});

function fillDefaultValues(fieldName,value) {
    // Notice that you call fillDefaultValues, passing it: a) the Display Name of the column,
    // and b) the querystring key (i.e. NewForm.aspx?ReportID=1)
    JSRequest.EnsureSetup();
    var qs = JSRequest.QueryString[value];
    var x = $('select[title="'+fieldName+'"]');
    if (typeof(qs)=="undefined") {x.val(0)}
    else {x.val(qs)};
    x.change(function () {
        x.val(qs);
    })
    .change();
}

</script>

```

EditForm.aspx CEWP source:

```

<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script src="/scripts/jquery.SPServices.min.js" type="text/javascript"></script>

<script type="text/javascript">
$(document).ready(function() {
    $("input[value='Save']").hide();
    $("table.ms-formtoolbar tr:first").hide();
    var rptItem = $('select[title="Injury Report"]'); // Use title shown in source in step 3
    rptItem.hide();
    var rptText = $('select[title="Injury Report"] option:selected').text());
    // Use title shown in source in step 3
    rptItem.after(rptText);

    var SurveyID = getQueryString('ID');
    var ReportID = null;

    $.SPServices({
        operation: "GetListItems",
        async: false,
        listName: "BPE Survey", // Use Display Name of your survey list
        CAMLQuery: "<Query><Where><Eq>" +
            "<FieldRef Name='ID' />" +
            "<Value Type='Integer'>" + SurveyID + "</Value>" +
            "</Eq></Where></Query>",
        completefunc: function(xData, Status) {
            ReportID = $(xData.responseXML).find("[nodeName='z:row']").attr
                ("ows_ReportID");
            // Prepend 'ows_' to the StaticName of the Lookup column in your survey list
        }
    })
}

```

```

});

$.().SPServices({
  operation: "UpdateListItems",
  async: false,
  listName: "Injury Report", // Use Display Name of your source list
  updates: "<Batch OnError='Continue'" +
    "<Method ID='1' Cmd='Update'" +
    "<Field Name='SurveyID'" + SurveyID + "</Field'" +
    // Use StaticName of the Lookup column in your source list
    "<Field Name='ID'" + ReportID + "</Field'" +
    "</Method'" +
    "</Batch'",
  completefunc: function(xData, Status) {
  }
});

function getQuerystring(key, default_)
{
  if (default_ == null) default_ = "";
  key = key.replace(/[\[\]\/, "\\\[\"].replace(/[\]\/, "\\\]");
  var regex = new RegExp("[\\?&]" + key + "=(^&#)*");
  var qs = regex.exec(window.location.href);
  if(qs == null)
    return default_;
  else
    return qs[1];
}

</script>

```

I may make it sound really easy, but figuring it all out wasn't that straightforward. If you'd like to follow the dialog and thought process that actually led to this solution, it's all archived at <http://spservices.codeplex.com/Thread/View.aspx?ThreadId=76735>.

Labeled Sections on Default Forms

Have you ever wanted to break up the columns on a list item into sections without having to unghost or customize the page in SPD (see [Figure 3-6](#))?

Checklist:	
Checklist Date	3/23/2010
IP / Computer Name	
Hardware	
	Yes
	Yes
	No
Test UPS	Yes
Computer and peripherals are clean	Yes
Verify machine has 4GB of RAM	Yes
If PC is routed through phone, enter in comments so we can get a separate drop	
Verify Cable is DVI on 21" monitor	Yes
Is it a VGA instead of DVI?	No
	Yes
Headset hook is present	Yes
Software	
Computer name matches the site name (2)	Yes
Computer is in domain (3)	Yes

FIGURE 3-6. *UnghostedDispForm with group labels*

It's a fairly simple process with jQuery.

1. Add a column to your list for each label you want.
2. Name each label with a convention that will let you find it easily with jQuery (for example, mine are `lblHardware` and `lblSoftware`). Make sure your convention prepends something identifying.
3. Set the type to Single line of text.
4. Set the default value to what you want the label to read. Refer to [Figure 3-7](#).

Column name:

The type of information in this column is:

Single line of text
 Multiple lines of text
 Choice (menu to choose from)
 Number (1, 1.0, 100)
 Currency (\$, ¥, €)
 Date and Time

Description:

Require that this column contains information:
 Yes No

Maximum number of characters:

Default value:
 Text Calculated Value

FIGURE 3-7. Setting up the label column

- Sort your columns to put the section label where you want it, as shown in [Figure 3-8](#).

Column (click to edit)	Type
Checklist Date	Date and Time
IP/Computer Name	Single line of text
lblHardware	Single line of text
	Yes/No
	Yes/No
	Yes/No
Test UPS	Yes/No
Computer and peripherals are clean	Yes/No
Verify machine has 4GB of RAM	Yes/No
TFPC is routed through phone, enter in comments	Multiple lines of text

FIGURE 3-8. Put the labels where you want them to fall on the page

- On your NewForm and EditForm, add a CEWP with the following code in the Source Editor:

```

<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

$(document).ready(function() {
  $("nobr").css("white-space", "normal"); // Lets labels wrap
  $("input[title^='lbl']").closest("tr").prev().each(function(index){

```

```

// Finds the TR that contains the label (change lbl to your prepend convention)
var lblText = $(this).next().find("span > input").val();
// Gets the label text (the default value)
$(this).parent().before("<tr class='ms-SPButton ms-WPAddButton'>" +
    "<td nowrap='true' valign='top' width='100%' class='ms-formlabel' +
    colspan='2'>" +
    "<h3 class='ms-standardheader ms-WPTitle'>" + lblText +
    "</h3></td></tr>"); // Adds a TR formatted as label
$(this).parent().hide(); // Hides the default TR
});
});
</script>

```

7. DispForm renders differently, so add a CEWP with the following code in its Source Editor:

```

<script src="/scripts/jquery.min.js" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

$(document).ready(function() {
    $("nobr").css("white-space", "normal"); // Lets the labels wrap
    $("a[name^='SPBookmark_lbl']").closest("tr").each(function(index){
        // Finds the TR that contains the label (change lbl to your prepend convention)
        var lblText = $(this).find("td#SPFieldText").text();
        // Gets the label text (the default value)
        $(this).before("<tr class='ms-SPButton ms-WPAddButton'>" +
            "<td nowrap=true valign='top' width='100%' class='ms-formlabel' +
            colspan='2'>" +
            "<h3 class='ms-standardheader ms-WPTitle'>" + lblText +
            "</h3></td></tr>"); // Adds a TR formatted as label
        $(this).hide(); // Hides the default TR
    });
});
</script>

```

You're done.

Changing Labels

Because the end user is never presented with the lbl* field to modify, only a list maintainer can change the labels.

In List Settings, the maintainer can change the order, change the groupings under each label, and/or add a new one using steps 1–4 above.

One last step you'll need to do if you already have data in this list is to set the labels for all of the old data so that the labels work on the EditForm and DispForm views of those list items. The easiest way to do this is to put the list in Data Sheet view and set them for the entire list.

Where To from Here?

Articles are still being regularly posted on EndUserSharePoint and elsewhere that delve into more examples for when to use jQuery to tweak the SharePoint experience for your users. Perhaps you would like to write one! The great thing about the SharePoint community is that it is collaborative, and we all learn from each other.

Summary

As I mentioned at the beginning of the chapter, these five examples are just the beginning of what you can do with jQuery. But, now that you've walked through the DOM and seen how selectors work, when to employ SPServices, and how to manipulate the presentation layer, hopefully you've come up with some ideas of your own for how you, too, can claim: "jQuery to the rescue!"

Unlocking the Mysteries of the SharePoint Data View Web Part XSL Tags

Marc D. Anderson

The article I chose for inclusion in this book may seem at first glance to be an odd one. After all, it is not much more than an outline for a series of articles I did over four months at EndUserSharePoint.com (EUSP). However, this first article laid the foundation for that series, which was well read and which later became an ebook that we sold on the EUSP site. People continue to buy from me today. This is not about me advertising that, but about the fact that a single, simple article at EUSP, with the questions, comments, and feedback attached to it, can grow and stretch and can help a great many more people than its author expected.

Writing for EUSP has always been a pleasure. If you think about it, there is really no pressure—no more than writing on one’s own blog—yet one can help a great number of people. When Mark Miller first approached me to write for EUSP a few years ago, I felt honored, humbled, and secretly thrilled. EUSP had been a source of great content for me as a reader, and I felt like I had become a member of a little club of which I never thought I would be a member.

I have always really believed in education and collaboration. I am not the alpha dog type; I work best with people who can treat everyone on the team as equals. That does not mean that everyone is the same. No, far from it. It means that no one of us is better than the other simply by dint of some artificial rank or job title. We all bring skills to the table that have value, sometimes unexpectedly.

This is the promise of SharePoint as a platform to me. If implemented well, it can enable the entire organization using it to improve its performance. This has been a clear focus for me professionally for a long time, certainly as a management consultant focused on knowledge management and process performance in the mid-1990s, and earlier, as the Manager of Delivery Systems at Staples. Even when I was away at Phillips Exeter Academy for my last two years of high school, I saw the tremendous value of the Harkness system, in which everyone sits around a large, oval table to discuss the topics at hand—there is nowhere to hide. By attacking a challenge together, collaboratively, in almost a hive mentality, we can do great things together.

By educating everyone in the organization in the basic tenets of working together well and tying their own work to the overall strategy of the organization, that organization gains far more productive, effective, and efficient workers. If those workers can do the basics well, they have more time to focus on the higher-level endeavors for which they rarely have the time. That is where we see the innovation and serendipity that can drive an organization to another level of success.

You may wonder what all of this has to do with EUSP. Well to me, EUSP has always been a platform where people can learn. Usually, they learn something that they did not even know they needed to learn when they got there. The “End User” part of EUSP is important, too. Many of the topics that I, and others, have written about on EUSP are considered by some to be beyond the so-called End User. I say pshaw to that. There is no real boundary to what people can learn if they choose to. The labels we slap onto people seem like an attempt to lock down the potential of their capabilities, and that just should not happen. The more an End User knows, the better he can perform and the better he can use the massive technology that is SharePoint. Perhaps even more important, the more End Users know about SharePoint, the better they can work with the people who build things for them on top of SharePoint. When End Users see the possibilities, or at least see that there are possibilities, End User conversations with developers and administrators can have more meat to them. This happens all the time based on what people learn by reading articles on EUSP.

Most people are more mercenary than martyr, and I suppose I have to admit that is somewhat true of me as well. I want to be able to support my family in comfortable style, but I also truly love to educate. That is why I have written so many articles on my own blog and for EUSP over the last few years. However, EUSP has a far larger audience than my blog probably ever will. Writing for EUSP has gained me some notoriety, and I can parlay that into well-paying gigs as a consultant. For me, the real motivation is the comment on an article that says it helped someone solve a seemingly unsolvable business problem or prove that something was indeed possible (or was far simpler than everyone she had asked had told her) or give someone the courage or spark to change his or her career direction. Unbelievably, each of those things has happened based on my writing for EUSP. I have felt that at least some of my writings there have made a big difference for some people. That is the whole point, indeed.

More About Data View Web Parts

The humble Data View web part (DVWP) is far more than just the list of XSL tags I included in my original EUSP article. I mentioned then that DVWPs are sometimes called the Swiss Army Knife of SharePoint, and to me they truly are, more so than the Content Query web part (CQWP) or even the Content Editor web part (CEWP). If you have content in SharePoint and want to display it in unique and useful ways, the DVWP is the tool you need to understand. It can help you to display simple list content in exactly the format you want, show business intelligence information using graphics and summary information, customize forms to your heart's content, and even more. Knowing that all of these things are possible means that you can imagine more than you would otherwise.

To be a master of the DVWP, you need to understand its parts. Let's go through the major components of the DVWP and see how they all fit together. Once you understand those parts, you can use the XSL tags covered in the original article, "[Unlocking the Mysteries of Data View Web Part XSL Tags](#)", and the subsequent articles in the series to build out whatever you want.

NOTE

You will hear people talking about the DVWP and the Data Form web part (DFWP) interchangeably. In earlier versions of SharePoint, it was the DVWP, and in SharePoint 2007, it was renamed the DFWP. We veterans of SharePoint all kept calling it the DVWP simply because it was easier. Any time that you see either name, know that the writer is talking about the same thing. The DVWP is not, however, the same thing as the Extended List View web part (XLV), which was introduced in SharePoint 2010. While they both are XSL-driven, there are other differences. I won't go into those differences here, but I will say that I find the DVWP to be far more powerful and malleable than the XLV.

Adding a DVWP to Your Page

I'll run through how to add a DVWP with SharePoint Designer 2010. (The steps are similar in SharePoint Designer 2007, but I won't cover them here.) You can create DVWPs only in SharePoint Designer. This imposes a restriction in some cases in which SharePoint Designer usage is locked down or turned off. If you are in that situation, reach out to the team that manages your governance to find out what the concerns are and whether you can have the restrictions lifted.

Once you have a page open in SharePoint Designer, you should position your cursor where you would like to add the DVWP. I suggest working in Split mode (see [Figure 4-1](#)) when you are using SharePoint Designer so that you can see what effect your actions have in the page's code.



FIGURE 4-1. Choose the Split view

To insert the DVWP, choose Insert on the top-level menu, then Data View, then Empty Data View, as shown in Figure 4-2.

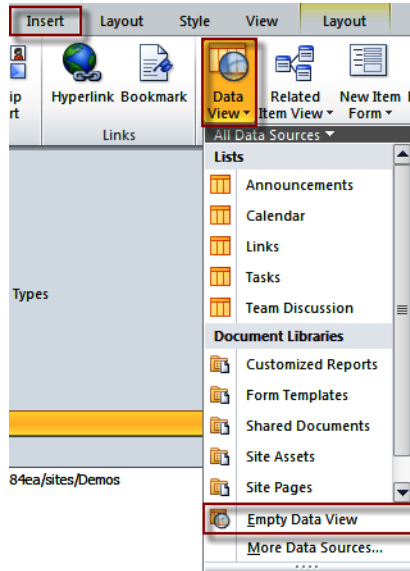


FIGURE 4-2. Adding a new DVWP to the page

If you choose one of the lists or libraries in the All Data Sources panel instead, you will get an XLV in your page rather than a DVWP.

After SharePoint Designer chugs a little bit, you will see the start of your DVWP in the page, as you can see in Figure 4-3.

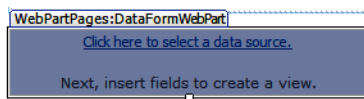


FIGURE 4-3. The empty DVWP shell

At this point, you have only the outer shell of the DVWP in place. As with all web parts that you add to a SharePoint page, the DVWP has an outer shell that declares its main characteristics.

A typical outer shell might look something like the Code view shown in [Figure 4-4](#) (I have cleaned it up a little to make it easier to read; I have done this cleanup with most of the following examples as well).

```
1 <WebPartPages:DataFormWebPart
2     runat="server"
3     IsIncluded="True"
4     AsyncRefresh="True"
5     FrameType="None"
6     NoDefaultStyle="TRUE"
7     ViewFlag="8"
8     Title="DataView 1"
9     PageType="PAGE_NORMALVIEW"
10    __markuptype="vsattributemarkup"
11    __WebPartId="{625B67E3-8642-4484-95E4-550907B6A73F}"
12    id="g_625b67e3_8642_4484_95e4_550907b6a73f">
13 <DataSources>
14 </DataSources>
15 <datafields/>
16 <XSL>
17 </XSL>
18 </WebPartPages:DataFormWebPart>
```

FIGURE 4-4. Code for the DVWP shell

As you can see, there is not much there yet. The DVWP is declared and a few of its gross characteristics are set, such as its Title (which is displayed as part of its chrome by default), whether or not it will autorefresh (`AsyncRefresh="TRUE"`), its ID (which is based on a GUID), etc. At this point, there is no data source or XSL, so there is not any meat in it yet.

The next step is to click “Click here to select a data source link”. This brings up the available data sources, which, in an out-of-the-box Team Site, will look something like the list shown in [Figure 4-5](#).

DVWPs can consume and display information from different types of data sources, but by far, the most common are lists and libraries. The other possibilities are linked data sources (this allows you to use more than one list or library in your DVWP), database connections (such as SQL), SOAP service connections (also known as web services, such as the lists web service), REST service connections (such as *ListData.svc*), and XML file connections (any XML file that you can refer to by URL). You can see these possibilities in [Figure 4-6](#), which shows them in the Data Sources ribbon.

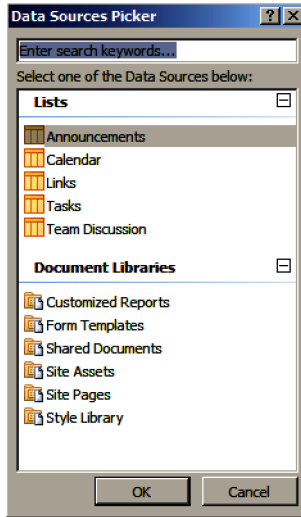


FIGURE 4-5. Selecting a data source for the DVWP

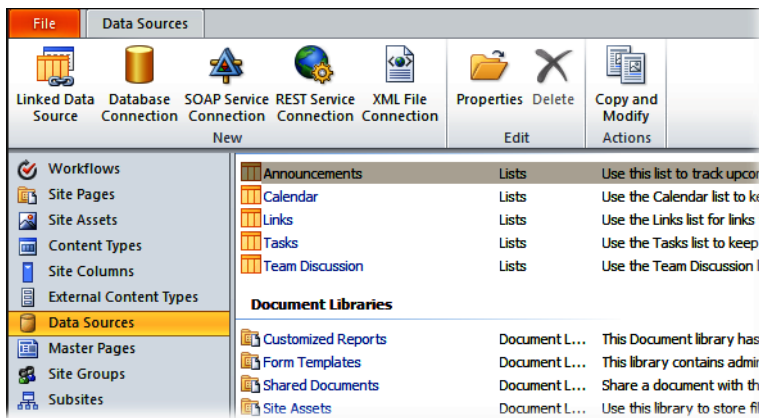


FIGURE 4-6. The ribbon, showing the available types of data sources

Let's stick with a plain old list as we run through the process; we will select the Announcements list. Once you make that selection, the Data Source Details pane should open on the right side of your screen, and you will see the first five columns highlighted by default, as shown in [Figure 4-7](#).

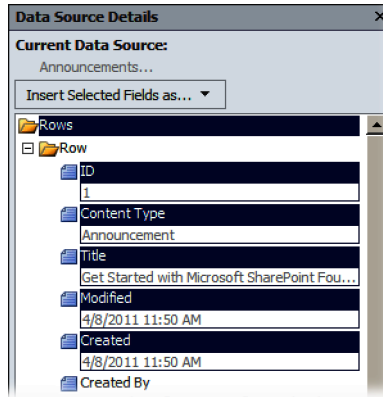


FIGURE 4-7. Selecting columns to display a DVWP based on an Announcements list

To keep things simple, we will only select the Title column and then choose Multiple Item View in the drop-down shown above the columns, as shown in Figure 4-8. This will show us just the Title column for all of the items in the list.

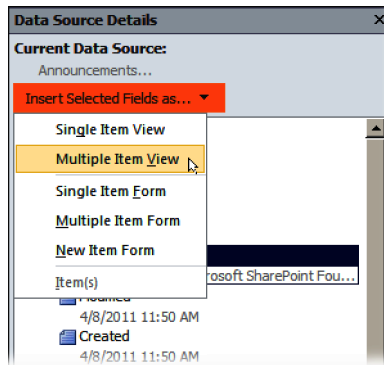


FIGURE 4-8. Inserting a multiple item view

Now you should see something like Figure 4-9 in the Design pane.

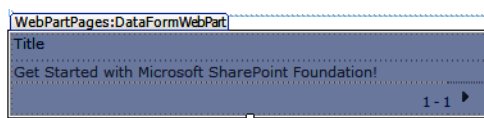


FIGURE 4-9. The simple DVWP showing one Announcement

Notice that we are seeing one item here—the default announcement placeholder that SharePoint creates in an Announcements list—and that paging is on by default. At this point, you can make further adjustments to your DVWP using the Data View Tools section in the ribbon, shown in [Figure 4-10](#).

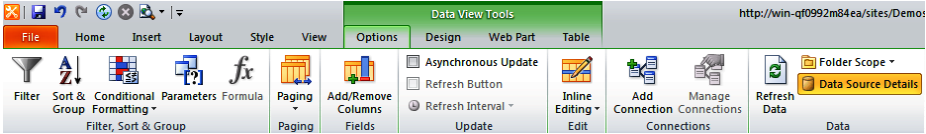


FIGURE 4-10. The ribbon, showing all of the available options

However, we are looking to do far more with the DVWP than what you can do by clicking on buttons in the ribbon. The XSL tags in the original EUSP article series are contained in the code for the DVWP, and are often not available by clicking on options in the ribbon.

Customizing the DVWP in Code

You can customize each of the main sections in the DVWP based on what you would like to accomplish. There is no possible way to cover all of the potential variations in one chapter of a book, but if you understand the reason each section is present, you may see some of the possibilities.

DataSources

Look at the `DataSources` section for our simple example in [Figure 4-11](#). You can see that the `DataSourceMode` is `List`, meaning that we are pulling data from a list, of course. The `selectcommand` shows the simplest CAML there is, as it is only the shell `<View></View>`. The ampersands and other characters you see are simply the greater than and less than signs encoded so the server interprets them correctly.

DVWPs are able to perform all four CRUD (create, read, update, and delete) operations, though in SharePoint's parlance, we have `Select`, `Delete`, `Update`, and `Insert`. Since we are only displaying the contents of the list and do not want to do any updating, we can remove the sections other than `Select`. We also don't want paging, so we can remove those parameters. That leaves us with the much more manageable `DataSources` section, like the one shown in [Figure 4-12](#).

```

1 <DataSources>
2   <SharePoint:SPDataSource
3     runat="server"
4     DataSourceMode="List"
5     UseInternalName="true"
6     UseServerDataFormat="true"
7     selectcommand="<View></View>"
8     id="dataformwebpart1">
9     <SelectParameters>
10      <WebPartPages:DataFormParameter
11        Name="ListID"
12        ParameterKey="ListID"
13        PropertyName="ParameterValues"
14        DefaultValue="{1C61BAD6-4174-4183-B3C8-F9FC38161F06}"/>
15      <asp:Parameter
16        Name="StartRowIndex"
17        DefaultValue="0"/>
18      <asp:Parameter
19        Name="nextpagedata"
20        DefaultValue="0"/>
21      <asp:Parameter
22        Name="MaximumRows"
23        DefaultValue="10"/>
24    </SelectParameters>
25    <DeleteParameters>
26      <WebPartPages:DataFormParameter
27        Name="ListID"
28        ParameterKey="ListID"
29        PropertyName="ParameterValues"
30        DefaultValue="{1C61BAD6-4174-4183-B3C8-F9FC38161F06}"/>
31    </DeleteParameters>
32    <UpdateParameters>
33      <WebPartPages:DataFormParameter
34        Name="ListID"
35        ParameterKey="ListID"
36        PropertyName="ParameterValues"
37        DefaultValue="{1C61BAD6-4174-4183-B3C8-F9FC38161F06}"/>
38    </UpdateParameters>
39    <InsertParameters>
40      <WebPartPages:DataFormParameter
41        Name="ListID"
42        ParameterKey="ListID"
43        PropertyName="ParameterValues"
44        DefaultValue="{1C61BAD6-4174-4183-B3C8-F9FC38161F06}"/>
45    </InsertParameters>
46  </SharePoint:SPDataSource>
47 </DataSources>

```

FIGURE 4-11. The data source as SharePoint Designer creates it

I point out this simple change to illustrate the fact that the DVWP is simply made up of code, and you can change any of that code to make your DVWP do what you want. Until you are comfortable working in the Code view, you can use the ribbon buttons, but watch the effect each change you make has on the code. After you see what changes for a while, you may feel more comfortable working in the Code view, or at least working in the Split view with confidence.

The DataSources section will look different based on what type of data source(s) you choose. Let's look at a couple of examples.


```

1 <DataSources>
2   <SharePoint:SPDataSource
3     runat="server"
4     DataSourceMode="List"
5     UseInternalName="true"
6     UseServerDataFormat="true"
7     selectcommand="&lt;View&gt;&lt;/View&gt;"
8     id="dataformwebpart1">
9     <SelectParameters>
10      <WebPartPages:DataFormParameter
11        Name="ListID"
12        ParameterKey="ListID"
13        PropertyName="ParameterValues"
14        DefaultValue="{1C61BAD6-4174-4183-B3C8-F9FC38161F06}"/>
15    </SelectParameters>
16  </SharePoint:SPDataSource>
17 </DataSources>

```

FIGURE 4-12. Cleaned up code for the same data source

Figure 4-13 shows what a linked data source looks like. In this case, the linked data source includes the Calendar list and the Tasks list. I have reformatted things again to make it easier to read, and I have also removed the Insert, Update, and Delete sections again for brevity.

```

1 <DataSources>
2   <SharePoint:AggregateDataSource
3     runat="server"
4     IsSynchronous="false"
5     id="dataformwebpart2">
6     <sources>
7       <SharePoint:SPDataSource
8         runat="server"
9         DataSourceMode="List"
10        UseInternalName="true"
11        UseServerDataFormat="true"
12        selectcommand="&lt;View&gt;&lt;/View&gt;">
13        <SelectParameters>
14          <asp:Parameter
15            Name="ListID"
16            DefaultValue="{2A9A0193-996A-479C-873F-85172FC28C40}"/>
17        </SelectParameters>
18      </SharePoint:SPDataSource>
19      <SharePoint:SPDataSource
20        runat="server"
21        DataSourceMode="List"
22        UseInternalName="true"
23        UseServerDataFormat="true"
24        selectcommand="&lt;View&gt;&lt;/View&gt;">
25        <SelectParameters>
26          <asp:Parameter
27            Name="ListID"
28            DefaultValue="{9B071A36-5984-4E53-B346-A82B0013A9BE}"/>
29        </SelectParameters>
30      </SharePoint:SPDataSource>
31    </sources>
32    <aggregate>
33      <concat name="data source">
34        <datasource name="Calendar" id="0" Type="SPList"/>
35        <datasource name="Tasks" id="1" Type="SPList"/>
36      </concat>
37    </aggregate>
38  </SharePoint:AggregateDataSource>
39 </DataSources>

```

FIGURE 4-13. An *AggregateDataSource*

When you have an `AggregateDataSource` like this, you can refer to each data source separately in your XSL based on the names you give each data source in the aggregate section. For instance, you might create a variable that contains the rowset from each data source, as shown in [Figure 4-14](#).

```
<xsl:variable name="CalendarRows" select="/dsQueryResponse/Calendar/Rows/Row"/>
<xsl:variable name="TasksRows" select="/dsQueryResponse/Tasks/Rows/Row"/>
```

FIGURE 4-14. Selecting items from each DataSource separately

You also can use a SOAP web service data source to grab data locally, from another Site Collection, or even from an external source. The `DataSource` for this type of connection looks something like the one shown in [Figure 4-15](#). Here, I am simply using the Lists web service to connect to the Announcements list with the `GetList` operation. This operation gives us information about the list structure and its columns. SharePoint Designer constructs the SOAP envelope for us and takes care of all of the other connection information as well.

```
1 <DataSources>
2   <SharePoint:SoapDataSource
3     runat="server"
4     id="SoapDataSource1"
5     AuthType="None"
6     WsdlPath="http://win-qf0992m84ea/sites/demos/_vti_bin/Lists.asmx?WSDL"
7     SelectUrl="http://win-qf0992m84ea/sites/demos/_vti_bin/Lists.asmx"
8     SelectAction="http://schemas.microsoft.com/sharepoint/soap/GetList"
9     SelectPort="ListsSoap"
10    SelectServiceName="Lists">
11    <SelectCommand>
12      <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
13        <soap:Body>
14          <GetList xmlns="http://schemas.microsoft.com/sharepoint/soap/">
15            <listName>Announcements</listName>
16          </GetList>
17        </soap:Body>
18      </soap:Envelope>
19    </SelectCommand>
20  </SharePoint:SoapDataSource>
21 </DataSources>
```

FIGURE 4-15. SOAPDataSource

Many times, simply configuring your data sources using the options on the ribbon will be adequate, but by looking at the code that generates, you will be able to see how you might make changes to the `DataSources` code manually to accomplish things that are more complex.

datafields

The next section that shows up in the code is the `datafields` section. I have never been able to figure out exactly what this section provides, frankly, as you can delete it with no ill effects. The one benefit I can see for it is that it shows the `InternalName` and `DisplayName` for all of the fields in your data source. If nothing else, that provides a nice little catalog for you to use as a

reference as you are working. [Figure 4-16](#) shows an example of the `datafields` section for the Announcements list, greatly truncated.

```
<datafields>@ID, ID; @ContentType, Content Type; @Title, Title; @Modified, Modified; @Created, Created; @Author, Created By; @Editor, Mod
```

FIGURE 4-16. `datafields`

parameterbindings

Next up is the `parameterbindings` section. Think of this section as a way to identify values you want to work with that come from outside the DVWP but are not contained in the `DataSources`. By default, you will get a `parameterbindings` section that looks something like the one shown in [Figure 4-17](#).

```
<parameterbindings>
  <ParameterBinding Name="ListID" Location="None" DefaultValue="{1C61BAD6-4174-4183-B3C8-F9FC38161F06}"/>
  <ParameterBinding Name="dvt_apos" Location="PostBack;Connection"/>
  <ParameterBinding Name="ManualRefresh" Location="WPPProperty[ManualRefresh]"/>
  <ParameterBinding Name="UserID" Location="CAMLVariable" DefaultValue="CurrentUserName"/>
  <ParameterBinding Name="Today" Location="CAMLVariable" DefaultValue="CurrentDate"/>
  <ParameterBinding Name="dvt_startposition" Location="PostBack" DefaultValue="" />
  <ParameterBinding Name="dvt_firstrow" Location="PostBack;Connection"/>
  <ParameterBinding Name="dvt_nextpagedata" Location="PostBack;Connection"/>
</parameterbindings>
```

FIGURE 4-17. `parameterbindings`

SharePoint Designer adds each of these parameters in a sort of opportunistic way, expecting that you might need each of them in your DVWP. The other way to look at it is that SharePoint Designer is fundamentally a code generator, and it sets things up for all of the possible choices you might make using the options on the ribbon.

One of the reasons people get into trouble when they start to customize the DVWP is that they go beyond what SharePoint Designer understands. This can be the cause of those annoying hangs and crashes everyone complains about with SharePoint Designer. As you become more familiar with how things work, you will find that those unpleasant events happen less frequently.

Most of the options in the `parameterbindings` shown in [Figure 4-17](#) are probably reasonably self-explanatory. As you add new parameters, whether by using the `Parameters` button on the ribbon or by typing in the `ParameterBinding` line yourself, you will get new lines in the `parameterbindings` section.

Starting from the ribbon, the sequence to add a new parameter through the dialogs works something like this. First, click on the Parameters button in the ribbon, shown in [Figure 4-18](#).

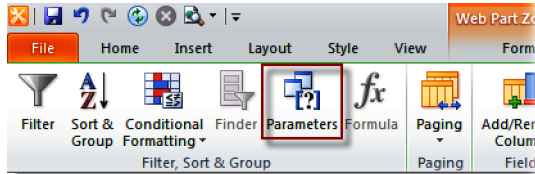


FIGURE 4-18. Adding a new parameter using the ribbon

[Figure 4-19](#) shows the dialog that will then pop up. Make the appropriate selections, enter the desired values in the dialog, and click OK. No, there is nothing here to help you out. You just have to know what the acceptable values for things like the Server Variable text box might be, unfortunately.

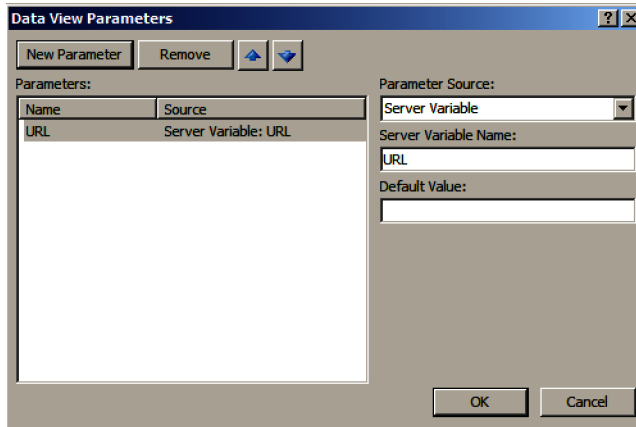


FIGURE 4-19. Filling in the properties of the parameter

Finally, you end up with a line like what is shown in [Figure 4-20](#) in the parameterbindings based on what you entered. Once you get the hang of it, you may choose to type in the line yourself.

```
<ParameterBinding Name="URL" Location="ServerVariable(URL)" DefaultValue="" />
```

FIGURE 4-20. The resulting ParameterBinding

XSL

The final section is the XSL section. This section is the most important, as it determines what ends up on the page. I always say XSL, which stands for eXtensible Stylesheet Language, rather than XSLT, which you may hear more people saying. The T stands for Transformation and, in my mind, you write the XSL that then transforms the data in the XML from the data source.

The XSL section can be as simple or as complex as you need it to be, and it is where most of the hardcoding work happens. You are limited only by your imagination, as you can transform your data from its starting state into just about anything using XSL and the XPath functions. That is what the original EUSP article and the follow-on series was all about—how you can write your own XSL using the DVWP's XSL tags to make magic happen. A list of the most frequently used XSL tags follows.

<xsl:template>

Think of a template as a subroutine. It is a unit of XSL to which you pass control.

<xsl:call-template>

This is how you call a template that you have defined with <xsl:template>.

<xsl:with-param>

You use this with <xsl:call-template> when you want to pass a value into a template, usually a value that varies.

<xsl:param>

A value you've passed into a template with <xsl:with-param>. You need to have an <xsl:param> at the top of the template for each value you expect to be passed into it.

<xsl:variable>

A value you create for use within a template that is only defined within the scope of that template.

<xsl:for-each>

A way to iterate over a nodeset (group of rows).

<xsl:sort>

Used within an <xsl:for-each> to determine the sort order in which the nodeset (group of rows) is processed.

<xsl:if>

A single conditional test. If the test is true, the contained code is executed.

<xsl:choose>

Like <xsl:if>, but with multiple possibilities, more like if-then-else.

<xsl:when>

Used within <xsl:choose> as a conditional test. If the test is true, the contained code is executed and the <xsl:choose> is exited.

`<xsl:otherwise>`

Used within `<xsl:choose>` as the “fall-back” condition. If none of the prior `<xsl:when>`s have been executed, the code contained here is executed.

`<xsl:value-of>`

Outputs the value to which it evaluates, such as the value of a column, a variable, etc.

`<xsl:comment>`

Allows you to put comments into your XSL. Comments: what a novel concept!

`<xsl:text>`

Use this tag when you want to include a piece of text that will be treated entirely literally.

`<xsl:attribute>`

`<xsl:attribute>` allows you to set an attribute on an element programmatically.

`<xsl:import>`

`<xsl:import>` allows you to import a section of XSL from an external file.

There are more XSL tags available, but this set is what you are most likely to find useful as you build your own DVWPs. SharePoint still uses the XSL 1.0 specification, so the list is shorter than you might see in other contexts.

Summary

I hope this overview of the DVWP gives you some more context with which to think about the DVWP and gives you a few ideas about how you might use one.

The original little EUSP article led to a 20-part series about the humble XSL tags and some of the XPath functions, with a few dalliances along the way. There were, in fact, more parts to that series than I had predicted in the introductory article. Because the series had such great readership, I tacked on quite a few more articles. To tell the truth, I didn't think the outline through as well as I might have, and there was simply more to cover than I thought at first.

Thanks to Mark Miller for believing in my early scratchings enough to give me the chance to write for EUSP. Thanks also to all of the other talented writers at EUSP; their articles have often taught me something significant or caused me to rethink my own ways of doing things. Most of all, thanks to the quiet orchestrator of all that is EUSP, Natasha Felshman. I have greatly enjoyed my collaborations with Natasha, whether working on my own articles, trying to help her to format others' articles, or wrangling the underlying technologies into submission. I am hoping there will be many more collaborations to come.

Hyperlinks in the Data View Web Part

Laura Rogers

In SharePoint, List and Library views are used to look at the same set of data in different ways. When views are created, columns, filters, sorting, and grouping are set up in an easy user interface. There are some additional settings such as totals, inline editing, and styles, but in a lot of cases, there is a point at which the out-of-box settings are not sufficient for the goals of the business.

When the settings do not provide the needed customization for the List or Library view requirements, you can use SharePoint Designer to create a Data View web part (DVWP) or an XSLT List View web part. There is a plethora of view modifications that you can make. The following are the most common reasons for customizing List views:

- A need for conditional formatting
- Customization of the displayed message when the view has no items
- Creation of parameters
- A need for complex filters

In this chapter, you will learn that there is a difference between Data View and XSLT List View web parts, and how hyperlinks are customized in each. The hyperlinks will also be structured differently for libraries than they are for lists. Therefore, an overview of SharePoint URL structures will be provided.

In May 2009, I originally wrote an article with an associated screencast, called “[Data View Web Part, The Basics—Add a Hyperlink](#).” It has had many site visits over the years, and is my most popular EndUserSharePoint (EUSP) article.

I wrote this article before SharePoint 2010 was released. Back in SharePoint 2007, there was only one way to modify a List view with SharePoint Designer, and that was to create a DVWP. My article is very simple. It includes eight steps for a document library and six steps for a list. This chapter is based on that original article, but there is much more to it. When SharePoint 2010 was released, it introduced the concept of XSLT List View web parts. This means that any List or Library view can be edited directly in SharePoint Designer, and can also still be edited in the browser’s view settings, even after the view has been edited using SharePoint Designer. Instead of having two simple sets of instructions for a list DVWP and a library DVWP, this chapter has four sets of instructions. Now there is an additional set of steps for a library XSLT List View web part, and different steps for a list’s XSLT List View web part.

Setup for Walkthroughs

Some lists and libraries will be needed for the test environment for the step-by-step instructions in this chapter. Here is a high-level overview of the content that will be used:

- Contacts list
 - Add hyperlinks to an XSLT List View web part.
 - Add hyperlinks to a DVWP.
- Project files document library
 - Add hyperlinks to an XSLT List View web part.
 - Add hyperlinks to a DVWP.
- Web Part Pages library
 - Create temporary ASPX pages to create the DVWPs.

Notice that the temporary ASPX pages are created as a test location for the creation and modification of DVWPs. This is not done for XSLT List View web parts. With a DVWP, the idea is that it is created in a temporary location and then exported from there and finally imported cleanly onto the site home page or wherever it needs to be displayed. Unfortunately, this same functionality just doesn’t quite work with XSLT List View web parts. When an XSLT List View web part is created in a temporary location, exported, and then imported onto another page, at first it looks just fine. As soon as you exit Edit mode on the page, however, all of your formatting changes are lost and it reverts back to the default view of the list. You’ll notice that with the XSLT List View web parts, we create a new view, but we do not bother with a temporary location on a web part page.

In this walkthrough, the test site and the example list and library will be created. Since lists and libraries have different URL structures, each one will be demonstrated.

1. Create a test site for the lists and web parts in this chapter. On your SharePoint site, click Site Actions, and choose New Site.
2. Choose the Document Workspace template. As shown in [Figure 5-1](#), name the site “Chapter 5 Test”, with a URL of chapter5. Click Create.

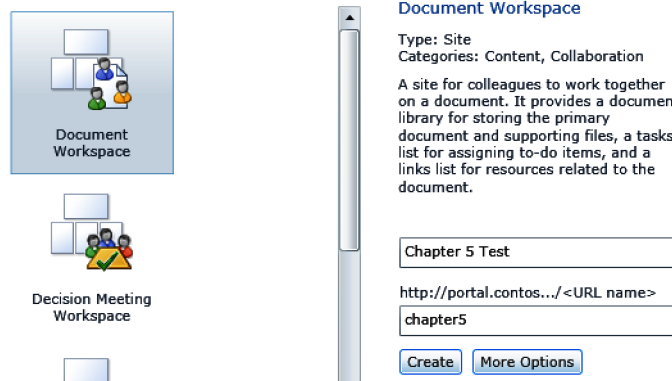


FIGURE 5-1. Create a site

3. Click Site Actions and choose New Document Library.
4. For the Name, type ProjectFiles. When a simple name is used, with no spaces or weird characters, the URL to the list or library is easier to work with later.
5. Click Create.
6. In the new library, on the Library tab in the ribbon, click Library Settings.
7. Click Title, description and navigation.
8. Put a space in the library name so that it is now “Project Files.” Click Save.

NOTE

Spaces in list and library names are pet peeves of mine because of the extra “%20” characters that are added where the spaces are. I always create lists, libraries, and even columns without spaces at first.

9. Upload a few files to the Project Files library. Even when testing, it helps to have example data to see in your new view.
10. Click Site Actions and choose More Options.
11. In the list of templates, select Contacts, name the list “Contacts”, and click Create (see [Figure 5-2](#)).

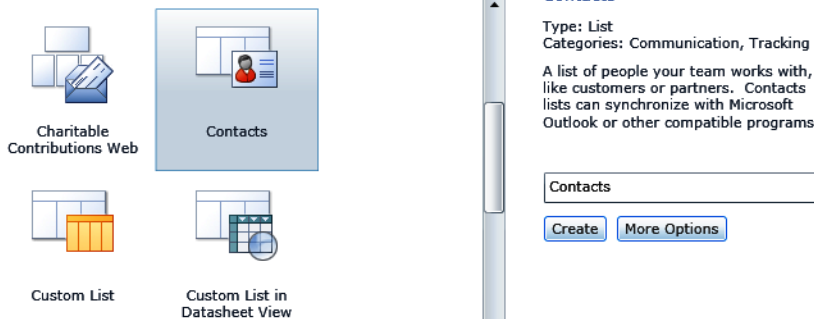


FIGURE 5-2. Create a Contacts list

12. Add a few contacts to this new list. Be sure to fill out the full name column and the email address for each contact.

URLs in SharePoint

Since this chapter is all about creating hyperlinks, it helps to know a bit about how hyperlinks in SharePoint are structured. This section covers some basics of list and library hyperlinks, which will give you a great foundation of understanding for the rest of the chapter.

Document Libraries

The URL of a document library can sometimes be confused with the URL of a subsite. A library called “ProjectFiles” under the site called “chapter5” will have the following URL:

<http://portal.contoso.com/chapter5/projectfiles>

Within document libraries, the document URLs consist of the names of each document. For example, if I upload an Excel file called *Project Hours.xlsx* to my ProjectFiles library, the URL to that file is <http://portal.contoso.com/chapter5/projectfiles/Project%20Hours.xlsx>.

Lists

Lists actually do live under a folder called *Lists* in each site, so they’re easy to identify. A Contacts list will have the URL <http://portal.contoso.com/chapter5/Lists/Contacts>.

Views

Views in lists and libraries each have their own URLs. Each view is an *.aspx* file under the URL of the list. In libraries, there is a *Forms* folder, and in lists, there is no *Forms* folder. For example, a tasks list in SharePoint has several default views, each with its own URL, such as:

- <http://portal.contoso.com/promos/Lists/Tasks/AllItems.aspx>
- <http://portal.contoso.com/promos/Lists/Tasks/active.aspx>

Library views will have this structure:

- <http://portal.contoso.com/ProjectFiles/Forms/AllItems.aspx>
- <http://portal.contoso.com/ProjectFiles/Forms/MyDocuments.aspx>

Items

Each individual item has its own URL, based on the type of form you want to open it in. Also, each file in a library has properties to edit and view, similar to editing a list item. There are three default forms in every list and library:

NewForm.aspx

This is the very first form used. Use it when creating a new item in a list.

DispForm.aspx

This is the display form. Use it to look at the item as read-only after it has been created.

EditForm.aspx

This is the form used to edit an item and save changes.

Every list and library item in SharePoint has its own ID, which is unique within that list or library. This is called the *ID field*, and it is created in numerical order as each list item is created. This ID cannot be changed.

This is the syntax of the very first task created in a task list:

<http://portal.contoso.com/promos/Lists/Tasks/NewForm.aspx>

This is the syntax of the very first task created in a task list, *after* it has been created, when you're simply viewing it:

<http://portal.contoso.com/promos/Lists/Tasks/DispForm.aspx?ID=1>

Click Edit Item, and this is the URL:

<http://portal.contoso.com/promos/Lists/Tasks/EditForm.aspx?ID=1>


XSLT List View Web Part Hyperlinks

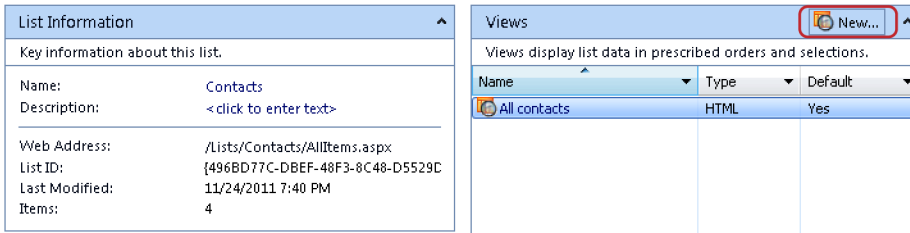
Every view in SharePoint 2010 is an XSLT List View web part. This means that all List views and List web parts can be added, edited, and removed via either the browser or SharePoint Designer. Now that some URL fundamentals have been covered, it's time to start with a series of step-by-step walkthroughs.

SharePoint List

In this walkthrough, a new view will be created and modified, starting with the Contacts list.

1. On your Chapter 5 test site in the browser, click Site Actions and choose Edit in SharePoint Designer.
2. In the Navigation pane on the left, click Lists and Libraries.
3. Click the name of the Contacts list.
4. On the right side in the Views section (as shown in [Figure 5-3](#)), click New.

 Use this page to view and manage settings for this list.



The screenshot shows two panels. The left panel, titled 'List Information', contains the following details:

Name:	Contacts
Description:	< click to enter text >
Web Address:	/Lists/Contacts/AllItems.aspx
List ID:	{496BD77C-DBEF-48F3-8C48-D5529C}
Last Modified:	11/24/2011 7:40 PM
Items:	4

The right panel, titled 'Views', shows a table of views with the 'New...' button highlighted in a red box. The table contains one view:

Name	Type	Default
All contacts	HTML	Yes

FIGURE 5-3. The New view button

5. Name this view "Test List View" and click OK.
6. Click "Test List View" to open it. Now that the view is open in SharePoint Designer, use the F12 keyboard shortcut to open the same view in the browser. In the browser, notice that the Last Name field is the default drop-down box with the list item menu.
7. Go back to SharePoint Designer. With the cursor in the web part table (in any column), click the Add/Remove Columns button on the Options tab.
8. Click Remove to remove all of the columns, so that the right side is empty. Add the following columns in this order: Full Name, Email Address, Job Title, and Company. Click OK when the screen looks like [Figure 5-4](#).

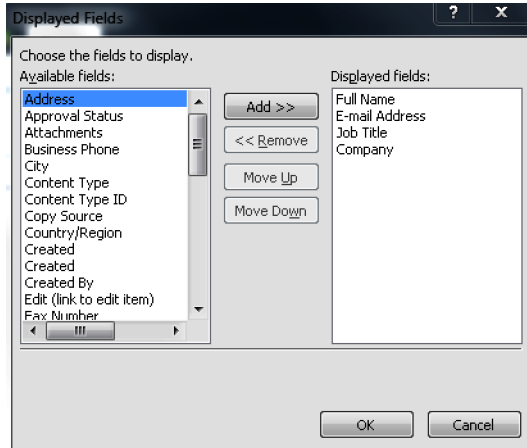


FIGURE 5-4. Add and remove columns

9. Click the Save button at the top left, go back to the browser, and refresh the web page to see the changes. Notice that any row can be selected, and the ribbon contains commands such as Edit Item, but there is no drop-down menu on the list items.
10. In SharePoint Designer, click to select the first person's name in your contact list. Click the little gray chevron next to the name and check the box next to Show List Item Menu and the box next to Show Link To Item, as shown in Figure 5-5.

NOTE

If an item in the list does not have a value in the Full Name column, no gray chevron will exist. It is a best practice to have a few test items and fill out all of the fields for those items when making these view changes.

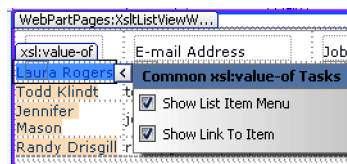


FIGURE 5-5. Choose List Item menu

11. Save the page again, go back to the browser, and refresh the page. Notice that each full name now has the list item drop-down.

12. It is also possible to create a hyperlink to the display form of an item without a drop-down menu. Click to select the first contact's job title, click the gray chevron, and check the box next to Show Link To Item.
13. Save the page and refresh the page in the browser again.

Notice in [Figure 5-6](#) that when the contact's full name is clicked and when the contact's job title is clicked, the display form pops up with all of the details about that contact.

Full Name	E-mail Address	Job Title
Laura Rogers	laurar@company.com	SharePoint Nerd
	todd@company.com	Professional Geek
	jenniferm@company.com	SharePoint Consultant
	randyg@company.com	Uber Rockstar

- View Item
- Edit Item
- Alert Me
- Manage Permissions
- Delete Item

FIGURE 5-6. Full name drop-down menu

Document Library

Since document library hyperlinks are structured differently, the same exercise will be done now, with a library instead of a list.

1. In your [Chapter 5](#) Test site in SharePoint Designer 2010, click Lists and Libraries on the left.
2. Click the name of the Project Files library to open it.
3. In the Views section, click New. Name the view "Test List View" and click OK.
4. Click the name of Test List View to open it.
5. With the cursor in one of the rows where the documents are listed, click the Add/Remove Columns button on the Options tab in the ribbon.
6. Remove all of the columns with the Remove button.
7. Add the following columns in this order: Type (icon linked to document), Name (linked to document), Title, Modified, and Modified By. When the screen looks like [Figure 5-7](#), click OK.

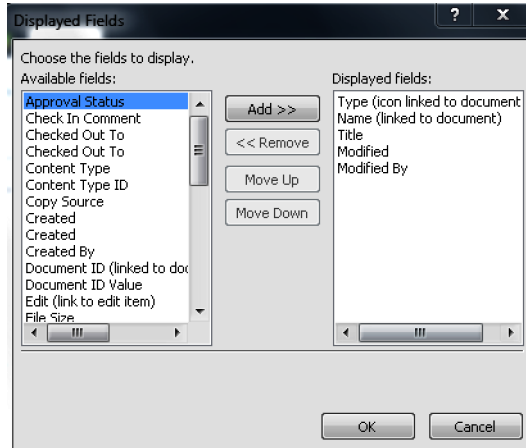


FIGURE 5-7. Select the library fields

8. Use the F12 shortcut key on the keyboard to go to the Browser view. On the library tab in the ribbon, click DatasheetView. Quickly type in titles for any documents that do not have them. The purpose of this is to show what was done with the SharePoint list in the previous section, where the drop-down box was placed on another field.
9. After all of the documents have titles, go back to SharePoint Designer and click the Refresh button at the top, near the Save button.
10. Click to select the title of the first document in the list. Click the little gray chevron next to it and check the boxes next to Show List Item Menu and Show Link To Item.
11. When you save and preview this page in the browser, you can tell that neither the new drop-down nor the hyperlink function as expected. The drop-down says “Edit in undefined,” which is a known bug. Also, the hyperlink that is supposed to open the document does not. It opens only the properties page for the file. Repeat the previous step and uncheck the box Show List Item Menu. Also uncheck Link To Item.

NOTE

There is a fairly complicated workaround to the bug regarding the “Edit in undefined” error here: <http://blogs.msdn.com/b/chunliu/archive/2010/09/29/enabling-ecb-menu-on-a-custom-column-in-sharepoint-2010-part-2.aspx>.

12. Click to select the Title of the first document. On the Insert menu, click Hyperlink.
13. Click the fx (function) button next to the Address.
14. Select the field called FileRef.urlencodeasurl and click OK.

15. Click OK on the Insert Hyperlink screen (shown in [Figure 5-8](#)).

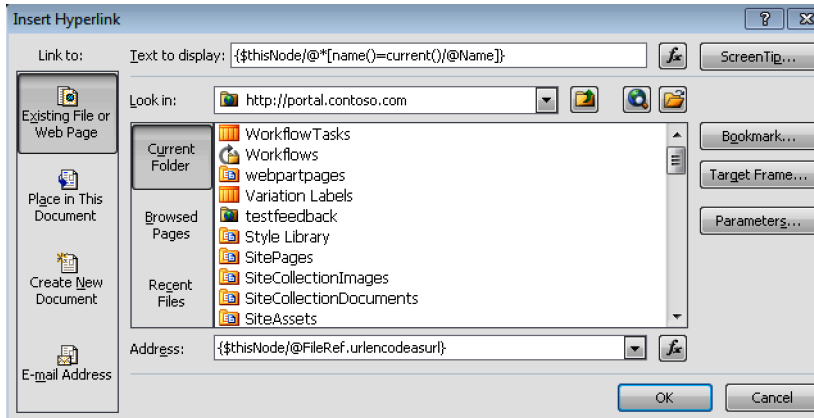


FIGURE 5-8. Select the field with the full URL

16. Save the page and press the F12 key to go over to the browser. Notice that when you click the title of any document, it now opens that document.

Sometimes you may want to display the date/time field with the date and not the time. If you remember back in SharePoint 2007, there was an easy user interface to uncheck a box next to Time. That helpful setting no longer exists, so here are the steps to accomplish the same thing:

1. In your XSLT List View web part in SharePoint Designer, click to select the actual date in the any row of the list, as shown in [Figure 5-9](#).

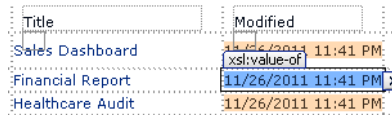


FIGURE 5-9. Select any date

2. Right-click the date and choose Edit Formula.
3. Replace the formula with this:
substring-before(string(\$thisNode/@*[name()='current()']/@Name), ' ')
4. The screen will look like [Figure 5-10](#). Click OK. Notice that the time is not showing anymore.

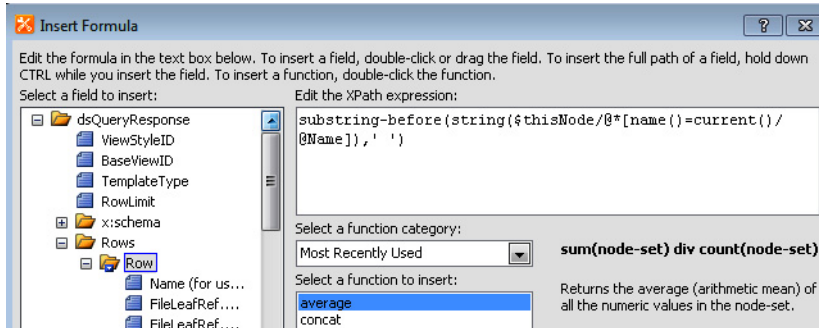


FIGURE 5-10. Show only the date

5. Save the page and close SharePoint Designer.

That concludes the work in the XSLT List View web part. The rest of the chapter covers “old school” DVWPs, which are similar to the way they were in SharePoint 2007 but different than the XSLT List views.

DVWP Hyperlinks

When it is time to create a DVWP, it can be created in a temporary location. This allows you to make changes and mistakes without affecting any web pages that are currently in use.

This first series of steps shows you how to create the web part page library, which can be used as the general location where DVWPs are developed and tested.

1. On your site, click Site Actions and choose New Document Library. This will be the location for creating your temporary web part pages.
2. In the Name box, type Web Part Pages.
3. In the Navigation section, select No so that this library is not displayed in the QuickLaunch. The simple reason is that this location is for temporary pages, and end users do not need to be confused by seeing an extra link in their navigation.
4. For the Document Template, choose Web Part page in the drop-down box.
5. Click Create.

NOTE

In addition to hiding the library from the QuickLaunch, you may want to change the permissions on this web part library so that only you can see your works in progress.

6. In the new Web Part Pages library, on the Documents tab in the ribbon, click New Document.
7. As shown in [Figure 5-11](#), name the new file TestListDVWP and click Create.

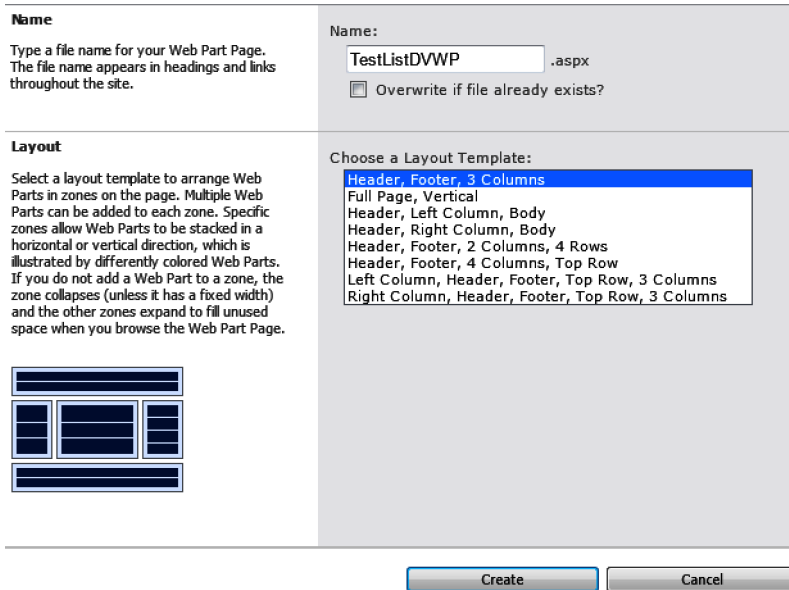


FIGURE 5-11. Create a new web part page

8. Click Stop Editing.
9. Repeat the last two steps to create another file called *TestLibraryDVWP.aspx*.
10. Click Stop Editing and close the browser.

Now that the two web part pages have been created, the web parts can be created on them. Remember that since this location is temporary, these web part pages can be deleted after the *.webpart* files have been put in production.

SharePoint List—DVWP

First, the Contacts list will be used as the basis of a new DVWP. Inherently, a DVWP will never automatically have any hyperlinks included, so it is important to understand how to create a hyperlink. If there is no link, people will be able to see your list of information, but will not be able to click on anything to see more details.

1. Open your [Chapter 5](#) Test site in SharePoint Designer 2010.
2. In the list of Site Objects on the left, click the little pin (shown in [Figure 5-12](#)) next to All Files to expand it.

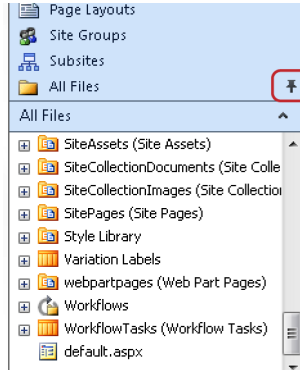


FIGURE 5-12. View the site file structure

3. Click the + (plus) next to the new Web Part Pages library to expand it.
4. Double-click TestListDVWP.aspx.
5. In the Customization section in the middle of the page, click Edit file.
6. Put your cursor in one of the blue rectangles on the page. These are the web part zones. Click the Insert tab in the ribbon.
7. Click Empty Data View, as shown in [Figure 5-13](#).

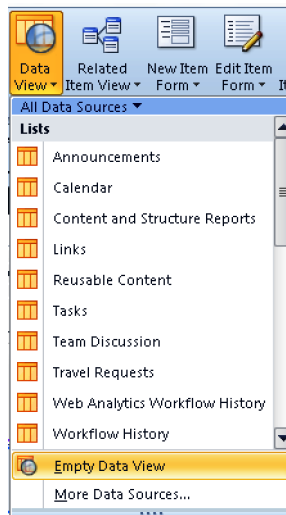


FIGURE 5-13. Insert a DVWP

8. As shown in [Figure 5-14](#), click the text that says Click here to select a data source.

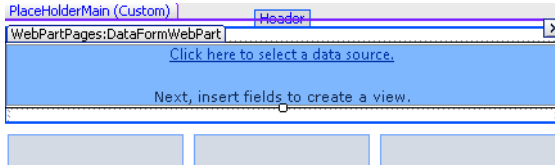


FIGURE 5-14. Select a data source

9. Select the Contacts list and click OK.
10. In the Data Source Details pane on the right, select the Full Name field. Hold down the Ctrl button and select the Email Address, Job Title, and Company fields, in that order.
11. Click the Insert Selected Fields As button at the top of the Data Source Details pane and choose Multiple Item View.
12. Click the Save button at the top left. Use the F12 shortcut key to quickly preview your page in the browser.

Now that this simple Contacts list has been inserted on the page, there are a few things to notice about this list compared to the default List view in SharePoint. There is no hyperlink to open or edit each contact, and the email address for each contact is not a hyperlink.

The next set of steps will teach you how to structure and create hyperlinks on this new DVWP.

1. In SharePoint Designer, click to select the first person's full name in the list of people.
2. On the Insert tab in the ribbon, click Hyperlink. Another way to accomplish the same thing is to right-click the first person's full name, choose Format As, and choose Hyperlink.
3. Click the fx button next to the Address box, shown in Figure 5-15.

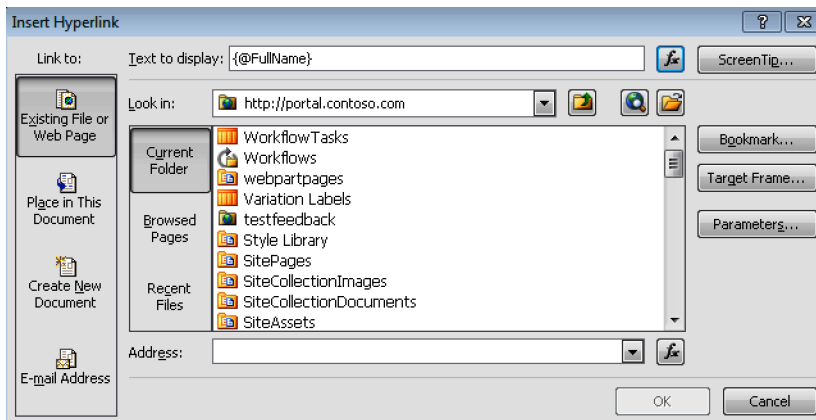


FIGURE 5-15. Insert a hyperlink

4. Check the box next to Show data values. With the data values shown, the screen will look similar to [Figure 5-16](#).

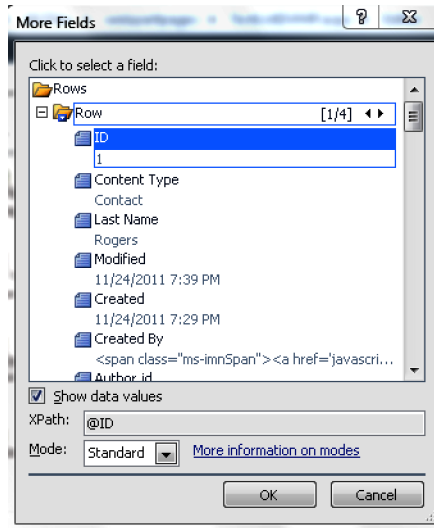


FIGURE 5-16. More fields

5. Scroll down and select the Path field. Click OK.
6. Put the cursor after the last } in the Address box.
7. Type the following:
`/DispForm.aspx?ID=`
8. Click the fx button again. This time, select the ID field and click OK.
9. When the new hyperlink looks like [Figure 5-17](#), click OK. Click the Save button at the top left.

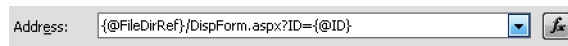


FIGURE 5-17. The completed hyperlink

The link to the display form is now complete! Now it's time to replicate the functionality of the Edit column that exists in regular List views. This entails adding a new column and inserting the Edit icon, then creating the hyperlink.

1. Put the cursor in the Full Name column of the DVWP
2. On the Table tab in the ribbon, click the Insert Left button.

3. In the header row of the new column, type the word **Edit**.
4. Put the cursor in the new column, in the second row in the Contact List table. On the Insert tab in the ribbon, click the Picture button.
5. For the filename, type the following (with your own site URL before the _): *http://portal.contoso.com/_layouts/images/edititem.gif*.

NOTE

The SharePoint Edit icon is being used here, but this can be any image at all, as long as you have the image URL.

6. Click the Open button.
7. On the Accessibility Properties dialog that pops up, type **Edit Item** in the Alternate Text box. Click OK.
8. At the bottom of the screen, click the Split button to see a partial Code view.
9. In the bottom half of the split screen, click to select the icon that you just inserted. In the Code view at the top, remove the “..” from the URL so that it looks like [Figure 5-18](#).

```
>img alt="Edit Item" src="/_layouts/images/edititem.gif" /</pre>
```

FIGURE 5-18. Edit icon relative URL

10. At the bottom of the screen, click the Design button so that the page is not in Split mode anymore.
11. Next, the hyperlink on the Full Name field will be copied to create a similar hyperlink for the Edit button. Click to select the first person's full name in the Full Name column. On the Insert tab, click Hyperlink. Select all of the text in the Address box and copy it to the clipboard. Click Cancel.
12. Click to select the Edit icon in the far left column. On the Insert tab in the ribbon, click Hyperlink.
13. In the Address box, paste the contents of the clipboard. Edit the part of the hyperlink to say *EditForm.aspx* instead of *DispForm.aspx*, as shown in [Figure 5-19](#). Click OK.

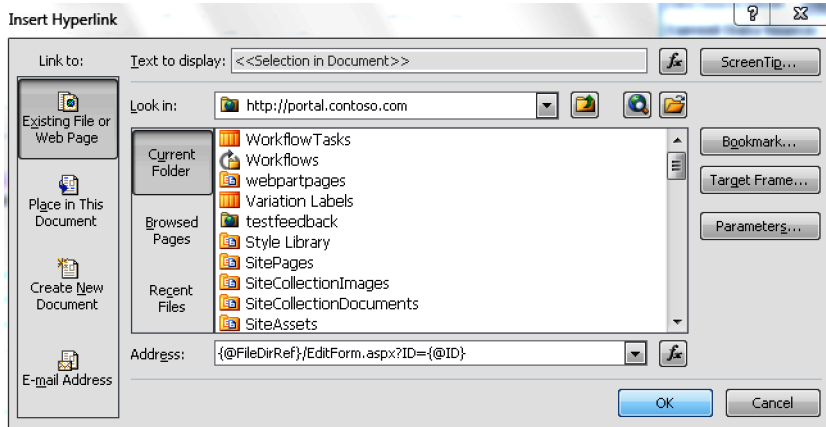


FIGURE 5-19. The edit form hyperlink

14. Click the Save button at the top left of SharePoint Designer.

Try out your DVWP in the browser. Notice that you can click the Edit button to get to the item's Edit form.

In the next set of steps, the Email column will be modified so that it is a hyperlink to create a new email message.

1. In the DVWP, click to select the first person's email address in the Email Address column. On the Insert tab in the ribbon, click Hyperlink.
2. On the Insert Hyperlink screen, put your cursor in the Address box and type the following: **mailto:**
3. Use the fx button next to the address box to select the Email Address field. Click OK.
4. When the Address box looks like Figure 5-20 on the Insert Hyperlink screen, click OK.

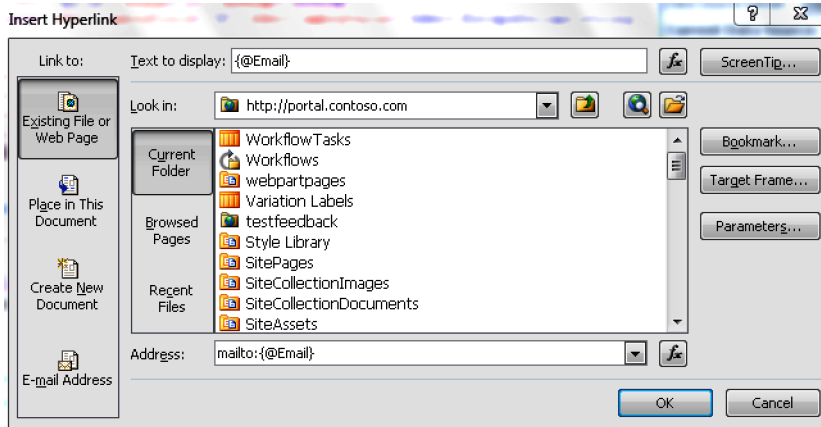


FIGURE 5-20. Insert email address hyperlink

5. Click the Save button at the top left. Press F12 on the keyboard to see a quick preview of this page in the browser.

Try each hyperlink that you have created. The final product is shown in Figure 5-21. Click the Edit icon to get to the Edit page, click the name of each contact to display the details about that contact, and click any contact's email address to quickly compose an email message to that person.

Contacts





	Full Name	E-mail Address	Job Title
	Laura Rogers	laurar@company.com	SharePoint Nerd
	Todd Klindt	toddk@company.com	Professional Geek
	Jennifer Mason	jenniferm@company.com	SharePoint Consultant
	Randy Drisgill	randyg@company.com	Uber Rockstar

FIGURE 5-21. Contacts list with hyperlinks

There are some additional tweaks that can make this DVWP a bit more useful. First, the column headings can be set up to be filterable and sortable.

1. On the *TestListDVWP.aspx* page in SharePoint Designer, put your cursor inside of the Contact list that is on the page. This is so that the DVWP's contextual ribbon will show all of the needed options at the top of the screen.
2. On the Design tab in the ribbon, check the box next to Sort & Filter on Headers.

- The first column (called “Edit”) will not look quite right. Put your cursor in the cell that says “Edit ”, and click the button at the bottom of the page that says <th.ms-vh>, as shown in [Figure 5-22](#).



FIGURE 5-22. The Edit column

- Press Delete on the keyboard and type the word **Edit** again. This is the easiest way to make the correction without having to modify a bunch of code.
- Save the page again, go back to the browser, and refresh the page.

When you save the page and preview it in the browser, notice that each of the column headings can now be sorted and filtered, just as with a regular SharePoint List view.

The next modification will be to mimic the shaded style of formatting in a List view. The row background will be set up so that every other row is slightly shaded—this will make it easier for viewers to discern one row’s data from the next.

- In the DVWP in SharePoint Designer, put the cursor in the first person’s row. On the Table tab in the ribbon, click the Select button and choose Select Row. With the row selected, the page will look like [Figure 5-23](#).

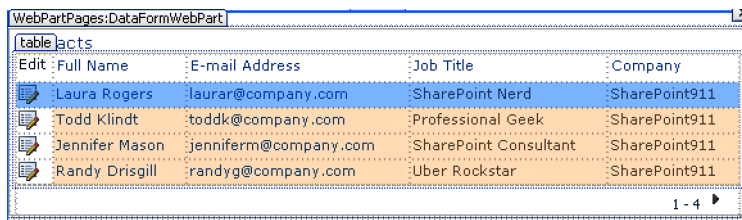


FIGURE 5-23. Select a row

- On the Options tab in the ribbon, click the Conditional Formatting button and choose Format Row.
- For the Field Name, choose [Row Number] and, in the Comparison column, choose Is Odd.

4. Click the Set Style button, shown in [Figure 5-24](#).

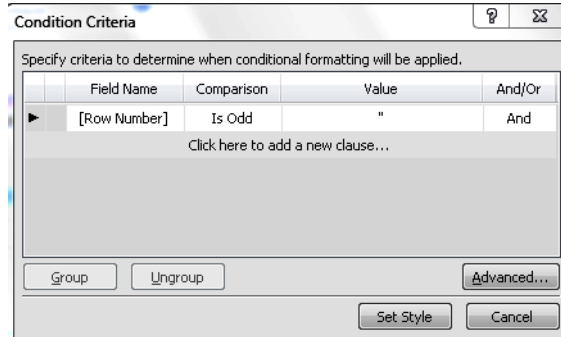


FIGURE 5-24. Row conditional formatting

5. On the Modify Style screen, select Background on the left. In the background-color drop-down box, choose a light gray color, as shown in [Figure 5-25](#). Click OK.

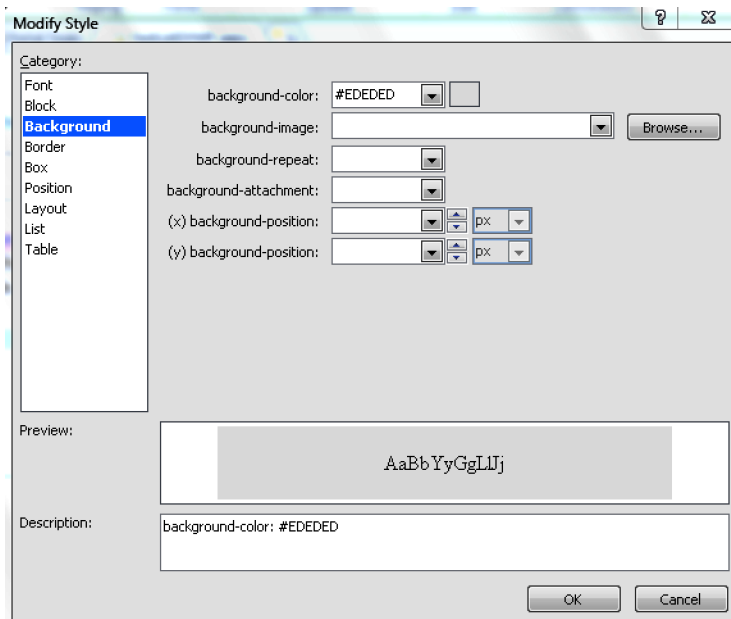


FIGURE 5-25. Set conditional formatting style

6. Click the Save button at the top left corner.

When you preview the page in the browser again, notice that the rows alternate color. The Contacts DVWP is done!

Moving the Web Part

Now that the DVWP has been finalized and tested on the temporary web part page, it can be moved to your home page or wherever its final resting place will be. Again, this method of using a temporary web part page allows you to create, modify, and test a DVWP in a separate location. The best practice is that each time the web part needs to be modified, it will be modified in that temporary page first, tested, and then moved to the home page again. This method is a bit time-consuming, but it's worth it when the web part blows up and the only thing that is broken is the temporary web part page, and not the home page.

1. In SharePoint Designer, with the cursor in your DVWP, click the Web Part tab in the ribbon.
2. In the Save Web Part section, click To File.
3. Pick a location on your computer (remember the location), and name the file *ContactsDVWP.webpart*. Click Save.
4. Navigate to the home page of the Chapter 5 Test site where you created the web part. Click Site Actions and choose Edit Page.
5. Click the Add a Web Part button.

NOTE

The interface for inserting web parts will be slightly different depending on whether the page is a web part page, a wiki page, or a publishing page. Since this site was created at the beginning of the chapter using the Document Workspace template, the home page is a web part page.

6. Click the Upload a Web Part button (see [Figure 5-26](#)).

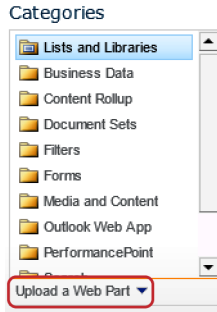


FIGURE 5-26. The Upload a Web Part button

7. Click the Browse button and navigate to the location on your computer where the `.webpart` file was saved in step 3.
8. Click the Open button.
9. Click the Upload button (see [Figure 5-27](#)).

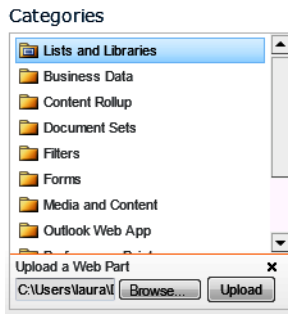


FIGURE 5-27. Upload a web part

10. Click Add a Web Part yet again.
11. In the Imported Web Parts category (shown in [Figure 5-28](#)), click to select the Contacts web part.

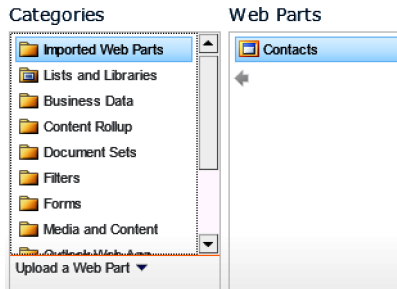


FIGURE 5-28. The Imported Web Parts category

12. Click the Add button on the right side of the screen.
13. Stop editing the page, and save if applicable. Notice that wiki pages and publishing pages have Save buttons, but web part pages do not.

Now you should see your custom Contacts web part on the home page, just like the one in [Figure 5-29](#)!

Contacts				
Edit	Full Name	E-mail Address	Job Title	Company
	Laura Rogers	laurar@company.com	SharePoint Nerd	SharePoint911
	Todd Klindt	toddk@company.com	Professional Geek	SharePoint911
	Jennifer Mason	jenniferm@company.com	SharePoint Consultant	SharePoint911
	Randy Drisgill	randyd@company.com	Uber Rockstar	SharePoint911

FIGURE 5-29. The final Contacts web part

NOTE

For Office 365 users: In SharePoint Online with Office 365, you will not see that Upload a Web Part button as in [Figure 5-26](#). As of the writing of this book, the version of SharePoint online does not offer the ability to import a web part. Here is the trick: on your home page, add `?ToolPaneView=2` to the end of the URL. In the pane on the right, click the blue word Browse and change it to Import. Then you'll have the ability to browse to the file, upload it, and add it to the page.

Document Library—DVWP

Since library hyperlinks are structured differently in libraries, there is a slightly different set of instructions for creating a link to open a document from a DVWP that has been created for a document library.

1. Open your site in SharePoint Designer 2010, and in the list of Site Objects on the left, click the little pin next to All Files to expand it.
2. Click the + (plus) next to the new Web Part Pages library to expand it.
3. Double-click TestLibraryDVWP.aspx. Click Edit file.
4. Put your cursor in one of the web part zones. Click the Insert tab in the ribbon.
5. Click Empty Data View, then click Click here to select a data source.
6. Select the Project Files library and click OK.
7. In the Data Source Details pane on the right, select the FileLeafRef.Name field. Click Insert Selected Fields As and choose Multiple Item View.
8. On the Options tab, click Add/Remove Columns. Select the appropriate columns so that the screen looks like [Figure 5-30](#). Click OK.

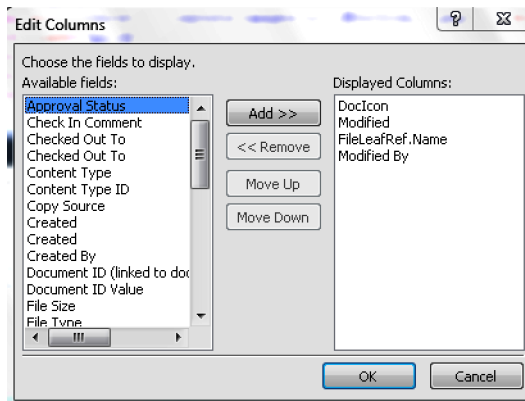


FIGURE 5-30. Add library columns

9. The Icon column will contain the name of the file type and, in a later step, we will change it to an icon.
10. The name of the file needs to become a link to open that file. Click to select the name of the first file in your library. On the Insert tab in the ribbon, click the Hyperlink button.
11. Next to the Address box, click the fx button. Check the box to Show data values.
12. Click to select the field called FileRef.urlencodeasurl, as shown in [Figure 5-31](#). Click OK.

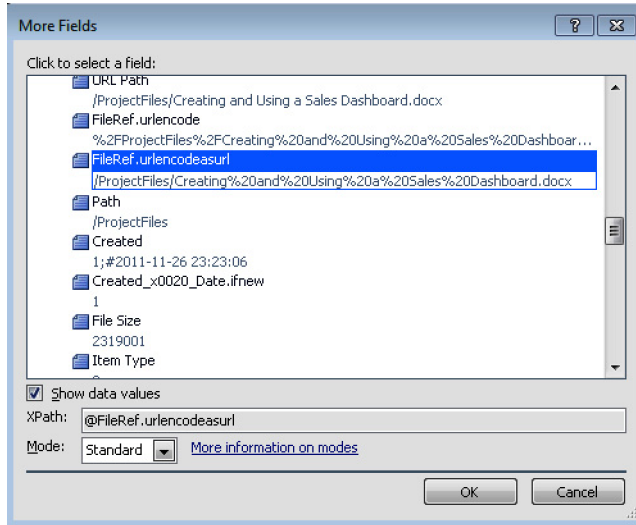


FIGURE 5-31. Select the file URL field

13. Click OK on the Insert Hyperlink screen (shown in Figure 5-32).

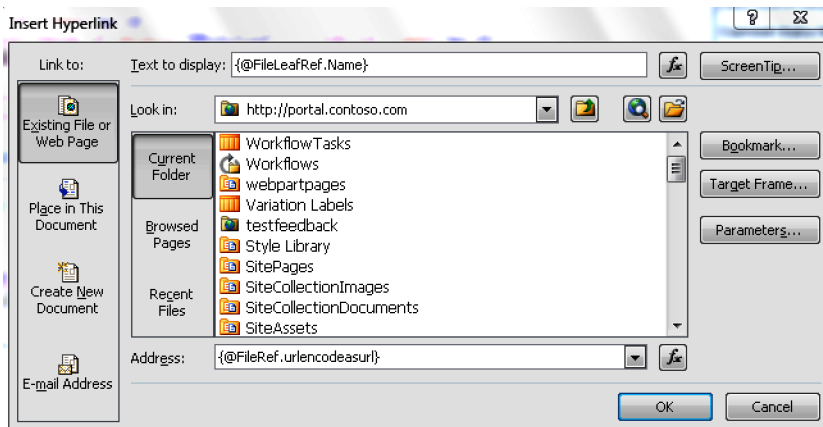


FIGURE 5-32. The Insert Hyperlink screen

14. Click the Save button at the top left corner, then use the F12 shortcut key to preview in the browser. Click any filename in the library to open that file.
15. Now that the filename hyperlink works, you can set up the file icon. In the DocIcon column, select the file type for the first document in the list. Click the Split button at the bottom of the screen to see the code.

16. When you select the file type value, the program should highlight the code for the item selected. If it does not, click the Design tab and click the Customize XSLT button, shown in [Figure 5-33](#). Choose Customize Item.

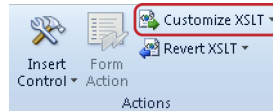


FIGURE 5-33. Customize the XSLT

17. In the code, select the DocIcon line, which is row 171 in [Figure 5-34](#).

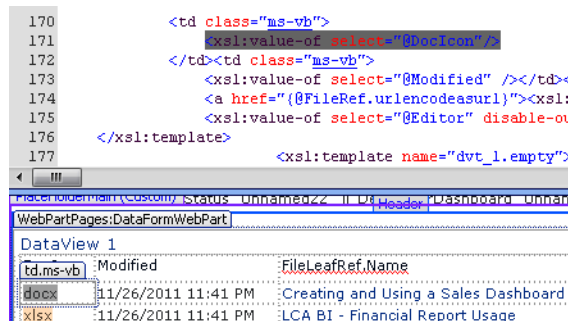


FIGURE 5-34. The DocIcon field

18. Paste the following code in place of the selected code:

```
</img>
```

19. Click the Save button at the top left, and see each office file icon displayed in the left column, as shown in [Figure 5-35](#). Preview your page in the browser.

DocIcon	Modified
	11/26/2011 11:41 PM
	11/26/2011 11:41 PM
	11/26/2011 11:41 PM
	11/27/2011 4:31 PM
	11/27/2011 4:31 PM

FIGURE 5-35. The DocIcon Image

Speaking of hyperlinks, it is often requested that hyperlinks to common items are *removed*, such as people's names and lookup items. Sometimes it is a bit confusing to site visitors to see several

different hyperlinks to choose from in a row item. This walkthrough will show you how to edit the Modified By column so that it is not a hyperlink.

1. Open the *TestLibraryDVWP.aspx* file in SharePoint Designer and make sure you are using Split view.
2. Click to select the first person's name in the Modified By column in the library.
3. In Code view, change the Editor field to say **Editor.title** (see line 175 in Figure 5-36).

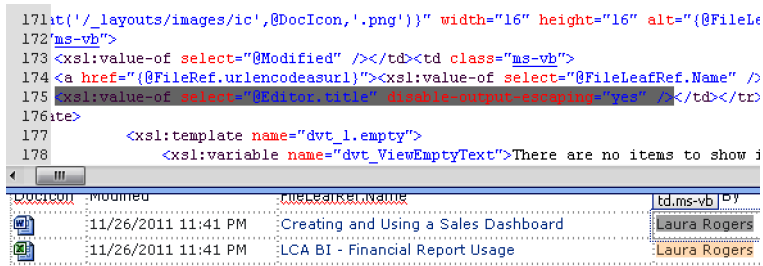


FIGURE 5-36. Remove the person hyperlink

4. Quickly clean up the column titles by simply typing new names. Change DocIcon to **Type** and change FileLeafRef.Name to **Name**.
5. Save the page and preview it in the browser.

If you have Office Web Apps installed on your SharePoint server, you can create a hyperlink that will open the file directly in the browser instead of with the client application. Here are the steps to add an Office Web Apps hyperlink. In this example, the file's icon will be used as the link, but feel free to use text in a new column, or just use a different image.

1. Open the *TestLibraryDVWP.aspx* file in SharePoint Designer.
2. Select the office icon in the far left column (Type) of the first document.
3. On the Insert tab, click the Hyperlink button.
4. Next to the Address box, click the fx button.
5. Select the field called serverurl.progid (as shown in Figure 5-37) and click OK.

NOTE

The URL looks correct, except that it has a number 1 at the beginning. This is the URL we need so that a substring function will get the portion of the field after the 1.

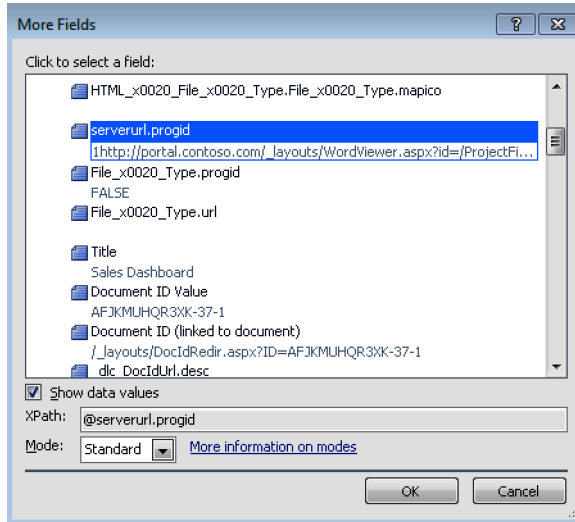


FIGURE 5-37. The *serverurl.progid* field

6. On the Insert Hyperlink screen, modify the address as shown in Figure 5-38. Click OK.

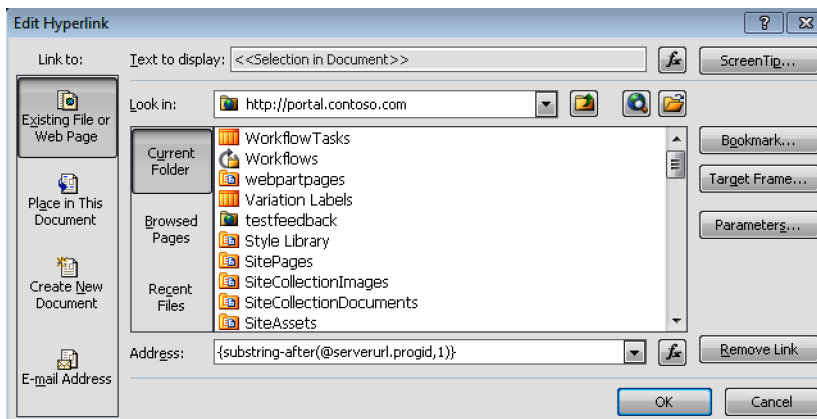


FIGURE 5-38. The Office Web Apps hyperlink

7. Save the page. Notice that now when you open this page in the browser and click the icon next to any file, that file will open in the browser.

Office Web Apps are great, because it doesn't matter if the end users have the Office Suite of software installed on their computers.

If you would like to place this DVWP on another page on your site, follow the instructions in [“Moving the Web Part” on page 105](#).

Modal Dialog Box

In the DVWPs we just created, notice that when you click to display or edit any item, the forms do not pop up in a modal dialog box. In SharePoint 2010 XSLT List View web parts, by default all list items pop up in a dialog box without navigating the user to a different web page. To mimic this functionality in a DVWP, we need to use some JavaScript code.

NOTE

There is a feature in the list and library advanced settings, called Launch forms in a dialog. This setting is set to Yes by default, but you can change it to No to turn off the modal dialog box.

Since our “test” version of the DVWP is still on each temporary web part page, we can still make changes there.

1. In SharePoint Designer, open the *TestListDVWP.aspx* file and click Edit File.
2. Instead of the full name field opening the *dispform.aspx* page, it is going to be changed to open the display form in a modal (pop-up) window. Click the Split button at the bottom of the screen.
3. Select the Full Name field. On the Insert tab, click the Hyperlink button.
4. Click the Remove Link button.
5. Click to select the Edit icon in the far left column. Click the Hyperlink button again and click the Remove Hyperlink button, shown in [Figure 5-39](#).

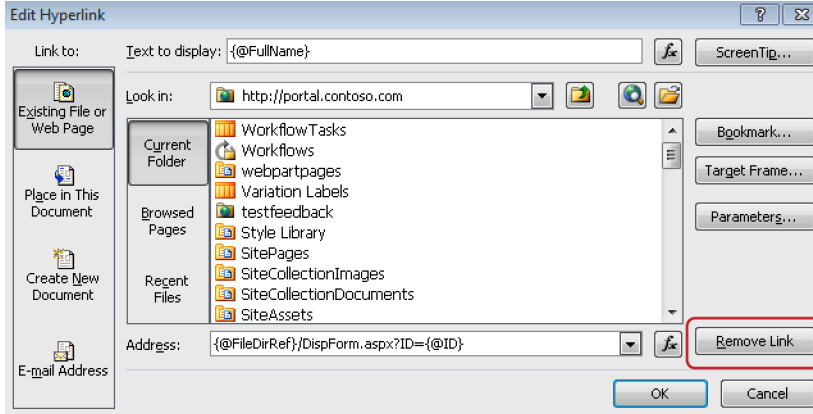


FIGURE 5-39. Remove the hyperlink

6. Click to select the Edit icon again. In the Code view, put the cursor to the left of `<imgtag` and add a couple of empty lines using the Enter key.
7. Enter the following code:

```
<script type="text/javascript">
function ModalPopup(url, title)
{
    var options = SP.UI.$create_DialogOptions();
    options.url = url;
    options.title = title;
    options.dialogReturnValueCallback = function() {
        SP.UI.ModalDialog.RefreshPage(SP.UI.DialogResult.OK); };
    SP.UI.ModalDialog.showModalDialog(options);
}
</script>
```

8. Now that you have inserted the JavaScript, you can create the display form and edit form hyperlinks again. Put the cursor in front of `<img alt="Edit Item"`. Use the Enter key to add a blank line.
9. Enter the following code in the new blank line and type `` after the image.

```
<a href="javascript:ModalPopup('/Lists/Contacts/EditForm.aspx?ID={@ID}',
    'Edit {@FullName}');">
```

10. Put the cursor in front of `<xs1:value-of select="@FullName" />` and insert another blank line.
11. Enter the following code in the new blank line and type `` after the full name field:

```
<a href="javascript:ModalPopup('/Lists/Contacts/DispForm.aspx?ID={@ID}',
  'Display {@FullName}');">
```

The code is now complete. The following snippet shows the JavaScript and the first two columns of the table:

```
<td class="ms-vb">
<script type="text/javascript">
function ModalPopup(url, title)
{
  var options = SP.UI.$create_DialogOptions();
  options.url = url;
  options.title = title;
  options.dialogReturnValueCallback = function() {
    SP.UI.ModalDialog.RefreshPage(SP.UI.DialogResult.OK); };
  SP.UI.ModalDialog.showModalDialog(options);
}
</script>
<a href="javascript:ModalPopup('/Lists/Contacts/EditForm.aspx?ID={@ID}',
  'Edit {@FullName}');">

</a></td>
<td class="ms-vb">
<a href="javascript:ModalPopup('/Lists/Contacts/DispForm.aspx?ID={@ID}',
  'Display {@FullName}');">
<xsl:value-of select="@FullName" />
</a></td>
```

If your links do not work, try using absolute URLs instead of relative ones in the hyperlink code.

Summary

In this chapter you learned that there is a difference between XSLT List View web parts and DVWPs, and that hyperlinks are configured a bit differently in each. You also learned a couple of extra tidbits such as how to remove hyperlinks where they are not needed.

Check out my blog, [SharePoint for Non-Developers](#).

Building a Quote of the Day Web Part in SharePoint 2010

Waldek Mastykarz

Back in 2009, I wrote an article about how to create a simple Quote of the Day web part by leveraging SharePoint's 2007 customization capabilities (see [Figure 6-1](#)).

Although the code behind the web part was very simple, the web part itself had a few limitations such as dependency on the jQuery library or use only in the same site as the location of the Quotes list.

One of the new pillars of SharePoint 2010 is the JavaScript Object Model (JSOM), which allows you to build dynamic web applications (<http://msdn.microsoft.com/en-us/library/ee538253.aspx>). In the first part of this chapter, you will learn how to build your own Quote of the Day web part using the new SharePoint 2010 JSOM.

Whenever you want to redistribute your SharePoint solution, you should create a SharePoint Solution Package (WSP). Unfortunately, in many environments, deploying WSPs is restricted and allowed only after passing reviews. This process might be very expensive for a simple solution such as the Quote of the Day web part. The concept of the *Sandbox* has been introduced in SharePoint 2010 specifically for such scenarios. Sandbox allows you to deploy SharePoint Packages in an isolated process without the involvement of IT and can be done by the site collection administrator.

In the second part of this chapter, we will create a Sandboxed Solution that will wrap the Quote of the Day web part with all of its assets, making it ready for redistribution.

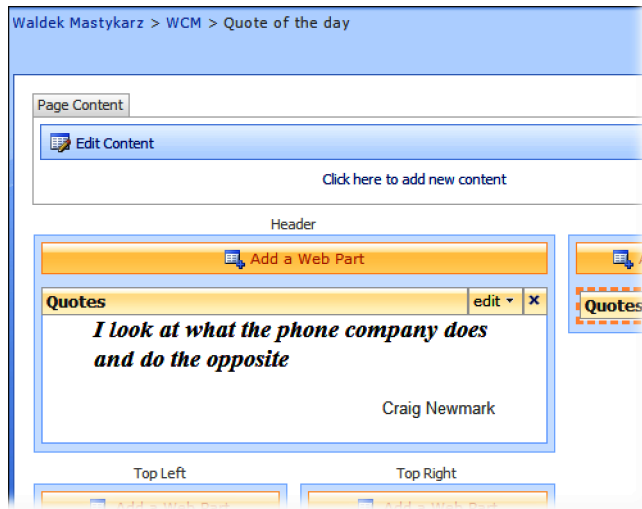


FIGURE 6-1. Quote of the Day web part in SharePoint 2007

Part I: Building the Quote of the Day Web Part

Before we start, let's take a moment to look at what we want to achieve. We want our Quote of the Day web part to display a randomly chosen quote and its author from a list of quotes. We want to pick one quote for each day instead of a random one each time the user opens the page. To simplify the maintenance of quotes and authors, we want to store them in a SharePoint list.

Quote of the Day Ingredients

Let's break the above description into requirements:

- List with quotes and authors
- Randomly picking a quote from the list of quotes
- The random value should be chosen based on the current day

Preparing the Quotes List

The first step in creating a working Quote of the Day web part is to create and configure a list we will use for managing quotes.

Creating the Quotes list

Let's start off by creating the Quotes list.

1. In the Site Actions menu, choose More Options (Figure 6-2).

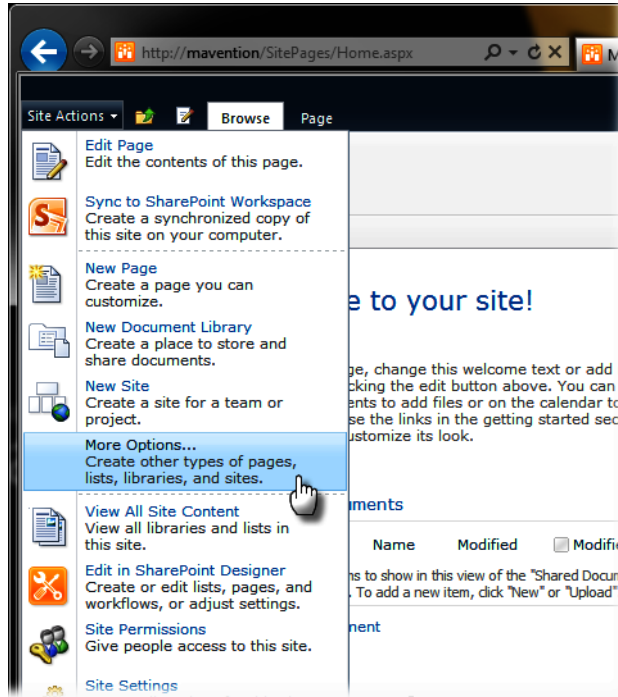


FIGURE 6-2. "More Options..." highlighted in the Site Actions menu

2. From the list of available templates, choose Custom List (shown as #1 in Figure 6-3) and type **Quotes** for the Title (#2 in Figure 6-3). Confirm your selection by clicking the Create button (#3 in Figure 6-3).

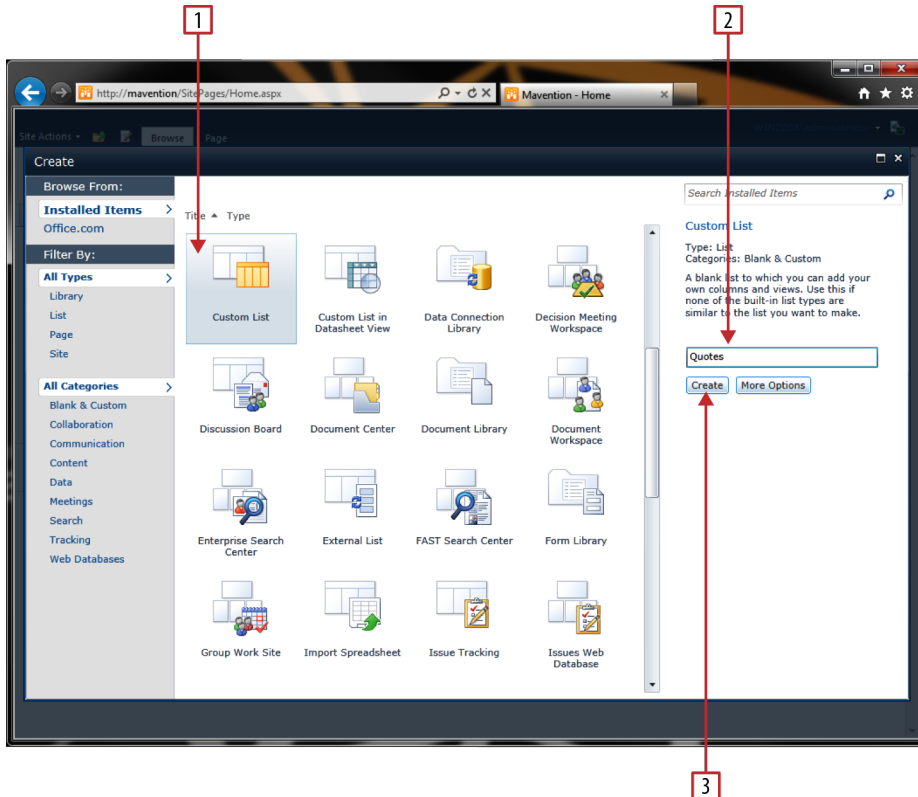


FIGURE 6-3. Creating new Quotes List

Configuring columns of the Quotes list

Now that we have the Quotes list created, let's proceed with configuring its columns so that, for every quote, we can store the quote itself and its author.

First we will change the name of the Title column to Author.

1. In the ribbon, activate the List tab and, in the Settings group, click the List Settings button (Figure 6-4).
2. From the list of List Columns, choose the Title column. Change the name of the column to Author (#1 in Figure 6-5) and confirm your changes by clicking OK (#2 in Figure 6-5).

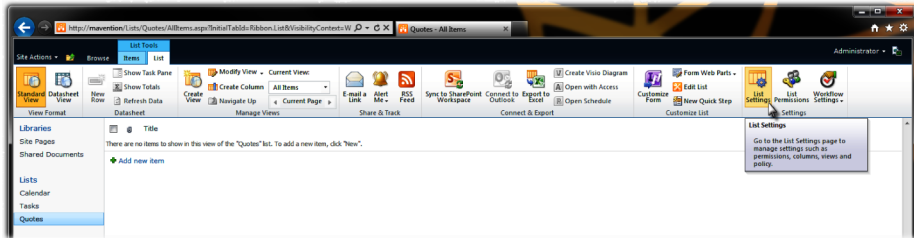


FIGURE 6-4. List Settings button highlighted in the ribbon

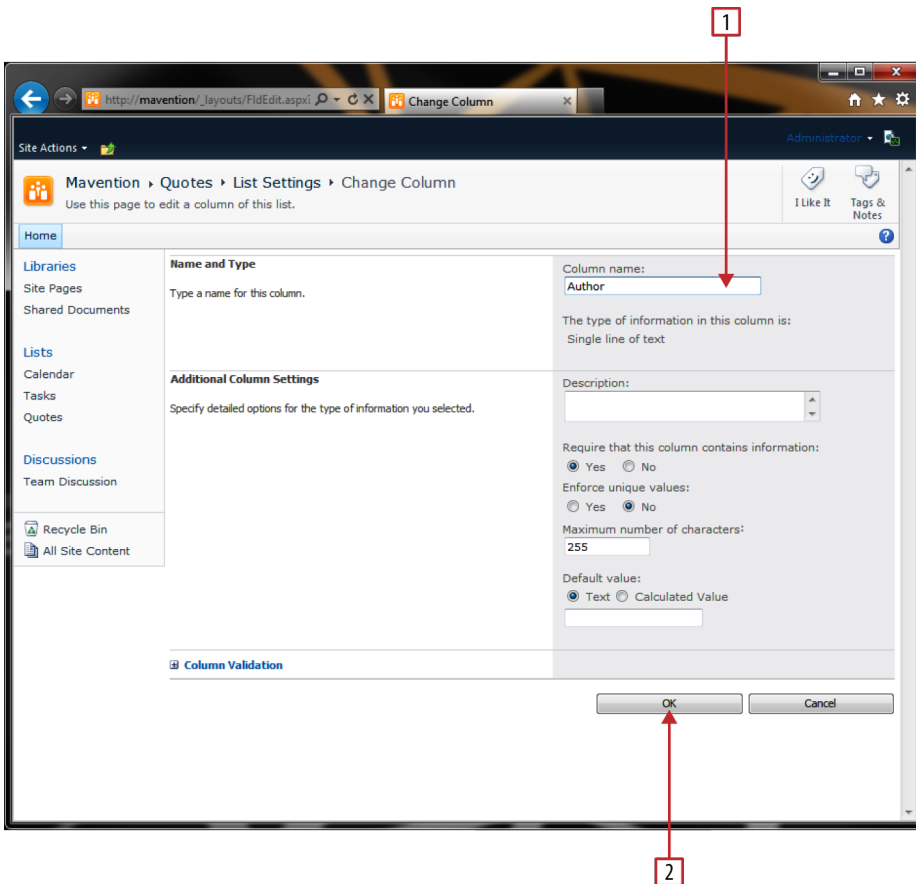


FIGURE 6-5. Changing the title of the Title column

3. The next step is to add a new column for storing quotes. On the List Settings page, in the Columns section, click the Create column link. Type **Quote** for the column name (#1 in

Figure 6-6). From the list of available types, select the Multiple lines of text field type (#2 in Figure 6-6). In the Additional Columns Settings section, choose Plain text (#3 in Figure 6-6). Confirm your choices by clicking OK (#4 in Figure 6-6).

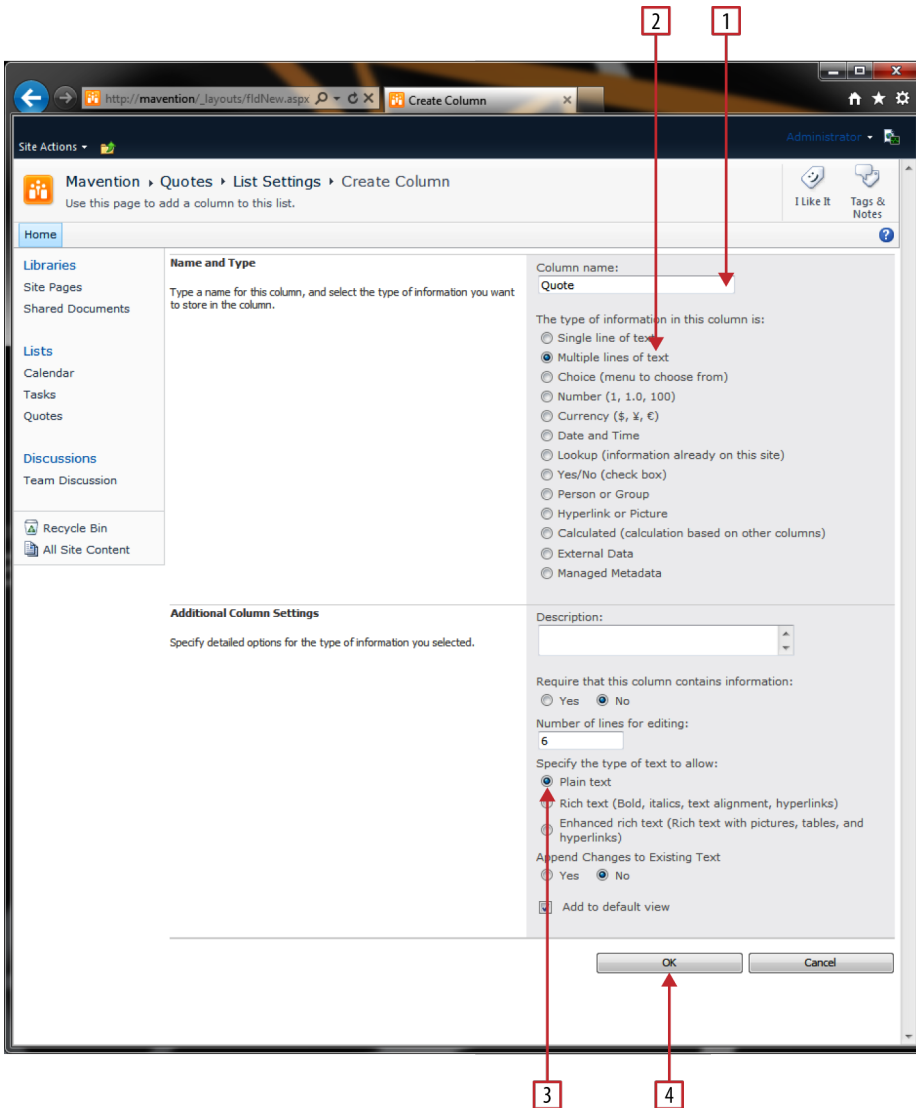


FIGURE 6-6. Creating the Quote column

After adding the Quote column, it should appear in the list of available columns, as shown in Figure 6-7.

Columns

A column stores information about each item in the list. The following columns are currently available in this list:

Column (click to edit)	Type	Required
Author	Single line of text	✓
Quote	Multiple lines of text	
Created By	Person or Group	
Modified By	Person or Group	

[Create column](#)

FIGURE 6-7. List of columns in the Quotes list

To finish configuration of the Quotes list, let's add a couple of quotes. Figure 6-8 shows a sample Quotes list filled with a few quotes.

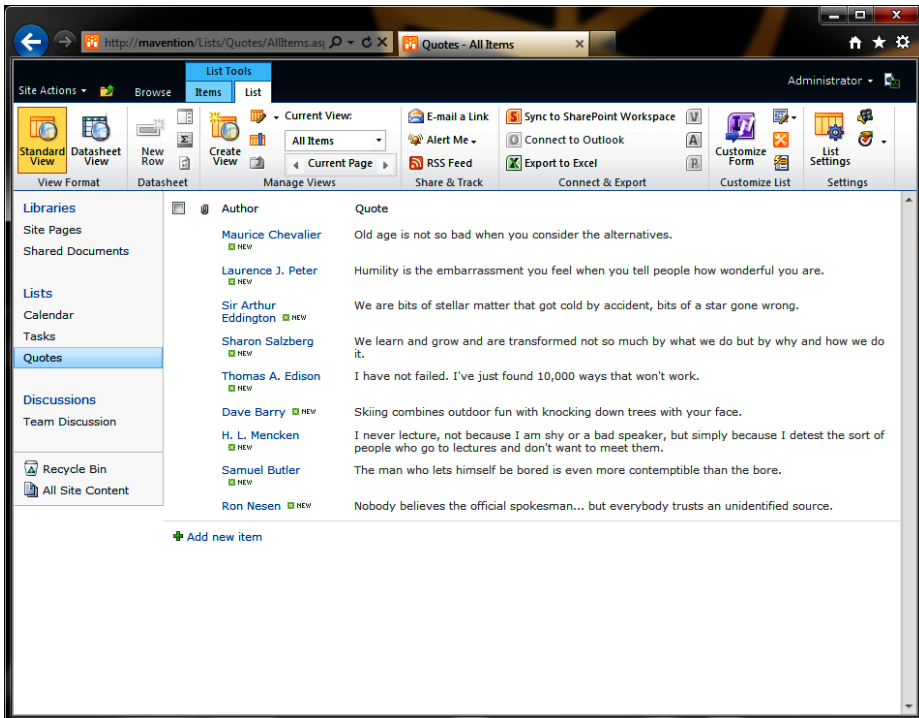


FIGURE 6-8. Quotes list containing a few quotes

Now that the Quotes list is ready, let's proceed with creating the Quote of the Day web part.

Building the Quote of the Day Web Part

To keep the web site as simple as possible, in this part of the chapter we will use the Content Editor web part (CEWP) to store all of our JavaScript. Using the SharePoint 2010 JSOM, we will retrieve the Quote of the Day and display it on the screen. In the second part of this chapter we will consider a slightly more advanced scenario and prepare our solution for redistribution using a Sandboxed Solution.

1. Let's start off by adding the CEWP to the page. We'll call it Quote of the Day (Figure 6-9).

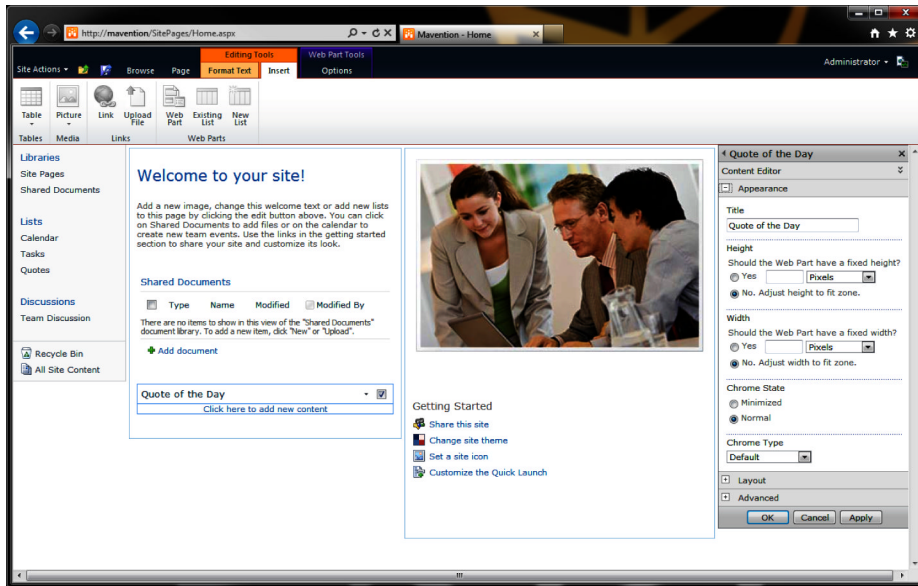


FIGURE 6-9. Quote of the Day web part added to page

2. Click the Click here to add new content link. In the ribbon, from the Markup group, click the HTML drop-down control and choose the Edit HTML Source option. In the HTML Source dialog, enter the following code snippet:

```
<style type="text/css">
.message { text-align: center; }
.message img { vertical-align: middle; }
blockquote p { font: italic bold 1.6em/1.2 "Times New Roman" , serif; }
blockquote p.author {
    font: normal 1em/1.2 sans-serif;
    color: #666;
    text-align: right;
}
```

```

</style>
<div id="quoteOfTheDay"></div> ❶
<script type="text/javascript">
Date.prototype.getFullDate = function () { ❷
    var dateAsString = '';
    dateAsString += this.getFullYear();
    var month = this.getMonth() + 1;
    if (month < 10) {
        dateAsString += 0;
    }

    dateAsString += month;

    var day = this.getDate();
    if (day < 10) {
        dateAsString += 0;
    }

    dateAsString += day;

    return parseInt(dateAsString);
}

SP.SOD.executeOrDelayUntilScriptLoaded(function() { ❸
    Type.registerNamespace('Mavention.QuoteOfTheDay');
    Mavention.QuoteOfTheDay.displayQuote = ❹
function (webUrl, listName, container) {
    container.innerHTML = '<div class="message">\ ❺
\
Loading Quote of the Day...</div>';

    context = SP.ClientContext.get_current();
web = context.get_site().openWeb(webUrl);
list = web.get_lists().getByTitle(listName);
    query = new SP.CamlQuery();
    query.set_viewXml('<View><ViewFields><FieldRef Name="Title"/>\
<FieldRef Name="Quote"/></ViewFields></View>');
    items = list.getItems(query); ❻
    context.load(items); ❼
    context.executeQueryAsync(Function.createDelegate(this, function () {
        numberOfQuotes = items.get_count(); ❽ ❾

```



```

        if (numberOfQuotes > 0) {
            now = new Date();
            quoteNumber = now.getFullDate() % numberOfQuotes;
            quoteItem = items.get_item(quoteNumber);
            quote = quoteItem.get_item('Quote'); ❶
            author = quoteItem.get_item('Title');

            container.innerHTML = '<blockquote><p>' + quote +
'</p><p class="author">' + author + '</p></blockquote>';
        }
        else { ❷
            container.innerHTML = '<div class="message">
There are no quotes in the list yet</div>';
        }
    }, Function.createDelegate(this, function () { ❸
        container.innerHTML = '<div class="message">

An error has occurred while loading Quote of the Day</div>';
    }));
}

Mavention.QuoteOfTheDay.displayQuote('/', 'Quotes', ❹
document.getElementById('quoteOfTheDay'));
}, 'sp.js'); ❺
</script>

```

- ❶ We start the snippet by defining a container where we will display the Quote of the Day.

```
<div id="quoteOfTheDay"></div>
```

B) Next we add the JavaScript code responsible for loading the Quote of the Day. To avoid naming conflicts, we define our custom code in a namespace.

```

<script type="text/javascript">
Type.registerNamespace('Mavention.QuoteOfTheDay');
Mavention.QuoteOfTheDay.displayQuote = function () {
}
</script>

```

- ❷ We want our web part to be reusable and work with Quotes lists no matter their location, so let's add parameters that will allow us to specify which list we want to use. Since loading the data is asynchronous, we will also need a parameter to determine where on the page the quote should be displayed.

```

<script type="text/javascript">
Type.registerNamespace('Mavention.QuoteOfTheDay');
Mavention.QuoteOfTheDay.displayQuote = function (webUrl, listName, container) {
}
</script>

```

Now we'll add the code responsible for loading quotes from the given Quotes List.

- 5 Because loading quotes can take a while, we start with displaying a message communicating that loading the quote is in progress.

```

container.innerHTML = '<div class="message">\
\
Loading Quote of the Day...</div>';

```

- 6 Next, we start the process of loading all available quotes. Since there is no function available that would allow us to directly load a random quote, we will first retrieve all quotes from the Quotes list and pick a random one for today. To minimize the amount of data that will be downloaded, we specify in the CAML query that only two columns (Author and Quote) should be retrieved.

```

context = SP.ClientContext.get_current();
web = context.get_site().openWeb(webUrl);
list = web.get_lists().getByTitle(listName);
query = new SP.CamlQuery();
query.set_viewXml('<View><ViewFields><FieldRef Name="Title"/>\
<FieldRef Name="Quote"/></ViewFields></View>');
items = list.getItems(query);

```

- 7 With that, we're ready to load all available quotes. Because the process is asynchronous, we have to provide two delegate functions that will be executed: one for when the loading completes and one should there be any errors.

```

context.load(items);
context.executeQueryAsync(Function.createDelegate(this, function () {
// success
}), Function.createDelegate(this, function () {
// error
}));

```

- 8 After loading completes, we need to get the total number of quotes in the list. We will use it to calculate the number of the random quote for today.

```

numberOfQuotes = items.get_count();

```

- 2 As I mentioned before, we don't want to display a random quote on each page load. Instead, we want to pick one quote for today and display it during the whole day for

all users. To do that, we have to calculate a random number based on today's date, so we will need a function that will return today's date as number (lines 13–31).

```
Date.prototype.getFullDate = function () {
    var dateAsString = '';
    dateAsString += this.getFullYear();
    var month = this.getMonth() + 1;
    if (month < 10) {
        dateAsString += 0;
    }

    dateAsString += month;

    var day = this.getDate();
    if (day < 10) {
        dateAsString += 0;
    }

    dateAsString += day;

    return parseInt(dateAsString);
}
```

- 9 By using the prototype notation, we add the `getFullDate` function to all JavaScript Date objects, which makes it easier for us to use the new function. With that, we can retrieve the quote for today.

```
numberOfQuotes = items.get_count();
if (numberOfQuotes > 0) {
    now = new Date();
    quoteNumber = now.getFullDate() % numberOfQuotes;
    quoteItem = items.get_item(quoteNumber);
}
```

- 10 Once we have our quote for today picked, we can display it, replacing the progress message (lines 55–59).

```
quote = quoteItem.get_item('Quote');
author = quoteItem.get_item('Title');

container.innerHTML = '<blockquote><p>' + quote +
    '</p><p class="author">' + author + '</p></blockquote>';
```

- 11 Should we choose an empty Quotes list, we can display an information message stating that no quotes have been entered yet.

```

if (numberOfQuotes > 0) {
    // omitted for brevity
}
else {
    container.innerHTML = '<div class="message">\
There are no quotes in the list yet</div>'
}

```

- 12 With our code in place, it's time to call our function passing the URL of the site, the name of our Quotes list, and where on the page the Quote of the Day should be displayed.

```

Mavention.QuoteOfTheDay.displayQuote('/', 'Quotes', document.getElementById
('quoteOfTheDay'));

```

- 13 If you tried to run the script at this moment, you would end up with an error. Since SharePoint 2010 loads JSOM asynchronously, and because our code depends on it, we have to wait on the execution of our script until JSOM has been loaded. We can do this by using the SharePoint SOD (Script On Demand) capability provided with SharePoint 2010 JSOM and wrap our code in the `SP.SOD.executeOrDelayUntilScriptLoaded` function.

```

SP.SOD.executeOrDelayUntilScriptLoaded(function() {
    // code goes here
}, 'sp.js');

```

3. This completes our code snippet. After you paste it in the CEWP's HTML Source dialog, confirm the changes by clicking the OK button. Next, click the OK button in the CEWP's Editor pane—you should see the Quote of the Day as shown in [Figure 6-10](#). (You can ignore the HTML modification warning.)

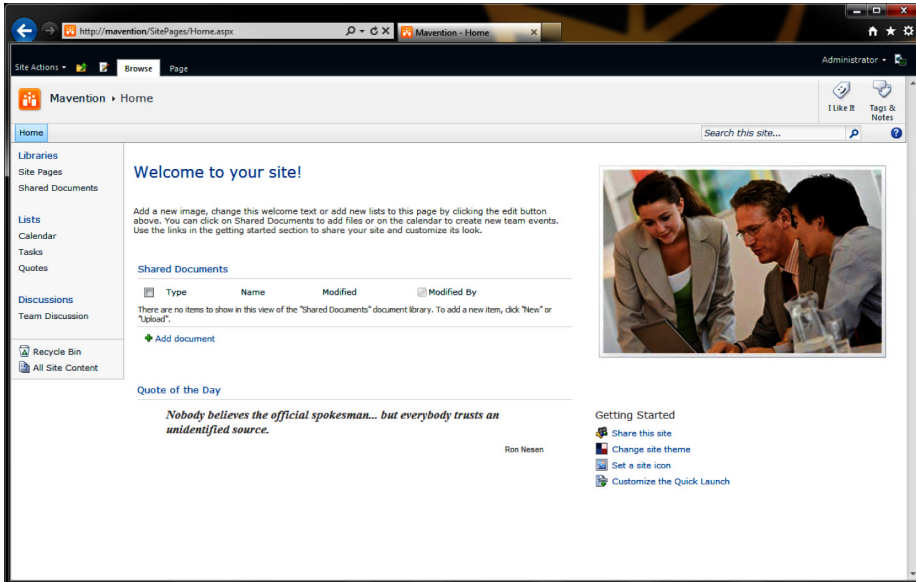


FIGURE 6-10. Quote of the Day displayed on the page

With that, we have created a Quote of the Day web part that dynamically loads a quote for today using the SharePoint 2010 JSOM. In the next part of this chapter, we will prepare the Quote of the Day web part for redistribution by including it in a Sandboxed Solution.

Part II: Preparing Quote of the Day Web Part for Redistribution

During the creation of the Quote of the Day web part, we leveraged the SharePoint 2010 JSOM to retrieve and display a random quote from a list of quotes, all without using a single line of custom code.

Although the Quote of the Day web part is working, it has a few disadvantages in its current configuration. First, managing JavaScript through the CEWP is fragile and error-prone. Yes, the CEWP allows Power Users to enrich pages with dynamic functionality. However, all it takes is one wrong character to break the whole page! Additionally, all of that JavaScript is included as-is on the page. While adding one Quote of the Day Web Part to a page would probably not make much difference, adding a few of them, especially combined with other similar web parts, will have a significant impact on the size and performance of the page.

Another thing worth considering is the fact that redistribution of the Quote of the Day web part at this moment is rather inconvenient. To get it right, you have to provide some kind of instructions to whoever would want to use the web part as well. The whole process consists of multiple steps, and getting even one of them wrong might break the web part.

Whenever you want to make your SharePoint customizations redistributable, you should provide them as SharePoint Packages. The challenge with SharePoint Packages, in SharePoint 2007, is that the deployment process requires IT involvement and introduces some risks to the farm. For this reason, many administrators weren't keen on deploying new SharePoint Packages without reviewing and testing their contents first, which is limiting from a business point of view.

One of the big improvements in SharePoint 2010 is the introduction of Sandboxed Solutions. From a deployment point of view, they are SharePoint Packages as well, but they can be deployed by a site collection administrator without any involvement by IT! Additionally, to keep your website safe, SharePoint 2010 Sandbox uses a number of counters that monitor how well Sandboxed Solutions is running. As soon as the counter values are exceeded, SharePoint 2010 Sandbox will shut down, keeping your website operational.

In this part of the chapter, we will create a Sandboxed Solution for our Quote of the Day web part. Although you can follow the process described in the first part as a power user, you will need some developer skills and knowledge of the SharePoint 2010 platform to understand the steps described in this part. If you need more information about the different components we discuss here, I suggest you visit the [SharePoint Developer Center website on MSDN](#), which contains a lot of valuable information on developing solutions on the SharePoint 2010 platform.

Quote of the Day Sandboxed Solution Ingredients

Before we open up Visual Studio, let's have a look at what we need to accomplish in order to make our Quote of the Day web part redistributable.

One of the key requirements of the Quote of the Day web part is that it displays a random quote using a Quotes list. To do this, we will need a Quote list. Since we want our users to be able to create the Quotes list themselves, we will provide them with a Quote list list definition so they can create a fully configured list with a single mouse click.

For loading and displaying a random quote from a selected Quotes list, we will keep using the SharePoint 2010 JSOM. Although we are allowed to use managed code in Sandboxed Solutions, using JavaScript is more beneficial. Because all of it is being executed in browser, it doesn't cost any server resources and therefore doesn't increase the Sandbox counters.

As I mentioned before, at this moment the Quote of the Day web part includes its JavaScript in the page, which increases the page size. When preparing our web part for redistribution, we will extract that JavaScript and store it in a separate file. This will allow us to load it only once when there are multiple Quote of the Day web parts on one page. Additionally, the browser will cache it, so it won't have to be downloaded on subsequent visits.

Finally, when all of our work is ready, we should make the Quote of the Day web part available to users so they can add it to pages.

Preparing SharePoint Project

Let's start off by creating a SharePoint Project that helps us build a Sandboxed Solution for our Quote of the Day web part.

1. In Visual Studio 2010, in the New menu, choose Project. In the list of available Project Templates, choose Empty SharePoint Project (#1 in Figure 6-11). Enter **Mavention.SharePoint.QuoteOfTheDay** for the Project Name (#2 in Figure 6-11). Optionally, you can change your project's location. Finally, confirm the project creation by clicking OK (#3 in Figure 6-11).

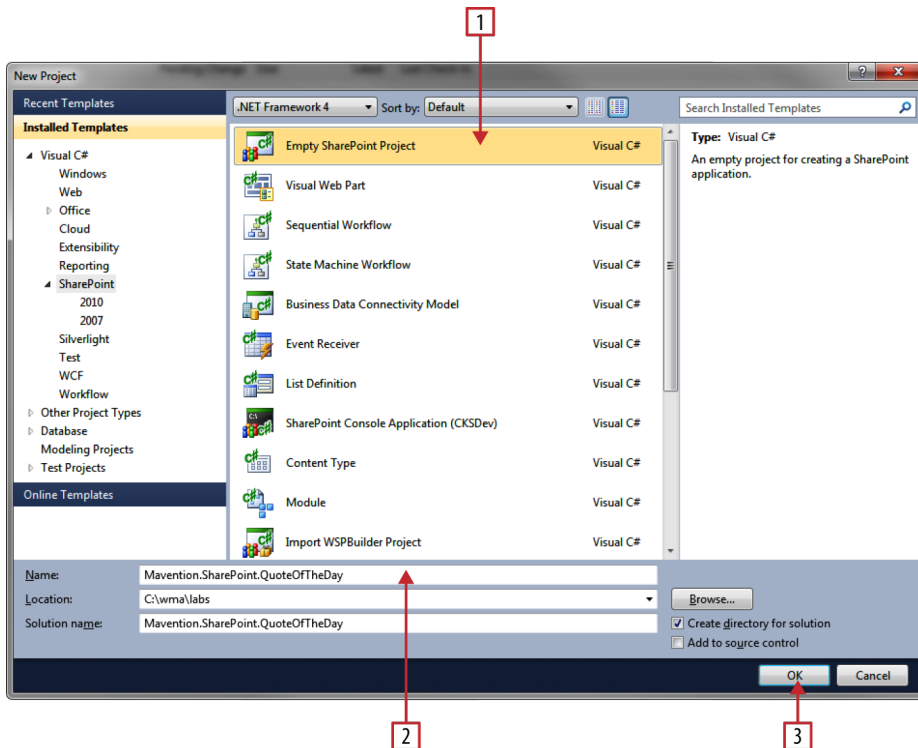


FIGURE 6-11. Creating new SharePoint project for the Quote of the Day web part

2. In the SharePoint Customization Wizard, provide a valid URL of a SharePoint site that you can use to test the solution (#1 in Figure 6-12). For the desired trust level, choose Deploy as sandboxed solution (#2 in Figure 6-12). Confirm your choices by clicking the Finish button (#3 in Figure 6-12).

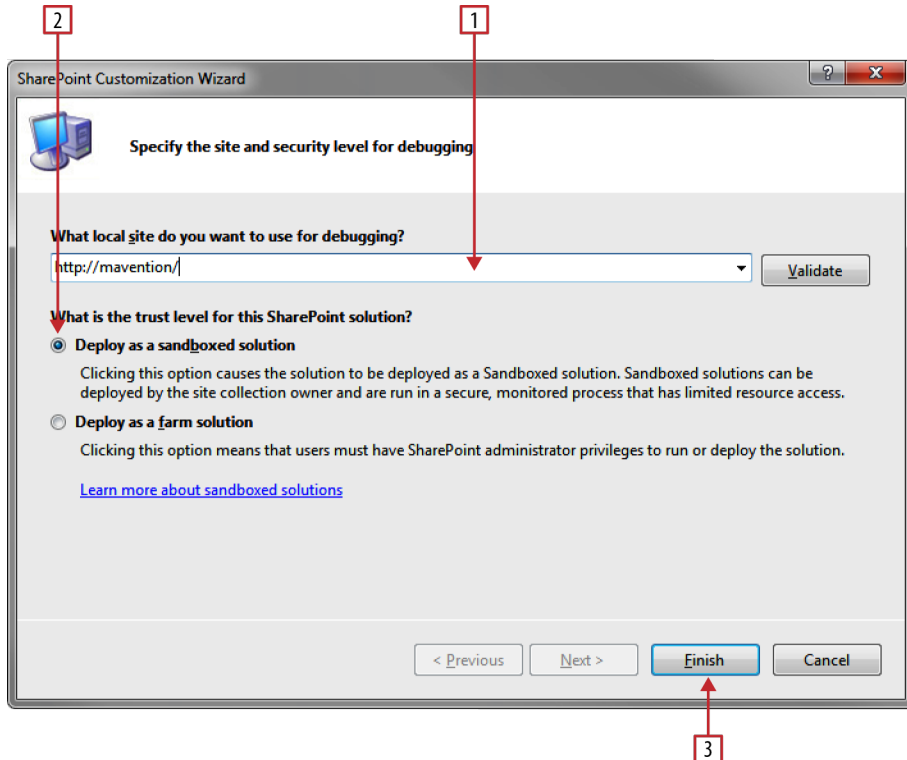


FIGURE 6-12. Configuring SharePoint project deployment target and type

3. Because our solution is pretty simple, we will use a single Feature to provision all required assets. To create a Feature, access Solution Explorer, right-click the Features node under our project and, in the context menu, choose the Add Feature option as shown in Figure 6-13.
4. In the Title field, enter **Mavention Quote of the Day Web Part**. For the Description, use Installs the Mavention Quote of the Day web part and all its assets. In the Scope drop-down list, choose Site. Finally, change the name of the new Feature from Feature1 to **Core** (because all Feature names are prepended with the project name, and as we have only one Feature in our project, this name is sufficient). Figure 6-14 shows the configured Core Feature.

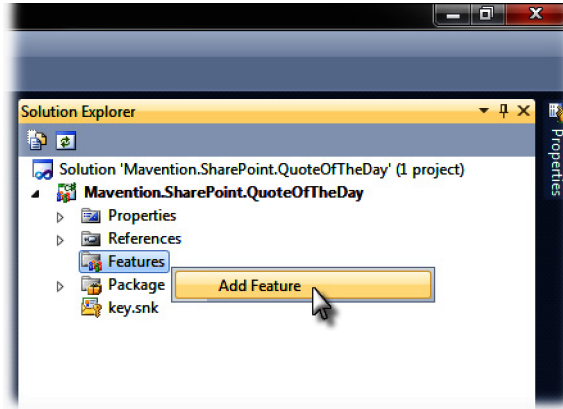


FIGURE 6-13. Adding a new Feature to our SharePoint project

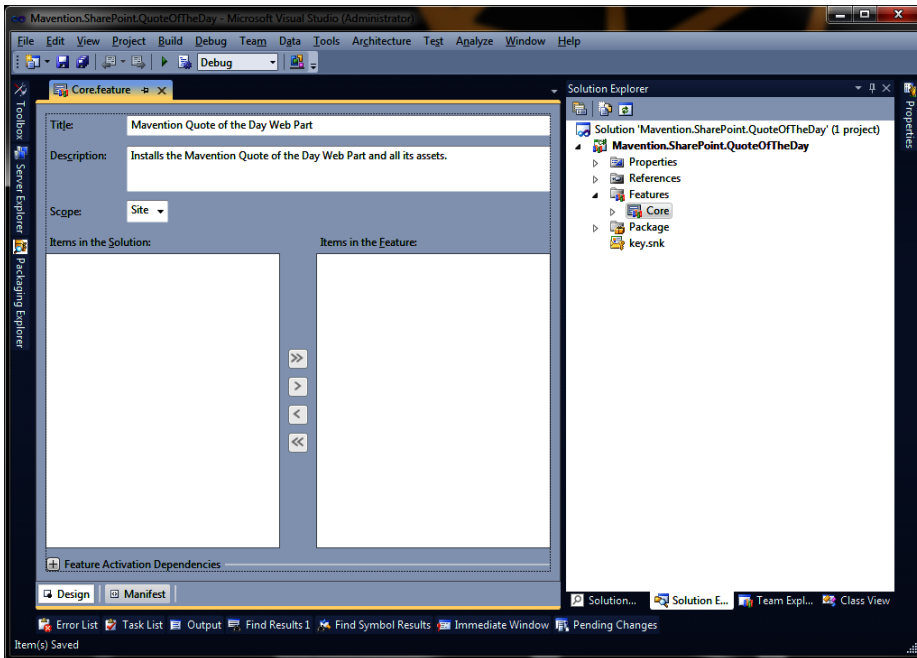


FIGURE 6-14. Configuring the Core Feature

With that, we are ready to start adding assets to our Quote of the Day Sandboxed Solution.

Creating the Quotes List List Definition

The first asset we will add to our project is the Quotes list list definition. This will allow users to easily create preconfigured Quotes lists.

1. To add a list definition in Solution Explorer, right-click the project, select the context menu, and choose Add→New Item, as shown in [Figure 6-15](#).

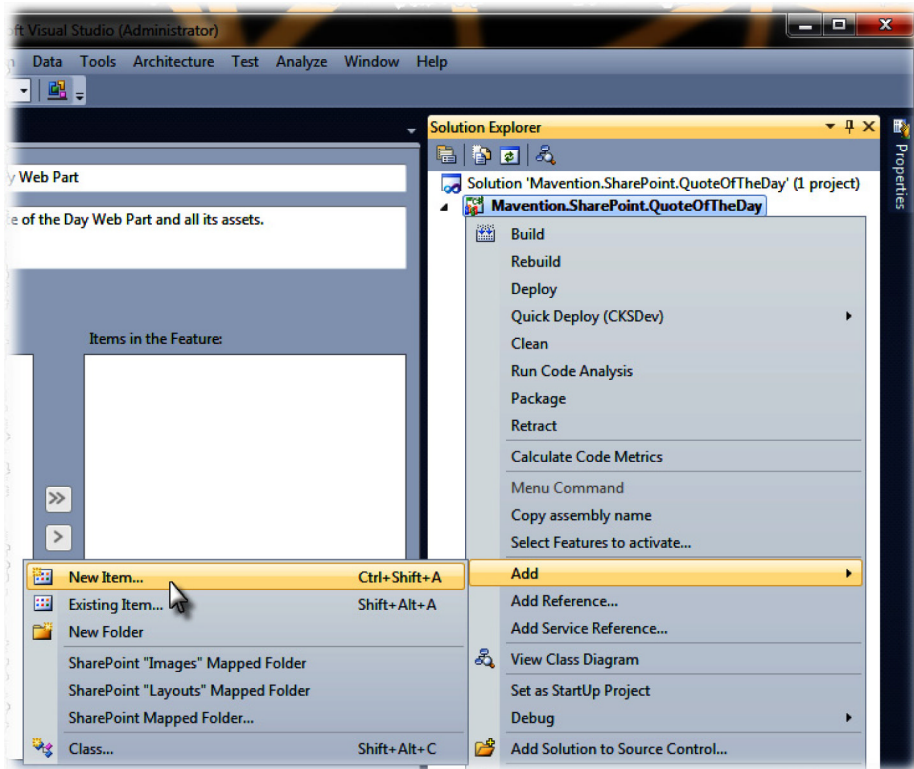


FIGURE 6-15. Adding a new project item

2. In the list of available Project Item Templates, choose List Definition, and enter **QuotesListListDefinition** for the name (see [Figure 6-16](#)).
3. In the SharePoint Customization Wizard, enter the Display Name **Quotes List** and, for the Type, choose Custom List. Because we don't want to create a Quotes list automatically, clear the Add list instance for this list definition checkbox, as shown in [Figure 6-17](#). Click Finish.

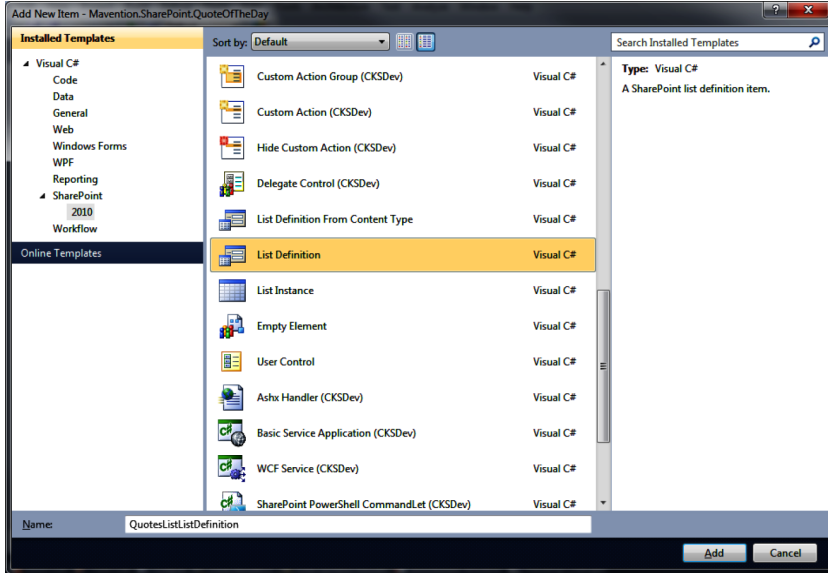


FIGURE 6-16. Adding the QuotesListListDefinition to our project

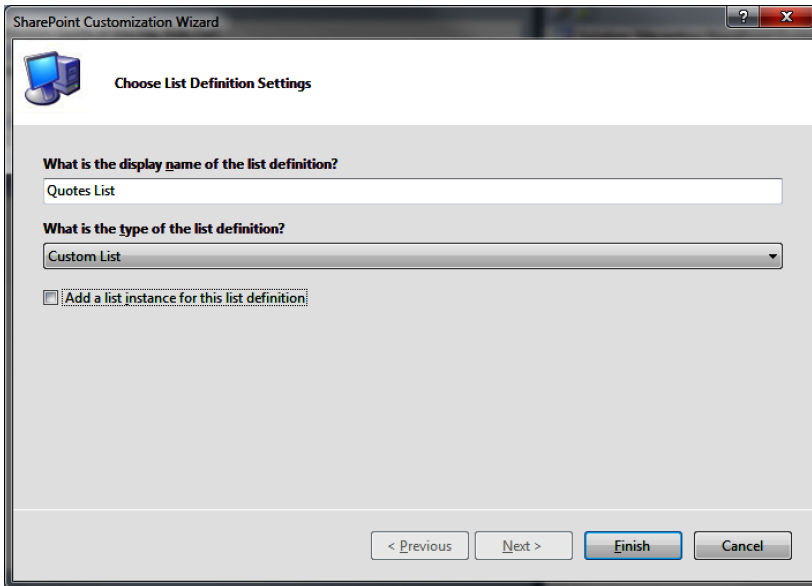


FIGURE 6-17. Configuring list definition options

- After creating the list definition, change its description by navigating to the *Elements.xml* file under the QuotesListListDefinition SharePoint Project Item (SPI). Change the value of the Description attribute to the following (see [Figure 6-18](#)):

A list of quotes to be used by the Quote of the Day web part

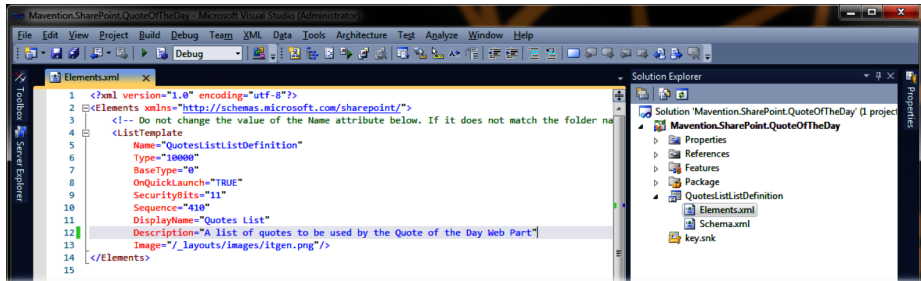


FIGURE 6-18. Changing the Quotes list list definition description

- Next, we will edit the schema of the Quotes list list definition. Open the *Schema.xml* file and enter the following code snippet:

```

<?xml version="1.0" encoding="utf-8"?>
<List xmlns:ows="Microsoft SharePoint" Title="Quotes List"
FolderCreation="FALSE" DisableAttachments="TRUE"
Direction="$Resources:Direction;" Url="Lists/Quotes" BaseType="0"
xmlns="http://schemas.microsoft.com/sharepoint/">
<Metadata>
<Fields>
<Field DisplayName="Author" Name="Title" Type="Text"/> ❶
<Field DisplayName="Author" Name="LinkTitle" Type="Text"/> ❶
<Field DisplayName="Quote" Name="Quote" Type="Note" /> ❷
</Fields>
<Views>
<View BaseViewID="0" Type="HTML" MobileView="TRUE" TabularView="FALSE">
<Toolbar Type="Standard" />
<Xslink Default="TRUE">main.xsl</Xslink>
<RowLimit Paged="TRUE">30</RowLimit>
<ViewFields>
<FieldRef Name="LinkTitleNoMenu"></FieldRef>
</ViewFields>
<Query>
<OrderBy>
<FieldRef Name="Modified" Ascending="FALSE"></FieldRef>
</OrderBy>

```

```

</Query>
<ParameterBindings>
<ParameterBinding Name="AddNewAnnouncement"
Location="Resource(wss,addnewitem)" />
<ParameterBinding Name="NoAnnouncements"
Location="Resource(wss,noXinviewofY_LIST)" />
<ParameterBinding Name="NoAnnouncementsHowTo"
Location="Resource(wss,noXinviewofY_ONET_HOME)" />
</ParameterBindings>
</View>
<View BaseViewID="1" Type="HTML" WebPartZoneID="Main"
DisplayName="$Resources:core,objectiv_schema_mwsidcamlidC24;"
DefaultView="TRUE" MobileView="TRUE" MobileDefaultView="TRUE"
SetupPath="pages\viewpage.aspx" ImageUrl="/_layouts/images/generic.png"
Url="AllItems.aspx">
<ToolBar Type="Standard" />
<XslLink Default="TRUE">main.xsl</XslLink>
<RowLimit Paged="TRUE">30</RowLimit>
<ViewFields>
<FieldRef Name="LinkTitle" DisplayName="Author"/> ❶ ❸
<FieldRef Name="Quote"/> ❷
</ViewFields>
<Query>
<OrderBy>
<FieldRef Name="ID"></FieldRef>
</OrderBy>
</Query>
<ParameterBindings>
<ParameterBinding Name="NoAnnouncements"
Location="Resource(wss,noXinviewofY_LIST)" />
<ParameterBinding Name="NoAnnouncementsHowTo"
Location="Resource(wss,noXinviewofY_DEFAULT)" />
</ParameterBindings>
</View>
</Views>
<Forms>
<Form Type="DisplayForm" Url="DispForm.aspx" SetupPath="pages\form.aspx"
WebPartZoneID="Main" />
<Form Type="EditForm" Url="EditForm.aspx" SetupPath="pages\form.aspx"
WebPartZoneID="Main" />
<Form Type="NewForm" Url="NewForm.aspx" SetupPath="pages\form.aspx"

```

```

WebPartZoneID="Main" />
</Forms>
</MetaData>
</List>

```

The most important modifications are:

- ❶ Changing the DisplayName of the Title field to Author
 - ❷ Adding the Quote field
 - ❸ Making sure that both Author and Quote are visible in the default List view
6. After saving all the changes, let's test our Quotes list list definition by deploying the Quote of the Day Sandboxed Solution to our site. In the Debug menu, choose Start Debugging (or press F5). A browser window with your site will open. In the Site Actions menu, choose More Options. From the list of available templates, choose Quotes List and enter **Famous Quotes** for the name (Figure 6-19). Confirm your choice by clicking the Create button.

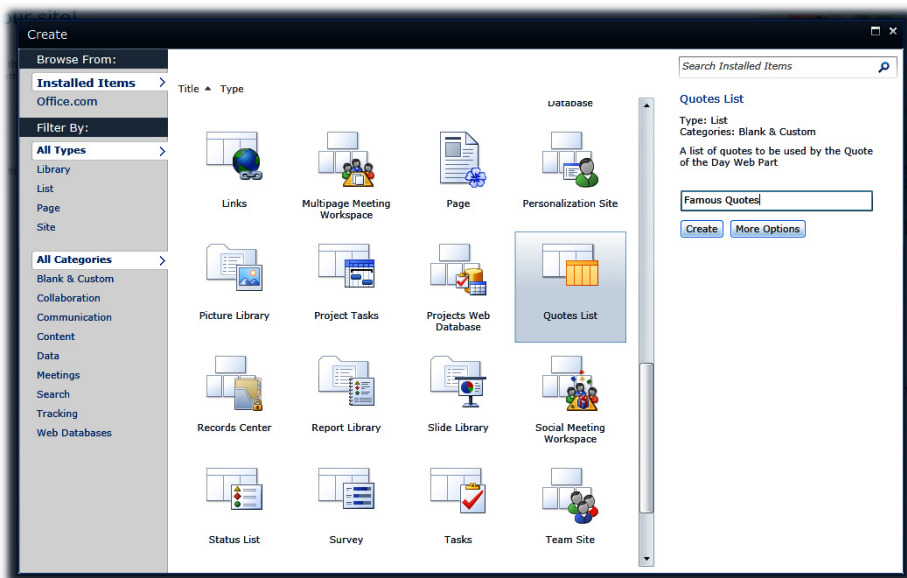


FIGURE 6-19. Creating a Quotes list

In the default view, you will see the Author and Quote columns, just as when adding new quotes. Figure 6-20 shows the Famous Quotes list filled with some quotes.

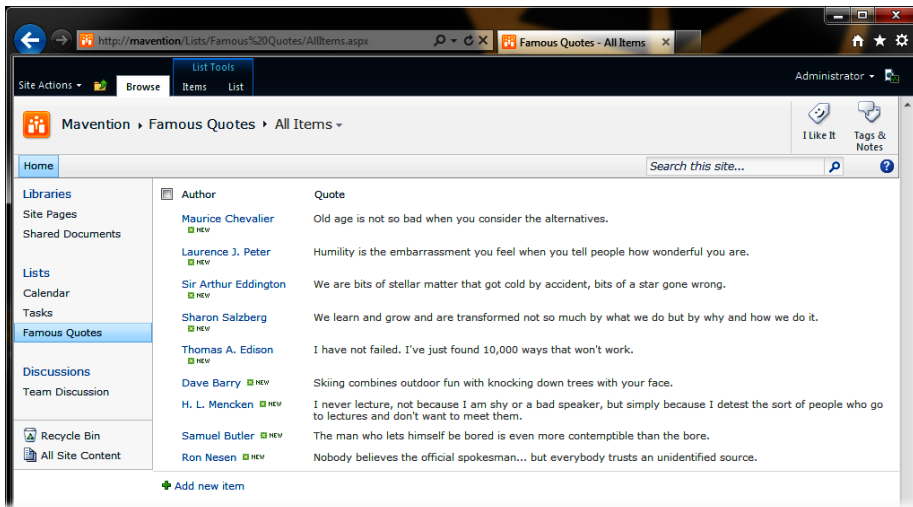


FIGURE 6-20. Quotes list filled with quotes

With that, we are ready to proceed with building the Quote of the Day web part.

Preparing for Building the Quote of the Day Web Part

Let's add a new web part SPI to our project.

1. Right-click the project and, in the context menu, choose Add → New Item. From the list of available Project Item Templates, choose Visual Web Part (Sandboxed) and call it **QuoteOfTheDayWebPart**, as shown in Figure 6-21. Confirm your choice by clicking the Add button.

NOTE

If you don't see this Project Item Template, install Visual Studio 2010 SharePoint Power Tools from <http://visualstudiogallery.msdn.microsoft.com/8e602a8c-6714-4549-9e95-f3700344b0d9>.

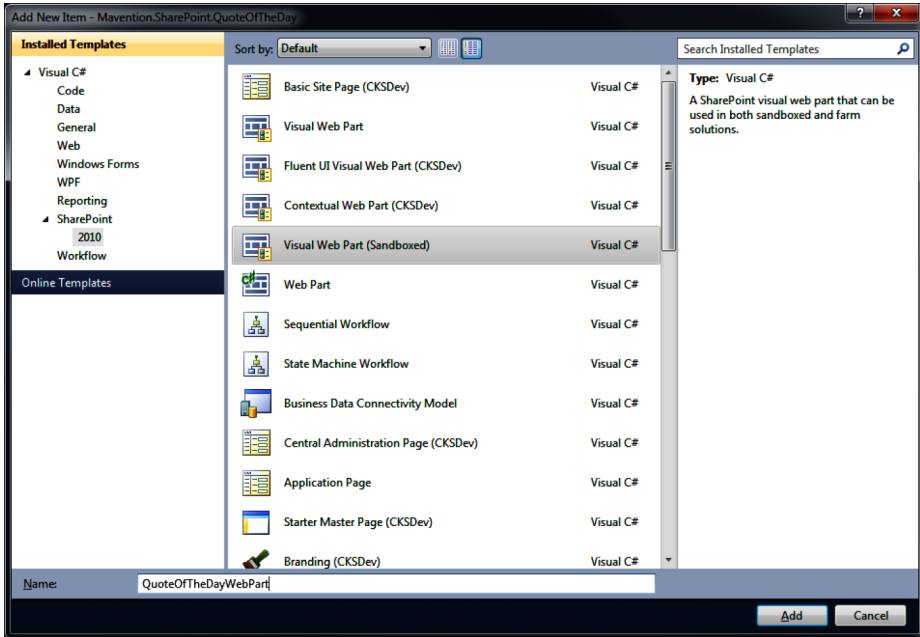


FIGURE 6-21. Adding new web part to our project

- Next, edit the web part definition file to make the description more user-friendly. In Solution Explorer, open the *QuoteOfTheDayWebPart.webpart* file and enter the following code snippet:

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
<webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
<metadata>
<type
name="Mavention.SharePoint.QuoteOfTheDay.QuoteOfTheDayWebPart.
QuoteOfTheDayWebPart, $SharePoint.Project.AssemblyFullName$" />
<importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>
</metadata>
<data>
<properties>
<property name="Title" type="string">Quote of the Day Web Part</property>
<property name="Description"
type="string">Displays quote of the day</property>
</properties>
</data>
```



```
</webPart>
</webParts>
```

All of our web part's logic will be executed in the browser. For this we need a JavaScript file that will contain all the necessary scripts. To deploy a JavaScript to a SharePoint site, we will need a module. Another advantage of using a module is that we can move CSS styles of the Quote of the Day web part to a separate file and have it automatically provisioned.

3. To add a module, right-click the QuoteOfTheDayWebPart SPI and, in the context menu, choose Add→New Item. From the list of available templates, choose Module and call it **QuoteOfTheDayWebPartAssets** (as shown in Figure 6-22). Confirm your choice by clicking the Add button.

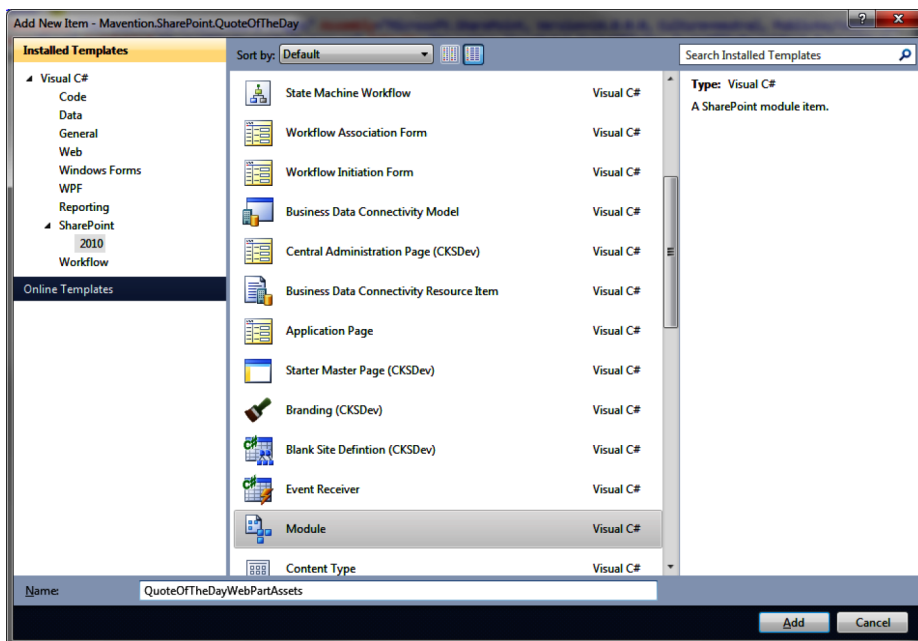


FIGURE 6-22. Adding the module for deploying the Quote of the Day web part's assets

4. In the QuoteOfTheDayWebPartAssets module, change the name of the *Sample.txt* file to *mavention.quoteoftheday.js*. Next, right-click the QuoteOfTheDayWebPartAssets SPI and, in the context menu, choose Add → New Item. From the list of available Project Item Templates, choose Style Sheet and call it **mavention.quoteoftheday.css**, as shown in Figure 6-23.

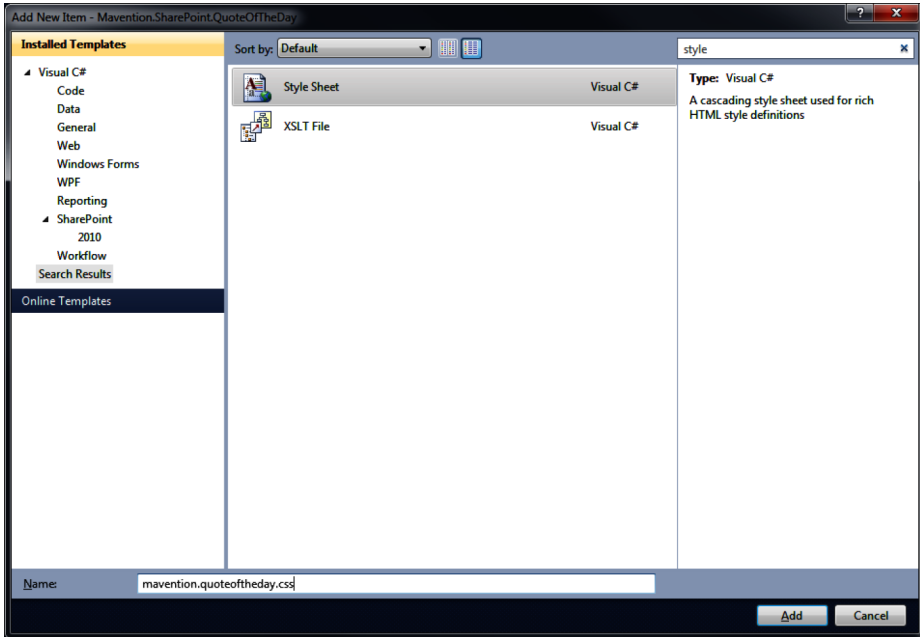


FIGURE 6-23. Adding a CSS styles file to the QuoteOfTheDayWebPartAssets module

- Next, we have to ensure our assets will be provisioned correctly. In the *Elements.xml* file, enter the following code snippet:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
<Module Name="QuoteOfTheDayWebPartAssets" Url="style library/quoteoftheday">
<File Path="QuoteOfTheDayWebPartAssets\mavention.quoteoftheday.js"
Url="mavention.quoteoftheday.js" />
<File Path="QuoteOfTheDayWebPartAssets\mavention.quoteoftheday.css"
Url="mavention.quoteoftheday.css" />
</Module>
</Elements>
```

If you were to deploy the project at this stage, you would see the *mavention.quoteoftheday.js* and *mavention.quoteoftheday.css* files deployed to the Style Library in your SharePoint site.

Building the Quote of the Day Web Part

We now have everything in place to start adding functionality to our Quote of the Day web part.

1. We start by adding some properties to the Quote of the Day web part that will allow users to configure the web part. In Solution Explorer, open the *QuoteOfTheDayWebPart.ascx.cs* file and enter the following code snippet:

```
using System;
using System.ComponentModel;
using System.Web.UI.WebControls.WebParts;

namespace Mavention.SharePoint.QuoteOfTheDay.QuoteOfTheDayWebPart {
    [ToolboxItem(false)]
    public partial class QuoteOfTheDayWebPart :
        System.Web.UI.WebControls.WebParts.WebPart {
        [WebDisplayName("Site URL"), ❶]
        WebDescription(@"Server-relative URL of the site where the Quotes List
        is located"), Category("Quote of the Day Configuration"),
        Personalizable(PersonalizationScope.Shared), WebBrowsable(true)]
        public string WebUrl { get; set; }

        [WebDisplayName("List Name"),
        WebDescription("Name of the Quotes List that you want to use"),
        Category("Quote of the Day Configuration"),
        Personalizable(PersonalizationScope.Shared), WebBrowsable(true)]
        public string ListName { get; set; }

        public bool Configured {
            get {
                return !String.IsNullOrEmpty(WebUrl) &&
                    !String.IsNullOrEmpty(ListName);
            }
        }

        protected override void OnInit(EventArgs e) {
            base.OnInit(e);
            InitializeControl();
        }

        protected void Page_Load(object sender, EventArgs e) {
        }
    }
}
```

- ❶ Start by defining two properties called **WebUrl** and **ListName** that allow users to select which Quotes list the Quote of the Day web part should load the quote from.

Additionally, we define the `Configured` property, which allows us to determine if the web part has been configured to prevent invalid calls.

2. The next step is to take care of the UI part of our Quote of the Day web part and reference all of our assets. In Solution Explorer, open the `QuoteOfTheDayWebPart.ascx` file and enter the following code snippet:

```
1 <%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>
2 <%@ AssemblyName="Microsoft.Web.CommandUI, »
3 Version=14.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
4 <%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
5 Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, »
6 PublicKeyToken=71e9bce111e9429c" %>
7 <%@ Register Tagprefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
8 Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, »
9 PublicKeyToken=71e9bce111e9429c" %>
10 <%@ Import Namespace="Microsoft.SharePoint" %>
11 <%@ Register Tagprefix="WebPartPages" Namespace="Microsoft.SharePoint.
    WebPartPages"
12 Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, »
13 PublicKeyToken=71e9bce111e9429c" %>
14 <%@ Control Language="C#" AutoEventWireup="true"
15 CodeBehind="QuoteOfTheDayWebPart.ascx.cs" ❶
16 Inherits="Mavention.SharePoint.QuoteOfTheDay.QuoteOfTheDayWebPart.»
17 QuoteOfTheDayWebPart" %>
18 <% if (Configured) { %>
19 <div class="quoteOfTheDay" id="<%= ClientID %>_body"></div>
20 <script type="text/javascript">
21     Type.registerNamespace('Mavention.QuoteOfTheDay'); ❷
22     if (!Mavention.QuoteOfTheDay.assetsLoaded) {
23         Mavention.QuoteOfTheDay.assetsLoaded = true;
24         var serverRelativeUrl =
25 ' <%= SPContext.Current.Site.ServerRelativeUrl.TrimEnd('/') %>/';
26         SP.SOD.registerSod('mavention.quoteoftheday.js',
27             serverRelativeUrl +
28 'style library/quoteoftheday/mavention.quoteoftheday.js');
29
30         // load CSS
31         if (document.getElementsByTagName) {
32             var link = document.createElement('link');
33             link.setAttribute('rel', 'stylesheet');
34             link.setAttribute('type', 'text/css');
35             link.setAttribute('href', serverRelativeUrl +
```

```

36  'style library/quoteoftheday/mavention.quoteoftheday.css');
37      var head = document.getElementsByTagName('head')[0];
38      head.appendChild(link); ❸
39  }
40  }
41
42      SP.SOD.execute('mavention.quoteoftheday.js',
43  'Mavention.QuoteOfTheDay.displayQuote', '<%= WebUrl %>', '<%= ListName %>',
44  document.getElementById('<%= ClientID %>_body'));
45  </script>
46  <% } %>

```

- ❶ First, we define the container that will contain the Quote of the Day after it has been loaded. Then we proceed with loading assets for the Quote of the Day web part. We do that keeping in mind the requirement of supporting multiple Quote of the Day web parts on the same page and loading all assets only once.
 - ❷ Unfortunately, SharePoint 2010 JSOM doesn't currently provide us with the server-relative URL of the Site Collection, which we need to properly load all assets. While we could do this using only JavaScript, it would add some complexity to our code. Instead, we can benefit from the fact that we are working with a Visual web part and retrieve the server-relative URL of the Site Collection directly using the server API.
 - ❸ Having registered JavaScript and CSS, all that is left is to call the `displayQuote` function from our external JavaScript file. By using the `SP.SOD.execute` function, SharePoint will automatically check if the JavaScript asset file has been loaded and will load it if necessary.
3. The next step is to edit the contents of our JavaScript file responsible for loading the quotes from the given Quotes list. In Solution Explorer, open the `mavention.quoteoftheday.js` file and enter the following code snippet:

```

Date.prototype.getFullDate = function () {
    var dateAsString = '';
    dateAsString += this.getFullYear();
    var month = this.getMonth() + 1;
    if (month < 10) {
        dateAsString += 0;
    }

    dateAsString += month;

    var day = this.getDate();
    if (day < 10) {

```

```

        dateAsString += 0;
    }

    dateAsString += day;

    return parseInt(dateAsString);
}

Type.registerNamespace('Mavention.QuoteOfTheDay');

Mavention.QuoteOfTheDay = function (webUrl, listName, container) {
    this.webUrl = webUrl;
    this.listName = listName;
    this.container = container;
}

Mavention.QuoteOfTheDay.prototype.init = function () {
    this.container.innerHTML = '<div class="message">\
\
Loading Quote of the Day...</div>';

    SP.SOD.executeOrDelayUntilScriptLoaded(Function.createDelegate(this,
function () {
    this.context = SP.ClientContext.get_current();
    this.web = this.context.get_site().openWeb(this.webUrl);
    this.list = this.web.get_lists().getByTitle(this.listName);
    this.query = new SP.CamlQuery();
    this.query.set_viewXml('<View><ViewFields><FieldRef Name="Title"/>\
<FieldRef Name="Quote"/></ViewFields></View>');
    this.items = this.list.getItems(this.query);
    this.context.load(this.items);
    this.context.executeQueryAsync(Function.createDelegate(this,
function () {
    this.numberOfQuotes = this.items.get_count();
    if (this.numberOfQuotes > 0) {
    this.now = new Date();
    this.quoteNumber = this.now.getFullDate() % this.numberOfQuotes;
    this.quoteItem = this.items.get_item(this.quoteNumber);
    this.quote = this.quoteItem.get_item('Quote');
    this.author = this.quoteItem.get_item('Title');

```

```

        this.container.innerHTML = '<blockquote><p>' + this.quote +
'</p><p class="author">' + this.author + '</p></blockquote>';
    }
    else {
        this.container.innerHTML = '<div class="message">\
There are no quotes in the list yet</div>'
    }
    }), Function.createDelegate(this, function () {
        this.container.innerHTML = '<div class="message">\
\
An error has occurred while loading Quote of the Day</div>';
    }));
    }}, 'sp.js');
}

Mavention.QuoteOfTheDay.displayQuote =
function (webUrl, listName, container) {
    var quoteOfTheDay =
new Mavention.QuoteOfTheDay(webUrl, listName, container);
    quoteOfTheDay.init();
}

```

```
SP.SOD.notifyScriptLoadedAndExecuteWaitingJobs('mavention.quoteoftheday.js');
```

Most of this code snippet should be familiar to you, as we used it in the first part of this chapter. The only difference is that we have moved the code out of the static `displayQuote` function to the instance `init` function. This allows us to avoid request conflicts when having multiple instances of the Quote of the Day web part on the same page. Another change is that at the end of the file, we call the `notifyScriptLoadedAndExecuteWaitingJobs` function, which allows us to load our script on demand and execute all calls to the `displayQuote` function.

One last thing left for us to do is to take care of the presentation of quotes. For this we will reuse the CSS styles we used in the first part of this chapter.

4. In Solution Explorer, open the *mavention.quoteoftheday.css* file and enter the following code snippet:

```

.message { text-align: center; }
.message img { vertical-align: middle; }
blockquote p { font: italic bold 1.6em/1.2 "Times New Roman" , serif; }
blockquote p.author {
font: normal 1em/1.2 sans-serif;
color: #666;
}

```

```
text-align: right;
}
```

5. Save all your changes and, in the Debug menu, choose the Start Debugging option or press F5. On your page, add an instance of the Quote of the Day web part. Edit the web part's Properties and in the Tool pane, navigate to the Quote of the Day Configuration section. Enter / for the Site URL and enter **Famous Quotes** for the List Name (this is the name of the Quotes list we created previously). Confirm your changes by clicking the OK button.

[Figure 6-24](#) shows the configuration settings of the Quote of the Day web part.

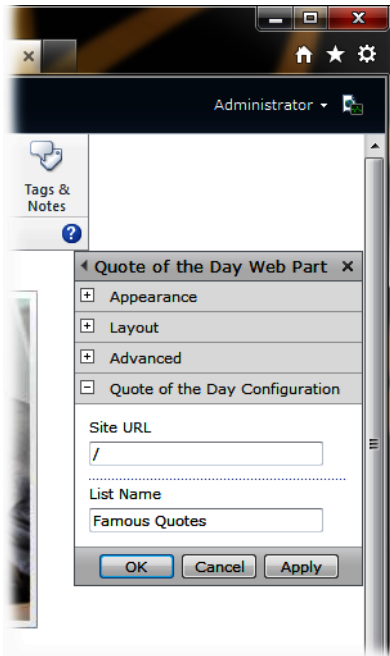


FIGURE 6-24. Configuring the Quote of the Day web part

If you have done everything correctly, you should see the Quote of the Day web part showing a quote from the Famous Quotes list, as shown in [Figure 6-25](#).

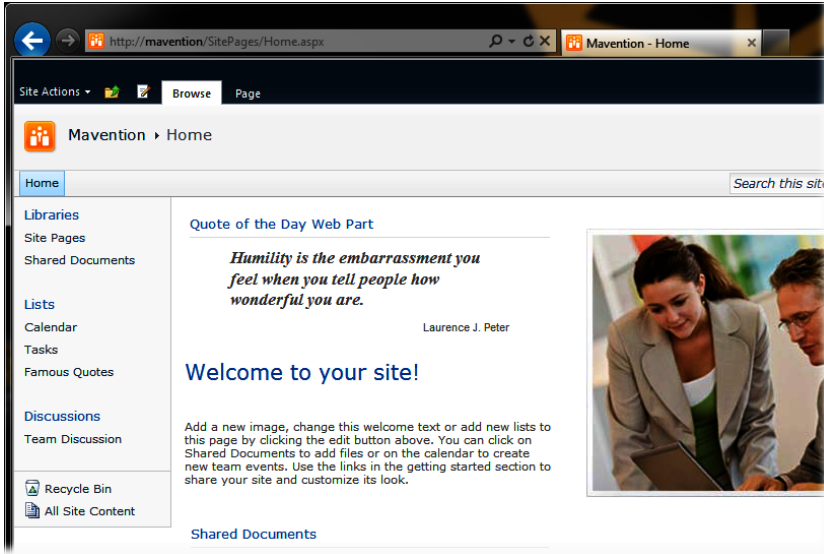


FIGURE 6-25. Quote of the Day web part showing a quote from the Famous Quotes list

- To verify that we have properly implemented support for multiple Quote of the Day web parts on the same page, let's create another quotes list, called SharePoint Quotes, and fill it with some quotes, as shown in Figure 6-26.

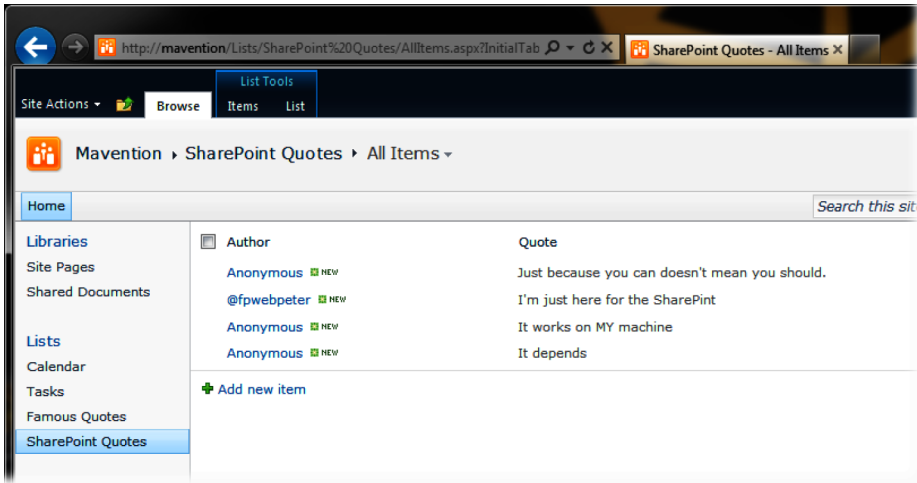


FIGURE 6-26. SharePoint quotes list filled with some SharePoint quotes

7. Next, add another Quote of the Day web part to the same page and configure it to load quotes from the SharePoint Quotes list we just created. [Figure 6-27](#) shows the configuration settings for this web part.

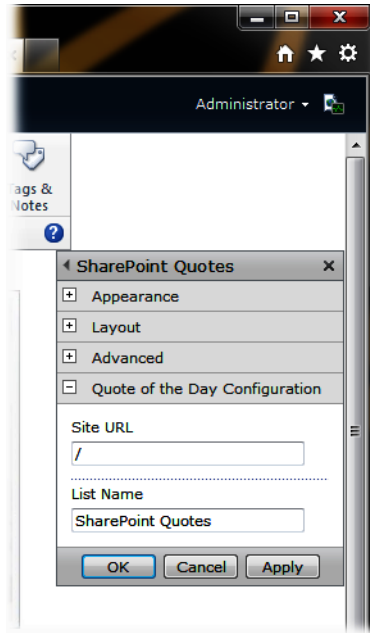


FIGURE 6-27. Configuring the Quote of the Day web part to load quotes from the SharePoint Quotes list

You should now see two Quote of the Day web parts showing quotes from different Quotes lists (see [Figure 6-28](#)).

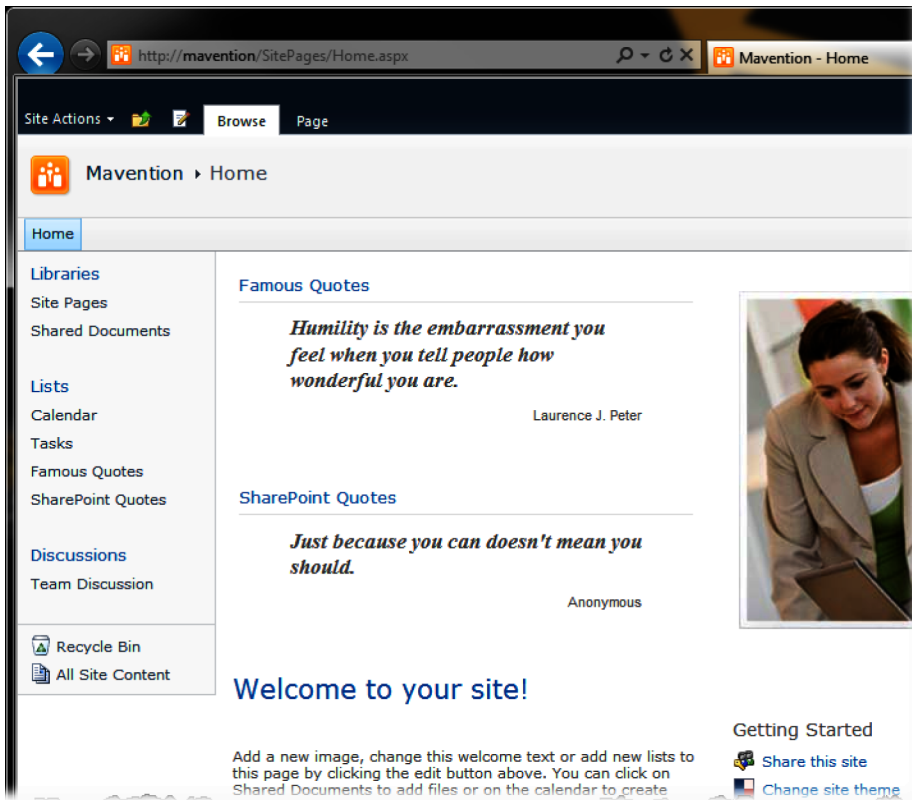


FIGURE 6-28. Two Quote of the Day web parts showing different quotes

Finally, we need to verify that we have fulfilled the requirement of loading Quote of the Day Web Part's assets only once, even when having multiple instances of that web part added on the page. There are a number of ways to verify this. Here I'm using the Firebug for Firefox add-on to examine all requests made from the current page. As you can see in Figure 6-29, although we have two instances of the Quote of the Day Web Part on the page, the *mavention.quoteoftheday.js* and *mavention.quoteoftheday.css* files are loaded only once.

Summary

In this chapter, we created a Quote of the Day web part that displays a quote for the day based on a list of quotes. We started with a simple approach by adding some JavaScript using the CEWP. In the second part of this chapter, we explored a slightly more advanced scenario and leveraged the Sandboxed Solutions capability to turn our Quote of the Day web part into a redistributable solution.

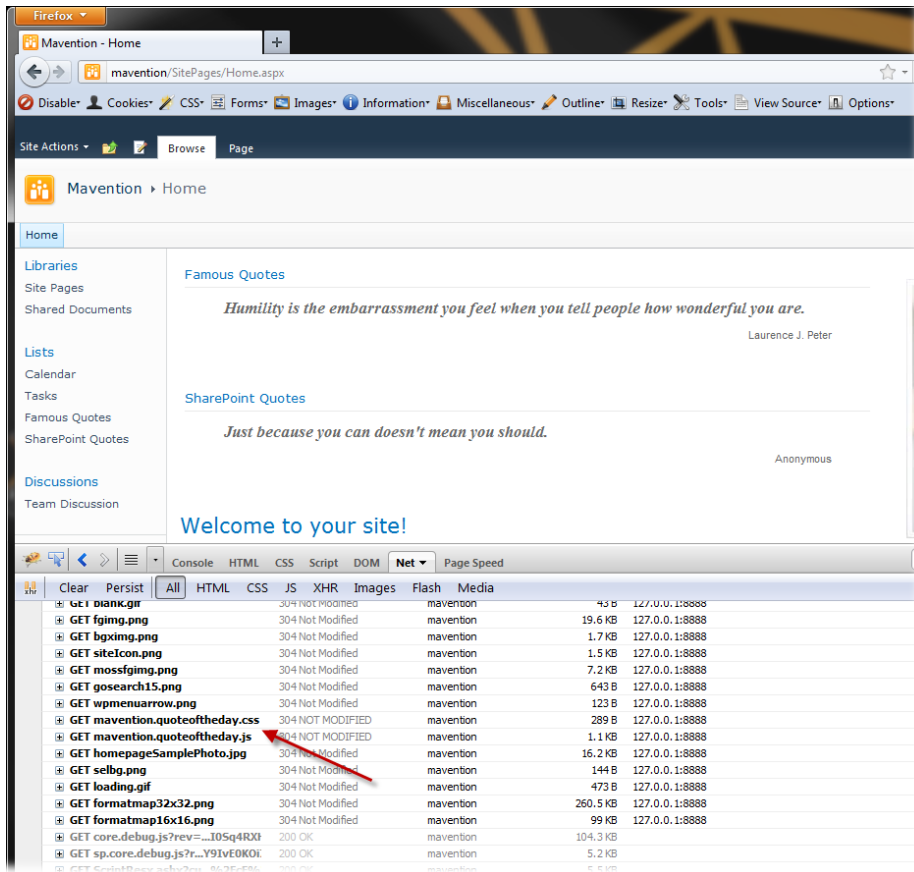


FIGURE 6-29. Quote of the Day web part's assets loaded only once for multiple instances

While working on the Quote of the Day web part, we used the new SharePoint 2010 JSOM, which allowed us to dynamically load the data in an efficient and interactive way. SharePoint 2010 JSOM allows you to not only interact with data stored in SharePoint, but also to load scripts whenever they are needed, which allows you to provide your users with the best experience possible.

SPJS Charts for SharePoint

Alexander Bautz

This solution is a bridge between Google's Visualization API and SharePoint. This enables you to get data from any SharePoint list or library within the site collection, and create dynamic charts with Google Visualization API.

The solution is used to get the data from the lists and present them to the Google Visualization API in the correct format. The charts are created with the Google Visualization API and therefore all chart-related issues must be checked with the documentation here: <http://code.google.com/intl/en-EN/apis/chart/>. You will find a link to the relevant configuration options for the selected chart from within the Edit Chart GUI.

This chapter will walk you through the setup of the solution and explain the various settings available in the Edit Chart GUI. It will not go into details about how you can create different charts. You will, however, find chart examples in my private blog, <http://sharepointjavascript.wordpress.com>.

The solution works in both SharePoint 2007 and 2010.

Technical Overview

The technique presented here is entirely client-side and all code and data are processed and rendered in the browser. No data is sent to any server.

The end user does not need to have more than design rights to the site to set it up for the first time. Creating charts after the initial setup can be done with contribute rights.

I have wrapped all code in one file to ease the setup process for the end user. The script-generated HTML that makes up the Edit Chart GUI accounts for the majority of the code lines.

The solution automatically creates the list necessary to store the configuration for the charts. We will use various web service calls to *lists.aspx*, *views.aspx*, *webs.aspx*, *sitedata.aspx*, and *usergroup.aspx* to gather the information required to configure the charts in the Edit Chart GUI and to render the charts when they are configured.

The inner workings of the code aren't discussed in detail here, but the code is commented, so anyone interested can understand it fairly easily.

NOTE

As this solution is client-side, the rendering of charts from large datasets (thousands of lines) can take its toll when it comes to load time and background data traffic. When querying the web service for data, the full XML blob will be downloaded to the client machine. When this is complete, the scripts will iterate through the XML and format it as required for use in the Google Visualization API. The line speed and the processing speed of the end user's computer must be taken into account.

The actual charts are made using the Google Visualization API, and this solution adds no extra functionality over that provided by Google. This solution may, however, need updates to support new features in upcoming versions of the Google Visualization API. The Google Visualization API team thoroughly tests each new version to ensure backwards compatibility, and I will do my best to keep this solution updated to support new features and fix potentially breaking changes.

SPJS Charts for SharePoint is under continuous development, and anyone can provide feature requests or bug reports. The solution is posted on [NothingButSharePoint](#) and in my personal blog, [SharePointJavaScript](#).

The SharePoint integration is licensed by Alexander Bautz under the MIT X11 License. The Google Visualization API Terms of Service can be found at <http://code.google.com/intl/en-EN/apis/visualization/terms.html>.

The Google Visualization API is loaded dynamically from Google. Under the license agreement, you cannot download the API locally.

Version History

I published the first version of my solution for using Google visualization API to create charts for SharePoint on May 4, 2010. Since then, I have updated the solution many times, but have never given the solution a distinct name. I figured in the event of writing this chapter, I needed to give the solution a name.

I have named it SPJS Charts for SharePoint.

Initial Setup

This solution is entirely client-side. All you need to get started are the rights to add lists (initial setup only), upload documents, and add a CEWP (Content Editor web part).

First, download the following files from <http://spjsfiles.com/> in the folder named *SPJS_ChartsForSharepoint*:

- *SPJS_ChartsForSharePoint_vX.X.js*, where X.X represents the version number
- *CEWP.txt*

Upload the *SPJS_ChartsForSharePoint_vX.X.js* file to a shared document library and ensure all users have read access. If you have access to SharePoint Designer, you can put the file in a folder on the root of your site for safer storage.

Edit the code in the *CEWP.txt* file and change the source of the last script reference to reflect your local copy of the *SPJS_ChartsForSharePoint_vX.X.js* file.

NOTE

If you want to use a local copy of jQuery, go to <http://jquery.com/> and download v1.6.4 of jQuery. It is important that you do not use v1.7, as there are some incompatibility issues between *SPJS_ChartsForSharePoint_v3.0.js* and jQuery v1.7.

Go to the page where you want the chart to appear and insert a CEWP. Use the Content link option to reference the code.

In SharePoint 2007, you can insert the code directly into the source editor, but I recommend using the Content link option. In SharePoint 2010, you must use the Content link option to prevent the HTML generated from the script from being appended to the CEWP when you re-edit the page. You could also use the HTML Form web part as this will not change the code in the same way as the CEWP. I would, however, recommend using the CEWP with the content link option due to the possibility to refer the same code in multiple web parts, thus making future updates more manageable.

To use the Content link option, put the CEWP code in a text file (use Notepad or a similar application) and upload it to the same location where you put the *SPJS_ChartsForSharePoint_v3.0.js* file. Copy the file URL by right-clicking the file and selecting “Copy shortcut” in the menu. Insert this URL in the Content link field of the CEWP.

NOTE

If your file is in a shared document library, right-click the file and select “Copy shortcut” in the menu. If your file is in a folder created in SharePoint Designer, right-click the file and select Properties. Copy the URL from Location.

You might want to make it a relative URL like the one shown in the example code in the file *CEWP.txt*.

Creating the Configuration List

When you access a page with this CEWP code for the first time, you are prompted to create the configuration list (see [Figure 7-1](#)).

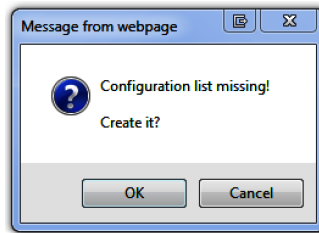


FIGURE 7-1. Configuration list prompt

Click OK to create the list in the current site. You will get a confirmation message (see [Figure 7-2](#)).

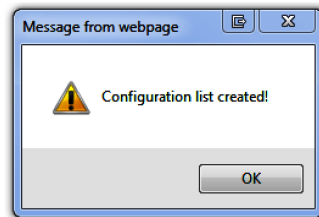


FIGURE 7-2. Confirmation message

This custom list is used by the solution to hold the configuration for all the charts created in this site. The list is named “SPJS Charts for SharePoint.” To configure this list, click the Edit Chart button in the top left corner of the chart container (the small arrow shown in [Figure 7-3](#)).

Click this little down-arrow to access the chart configuration.

FIGURE 7-3. Edit Chart button

If the chart is not configured, the little down arrow is the only thing visible.

The Edit Chart GUI

This section provides a description of the various configuration options in the GUI.

Figure 7-4 shows an example combo chart.

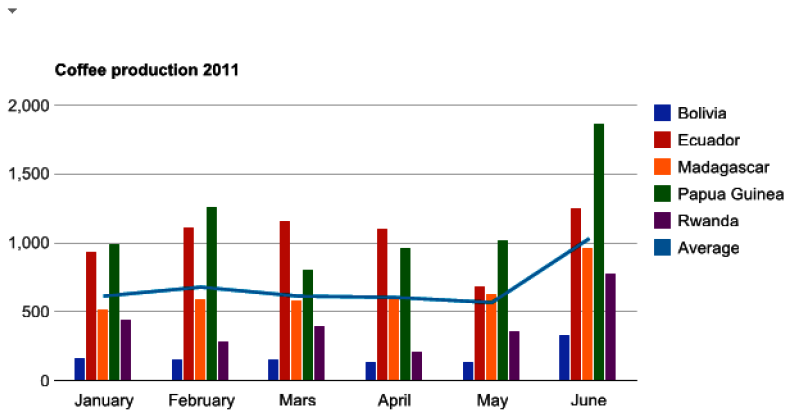


FIGURE 7-4. Example of a combo chart

Figure 7-5 shows the configuration for a combo chart.

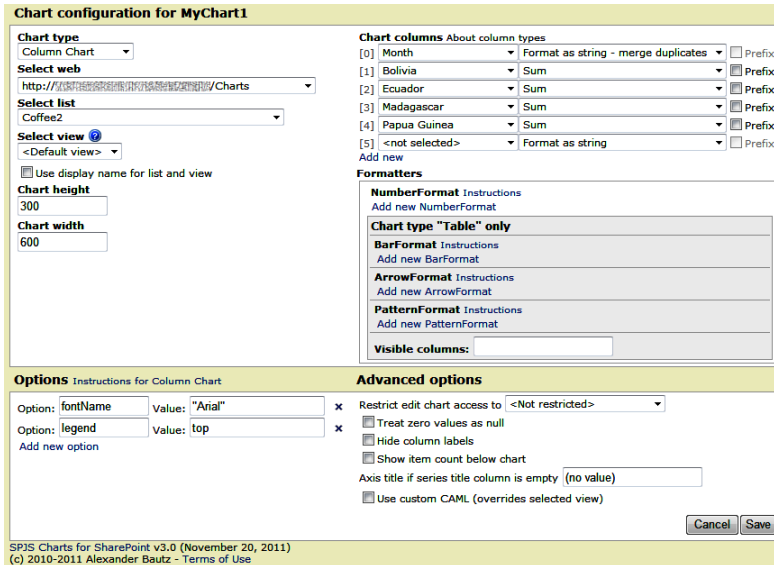


FIGURE 7-5. Configuration of the combo chart

Chart type

You can select among the following chart types:

- Area chart
- Bar chart
- Candlestick chart
- Combo chart
- Gauge
- Geo map
- Line chart
- Map
- Motion chart
- Org chart
- Pie chart
- Scatter chart
- Table

NOTE

The configuration options for each chart type differ. You'll find a link to the configuration options for the selected chart type to the right of the Options heading.

Select web

This control is rendered differently for site collection administrators (SCAs) and nonSCAs. For SCAs, the control is rendered as a drop-down menu that allows you to select among all the sites in the current site collection. For nonSCAs, the control is rendered as a text box where the user enters the URL and loads the lists with the click of a button. This is done because the method used to retrieve all the sites is available for SCAs only.

Select list

Once you have selected the web, you can select among all the lists and libraries in the site. After you select the list or library, the view selector and chart column selectors are populated.

Select view

The view defines the data source for your chart. The view selector is overridden if you enable the Use custom CAML option.

NOTE

Standard List views have paging turned on, most likely with a page size of 100 elements. This means that your chart will use the first 100 elements only. You must therefore either change the item limit of the view or use a custom CAML.

Use display name for list and view

This option is provided to support adding charts to a site template and to create additional subsites. What this option does is to replace the list and view GUID with the display name. This has to do with the fact that all lists will get a new GUID when the site is created from a site template.

NOTE

This option is only for those planning to use the site as a template and may not be appropriate in multilanguage setups where the list's display name has been translated.

Chart height

This sets the height of the chart container (in pixels).

Chart width

This sets the width of the chart container (in pixels).

Chart columns

These are the list columns that make up the chart. Different chart types require different setups. For more information about the chart type you have selected, click the link to the right of the Options heading.

The first drop-down list allows you to select the field. Use the second drop-down list to select the format.

The following formats are available:

Average

The total value of all the lines in this column are summed and then divided by the number of cells that are *not empty* in the column.

Count

This will count the number of cells that are *not empty* in the column.

Sum

This will add up all the values in that column.

Counting unique—including blank

Make a chart from a selected column in a list and show each option as a separate series. This option will make a series with the blank values as well.

Counting unique—excluding blank

This option will *not* make a series with the blank values.

Format as string

Outputs the value as a string.

Format as string—merge duplicates

Outputs the value as a string, but will merge duplicates.

Format as Boolean (true/false)

Outputs Boolean values.

Format as number

Outputs a floating point number.

*Format as % (value*100)*

Used in conjunction with, for example, the “Task % Complete” column.

Format as date

Outputs a JavaScript date object.

Any column containing numbers can be summed or averaged. The column can contain text, and numbers mixed as a regular expression will extract the number from the text. Only the first occurrence of an integer/float will be used.

NOTE

You cannot include columns of type **Sum**, **Count**, **Average**, or **Format as string**—merge duplicates with columns of type **String**, **Number**, **Boolean**, or **Date** in the same chart.

The last checkbox sets the prefix “Sum of,” “Count of,” or “Average of” before the label in the legend.

Formatters

The formatters are used to format the data as it is displayed in the chart. For regular charts, only the **NumberFormat** option is available.

One example of number formatting could be a value that should have the suffix “\$” (shown in [Figure 7-6](#)).

The screenshot shows the 'Chart columns' configuration interface. It features a table with three rows of columns and a 'Formatters' section below it.

	Column Name	Format	Prefix
[0]	Title	Format as string - merge duplicates	<input type="checkbox"/>
[1]	Num	Average	<input checked="" type="checkbox"/>
[2]	<not selected>	Format as string	<input type="checkbox"/>

[Add new](#)

Formatters

NumberFormat [Instructions](#)

Column index: Option: Value:

[Add new NumberFormat](#)

FIGURE 7-6. Example of number formatting

Click the **Instructions** link to the right of the **NumberFormat** heading for details.

NOTE

The additional formatters **BarFormat**, **ArrowFormat**, and **PatternFormat** are used in the **Table** chart only. You’ll find more information by clicking the **Instructions** link.

Visible columns

This setting is used in conjunction with **Table** chart and **PatternFormat** to select which columns to show.

Options

Here you can add all the configuration options available for the selected chart type. The configuration options can be found at Google’s site by following the link to the right of the **Options** heading.

NOTE

There is a bug in Internet Explorer regarding the default font name. This results in a font that is not very readable. To overcome this bug, insert an option:

Option: `fontName`, Value: "Arial"

Advanced options

Restrict edit access to

This option is used to restrict edit rights. When you select a SharePoint group, the script will check to see if the user is member of this group. If not, the Edit Chart button will be omitted. This has nothing to do with actual permissions, it's only a matter of whether the user is a member of this group.

NOTE

This setting is not available if your site is using a custom master page where the built-in variable `_spUserId` is missing.

Treat zero values as null

If you have holes in the source data for a line chart, enabling this option will prevent the line from dipping down to 0. This will result in a break in the line, but if you combine this setting with the `interpolateNulls` option set to true, you will have a continuous line that "guesses" the value for the missing data point.

Hide column label

Use this to hide the column label from the legend and the hover text.

Show item count below chart

Shows or hides the item count of the source data below the chart. If enabled, the prefix and/or suffix can be specified in the fields that are expanded.

Axis title if series column is empty

If the series column has no value, use this value as column label.

Use custom CAML (overrides selected view)

Specify a custom CAML instead of relying on the selected view.

[Figure 7-7](#) shows the available options for using custom CAML.

Use custom CAML (overrides selected view)

Filters:

Get filter value from this profile property: ID

Create a filter field above the chart using: <manual filter setup>

URL filter:
Insert {url:URL query string parameter} in the CAML where you want the URL query string variable to be inserted.

Show URL filter info below chart

Consume filter from list view web part

Custom CAML-query

Build CAML from selected filters | Get CAML from selected view

FIGURE 7-7. Custom CAML options

If you need help writing CAML, use a “dummy” view to harvest the CAML from. Set up the view the way you want, then go back to this tool and pick the view in the view selector. Click Get CAML from selected view (below the Custom CAML-query text area).

If you use one or more filters, you can also click Build CAML from selected filters. You might need to edit the CAML to insert the correct FieldInternalName.

NOTE

To find the FieldInternalName of a field, go to *DispForm.aspx*, right-click, and select View Source. Search for your field’s DisplayName and you will find the fields FieldInternalName like this:

FieldInternalName=“This is the FieldInternalName”

Get filter value from this profile property

This will look up the profile property of the current user and use it in the CAML (assuming you insert {0} in the CAML as a placeholder). This is the property from the built-in user list in SharePoint and not the Microsoft Office SharePoint Server (MOSS)/SharePoint Server Shared Service Provider profile.

NOTE

This setting is not available if your site is using a custom master page where the built-in variable `_spUserId` is missing.

Create a filter field above the chart using

This option lets you select among the columns in your list to make a drop-down filter above the chart. You must insert **{1}** in the CAML as a placeholder for the value.

If the column you want to filter by is of type Single line of text, you can manually create the filter by selecting manual filter setup, and hardcoding the options as described here:

Enter a label for the filter in the Filter label field. Enter the choices separated by a pipe character like this:

```
Choice 1|Choice2|Choice3
```

You can have a different display name and internal name, like this:

```
1^Choice 1|2^Choice2|3^Choice3
```

In this example, the valid choices are 1, 2, and 3, but the selections shown in the drop-down list are Choice 1, Choice2 and Choice3.

To preselect an option other than the first, append “#” to the option, like this:

```
Choice 1|Choice2#|Choice3
```

Show URL filter info below chart

If you are using a URL filter, either manually by inserting **{url:keyword}** in the CAML, or by using the option to Consume filter from List View web part, this setting will display the applied filter below the chart.

Consume filter from List View web part

To use this option, supply an initial custom CAML and place the chart in a list view with a single List View web part. This will connect the chart to a List View web part so that any filter applied to it will be picked up from the URL and used as a filter for the chart(s).

If you put more than one List View web parts in the page, the chart cannot determine which list the filter is applied to, and will therefore try to consume filters from multiple List Views when filtered.

NOTE

Filtering by columns of type DateTime is not supported.

Build CAML from selected filters

This option (located below Custom CAML-query) lets you build the CAML from the selected filters. If you have set up a filter using manual filter setup, you must edit the CAML and replace the text FieldInternalNameToMatch with the correct FieldInternalName.

Get CAML from selected view

This option (located below Custom CAML-query) lets you harvest the CAML from the selected view. You can then edit the CAML by hand.

How to Make Web Part Templates

To make adding charts more accessible to the end users, you might want to create web part templates and add them to the web part gallery.

WARNING

You must be a site collection administrator to make web part templates.

NOTE

If you followed my recommendation on using the Content link option when adding the CEWP code, you can maintain the code for all charts in a site or site collection by editing one file. This might come in handy if you want to update all charts when a new version of this tool is released.

1. Add a CEWP to a page and open the web part properties. Link to your CEWP code and give it a name, for example, "SPJS Charts for SharePoint" (see [Figure 7-8](#)).

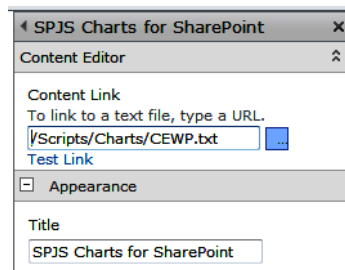


FIGURE 7-8. Adding a web part template

2. Save the changes and open the page again in Edit mode. From the drop-down menu on the CEWP, select Export (see [Figure 7-9](#)).

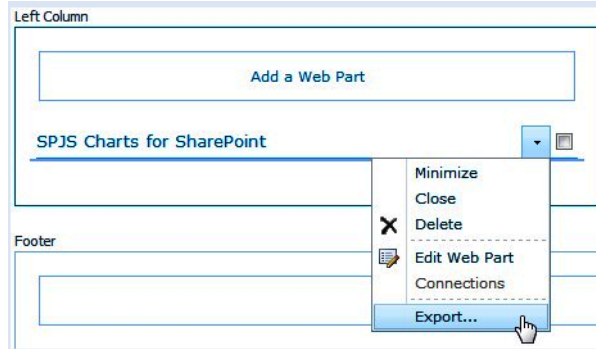


FIGURE 7-9. Exporting a web part template

3. Save this file locally and navigate to the web part gallery in the root of your site collection and upload the DWP file.
4. Add a description and a group name in the Upload File dialog and save the form.

The new web part is now available in all sites in this site collection, and end users can insert charts through the common Add a Web Part interface (Figure 7-10).

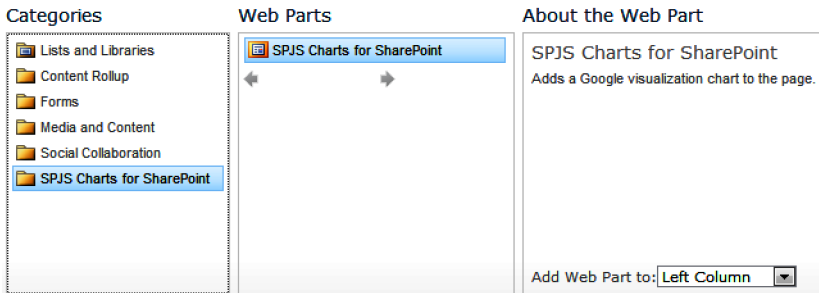


FIGURE 7-10. A new web part template added

Multiple Charts in One Page

To insert multiple charts in one page, edit the CEWP code and insert more placeholders in the top of the code. The one in the example code is:

```
<div id="MyChart1"></div>.
```

If you want to have another chart below the first, add another container like this:

```
<div id="MyChart1"></div>
<div id="MyChart2"></div>
```

You also need to add these IDs to the array `arrOfChartContainers` in CEWP code like this:

```
var arrOfChartContainers = ["MyChart1", "MyChart2"];
```

NOTE

A placeholder can be any HTML element with an ID attribute, such as `<div>`, ``, or `<td>`.

You can put the placeholders in another CEWP on the page, but place the containers above the CEWP with the code to ensure they are loaded and available when the code runs.

Summary

This chapter provided a description of the various configuration options found in SPJS Charts for SharePoint v3.0. The solution is under continuous development, and additional features may be added in future versions.

If you have any questions or want some chart examples, add a comment under the article on this solution at [NothingButSharePoint](#) or on my personal blog, [SharePointJavaScript](#).

Taming the Elusive Calculated Column— Logic Functions

Dessie Lunsford

One of the more common frustrations for users new to working with formulas in SharePoint is the usage of *logical functions*. Throughout this chapter, my hope is that I can help alleviate some of the confusion in how they work, how to use them, and where they can be of benefit in your lists and libraries by working through a few examples that detail the “logic” behind the formula. Each function has a purpose, and although the logic involved can become confusing at times with complexities needed to build robust solutions, I’m going to break each one down with an explanation of how they all fit together, and why.

The Functions

The *logical* functions include the following individual functions:

IF

Returns different values depending on whether a condition specified evaluates to TRUE or FALSE.

AND

Returns a value of TRUE if all arguments in a statement are TRUE. If one or more arguments are FALSE, the entire statement is FALSE.

OR

Returns YES if any argument is TRUE and NO if all arguments are FALSE.

NOT

Reverses the value of its argument. Used in cases where you want to test whether a value is not equal to another value.

TRUE

Returns the logical value TRUE (used primarily for compatibility with other programs; you can also enter the value TRUE directly into your formulas when a requirement exists to test for the TRUE value).

FALSE

Same as TRUE, but returns FALSE.

In this chapter, I will be detailing the usage of IF, AND, and OR.

In several of my online articles, I've discussed the NOT, TRUE, and FALSE types quite often, but some of the more common questions popping up on forums and blogs daily (and sometimes hourly) are, "How do I write out the logic?" or "What's the syntax for nesting?" or "Why is it not giving me the answer I wanted?"

The thing to keep in mind when working with these functions is that logic operations are essentially true/false questions. The answer will always be in this form no matter how many *nestings* there are or cases where you'll be testing for more than two possible outcomes. In the end, it will always come down to the most basic of terms:

If the answer to my question is TRUE, do this; if the answer is FALSE, do that.

It seems rather simple when you look at it like this, but then why is it so troublesome to come up with formulas (sometimes for myself as well) to test for multiple conditions?

If the above is a correct summation (always either true or false), how do we account for checking for multiple (more than two) scenarios? Wouldn't this make it a multiple choice answer instead of a true/false answer? In human logic terms you may think so, but in computer logic (which is what our calculations are based upon), the answer is no.

Let's look at a few examples.

The IFs

"If I'm hungry, I'll eat."

Basic human survival, we need to eat to survive. Am I hungry: yes or no? If yes, I'll eat. If no, I won't eat.

This is the most basic of IF tests, since it deals with a single question that will have one of two possible answers:

If hungry = true, result = eat

If hungry = false, result = don't eat

SharePoint translation:

=IF([Hungry]="True","Eat", "Don't Eat")

We'll test this by creating a custom list with two added columns. The first column is named Hungry, the type of information is Choice, and choices are displayed using Radio Buttons (see [Figure 8-1](#)).

The screenshot shows the configuration interface for a new column in SharePoint. The column name is 'Hungry'. The type of information is 'Choice (menu to choose from)'. The choices are 'True' and 'False', listed in a text box. The display method is 'Radio Buttons'. The default value is set to 'Choice'. The 'Add to default view' checkbox is checked.

FIGURE 8-1. First column

The second column, shown in [Figure 8-2](#), is named Result, and the type is Calculated with the following formula:

=IF([Hungry]="True","Eat","Don't Eat")

Column name:

Calculated (calculation based on other columns)
 External Data
 Managed Metadata

Formula:

Insert Column:

[Add to formula](#)

The data type returned from this formula is:
 Single line of text

FIGURE 8-2. Second column

Once we have our columns, create a new item on the list, as shown in [Figure 8-3](#).

Title *	<input type="text" value="Test 1"/>
Hungry	<input checked="" type="radio"/> True <input type="radio"/> False

FIGURE 8-3. New list item

The results are shown in [Figure 8-4](#).

<input type="checkbox"/>	@ Title	Hungry	Result
	Test 1 NEW	True	Eat

[+ Add new item](#)

FIGURE 8-4. Results view

Since we chose True for the answer to whether or not we were hungry, the result displayed is “Eat.”

Now go back and edit the item, but choose False this time, as shown in [Figure 8-5](#). The result is now “Don’t Eat.”

☐ @ Title	Hungry	Result
Test 1 <small>NEW</small>	False	Don't Eat

[+ Add new item](#)

FIGURE 8-5. Changed to False

Now that you've seen a basic IF function in action, let's dissect it to see exactly what is happening in the formula (we're going to trace the logic from the beginning to the end).

Function format:

`=IF(logical_test,value_if_true,value_if_false)`

The IF statement itself has three basic parts:

1. The logical condition to check for (our question)
2. Result to return if TRUE
3. Result to return if FALSE

In our case, the three pieces are made up of the following:

1. Condition: does the value in the Hungry column equal TRUE?
2. TRUE: display the result "Eat".
3. FALSE: display the result "Don't Eat".

The entire conditional check, or question, can be viewed as the following:

If the value displayed in the Hungry column equals True, display the result "Eat." If the value displayed in the Hungry column does not equal True, display the result "Don't Eat."

Simple and straight to the point—Yes/No, True/False, "Eat/Don't Eat."

Let's take this one step further and add in a second condition to test for.

"Am I thirsty?"

As before, we look at what our two possible answers for this can be. Am I thirsty: yes or no? If yes, I'll drink. If no, I won't drink.

If thirsty = true, result = drink
 If thirsty = false, result = don't drink

SharePoint translation:

`=IF([Thirsty]="True","Drink","Don't Drink")`

Test this by adding another column to the list in SharePoint.

Create a column named Thirsty, set the type to Choice type, and select Radio Buttons for the display (Figure 8-6).

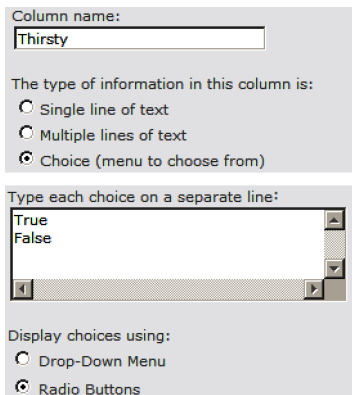


FIGURE 8-6. Creating a Thirsty column

Go back to the Result column and modify its formula to the following (Figure 8-7):

=IF([Thirsty]="True","Drink","Don'tDrink")

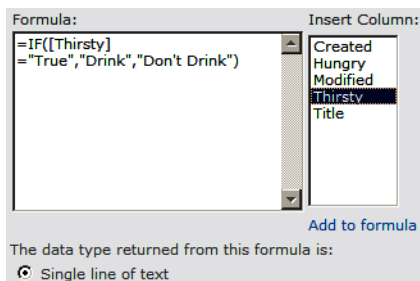


FIGURE 8-7. Modified formula for the Thirsty column

While you're still in the list settings, go ahead and change the order of the columns in your view (Figure 8-8) so that the Result column is last (just some cleanup as we go to keep things organized and ordered).

Display	Column Name	Position from Left
<input checked="" type="checkbox"/>	Title (linked to item with edit menu)	1
<input checked="" type="checkbox"/>	Hungry	2
<input checked="" type="checkbox"/>	Thirsty	3
<input checked="" type="checkbox"/>	Result	4

FIGURE 8-8. Reordering the columns

Back on our List View page, edit the existing item Title and select True for the Thirsty column value (Figure 8-9).

Title *	Test 1
Hungry	<input type="radio"/> True <input checked="" type="radio"/> False
Thirsty	<input checked="" type="radio"/> True <input type="radio"/> False

FIGURE 8-9. Editing the item

Since we've adjusted the formula in the Result calculated column to now test for the value of Thirsty, the three pieces of the IF statement are as follows:

1. Condition: does the value in the Thirsty column equal True?
2. True: display the result "Drink"
3. False: display the result "Don't Drink"

Our question can then be viewed as this: if the value displayed in the Thirsty column equals True, display the result "Drink." If the value displayed in the Thirsty column does not equal True, display the result "Don't Drink" (Figure 8-10).

Title	Hungry	Thirsty	Result
Test 1 <small>NEW</small>	False	True	Drink

[+ Add new item](#)

FIGURE 8-10. Verifying the answers

Looking at our list item and knowing the current state of the formula in the Result column, what we have makes sense in as far as the data being tested. But what about the Hungry column? How can we integrate that and test for its value at the same time we're testing Thirsty?

The approach we'll take is to *nest* the formulas together.

To nest a series of IF statements, let's look again at the three parts of a basic IF function:

1. The logical condition to check for (our question)
2. Result to return if true
3. Result to return if false

Here is the formula in its most basic form:

=IF(condition,True,False)

When we nest an additional IF into this, we're substituting a second instance of the exact same formula in place of either the True or False result in order to check a second condition.

If we want to test for a second condition based on an answer of True from our first condition, this becomes the following:

```
=IF(condition1,IF(condition2,True,False),False)
```

This *could* be read as the following:

If [condition1] equals True, then if [condition2] equals True, display the result specified in the [condition2] True value.

If [condition1] equals True, then if [condition2] does not equal True, display the result specified in the [condition2] False value.

If [condition1] does not equal True, display the result specified in the [condition1] False value.

So, even though we're only looking at two different conditions, we can actually have three different possibilities of answers.

But wait, how can this be possible if (as I stated earlier) a logical test can only have two possible answers (either true or false)?

Although it may appear that there are three different answers, we're still only using two, because the second condition is occurring inside the True answer for the first condition. When it comes time to display a result, the result of the logic that occurred within the nested formula will be sent up to the True value for the first condition and is displayed as its value (OK, that confused even me).

Let's take a look at the logic of how this is processed by using our test data of Hungry and Thirsty to illustrate why there are still only two possible answers.

Rewriting our formula from its basic form (adding in the real data), it becomes the following:

```
=IF(Hungry,IF(Thirsty,"Eat and Drink","Eat"),"Don't Eat")
```

Written out in a plain language for each step, this becomes:

Is it true that I am hungry?

If so, is it true that I am thirsty?

If so, I will eat and drink.

If not, I will eat.

If it is not true that I am hungry, I will not eat.

Enter the flowchart shown in [Figure 8-11](#).

Since the entire conditional check of "Am I thirsty?" takes place in the True value of "Am I hungry?" we're still within the boundaries of a single answer of True, so this fits in with the notion of a logical condition having only two possible answers (either True or False).

Regardless of which of the two answers from the second check we arrive at (True equaling "Eat and drink" or False equaling "Eat"), the answer is passed back to the original True value

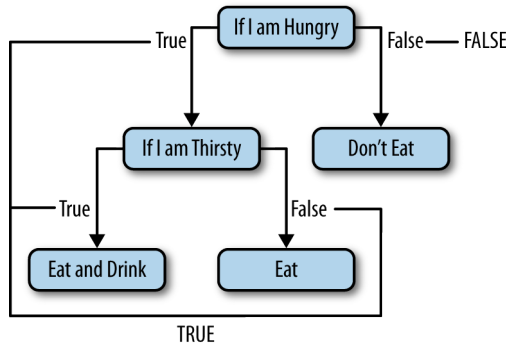


FIGURE 8-11. Basic flowchart

for the first conditional check, resulting in the True value for “Am I hungry?” displaying one of the results.

This is how nesting IF statements works. The first conditional check will display only a single result, whether it’s from itself or from a nested (child) conditional check, it can display only one result. The nested (child) conditional check passes its result back to its parent to be displayed, because the parent (first conditional check) will wait until it gets a value to satisfy its condition.

Understandably, this can be somewhat confusing, because the logic taking place is sometimes not intuitive, but if you look at the formula from left to right (the same manner in which it is processed) and work through the logic in a manner similar to what I’ve detailed (breaking it down into plain English), hopefully that will help to clear it up.

Let’s look at the formula once again and break down each step from left to right:

=IF(Hungry,IF(Thirsty,“Eat and drink”,“Eat”),“Don’t Eat”)

1. Starting with each condition equaling True:
 - a. Look at the formula in its basic form:

=IF(Hungry,True,False)
 - b. Look at the formula in its basic form with nesting:

=IF(Hungry,IF(Thirsty,True,False),False)
 - c. Expand the True path:

IF Hungry = True THEN IF(Thirsty,True,False)
 - d. Expand the True path of the nested condition:

IF Hungry = True THEN IF Thirsty = True, “Eat and drink”
 - e. Pass the values back:

IF Hungry = True THEN “Eat and drink”

- f. Pass the values back until you get to the beginning:
"Eat and drink"
 - g. Display the results:
Display = "Eat and drink"
2. Moving to the next set of answers (first condition equals True, second condition equals False):
- a. Look at the formula in its basic form:
=IF(Hungry,True,False)
 - b. Look at the formula in its basic form with nesting:
=IF(Hungry,IF(Thirsty,True,False),False)
 - c. Expand the True path:
IF Hungry = True THEN IF(Thirsty,True,False)
 - d. Expand the False path of the nested condition:
IF Hungry = True THEN IF Thirsty = False, "Eat"
 - e. Pass the values back:
IF Hungry = True THEN "Eat"
 - f. Pass the values back until you get to the beginning:
"Eat"
 - g. Display the results:
Display = "Eat"
3. Moving to the last set of answers (first condition equals "False"):
- a. Look at the formula in its basic form:
=IF(Hungry,True,False)
 - b. Look at the formula in its basic form with nesting:
=IF(Hungry,IF(Thirsty,True,False),False)
 - c. Expand the False path:
IF Hungry = False, "Don't eat"
 - d. Since we don't have any further conditions to check for in the False path of the first conditional check, pass the values back to the following:
"Don't eat"
 - e. Display the results:
Display = "Don't eat"

Once we've looked at each path, we can see there are only three possible answers:

1. "Eat and drink"

2. "Eat"
3. "Don't eat"

Since we're looking at the initial condition of "Am I hungry?" and it only has two possible paths (True or False), this again confirms the notion of only two possible answers for a logical operation (even though there are two values that could be displayed for the True answer, only one of them can be chosen and passed back to its parent—the first conditional check, where it will be returned as a result).

Moving back to SharePoint, let's see how this looks in our list.

Go back into the Result column and modify the formula to the following:

```
=IF([Hungry],IF([Thirsty],"Eat and drink","Eat"),"Don't eat")
```

Wait a minute. In the new formula, we're not actually telling it what to look for to test beyond the name of the column itself. Using the same structure we've built for the previous version of our formulas, it would logically make sense to keep them in the same format, such as this:

```
=IF([Hungry]="True",IF([Thirsty]="True","Eat and drink","Eat"),"Don't eat")
```

Rather than this:

```
=IF([Hungry],IF([Thirsty],"Eat and drink","Eat"),"Don't eat")
```

Notice the difference? In the versions we've been using so far, we've defined a value for it to look at (True). Since we're using a very basic test (just asking it to see if the value is True...heck, we're not even looking for a False value), we can use a bit of SharePoint shorthand and omit the True portion altogether, because it automatically fills that in for us when the formula runs!

Recall from the beginning of this chapter that the TRUE function returns the logical value TRUE (used primarily for compatibility with other programs, you can also enter the value TRUE directly into your formulas when a requirement exists to test for the TRUE value).

If you don't specify that you're testing for True, the system is smart enough to assume that and treat it as if you had. If you were to test for anything other than True, you'd need to add the "*value*" part (where *value* equals the literal text you want to test for).

With that in mind, we can now use our shorthand version from this point forward. and trust me, as your formulas gain complexity from adding in more nestings or values to test, you'll want to save as much space as possible, so we might as well start while it's relatively simple.

In our Result column, modify the formula to the following, shown in [Figure 8-12](#):

```
=IF([Hungry],IF([Thirsty],"Eat and drink","Eat"),"Don't eat")
```

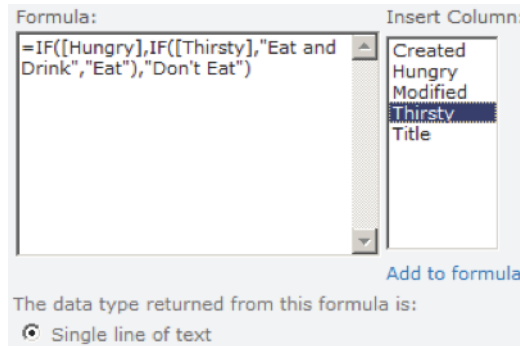



FIGURE 8-12. Modified formula

Save this, then go back to the list and edit the existing item by choosing True for each (see Figure 8-13).

<input type="checkbox"/> Title	Hungry	Thirsty	Result
Test 1 NEW	True	True	Eat and Drink
+ Add new item			

FIGURE 8-13. Verifying answers

As expected (based on our workthrough of the logic earlier), if it is True for both Hungry and Thirsty, we should see a result of “Eat and drink,” which we do.

Test the other various combinations of True and False to see if our logic is correct by editing the values of both Hungry and Thirsty to view the results:

The result of Hungry = True, Thirsty = False is shown in Figure 8-14.

<input type="checkbox"/> Title	Hungry	Thirsty	Result
Test 1 NEW	True	False	Eat
+ Add new item			

FIGURE 8-14. Testing with Hungry = True, Thirsty = False

The result of Hungry = False, Thirsty = True is shown in [Figure 8-15](#).

<input type="checkbox"/> Title	Hungry	Thirsty	Result
Test 1 <small>NEW</small>	False	True	Don't Eat

[+ Add new item](#)

FIGURE 8-15. Testing with Hungry = False, Thirsty = True

The result of Hungry = False, Thirsty = False is shown in [Figure 8-16](#).

<input type="checkbox"/> Title	Hungry	Thirsty	Result
Test 1 <small>NEW</small>	False	False	Don't Eat

[+ Add new item](#)

FIGURE 8-16. Testing with Hungry = False, Thirsty = False

Notice that the last two display the exact same results? This occurs because in the current form of the calculation, we're only moving to the second condition if the first condition returns a value of True.

Working through the logic, it's apparent that we were looking at this from the point of view of just being hungry. What we didn't take into account was the question of "What about if I am only thirsty?"

Therein lies one of the inherent difficulties when working with nested operations. To account for each possible scenario, you must add logical checks for each, which tends to make your formulas expand immensely with each additional question.

In our case, the formula checks the second question if, and only if, the result of the first question is True. If it is, the logic moves on to the next question to see if we are also Thirsty. If the answer to the first question happens to be False, however, we never get a chance to even ask the Thirsty question, so we need to remedy this by rewriting our logic in a fashion that will always ask the second question regardless of the answer from the first question.

We do this by adding in our second question to the False result of the first question.

Here is our original formula:

```
=IF(Hungry,IF(Thirsty,"Eat and drink","Eat"),"Don't eat")
```

Here is our new formula:

```
=IF(Hungry,IF(Thirsty,"Eat and drink","Eat"),IF(Thirsty,"Drink","Don't eat or drink"))
```

This changes our logic to the following:

Is it true that I am hungry?
If so, is it true that I am thirsty?
If so, I will eat and drink.
If not, I will eat.
If it is not true that I am hungry, is it true that I am thirsty?
If so, I will drink.
If not, I will not eat or drink.

By doing this we've, in effect, asked the same (second) question twice, but with different possible answers based on where (or when) it is asked (see [Figure 8-17](#)).

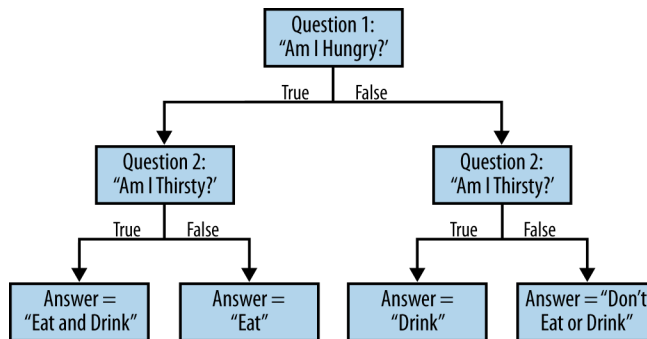


FIGURE 8-17. Adding the question a second time

Based on our questions, we can look at the total possible results by examining the logic of our questions through what is known as a *Truth Table* ([Figure 8-18](#)).

A Truth Table is a testing tool you can use to validate the expected results from a given logic problem by preloading each possible answer and giving a result for each combination of answers. Once set up, you can use it to do a run-through of a problem to see if it returns the expected results.

Most often, this tool is used by programmers when penciling out a piece of logic in an application to test whether it is performing as expected.

Although a “Calculated Column” is not specifically an application or program (in the traditional sense), since we’re working with formulas that have to follow a “Logical” path to arrive at a result, tools like this can help in troubleshooting when the results you get aren’t what you expected.

Question 1 and 2 - "Truth Table"		
Question 1: "Am I Hungry?"	Question 2: "Am I Thirsty?"	Result
TRUE	TRUE	"Eat and Drink"
TRUE	FALSE	"Eat"
FALSE	TRUE	"Drink"
FALSE	FALSE	"Don't Eat or Drink"

FIGURE 8-18. A Truth Table

The combinations of each possible answer will give us a list of what we could see as a final result.

- TRUE TRUE = "Eat and drink"
- TRUE FALSE = "Eat"
- FALSE TRUE = "Drink"
- FALSE FALSE = "Don't eat or drink"

- If Question 1 is True and Question 2 is True, display "Eat and drink".
- If Question 1 is True and Question 2 is False, display "Eat".
- If Question 1 is False and Question 2 is True, display "Drink".
- If Question 1 is False and Question 2 is False, display "Don't eat or drink".

Now that we have our logic flowing correctly and have validated it with a separate tool (the Truth Table), let's verify it again by moving it to SharePoint into our list.

Delete the initial test item from the list and add in four items (shown in Figure 8-19) that follow the pattern laid out in the above Truth Table.

<input type="checkbox"/> Title	Hungry	Thirsty	Result
Hungry and Thirsty NEW	True	True	Eat and Drink
Hungry only NEW	True	False	Eat
Thirsty only NEW	False	True	Don't Eat
Not Hungry or Thirsty NEW	False	False	Don't Eat

+ Add new item

FIGURE 8-19. Adding test items

We can see that in its current form (as we noticed earlier), the formula does not take into account all possible scenarios (notice the last two items on the list).

Now, let's go back in and modify our formula to include our change of checking the Thirsty side of things along with Hungry.

Modify the Results column formula to the following (shown in Figure 8-20):

=IF(Hungry,IF(Thirsty,"Eat and drink","Eat"),IF(Thirsty,"Drink","Don't eat or drink"))

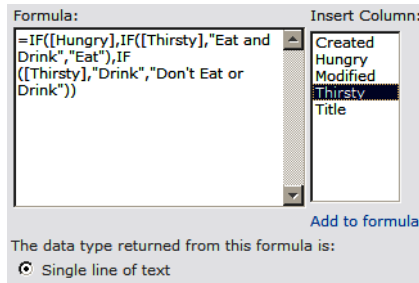


FIGURE 8-20. Modifying the formula

Save this, and look back at our existing list items, shown in [Figure 8-21](#).

<input type="checkbox"/> Title	Hungry	Thirsty	Result
Hungry and Thirsty <small>NEW</small>	True	True	Eat and Drink
Hungry only <small>NEW</small>	True	False	Eat
Thirsty only <small>NEW</small>	False	True	Drink
Not Hungry or Thirsty <small>NEW</small>	False	False	Don't Eat or Drink

[+ Add new item](#)

FIGURE 8-21. Viewing the list

Because we're now taking into account each possible scenario in our modified formula, we can display all of the possible answers based on the choices made.

As I briefly mentioned earlier, as we add in more checks (questions), our formula will grow larger and more complex, in some cases, to the point where it's almost impossible to visually inspect it to find problems.

So, if we were to add in the additional question, "Am I tired?" to the mix, our formula logic expands out to:

Is it true that I am hungry?
 If so, then is it true that I am thirsty?
 If so, then is it true that I am tired?
 If so, I will eat and drink and sleep.
 If not, I will eat and drink.
 If it is not true that I am thirsty, is it true that I am tired?
 If so, I will eat and sleep.
 If not, I will eat.
 If it is not true that I am hungry, is it true that I am thirsty?
 If so, then is it true that I am tired?
 If so, I will drink and sleep.
 If not, I will drink.

If it is not true that I am thirsty, is it true that I am tired?
 If so, I will sleep.
 If not, I will not eat or drink or sleep.

The formula for this becomes:

=IF(Hungry,IF(Thirsty,IF(Tired,“Eat and drink and sleep”,“Eat and drink”),IF(Tired,“Eat and sleep”,“Eat”)),IF(Thirsty,IF(Tired,“Drink then sleep”,“Drink”),IF(Tired,“Sleep”,“Don’t eat or drink or sleep”)))

Notice the three beginning IF statements:

IF(Hungry, IF(Thirsty,IF(Tired...

This literally translates to, “If I’m hungry, then if I’m thirsty, then if I’m tired...” then progresses, based on the results of each.

As complicated as it now appears, the formula is still using the exact same process it was before, just with an additional check along the way.

If we look at the Truth Table for this, we can see the extra growth in the list of possible answers (see [Figure 8-22](#)).

Question 1, 2, 3 - “Truth Table”			
Question 1: “Am I Hungry?”	Question 2: “Am I Thirsty?”	Question 3: “Am I Tired?”	Result
TRUE	TRUE	TRUE	“Eat and Drink and Sleep”
TRUE	TRUE	FALSE	“Eat and Drink”
TRUE	FALSE	TRUE	“Eat and Sleep”
TRUE	FALSE	FALSE	“Eat”
FALSE	TRUE	TRUE	“Drink and Sleep”
FALSE	TRUE	FALSE	“Drink”
FALSE	FALSE	TRUE	“Sleep”
FALSE	FALSE	FALSE	“Don’t Eat or Drink or Sleep”

FIGURE 8-22. The growing Truth Table

Anyone notice a pattern forming in the table?

In the third question column, the True/False values are alternating every other row. In the second question column, they’re alternating every two rows, and the first question column, they’re alternating every four rows.

If we were to add yet another condition (which we will, momentarily), can you guess what the pattern would look like?

If you guessed column one would alternate every eight rows, you’d be correct!!

This is part of what makes Truth Tables such a valuable tool when prototyping or working out the logic beforehand—they seem a bit daunting at first, but *once you figure out the pattern*, you can write them rather quickly and translate the information into a basic flowchart to see the path the logic needs to process through. Once you have this, you will also have the basics, or root, of the formula you need to build to move into SharePoint.

To test what we have so far, let’s look at our existing example, build upon it to make it even more complex, and use our Truth Table and a basic hierarchical flowchart to create our final formula.

NOTE

This exercise is still outside SharePoint—we will come back to the list after we’ve built the formula.

First, let’s see what we have and what we want to add to make it more complex. Our existing formula is as follows:

```
=IF(Hungry,IF(Thirsty,IF(Tired,“Eat and drink and sleep”,“Eat and drink”),IF(Tired,“Eat and sleep”,“Eat”)),IF(Thirsty,IF(Tired,“Drink then sleep”,“Drink”),IF(Tired,“Sleep”,“Don’t eat or drink or sleep”)))
```

With this, we’re checking for three conditions:

- “Am I hungry?”
- “Am I thirsty?”
- “Am I tired?”

To add a new layer of complexity, we’ll add in a fourth condition of “Am I sick?”

Building out the first part of our truth table, we add each column needed (see [Figure 8-23](#)):

Question 1, 2, 3, 4 - “Truth Table”				
Question 1: “Am I Hungry?”	Question 2: “Am I Thirsty?”	Question 3: “Am I Tired?”	Question 4: “Am I Sick?”	Result

FIGURE 8-23. Starting the Truth Table

Next, using the pattern we started to see in the last table, add in rows of cells with alternating patterns (the text True and False) matching the following sequences (starting with True as the first value in each column):

- Question 1: alternate every eight rows
- Question 2: alternate every four rows
- Question 3: alternate every two rows
- Question 4: alternate every other row

When completed, it should appear as shown in [Figure 8-24](#).

Question 1, 2, 3, 4 - "Truth Table"				
Question 1: "Am I Hungry?"	Question 2: "Am I Thirsty?"	Question 3: "Am I Tired?"	Question 4: "Am I Sick?"	Result
TRUE	TRUE	TRUE	TRUE	
TRUE	TRUE	TRUE	FALSE	
TRUE	TRUE	FALSE	TRUE	
TRUE	TRUE	FALSE	FALSE	
TRUE	FALSE	TRUE	TRUE	
TRUE	FALSE	TRUE	FALSE	
TRUE	FALSE	FALSE	TRUE	
TRUE	FALSE	FALSE	FALSE	
FALSE	TRUE	TRUE	TRUE	
FALSE	TRUE	TRUE	FALSE	
FALSE	TRUE	FALSE	TRUE	
FALSE	TRUE	FALSE	FALSE	
FALSE	FALSE	TRUE	TRUE	
FALSE	FALSE	TRUE	FALSE	
FALSE	FALSE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	

FIGURE 8-24. Building the Truth Table

By entering in the True/False values in this predefined pattern, we don't have to try to figure out how to make sure we cover each and every combination—it just works out.

Each additional condition added will double the total rows: 2 conditions = 4 rows, 3 conditions = 8 rows, 4 conditions = 16 rows, and so on.

Now that we have the table, insert text in the Result column for each cell that has a True value. For this, we can just use the simple terms "Eat," "Drink," "Sleep," and "I'm sick" (adding "and" in between to make it appear more sentence-like).

Row one (since each column has a True value) is as follows:

"Eat and drink and sleep, I'm sick"

Row two (having True in only the first three columns) is as follows:

"Eat and drink and sleep"

If we continue with this pattern, row 10 is as follows:

"Drink and sleep"

Fill out the rest of the Results column with answers following the same pattern, and you will come up with the table shown in [Figure 8-25](#).

Truth Table - "AND/OR"				
Question 1: "Am I Hungry?"	Question 2: "Am I Thirsty?"	Question 3: "Am I Tired?"	Question 4: "Am I Sick?"	Result
TRUE	TRUE	TRUE	TRUE	"Eat and Drink and Sleep, I'm Sick"
TRUE	TRUE	TRUE	FALSE	"Eat and Drink and Sleep"
TRUE	TRUE	FALSE	TRUE	"Eat and Drink, I'm Sick"
TRUE	TRUE	FALSE	FALSE	"Eat and Drink"
TRUE	FALSE	TRUE	TRUE	"Eat and Sleep, I'm Sick"
TRUE	FALSE	TRUE	FALSE	"Eat and Sleep"
TRUE	FALSE	FALSE	TRUE	"Eat, I'm Sick"
TRUE	FALSE	FALSE	FALSE	"Eat"
FALSE	TRUE	TRUE	TRUE	"Drink and Sleep, I'm Sick"
FALSE	TRUE	TRUE	FALSE	"Drink and Sleep"
FALSE	TRUE	FALSE	TRUE	"Drink, I'm Sick"
FALSE	TRUE	FALSE	FALSE	"Drink"
FALSE	FALSE	TRUE	TRUE	"Sleep, I'm Sick"
FALSE	FALSE	TRUE	FALSE	"Sleep"
FALSE	FALSE	FALSE	TRUE	"I'm Sick"
FALSE	FALSE	FALSE	FALSE	"Don't Eat or Drink or Sleep, I'm not Sick"

FIGURE 8-25. Building the Truth Table with results

Now that we have a complete list of all expected results, we can build a basic flowchart of how the logic should travel (based on these results).

If you're familiar with Visio, or similar technical drawing programs, start by creating our first question at the very top (this is the entrance point, or start, of our logic), and add *placeholders* for both the True and False results ([Figure 8-26](#)).

A placeholder is a temporary value that will be replaced later. Since we don't yet know what our final result will be for either the True or False values, we enter the text for the logical value in as a temporary marker, which we'll replace once we know the actual value to put in its place.

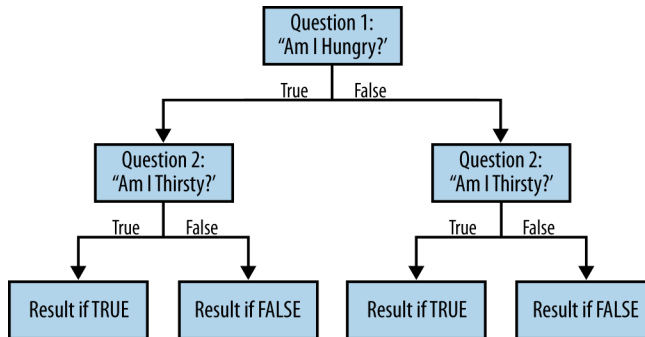


FIGURE 8-27. Flowchart with Question 2 placeholders

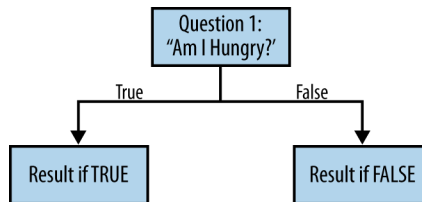


FIGURE 8-26. Flowchart with Question 1 placeholders

Next, add the second question (Thirsty) to both the True and False placeholders from Question 1 (literally replace the text “Result if TRUE” and “Result if FALSE” with the second question), then add placeholders for the True and False results for Question 2 (see [Figure 8-27](#)).

Continue this process by adding in the third question (Tired) to each of the True and False placeholders of the second question, then add the fourth question (Sick) to each of the True and False placeholders of the third question.

Once completed, you’ll have 16 True/False boxes at the very bottom, which (coincidence?) matches up with the total rows in our Truth Table.

The completed hierarchical diagram (logic flowchart), should now appear similar to the one shown in [Figure 8-28](#). I’ve added in the actual values for the True/False results of Question 4, since it’s the last question to be asked in the logic. To do this yourself, follow along in the Truth Table and enter each value, starting from the top, into each placeholder in the diagram starting from the left.

This is all good and fine—we’ve created a nice table of all possible results and have this neat diagram, but what the heck are we supposed to do with it to create the formula for our calculated column in SharePoint?

Believe it or not, this flowchart is a literal translation of our exact formula.

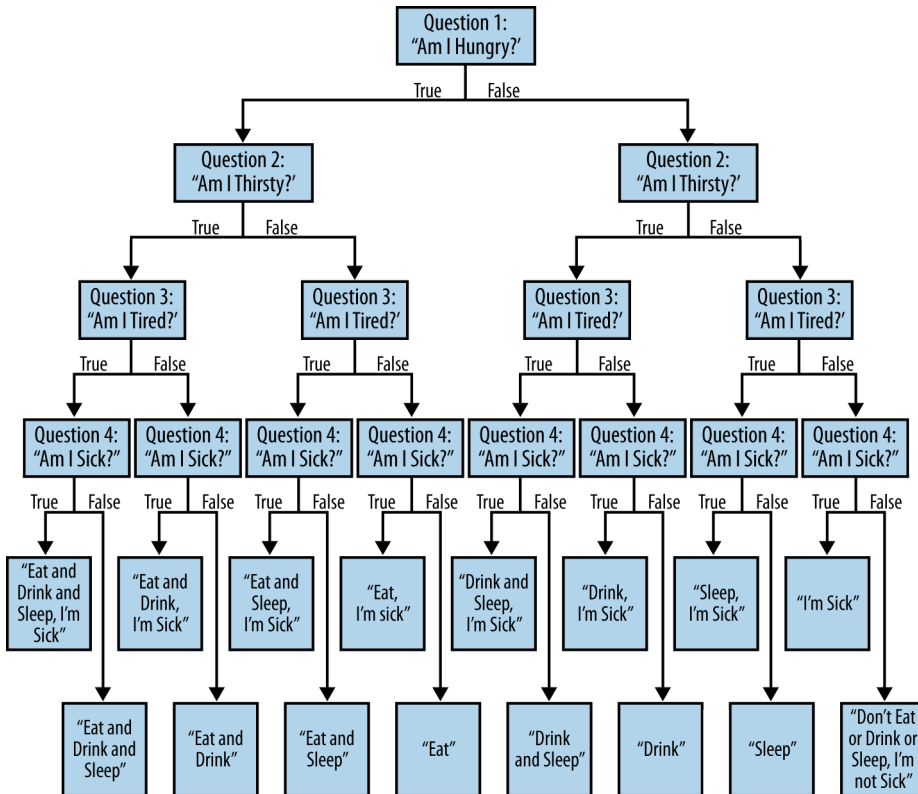


FIGURE 8-28. Flowchart with values

The logic of our formula (with all four questions included), works out to the following:

Is it true that I am hungry?
 If so, then is it true that I am thirsty?
 If so, then is it true that I am tired?
 If so, then is it true that I am sick?
 If so, I will eat and drink and sleep, I'm sick.
 If not, I will eat and drink, and sleep.
 If it is not true that I am tired, is it true that I am sick?
 If so, I will eat and drink, I'm sick.
 If not, I will eat and drink.
 If it is not true that I am thirsty, is it true that I am tired?
 If so, is it true that I am sick?
 If so, I will eat and sleep, I'm sick.
 If so, I will eat and sleep.
 If it is not true that I am tired
 Is it true that I am sick?
 If so, I will eat, I'm sick.

If not, I will eat.
If it is not true that I am hungry, is it true that I am thirsty?
If so, then is it true that I am tired?
If so, then is it true that I am sick?
If so, I will drink and sleep, I'm sick.
If not, I will drink and sleep.
If it is not true that I am tired, is it true that I am sick?
If so, I will drink, I'm sick.
If not, I will drink.
If it is not true that I am thirsty, is it true that I am tired?
If so, is it true that I am sick?
If so, I will sleep, I'm sick.
If not, I will sleep.
If it is not true that I am tired, is it true that I am sick?
If so, I'm sick.
If not, don't eat or drink or sleep, I'm not sick.

In our hierarchical flowchart diagram, we need to add the actual formula text for each of our questions (the IF(condition,True,False) pieces), so for each True/False value where we currently have our questions listed, add in a simple IF statement in the same manner you'd use to enter it into SharePoint (or Excel).

Question 1 is: IF(Hungry,True,False)
Question 2 is: IF(Thirsty,True,False)
Question 3 is: IF(Tired,True,False)
Question 4 is: IF(Sick,True,False)

You will end up with something similar to [Figure 8-29](#).

Now, look at the diagram and observe where each piece is. Since we've just added the text for our formulas into each question, we have a bunch of True and False placeholders throughout the entire diagram (in each formula).

Each of these placeholders will be housing the child branch directly beneath it, so we need to start moving things around to progressively build out our final formula.

The next step is simply to match up each box on the chart with its immediate parent in the correct placeholder (True/False) position.

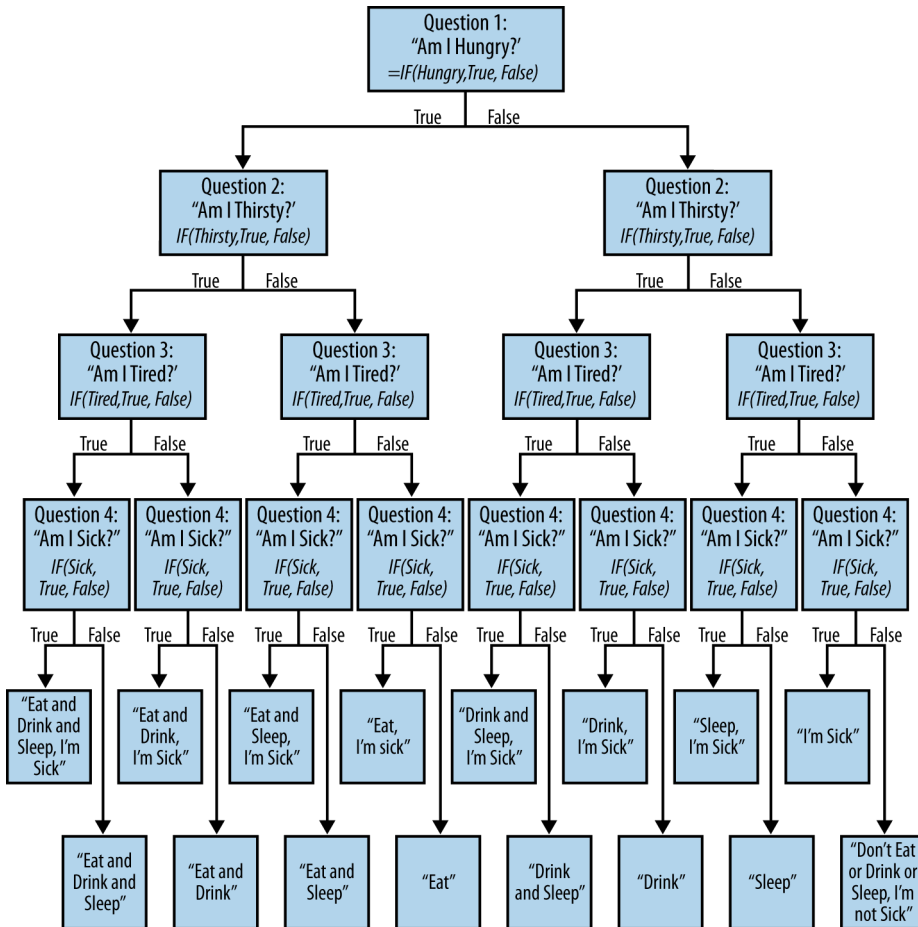


FIGURE 8-29. Flowchart with formulas

Do the following, starting from the extreme bottom left of the chart:

WARNING

Work slowly on this part—take your time with each step to make sure you’re placing the correct part into each section.

1. Take the result text from the True value below Question 4 and paste it in place of the literal text True in Question 4 directly above it (remember, True is our placeholder, so simply replace it with the answer text below it). This is shown in Figure 8-30, and the result is shown in Figure 8-31.

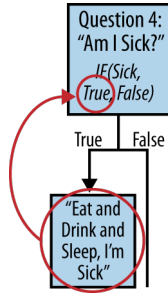


FIGURE 8-30. Replacing the Question 4 placeholders

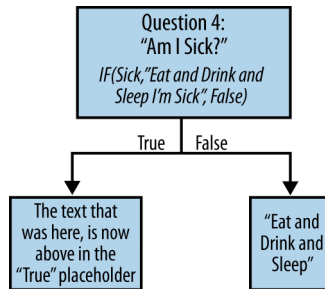


FIGURE 8-31. Question 4 with the True result text

- Do the exact same for the False answer by moving its text up to the False placeholder. The result is shown in Figure 8-32.

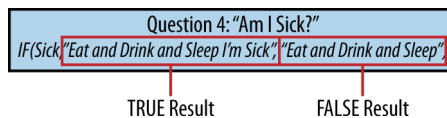


FIGURE 8-32. Question 4 with the False result text

- Continue this pattern, moving to the right for each True and False result of the 16 total Question 4 parts and moving them to the Question 4 placeholder directly above.

When completed, you will have eight different versions of Question 4, each with different values for True and False results (see Figure 8-33).

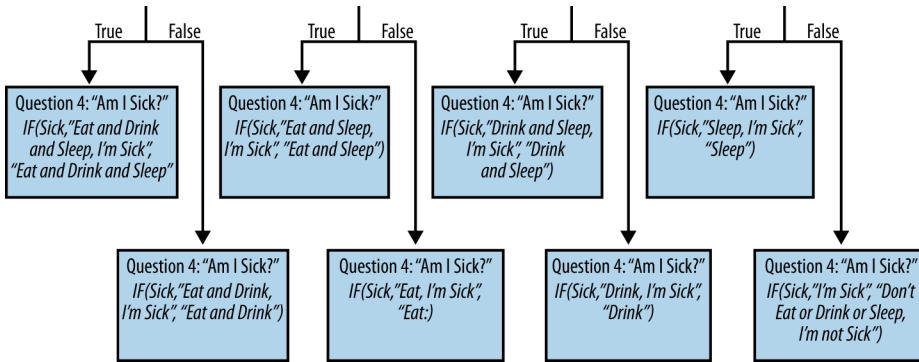


FIGURE 8-33. Completed Question 4 results

Our next step is to follow the exact same process for Question 3, copying each result back up one level to the placeholders (True/False).

Copy the IF statement of each of our eight Question 4 boxes and move each to its immediate parent's True/False placeholder (Figure 8-34).

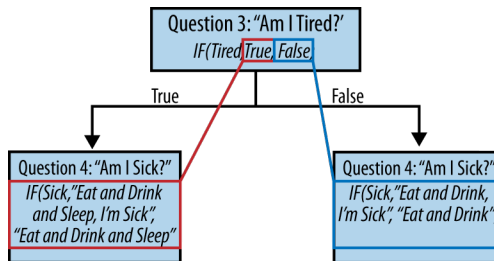


FIGURE 8-34. Replacing the Question 3 placeholders

The result is shown in Figure 8-35.

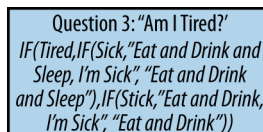


FIGURE 8-35. Completed Question 3 results

As you move through each of the eight Question 4 results and move each of their formulas up to their parent Question 3 True/False placeholder, you should be able to see the final formula coming together (Figure 8-36).

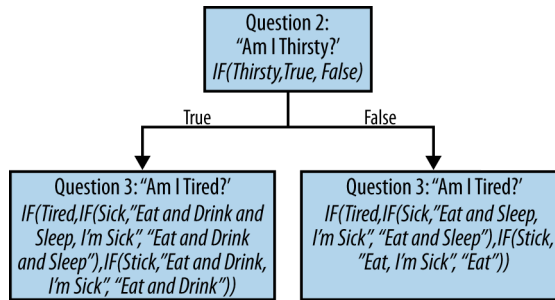


FIGURE 8-36. Questions 2 and 3 results

Perform this process again for Question 2 by moving each of the versions of Question 3 into their parent Question 2 results, then one last time by moving the two versions of Question 2 back up to the top-level Question 1 placeholders.

When completed, the top-level question (our entry point to the logic) will have the final formula, shown in Figure 8-37.

```

=IF(Hungry,IF(Thirsty,IF(Sick,"Eat and Drink and Sleep, I'm Sick","Eat and Drink and Sleep"),IF(Sick,"Eat and Drink, I'm Sick","Eat and Drink")),IF(Tired,IF(Sick,"Eat and Sleep, I'm Sick","Eat and Sleep"),IF(Sick,"Eat, I'm Sick","Eat"))),IF(Thirsty,IF(Tired,IF(Sick,"Drink and Sleep, I'm Sick","Drink and Sleep"),IF(Sick,"Drink, I'm Sick","Drink")),IF(Tired,IF(Sick,"Sleep, I'm Sick","Sleep"),IF(Sick,"I'm Sick","Don't Eat or Drink or Sleep, I'm not Sick"))))
  
```

FIGURE 8-37. The final formula

Neat, huh? You now have a complete formula that includes every step of the logic needed to ask each question in a manner that allows for each possible answer.

The process I've detailed does take a bit of time, but the end result is a complete formula without errors, and you get the benefit of not having to figure out where all the closing parentheses go (one of the most troublesome parts of working with complex nested formulas).

To test that our formula indeed is correct, we'll move to our list in SharePoint, where we can see if our expected results from our Truth Table match up.

1. In our list, we need to add in our additional two Question columns, so create two more Choice columns (in the same fashion as the existing Hungry and Thirsty columns), called Tired and Sick.
2. Modify the Column Ordering to display the items on the New Item form in the correct order (see Figure 8-38).

Field Name	Position from Top
Title	1
Hungry	2
Thirsty	3
Tired	4
Sick	5
Result	6

FIGURE 8-38. Reordering the fields on the form

3. Modify the view to display the fields in the same order (see Figure 8-39).

Display	Column Name	Position from Left
<input checked="" type="checkbox"/>	Title (linked to item with edit menu)	1
<input checked="" type="checkbox"/>	Hungry	2
<input checked="" type="checkbox"/>	Thirsty	3
<input checked="" type="checkbox"/>	Tired	4
<input checked="" type="checkbox"/>	Sick	5
<input checked="" type="checkbox"/>	Result	6

FIGURE 8-39. Reordering the fields in the view

4. In the Result column, replace the existing formula with the one we just built (see Figure 8-40).

<p>Formula:</p> <pre>=IF(Hungry,IF(Thirsty,IF(Tired,IF(Sick, "Eat and Drink and Sleep, I'm Sick","Eat and Drink and Sleep"),IF(Sick,"Eat and Drink, I'm Sick","Eat and Drink")),IF(Tired,IF(Sick,"Eat and Sleep, I'm Sick","Eat and Sleep"),IF(Sick,"Eat, I'm Sick","Eat"))),IF(Thirsty,IF(Tired,IF(Sic k,"Drink and Sleep, I'm Sick","Drink and Sleep"),IF(Sick,"Drink, I'm Sick","Drink")),IF(Tired,IF(Sick,"Sleep, I'm Sick","Sleep"),IF(Sick,"I'm Sick","Don't Eat or Drink or Sleep, I'm not Sick"))))</pre>	<p>Insert Column:</p> <div style="border: 1px solid gray; padding: 2px;"> <ul style="list-style-type: none"> Created Hungry Modified Sick Thirsty Tired Title </div>
---	--

FIGURE 8-40. Modifying the formula

NOTE

If you receive an error after entering the formula, the first thing to check is that your quotes are not formatted to a specific font style. Unfortunately, an issue that has persisted throughout the several versions of SharePoint in calculated columns is that if you are using a program that has any type of formatting for font styles (pretty much anything), you may need to clean the quotes before pasting in a formula.

Notice the style of the quotes in [Figure 8-41](#). The first line has curly quotes that have a beginning and ending style, whereas the second line has the same character for both (vertical lines rather than slightly curved). If you get an error after pasting your formula, paste it into a text editor and perform a “Find and Replace” operation to switch all the quotes to unformatted versions.

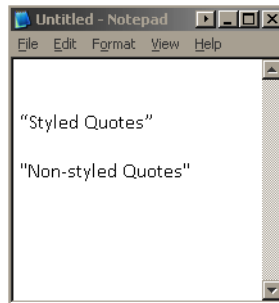


FIGURE 8-41. Styled quotes

Now that we have the columns set up, test everything by deleting all items currently on the list and create several items that cover the range of all possible answers (hint: use the Truth Table as a reference when creating the items—you should have 16 entries, and we’ll also create one additional, explained next). The results will appear like the one shown in [Figure 8-42](#).

<input type="checkbox"/> Title	Hungry	Thirsty	Tired	Sick	Result
Eat Drink Sleep Sick <small>NEW</small>	True	True	True	True	Eat and Drink and Sleep, I'm Sick
Eat Drink Sleep <small>NEW</small>	True	True	True	False	Eat and Drink and Sleep
Eat Drink Sick <small>NEW</small>	True	True	False	True	Eat and Drink, I'm Sick
Eat Drink <small>NEW</small>	True	True	False	False	Eat and Drink
Eat Sleep Sick <small>NEW</small>	True	False	True	True	Eat and Sleep, I'm Sick
Eat Sleep <small>NEW</small>	True	False	True	False	Eat and Sleep
Eat Sick <small>NEW</small>	True	False	False	True	Eat, I'm Sick
Eat <small>NEW</small>	True	False	False	False	Eat
Drink Sleep Sick <small>NEW</small>	False	True	True	True	Drink and Sleep, I'm Sick
Drink Sleep <small>NEW</small>	False	True	True	False	Drink and Sleep
Drink Sick <small>NEW</small>	False	True	False	True	Drink, I'm Sick
Drink <small>NEW</small>	False	True	False	False	Drink
Sleep Sick <small>NEW</small>	False	False	True	True	Sleep, I'm Sick
Sleep <small>NEW</small>	False	False	True	False	Sleep
Sick <small>NEW</small>	False	False	False	True	I'm Sick
Nothing <small>NEW</small>	False	False	False	False	Don't Eat or Drink or Sleep, I'm not Sick
Empty (no values chosen) <small>NEW</small>					Don't Eat or Drink or Sleep, I'm not Sick

[+ Add new item](#)

FIGURE 8-42. Creating items that match the Truth Table

Based on our Truth Table, we can see that our logic does indeed catch each and every possible result (with the addition of displaying a result even if fields were skipped).

Using the method of building out a Truth Table and a flowchart for each and every nested formula may not always be the best approach for you, given the time it takes to build them, but as the complexity of your formulas increases, you might find that having them can decrease the headache time of debugging and troubleshooting when your logic goes awry.

Hints:

- Use a spreadsheet program like Excel to quickly build out a Truth Table when needed (since it's nothing more than columns and rows anyway).
- Use a diagramming or flowcharting program like Visio (or a piece of scratch paper in a pinch) for quickly building hierarchical flowcharts ([OpenOffice](#) is another great option if you don't have the budget for Microsoft Office).

The Cousins: OR and AND

Now that we've covered how to build the logic of a basic formula (yes, even though it has multiple nested conditional checks, it's still a basic formula), we're going to look at bringing in

the additional logical functions OR and AND to test for combinations of logical tests all within the same check.

To begin, we first need to understand the logic associated with both new functions, so we'll revisit our Truth Table to illustrate how OR and AND work (Figure 8-43).

Truth Table - "OR"		
Question 1	Question 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

FIGURE 8-43. OR Truth Table

The idea behind the OR function is that if *any* of the values are true, the entire statement is true, and if none of them are true, the entire statement is false.

We can use the OR function to test for up to 30 different values together and, as long as at least one of them results in true, the end result will also be true.

The OR function has the following format:

`=OR(test1,test2,test3,...test30)`

It could be something as simple as the following:

`=OR(1+1=2)` : result = TRUE since 1+1 does equal 2

Or it could be more complex with a series of nested logical functions similar to our example earlier in this chapter (the OR function just puts them all together to determine if any of them results in a true result).

To explore the OR function, we'll look at a basic business model for paychecks. For someone to receive a paycheck, there are minimum criteria that must be met; the employee should be in the system (an employee on record) and should have worked a number of hours. There's the possibility that the employee took vacation days or sick leave as well, so we're going to take a look at these three types of hours (working, sick, and vacation) to see if the employee should be issued a paycheck.

For our new logic problem, our conditions (questions) will be:

- Regular hours
- Vacation
- Sick leave

If we add these conditions to our Truth Table, we get the table shown in Figure 8-44.

Truth Table - "OR"			
Regular Hours	Vacation	Sick Leave	Result
TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	FALSE	TRUE
TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE

FIGURE 8-44. OR Truth Table with conditions

The golden rule with an OR function is that no matter what conditions are, as long as one of them yields a true result, the entire formula is considered true.

Let's look at each row in the Truth Table and expand each question so we can fully understand it.

Row 1: The employee worked regular hours, took vacation, and submitted sick leave (true for all three conditions). Since at least one of the three is true (satisfying our golden rule), the final result is true, so the employee will be issued a paycheck.

Row 2: The employee worked regular hours and took vacation, but did not submit sick leave. Again, since at least one of the three is true, the final result is true, so the employee will be issued a paycheck.

Rows 3–7: At least one of the three is true, so the final result is true, which will allow for a paycheck to be issued to the employee.

Row 8: In all previous rows, at least one condition is true, so the final result is true. However, in the last row, none of the conditions have been met, so the final result is false and the employee will not be issued a paycheck.

Pretty simple so far—it's basically an "any or none" test. As long as one of the conditions is TRUE, the entire formula yields a TRUE result as well. What if, instead of any conditions being True, we want to test for a combination of conditions? This is where the cousin to the OR function comes in—the AND function.

The AND function is similar to the OR function in that it will look at up to 30 different values for comparison, but its difference is that for the final result to be True, *every condition must be True* in the entire check or the end result will be False.

It's important to understand this difference, because it will influence your choice to use one or the other and how you will build your logic.

Before we look at our Truth Table to see how the different combinations of values will produce results when using the AND function, we need to re-evaluate the conditions we're checking. If we were to look at the Truth Table now, before modifying our conditions, the only combination of values that would ever result in an employee receiving a paycheck would be if the employee had actually submitted all three types of hours (must be an interesting job if an employee can work, go on vacation, and take sick leave each and every week), so let's modify things a little before examining the table.

We know that in order for an employee to be issued a paycheck, he must first be in the system as an actual employee, so we'll use this as our first condition to check.

Employee

Next, we know (based on our test with the usage of the OR function) that in order for the employee to be issued a paycheck, he must have some sort of record of time to track how many hours he is to be paid for. This is in the form of three possible records:

- Regular hours
- Vacation
- Sick leave

Since we're not quite at the point where we can merge the OR function and the AND function (coming shortly), let's look at three different versions of the Truth Table that will process each type of hours tracked individually. Figures 8-45, 8-46, and 8-47 show Truth Tables for regular hours, vacation, and sick leave, respectively.

Truth Table - "AND" (Regular Hours)		
Employee	Regular Hours	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

FIGURE 8-45. AND Truth Table for regular hours

Truth Table - "AND" (Vacation)		
Employee	Vacation	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

FIGURE 8-46. AND Truth Table for vacation

Truth Table - "AND" (Sick Leave)		
Employee	Sick Leave	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

FIGURE 8-47. AND Truth Table for sick leave

Notice in all three Truth Tables that the only time the end result is True is when both conditions are True as well:

Person is "Employee" and has worked "Regular Hours" = "TRUE" (Issue paycheck)

Person is "Employee" and has taken "Vacation" = "TRUE" (Issue paycheck)

Person is "Employee" and has submitted "Sick Leave" = "TRUE" (Issue paycheck)

In all other cases in each table, at least one of the conditions is False, so the end result is also False (the Golden Rule for the AND function is all conditions must be True for the formula to be True).

Since we've now looked at the basics of both the OR and AND functions, we're going to merge the two to test for multiple conditions together. The approach we need to take is based on what we already know:

- For an employee to receive a paycheck, she must be in the system. In other words, she must be an employee.
- In addition to being an employee, she must have committed some sort of hours, either through the accumulation of regular hours (actually working) or through the usage or redemption of vacation or sick leave.

Notice the specific wording I've used in the above list: "...must be an employee," "In addition to...," and "...either through..."

Truth Table - "AND/OR"				
Employee	Regular Hours	Vacation	Sick Leave	Result
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	FALSE	TRUE
TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	TRUE	FALSE	FALSE	TRUE
TRUE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE

FIGURE 8-48. Merged AND/OR Truth Table

I wrote these two items in a specific manner to illustrate that by using plain English to describe the conditions, we can do half the work for ourselves. Based on these items and how they're written, we can immediately begin to see what we'll need to design our Truth Table in order to account for each possible scenario.

In the first item, since I stated the person must be an employee, that suggests the usage of an AND function, since in the AND, the values must all be true, regardless of how many there are.

The second items starts with, "In addition to being an employee." This makes the three conditions listed in this item the second part of the AND, and also suggests that we have three options to choose from to pair up with the first half of the AND (kind of sounds like an OR, doesn't it?).

So, we basically just need to list each of the four pieces of our criteria within the Truth Table to get a set of test data based on the results of the True/False values.

In our Truth Table, following the exact same model we've been using, create your five columns (four conditions and one result). Your table should look like the one shown in [Figure 8-48](#).

Upon examining this Truth Table, you should see very clearly that no matter what form of hours is submitted, if the submitter isn't an employee, the result is always False. As long as the person is an employee, the only time the value result is False is when there aren't any hours of any kind submitted. So, it makes sense by this table that any employee who has committed some form of hours (through working, vacation, or sick leave) will be issued a paycheck.

To simplify, as long as the person is an employee and at least one of the hours columns has a True value, the entire row is True (both golden rules have been satisfied—the rule of at least one True in the OR function results in a True value for the entire OR, which will be added to the AND rule that all values must be True).

To take what we've learned by using our Truth Table and convert it into a usable formula for our SharePoint list, we need to work through the logic to make sure we don't skip any steps, which can happen when designing complicated formulas on the fly.

Since we have only a single IF function to work with (is the person an employee who has submitted any hours), using the previous approach of building out a hierarchical diagram won't really help us too much in this case.

Instead, we'll write out the question we're trying to answer into plain English based on what we discovered in our Truth Table, then dissect the sentence and build out our formula.

Once again, here are our criteria:

1. For an employee to receive a paycheck, he must be in the system. In other words, he must be an employee.
2. In addition to being an employee, he must have committed some sort of hours, either through the accumulation of regular hours (actually working), or through the usage or redemption of vacation or sick leave.

Based on our two criteria, the logic that needs to occur is as follows:

If a person is an employee and has submitted hours by working regular hours, taking vacation, and taking sick leave, or if the person is an employee and has submitted hours by working regular hours and taking vacation, or the person is an employee and has submitted hours by working regular hours and taking sick leave, or the person is an employee and has submitted hours by working regular hours, or the person is an employee and has submitted hours by taking vacation and sick leave, or the person is an employee and has submitted hours by taking vacation, or the person is an employee and has submitted hours by taking sick leave, issue him a paycheck. If not, do not issue him a paycheck.

A bit long-winded, so let's simplify it to the following:

If a person is an employee and has submitted hours, either by working regular hours or by taking vacation or sick leave, issue a paycheck. If not, do not issue a paycheck.

Since an OR function is based on one or more of its list of conditions that evaluate to *true*, yielding an end result of true, the above simplified version will do just fine.

Looking back at our list of criteria, the first part is simply asking the question, “Is the person an employee?” We saw this in our expanded textual walkthrough. Since this is nothing more than a basic yes/no (true/false) question, we can further simplify it by making it a single-word question:

Employee?

The second part of the criteria is about looking at a list of options. What form of hours was submitted? Based on our simplified sentence, we can clearly see what this list is, so we’ll grab just the list of options:

Regular hours
Vacation
Sick leave

Being a list of options in which at least one of them must be true, we’ll place them in an OR function:

OR(“Regular Hours”,“Vacation”,“Sick Leave”)

The entire OR function itself will be a mandatory check, but its contents are all optional. The only caveat is that at least one of them must return *true* for the OR to be true, so since the OR itself is part of the overall condition to check for (in addition to the employee check), we’ll add it and the first criteria to an AND function:

AND(“Employee”,OR(“Regular Hours”,“Vacation”,“Sick Leave”))

We now have a completed formula that will work just fine, but since we haven’t yet defined just what is to occur based on the two possible results (true and false), we’ll need to place this entire formula inside an IF function to test it and decide what to do if the result is true and what to do if the result is false.

To review the format for each function, the IF functions is as follows:

IF(logical condition, result if true, result if false)

The OR function is written in the following form:

OR(test1, test2, test3,...test30)

The AND function is the same as the OR in the following form:

AND(test1, test2, test3,...test30)

So, as we build out each part of the overall formula, we need to start with the placeholders and replace each one with the actual values or functions to test. We’ll do this step by step as follows:

1. List the functions with placeholders.

IF(logical condition, result if true, result if false)

OR(test1, test2, test3,...test30)

AND(test1, test2, test3,...test30)

2. Start replacing placeholders with actual values.

```
IF(Condition,TRUE,FALSE)
```

```
OR("Regular Hours","Vacation","Sick Leave")
```

```
AND("Employee",test2)
```

3. Add the OR to the second test placeholder in the AND.

```
AND("Employee",OR("Regular Hours","Vacation","Sick Leave"))
```

4. Add the full condition (merged "AND/OR") to the "IF" condition placeholder.

```
IF(AND("Employee",OR("Regular Hours","Vacation","Sick Leave")),TRUE,FALSE)
```

5. Add the result values to the "IF" TRUE/FALSE placeholders.

```
IF(AND("Employee",OR("Regular Hours","Vacation","Sick Leave")),"Issue  
Paycheck","Do Not Issue Paycheck")
```

Now that we've built our formula, we'll move it into a list in SharePoint to test it and make sure it gives us the results we saw in our Truth Table.

1. In SharePoint, create a new custom list called Employee Paychecks.
2. Create a column called Employee, make it a Choice type, and add in the two choices of Yes and No as Radio Buttons (see [Figure 8-49](#)).

Column name:
Employee

The type of information in this column is:

- Single line of text
- Multiple lines of text
- Choice (menu to choose from)

Type each choice on a separate line:

Yes
No

Display choices using:

- Drop-Down Menu
- Radio Buttons

Default value:

- Choice
- Calculated Value

Add to default view

FIGURE 8-49. Creating a new column

NOTE

I chose the Choice column rather than the Yes/No checkbox option because the Yes/No type doesn't have a clear display as to what it really is (it doesn't actually display the text "Yes" or "No" and can be confusing to users). Using a Choice type instead will clearly display just what the column is for and will make it more readable.

3. Create three additional columns, called Regular Hours, Vacation, and Sick Leave, for the three types of hours. Use Choice type and Radio Buttons (see [Figure 8-50](#)).

Column (click to edit)	Type
Title	Single line of text
Employee	Choice
Regular Hours	Choice
Vacation	Choice
Sick Leave	Choice
Created By	Person or Group
Modified By	Person or Group

FIGURE 8-50. All columns

Since we're using Choice columns in our list, we'll need to add in a little bit more to our formula to test for the values that they'll be returning. We'll also need to change it to include our column names rather than their text equivalents (consider those as placeholders that were used prior to creating the columns).

4. Building our formula with the names of our columns in mind as straight text, simply replace the quotes surrounding them with brackets. I always model formulas that are based on column names with brackets added, even if the name is a single word with no spaces. This ensures that when I paste in the formula, I'll have less chance of errors.

Here is our formula so far:

```
IF(AND("Employee",OR("Regular Hours","Vacation","Sick Leave")), "Issue  
Paycheck", "Do Not Issue Paycheck")
```

- a. When we replace the quotes to reflect column names, we get the following:

```
IF(AND([Employee],OR([Regular Hours],[Vacation],[Sick Leave]), "Issue  
Paycheck", "Do Not Issue Paycheck")
```

- b. When we add the parts that reflect our options in the choice columns (condition="Yes"), our formula is as follows:

```
=IF(AND([Employee]="Yes",OR([Regular Hours]="Yes",[Vacation]="Yes",[Sick  
Leave]="Yes")), "Issue Paycheck", "Do Not Issue Paycheck")
```

This will allow us to check the return text value of each column to see if it's Yes or No (we could have used the same true/false shorthand approach as we did in the earlier examples, but I want to illustrate how you can check for any text value you want to use).

5. Next, we need to add in our formula, so create our final column called Issue Paycheck, make it a Calculated type with a return type of Text, and enter the modified formula (see [Figure 8-51](#)).

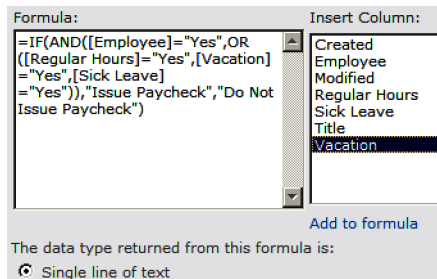


FIGURE 8-51. Calculated column and formula

6. Now that we have all our columns created and formula added, create a series of items on the list using the same combinations of values from the Truth Table (see [Figure 8-52](#)).

NOTE

I've added four extra items to show that it will handle empty or incomplete values as well (since empty does not equal Yes, the result will be False).

Looking at the list, we can see that we did, indeed, get the expected results we saw in our Truth Table and that our formula is working through the logic exactly as we worked it out.

Obviously, this isn't yet a complete solution for a business to use, since the list is showing nonemployees submitting accumulated hours, but it should serve as a basis for building in the extra functionality needed to get it closer. One of many things you could add in for extra functionality is to change Employee to a Choice column and add different categories of employees such as Volunteer, Full-time, or Part-time to help identify who should be getting paychecks.

Now that we've worked through the logic needed for this example, you may be wondering why we didn't use the same approach as the Hungry/Thirsty/Tired/Sick list. If we had taken the same direction as the first example (detailing the IF) to test for each possible combination of values with multiple nested IF functions, the formula would have been considerably longer and would have worked, but would have been much less efficient, since it would have needed to run through each and every scenario before coming up with a result.

<input type="checkbox"/>		Title	Employee	Regular Hours	Vacation	Sick Leave	Issue Paycheck
		Employee #1	Yes	Yes	Yes	Yes	Issue Paycheck
		Employee #2	Yes	Yes	Yes	No	Issue Paycheck
		Employee #3	Yes	Yes	No	Yes	Issue Paycheck
		Employee #4	Yes	Yes	No	No	Issue Paycheck
		Employee #5	Yes	No	Yes	Yes	Issue Paycheck
		Employee #6	Yes	No	Yes	No	Issue Paycheck
		Employee #7	Yes	No	No	Yes	Issue Paycheck
		Employee #8	Yes	No	No	No	Do Not Issue Paycheck
		Employee #9	No	Yes	Yes	Yes	Do Not Issue Paycheck
		Employee #10	No	Yes	Yes	No	Do Not Issue Paycheck
		Employee #11	No	Yes	No	Yes	Do Not Issue Paycheck
		Employee #12	No	Yes	No	No	Do Not Issue Paycheck
		Employee #13	No	No	Yes	Yes	Do Not Issue Paycheck
		Employee #14	No	No	Yes	No	Do Not Issue Paycheck
		Employee #15	No	No	No	Yes	Do Not Issue Paycheck
		Employee #16	No	No	No	No	Do Not Issue Paycheck
		Empty Test #1					Do Not Issue Paycheck
		Empty Test #2	Yes				Do Not Issue Paycheck
		Empty Test #3	No				Do Not Issue Paycheck
		Empty Test #4		Yes	Yes	Yes	Do Not Issue Paycheck

Add new item

FIGURE 8-52. List items that match the Truth Table

By using the OR and AND functions to test for combinations of values at the same time, we’ve not only lessened the development time needed to create the formula, we’ve also made debugging and testing much easier (always a good thing) and we’ve lowered the processing time needed by the system to get an answer. In a pure programmatic sense, efficiency in the logic process will make the page and list load quicker and lessen the load on the database. This isn’t always true though—sometimes it is better to take the long approach if complexity is needed, but in this case, a simple approach is far better.

Summary

Logic formulas can sometimes be difficult to manage, hard to build if they’re complex, and downright problematic to debug if something isn’t going as you expected. If you break the formulas down into chunks—look at each function, what it’s made of, how each piece works, where you can merge some together with other functions and work through them in basic terms as I’ve illustrated in the examples—you’ll find that you’re well on your way to taming your own “Elusive Calculated Columns.”

Creating Document Libraries with Mixed Content Sources

Eric Alexander

This chapter shows how both documents and list items can be contained in the same document library for a consistent user interface. Often, sites are constructed so that documents are separated from useful internal or external links that may be relevant to that content, but this doesn't have to be the case. Documents and list items can both live in the same spot and blend seamlessly together. In this chapter, we'll examine the steps to achieve this result in both Windows SharePoint Services 3.0 (WSS)/Microsoft Office SharePoint Server 2007 (MOSS) and SharePoint Foundations/SharePoint Server 2010.

Background

On NothingButSharePoint.com, formerly EndUserSharePoint.com, we would get great questions on the Stump the Panel forum. These questions were from users who ranged from absolute SharePoint beginners to seasoned SharePoint administrators, End Users, and Developers. This forum, which is now located at <https://www.nothingbutsharepoint.com/sites/eusp/Forum/Pages/default.aspx>, is a hotbed for great blog-worthy material. This is one of those great submissions from the forums.

I am new to SharePoint and am trying to set up a research library for our branch. I have created 8 bookshelves for my library using the Document Libraries. I would like to combine different document types (.pdf .doc and .xls) for a particular subject area (e.g. homelessness) and links to

websites with relevant subject area information on the same 'bookshelf' of my library. I know that I can put in links in a Links List Library but I would like to have both documents and website links for a subject area together. Can this be done?

This is an excellent question. It taps into what appears to have been a little-known feature in SharePoint, the Link to a Document content type. This content type allows you to blend both documents and links to other things (it could be websites or documents residing in other areas) in the same document library.

Benefits of this approach include a better user experience; users don't have to go to separate places to see similar content, and it minimizes content duplication when pointing to files that reside in other areas.

Think about that last sentence for a second. Imagine a scenario in which HR forms are uploaded to a central repository, perhaps at the root level of the web application. Users within site collections in the web application might want quicker access to these forms, so they create copies of them and store them in a document library within their site collections. Now HR has gone and posted new revisions to the forms, perhaps minor, so no one was notified. You now have a mismatch in the forms copied to your site.

If this is a large corporation, there could be dozens of copies of this one form floating around. This becomes especially problematic as the nature of these forms becomes more complex. The Link to a Document content type approach eliminates these pain points by linking directly to the source. The great thing about this is if the source file is modified and has the filename changed, the link to a document continues to function without any interruption, and the updated form is always available.

Configuring a Document Library in SharePoint Server 2010/ SharePoint Foundation 2010

Below are the steps needed to enable the Link to a Document content type for a library in SharePoint Server 2010 and SharePoint Foundation 2010.

1. Navigate to the desired library to contain both links and documents. In the ribbon, click the Library tab and click the Library Settings button (shown in [Figure 9-1](#)). This interface should look familiar if you've done any configuration of lists or libraries before.

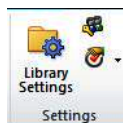


FIGURE 9-1. Click Library Settings in the ribbon

2. In the General Settings section, click the Advanced settings link (shown in [Figure 9-2](#)).

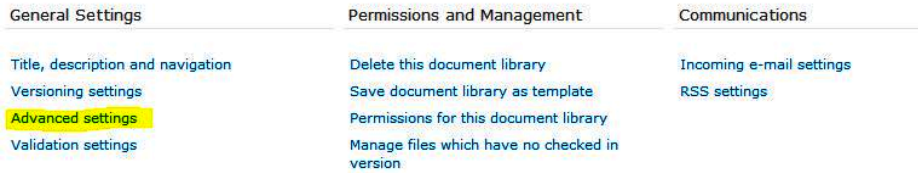


FIGURE 9-2. Click Advanced settings

On the resulting page, you'll be presented with several options. The first option in the list is the area we're concerned with. The Content Type option says, "Specify whether to allow the management of content types on this document library. Each content type will appear on the new button and can have a unique set of columns, workflows and other behaviors." This is exactly what we want.

3. Set the toggle to Yes and click OK to save the settings (see [Figure 9-3](#)).

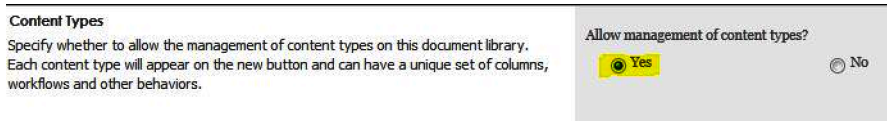


FIGURE 9-3. Toggle Allow management of content types to Yes

You'll then be redirected back to the document library settings screen. It will now look different than before because of the additional Content Type section. This is where we want to focus our attention next.

4. Click the "Add from existing site content types" link (see [Figure 9-4](#)).

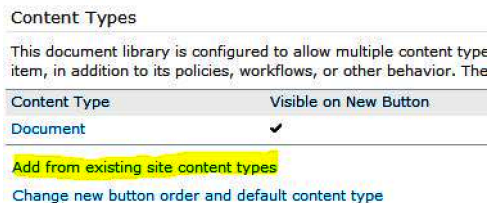


FIGURE 9-4. Click the "Add from existing site content types" link

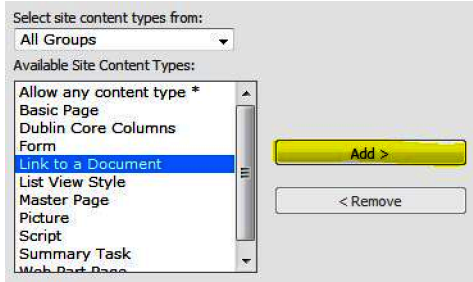


FIGURE 9-5. Highlight *Link to a Document* and click *Add*

You'll be shown a page that lists all the existing available content types at the current level of the site collection.

5. Select the *Link to a Document* content type from the list and click *Add*, then click *OK* to save the changes (Figure 9-5).

You'll now be redirected back to the *Settings* page for the document library. Now it's time to put our handy work into action.

6. Navigate back to the document library via the breadcrumbs, the *Quick Launch*, or via the *View All Site Content* link. Click the *Documents* tab. You'll now see the two content types available within the *New Document* drop-down list (see Figure 9-6).

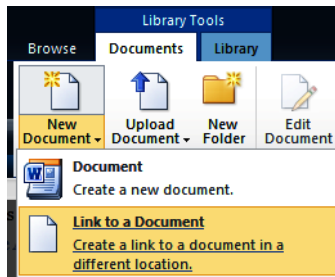


FIGURE 9-6. The new content type is available on the *New* menu

After adding some content into my library, I get a view that looks like the one shown in Figure 9-7.



FIGURE 9-7. List view with Type field included

However, modifying the view and removing the Type icon from the display results in a seamlessly blended listing of content (see Figure 9-8).



FIGURE 9-8. List view with Type field excluded

Configuring a Document Library in MOSS 2007/WSS 3.0

Below are the steps needed to enable the Link to a Document content type for a library in a Microsoft Office SharePoint Server 2007 (MOSS)/Windows SharePoint Services 3.0 (WSS) environment.

1. Navigate to the desired library to contain both links and documents. Click Settings and then click Document Library Settings (see Figure 9-9).

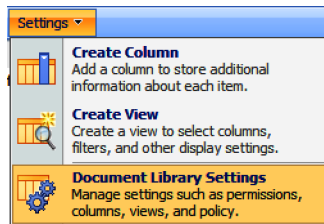


FIGURE 9-9. Go to the Library settings

2. In the General Settings column, click the Advanced Settings link (see Figure 9-10).

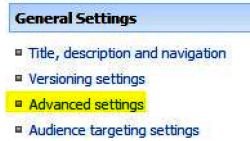


FIGURE 9-10. Click *Advanced settings*

On the resulting page, you'll be presented with several options. The first option in the list is the area we're concerned with. The Content Type option says, "Specify whether to allow the management of content types on this document library. Each content type will appear on the new button and can have a unique set of columns, workflows and other behaviors." This is exactly what we want.

3. Select Yes and click OK to save the settings (see [Figure 9-11](#)).



FIGURE 9-11. Toggle *Allow management of content types* to Yes

You'll then be redirected back to the document library settings screen. It will look different than before because of the additional Content Type section. This is where we want to focus our attention next.

4. Click the "Add from existing site content types link" (see [Figure 9-12](#)).

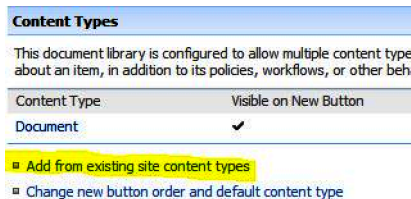


FIGURE 9-12. Click the "Add from existing site content types" link

You'll be shown a page that lists all the existing available content types at the current level of the site collection.

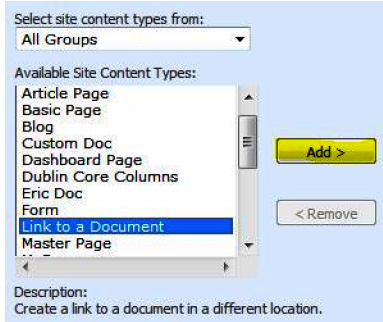


FIGURE 9-13. Highlight Link to a Document and click Add

5. Select the Link to a Document content type from the list and click Add, then click OK to save the changes (Figure 9-13).

After clicking OK, you'll be redirected to the Settings page for the document library. Now it's time to put our handy work into action.

6. Navigate back to the document library via the breadcrumbs. Click the down arrow on the New button. You'll now see the two content types available (Figure 9-14).

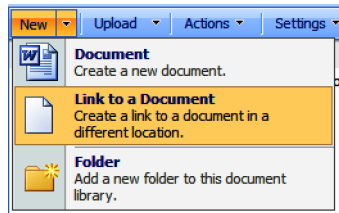


FIGURE 9-14. The new content type is available on the New menu

After adding some content into my library, I get a view that looks like the one shown in Figure 9-15.

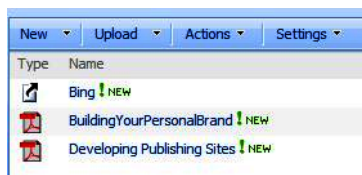


FIGURE 9-15. List view with Type field included

However, modifying the view and removing the Type icon from the display results in a seamlessly blended listing of content (Figure 9-16).

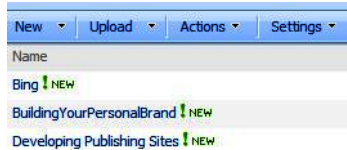


FIGURE 9-16. List view with Type field excluded

What the Content Type Does

The Link to a Document content type creates a small ASPX in the document library.

```
<%@ Assembly Name='Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c' %>
<%@ Register TagPrefix='SharePoint' Namespace='Microsoft.SharePoint.WebControls'
  Assembly='Microsoft.SharePoint' %>
<%@ Import Namespace='System.IO' %>
<%@ Import Namespace='Microsoft.SharePoint' %>
<%@ Import Namespace='Microsoft.SharePoint.Utilities' %>
<%@ Import Namespace='Microsoft.SharePoint.WebControls' %>
<html xmlns:mso="urn:schemas-microsoft-com:office:office" xmlns:msdt=
  "uuid:C2F41010-65B3-11d1-A29F-00AA00C14882">
<Head>
<META name="WebPartPageExpansion" content="full"><META Name='progid'
  Content='SharePoint.Link'>

<!--[if gtemso 9]><xml>
<mso:CustomDocumentProperties>
<mso:URLmsdt:dt="string">http://www.bing.com, http://www.bing.com</mso:URL>
<mso:ContentTypemsdt:dt="string">Link to a Document</mso:ContentType>
</mso:CustomDocumentProperties>
</xml><![endif]-->
</head><body>
<form id='Form1' runat='server'>
<SharePoint:UrlRedirector id='Redirector1' runat='server' />
</form>
</body>
</html>
```

This is simply creating a redirection behind the scenes. This redirection results in a 301 redirect, moved permanently, as interpreted by the browser when clicked. SharePoint uses the value stored in the Document Name field to use as a page name in the URL. Note that if you put a space in the Document Name field, it will result in a URL with encoded spaces (e.g., %20) in the filename. Keep this in mind if it's important to you to supply users with as tidy as possible a URL.

Extending the Link to a Document Content Type

Since the Link to a Document content type functions just like any other content type in SharePoint, it can be extended to include additional columns that other library content types might be using. This allows for a consistent UI design and allows for the Link to a Document's content to be displayed along with the rest of the content and filtered and sorted as necessary.

Adding Additional Fields in SharePoint 2010

Additional columns can be added to the content type. This section describes how to do so in SharePoint 2010.

1. Click Site Actions and select Site Settings.
2. From the Site Settings menu, click the Site Content Types link in the Galleries section (see [Figure 9-17](#)).



FIGURE 9-17. Click Site content types in Site Settings

3. Click the Create option to start the process of creating a new Link to a Document content type. Give the new content type a name and description. Select the parent category as Document Content Types and the parent as Link to a Document. You can create a new group if you'd like or just accept the default of Custom Content Types. When finished, click OK (see [Figure 9-18](#)).

When you click OK, you'll be shown the details page for that content type.

4. From here there are two options. If you already have site columns from existing content types, then click the "Add from existing site columns" link. If no pre-existing site columns exist, then click the "Add from new site column" link (see [Figure 9-19](#)).

Once all of the column(s) have been added to the content type, you can add it to the document library.

Adding Additional Fields in SharePoint 2010

Use this procedure to add columns to the content type in SharePoint 2010.

1. Click Site Actions and select Site Settings.

Name:

Description:

Parent Content Type:
 Select parent content type from:

Parent Content Type:

Description:
 Create a link to a document in a different location.

Put this site content type into:
 Existing group:

New group:

OK Cancel

FIGURE 9-18. Creating a new content type based on an existing content

Columns	
Name	Type
Name	File
URL	Hyperlink or Picture
PublishingDate	Date and Time

FIGURE 9-19. Content type after adding site columns to it

- In the Site Settings menu, click the Site Content Types link in the Galleries section (see [Figure 9-20](#)).

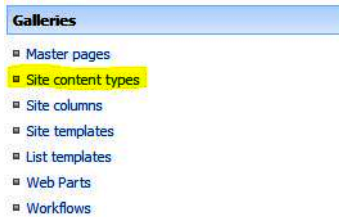


FIGURE 9-20. Click Site content types in Site Settings

3. Click the Create option to start the process of creating a new Link to a Document content type. Give the new content type a name and description. Select the parent category as Document Content Types and the parent as Link to a Document. You can create a new group if you'd like or just accept the default of Custom Content Types. When you are finished, click OK (see Figure 9-21).

A screenshot of a dialog box for creating a new content type. The dialog has a light blue background and contains the following fields and options:

- Name:** A text box containing 'Custom Link to a Document'.
- Description:** An empty text box with a vertical scrollbar.
- Parent Content Type:** A section with two dropdown menus. The first is labeled 'Select parent content type from:' and is set to 'Document Content Types'. The second is labeled 'Parent Content Type:' and is set to 'Link to a Document'.
- Description:** A text box containing the text 'Create a link to a document in a different location.'
- Put this site content type into:** A section with two radio buttons. The first is 'Existing group:' and is selected. Below it is a dropdown menu set to 'Custom Content Types'. The second is 'New group:' with an empty text box below it.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom right.

FIGURE 9-21. Creating a new content type based on an existing content

When you click OK, you'll be shown the details page for that content type.

4. From here, there are two options. If you already have site columns created from existing content types, click the "Add from existing site columns" link. If there are no pre-existing site columns, click the "Add from new site column" link (see Figure 9-22).

Columns	
Name	Type
Name	File
URL	Hyperlink or Picture
Publication Date	Date and Time

- ▣ Add from existing site columns
- ▣ Add from new site column
- ▣ Column order

FIGURE 9-22. Content type after adding site columns to it

Once all of the column(s) have been added to the content type, you can add it to the document library.

Summary

SharePoint gives users the ability to intermix content sources in document libraries by leveraging the Link to a Document content type. This seemingly little known out-of-the-box component allows site owners and designers to create concise, clean content libraries with ease. Owners and designers can leverage existing site columns and metadata by extending the base content type to suit the needs of the department or organization. This gives users the ability to sort and filter information in these content sources as needed to accomplish their day-to-day activities.

SharePoint 2010 Tab Page

Peter Allen

When the amount of content to be shown on a page exceeds the viewable real estate, users have to scroll down the page to view the content they wish to see. Scrolling is something to be avoided for an optimum user experience. Less scrolling means a much better experience for the user. SharePoint, however, does not provide a way to make this happen.

One of the better solutions to this problem in SharePoint is the use of tabs. Tabs allow the content manager to place more information on a single page while not forcing the user to scroll up and down the page to get to that information.

Wouldn't it be great to have a solution that allowed you to have a page that had tabs incorporated and ready for you to use?

The goal, then, is to create a Tab page that will do the following:

- Allow users to view content with little to no scrolling.
- Provide an option that has two to eight tabs.
- Enable the use of cookies so that tabs become *sticky*, so when returning to the tab page, users will land on the last tab they were on.
- Allow for multiple web parts to be viewed in one tab.

What, then, will the end product look like? Well, it will look something like the pages shown in [Figure 10-1](#) and [Figure 10-2](#).

1. Now edit the Content Editor Web Part (CEWP)
2. At this point you will then want to open up the Source Editor and past in the code below
3. Now you will need to decide the number of tabs you want.
 - To have less then 8 tabs I recommend that you do the following, add <!-- & --> to the start of this code and the end of this code
 - This will allow you to easily add a tab back if you need it in the future with out having to remember the code to add
 - Example: <!--TAB 8-->
4. Next you will want to change the title of the tabs.
 - The title is found between the code
5. Last do you want the [Tab Page Help](#) to show or NOT show
 - Add the <script> code to HIDE the help button
6. Below is the code to add to the CEWP

```

<!-- Code to hide Tab Page Help Link -->
<script type="text/javascript">
  $(document).ready(function() {
    $('#TabPageHelp').hide();
  });
</script>

<!-- Code to add Tabs -->
<div id="tabs">
  <ul class="tabNavigation ms-WPBody">
    <li><a href="#tab-1" class="selected ms-topnavselected"> <span>TAB 1</span></a></li>
    <li><a href="#tab-2" > <span>TAB 2</span></a></li>
    <li><a href="#tab-3" > <span>TAB 3</span></a></li>
    <li><a href="#tab-4" > <span>TAB 4</span></a></li>
    <li><a href="#tab-5" > <span>TAB 5</span></a></li>
    <li><a href="#tab-6" > <span>TAB 6</span></a></li>
    <li><a href="#tab-7" > <span>TAB 7</span></a></li>
    <li><a href="#tab-8" > <span>TAB 8</span></a></li>
  </ul>
  <div style="clear: both"></div>
</div>

```

FIGURE 10-1. SharePoint 2010 Tabs page

You will find that each tab has a header, left, middle, right, and footer section to add web parts. This provides a lot of customization for each tab, as the layout of your web parts provides flexibility in configuration and placement.

Implementation

Now let's implement the solution. Here is a brief overview of the three basic steps we will carry out to create a tab page:

1. Download the page and load it into your SharePoint 2010 instance.
2. Add a web part to the tab page, selecting the HTML Forms Web Part.
3. Add the code to the HTML Forms Web Part to customize your tabs.

Once these steps are completed, you will be able to add content to your new tab page. Now onto the details!

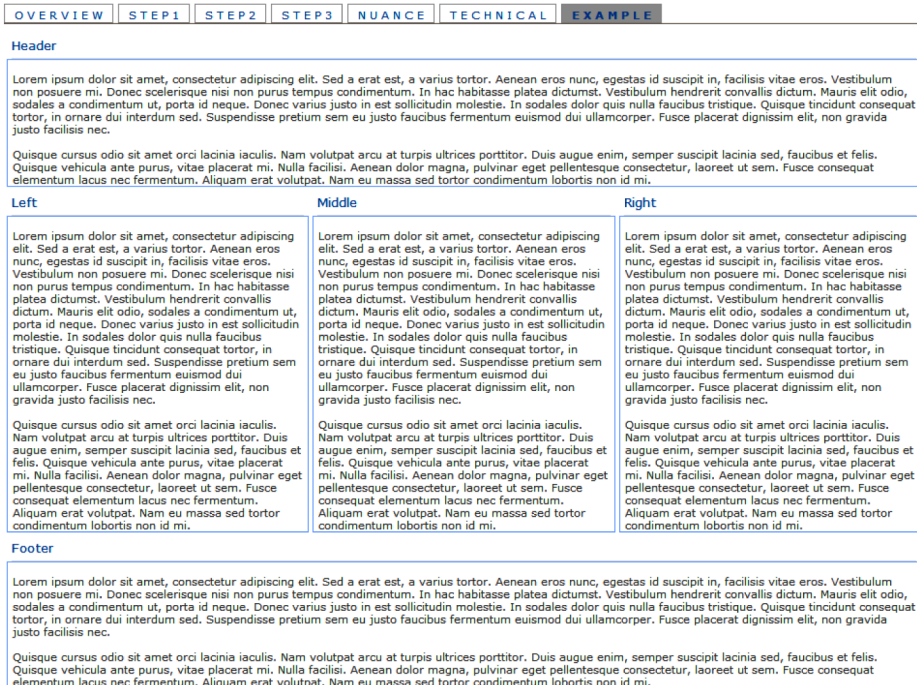


FIGURE 10-2. SharePoint 2010 Tab page example showing web part placement

NOTE

All of the code needed to implement this solution can be found at the following link:

examples.oreilly.com/9781449321000/

Step 1: Download and Load the Page

The first thing that you will need to do is identify the document/page library to house the tab page. This page can be added to any document library, so it gives you flexibility as to where you deploy it. You also have the option of loading it into your page layout library on your root site so that it is available to all sites within your site collection. For the purposes of this discussion, we will not be going through the process of adding it to the page layouts library.

Next, you will need to either enter the code at the end of this chapter into a blank *.aspx* page. Extract the contents of the *.zip* file and upload to a document or page library of your choosing.

Step 2: Add the Web Part

Open the page you have chosen to be your Tabs page. Access the Site Actions and select Edit Page. Now look for the section called Tab (shown in [Figure 10-3](#)).

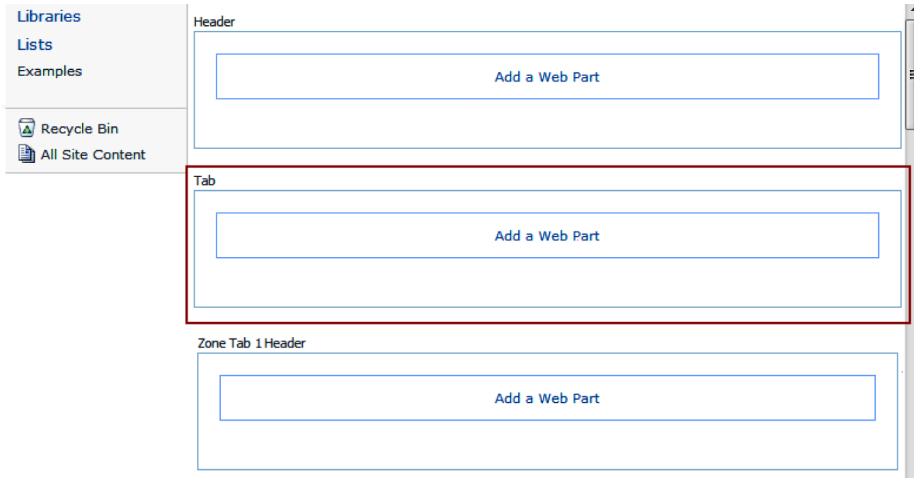


FIGURE 10-3. SharePoint 2010 Tab page location for HTML Form web part

On this page, click Add a Web Part. On the screen that pops up, [Figure 10-4](#), select Forms in the Categories list and then select the option HTML Form Web Part. Click Add.

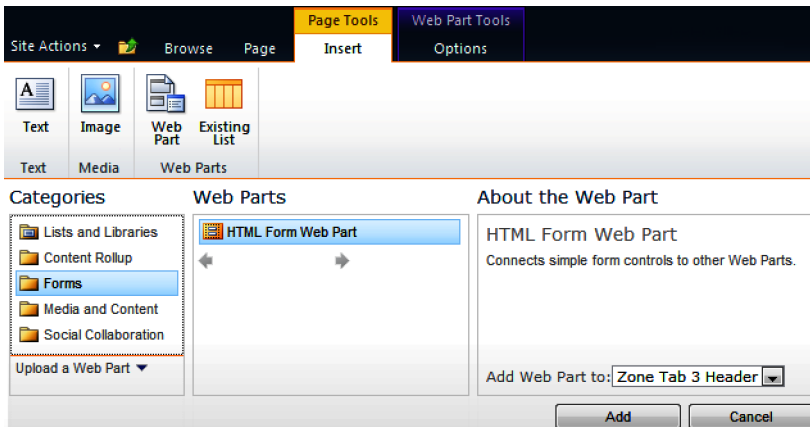


FIGURE 10-4. HTML Form web part

NOTE

Why are we using an HTML Form web part rather than the Content Editor web part for adding code? In SharePoint 2007, the Content Editor web part is the web part of choice when adding code to a page. However, SharePoint 2010 has changed the way the Content Editor web part works; it is much easier to edit content, but it will strip out code for no apparent reason. So, the web part to use for code is the HTML Forms web part, as it does not evaluate your code.

Step 3: Customize the Tabs

Now you can edit the HTML Forms web part. Open the Source Editor and enter the following code:

```
<!-- Code to add Tabs -->
<div id="tabs">
<ul class="tabNavigation ms-WPBody">
<li><a href="#tab-1" class="selected ms-topnavselected"><span>TAB 1</span></a></li>
<li><a href="#tab-2" ><span>TAB 2</span></a></li>
<li><a href="#tab-3" ><span>TAB 3</span></a></li>
<li><a href="#tab-4" ><span>TAB 4</span></a></li>
<li><a href="#tab-5" ><span>TAB 5</span></a></li>
<li><a href="#tab-6" ><span>TAB 6</span></a></li>
<li><a href="#tab-7" ><span>TAB 7</span></a></li>
<li><a href="#tab-8" ><span>TAB 8</span></a></li>
</ul>
<div style="clear: both"></div>
</div>
```

You will need to decide the number of tabs you want. To have fewer than eight tabs, I recommend that you add `<!--& -->` to the beginning and end of the lines of code that you use to denote the tabs (those that start with `` and end with ``). This will allow you easily to add a tab back if you need it in the future, without having to remember the code you are adding.

```
<!--<li><a href="#tab-8" ><span>TAB 8</span></a></li>-->
```

You can give the tabs different names by inserting the name for the tab between the code `` and `` for each tab:

```
<li><a href="#tab-2" ><span>TAB NAME</span></a></li>
```

Finally, decide if you want the Tab page Help link to be shown. If you decide that the Help link should be hidden, add the following code:

```
<!-- Code to hide Tab Page Help Link -->
<script type="text/javascript">
  $(document).ready(function() {
    $('#TabPageHelp').hide();
  });
</script>
```


NOTE

When you add a web part to any of the tabs, you will notice that each time the page refreshes, it takes you back to Tab 1. This will happen each time, but it does not affect the web part you are working on. When you click **Modify Shared web part** on Tab 3 and it refreshes to Tab 1, it does *not* mean you are editing the web part in Tab 1. You will notice that you can click Tab 3 and you will see that the web part has the dotted lines that indicate it is being edited.

jQuery Implementation

This solution uses jQuery and so, of course, jQuery must be loaded. The code calls a remote version of jQuery if jQuery has not already been loaded. This ensures that the site where this solution is implemented has jQuery loaded so that the Tab page will work. If your site collection already has jQuery loading through the master page, the script on the Tab page will check to see if jQuery is loaded. If it already has been loaded, the script will not load another version or copy. The following code snippet evaluates the status of jQuery on your site:

```
<script type="text/javascript">
  if(typeof jQuery=="undefined"){
    var jqPath="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/";
    document.write("<script src='"+jqPath,"jquery.min.js' type='text/javascript'>
                  </script>");
  }
</script>
```

Tab Page Layout Code

The following URL hosts the complete code for the page layout of this tab solution. There is standard SharePoint page layout code, a script to render and remember the tab you were last on, and CSS to render the tabs.

For the complete code and latest updates to this solution, visit examples.oreilly.com/9781449321000/.

Summary

Now that you have implemented your Tab page, you can begin to add your web parts and content as needed to each page. Be careful about how many web parts per page you add that reference lists and libraries. The more web parts you add of this type, the longer it will take to render your tab page. Even if users do not click any of the tabs, the page loads all the web parts at once, so performance can become an issue.

A Global Navigation Solution Across Site Collections

Peter Allen

Have you ever wanted an easier way to have global navigation that spans across site collections? SharePoint provides an elegant solution for global navigation within a site collection. However, when you architect a SharePoint instance to have multiple web apps and site collections, a global navigation system is not an easy solution to create. A global navigation system would need to be created on each site collection, and any changes to the global navigation need to be made in each and every site collection ...ouch!!! You would think there needs to be a better solution—and there is!

In this chapter, we will explore one possible solution that allows us to have the same global navigation across site collections and the ability to manage it from a central location.

The goal of this solution is to create a global navigation solution that will be capable of the following:

- Deploying once and centrally adding, editing, deleting, organizing, and managing tabs and their associated pull-down menus
- Controlling the order of tabs and the placement at the beginning or end of the Top Navigation bar
- Adding functionality to have pull-downs with our tabs (SharePoint 2010 Foundation does not provide this functionality)
- Allowing pull-down menu items to be pulled from a standard SharePoint Links list

- Ordering and grouping pull-down menu items
- Creating tabs or pull-down grouping headers can be links—or not
- Supporting themes for global navigation

Our end product will look similar to the page shown in [Figure 11-1](#).

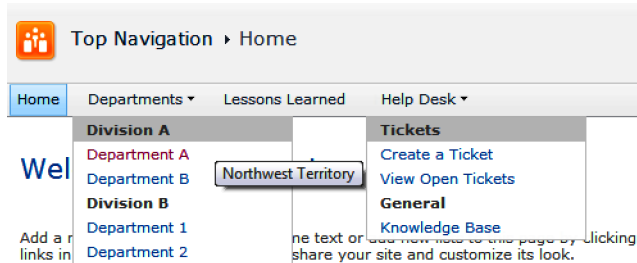


FIGURE 11-1. SharePoint 2010 global navigation features: tab pull-down arrow, pull-down lists, and pop-ups

Imagine that—one place that contains all your libraries, lists, and folders. [Figure 11-2](#) shows the central location for managing tabs and what their pull-downs look like. Notice that we have the two lists for tabs and pull-downs.

Implementation

Now let's implement the solution. As you make your way through this solution you will:

- Add lists and libraries
- Add code to your master page
- Change code in the script file
- Add your tabs and pull-downs to the lists

To execute this solution, you will need a number of components. Although this list may look like a lot, the solution is very simple:

- SharePoint Resource site
- SPServices by Marc Anderson
- jQuery
- Global Navigation Solution files
- SharePoint Designer
- SharePoint Links list for tabs and pull-downs

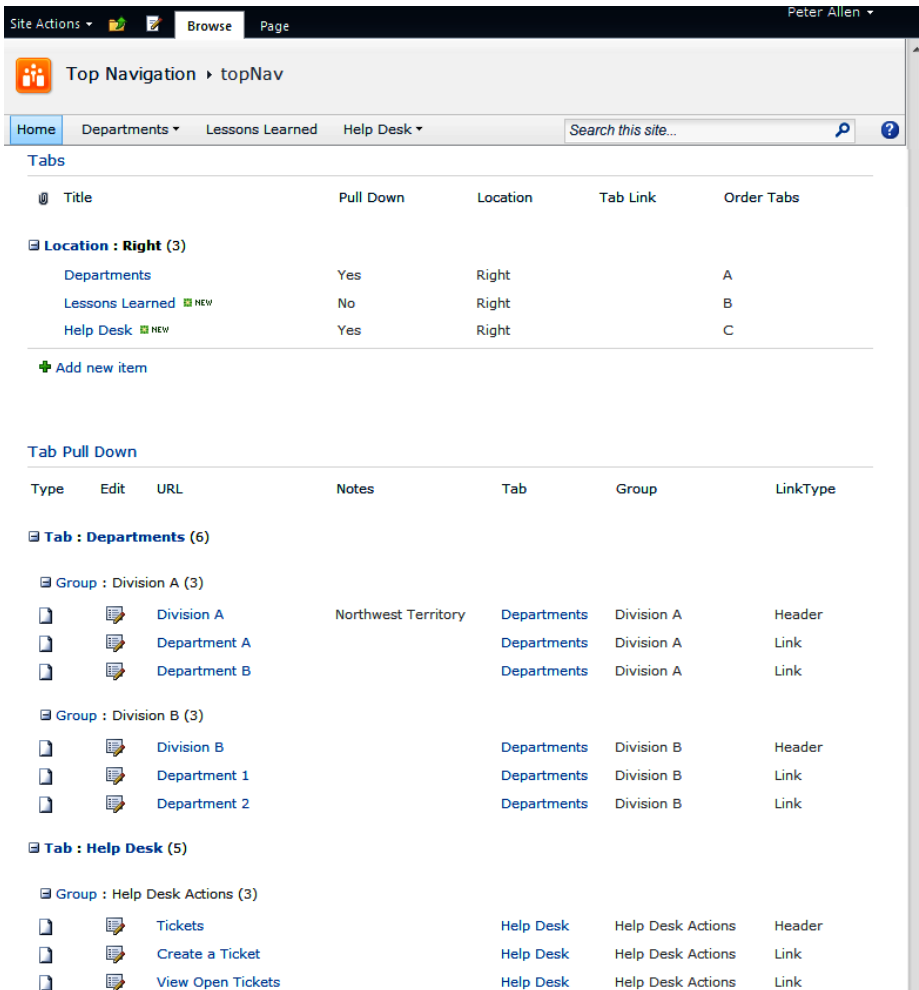


FIGURE 11-2. Global navigation tabs and tab pull-down lists

- Solution code
- Solution CSS

Now to tell you a bit more about each of these pieces and why they are essential to this solution as you work your way through the steps to make this a reality!

SharePoint Resource Site

The first component we need is a central site where all of the files and lists are to be housed to implement this solution. Accessible by everyone, this site will need to be set up with read access

only. Setting up this site allows you to have a document library where you can house your scripts for jQuery and SPServices. It will also allow you to create the required lists including the tab and pull-down lists.

For purposes of this implementation, we will create a site collection and name it Resources. You can find an article related to this very topic at [BitsOfSharePoint](#).

SPServices by Marc Anderson

Next, we will need to load SPServices to our Resource site. Marc Anderson created SPServices, which is a jQuery library for SharePoint web services. You can find it at <http://spservices.codeplex.com/>.

The script for this solution will use SPServices, so you need to add the file to the aforementioned document library where you are keeping scripts. In our case, it will go into the document library in the Resources site collection.

I recommend that you change the name of the file so that it does not have a version number; it should look something like this: *jquery.SPServices-latest.min.js*. This naming convention will allow you to update to the latest version of SPServices without having to go back to each master page and change to the new version name.

jQuery

We will need to load jQuery, as this is the glue that makes this solution adhere. Thanks to the folks at jQuery for developing a feature-rich scripting solution, there are two ways to work with a jQuery file. You have a choice of downloading the latest version and putting it in scripts document library, or you can point to Google's version, which will always be the latest version. If you do not already have it, you can download jQuery from <http://jquery.com/>.

You should then add this script to the document library in your Resources site collection.

NOTE

As of this writing, only jQuery 1.6.4 works with this solution. Download version 1.6.4 only.

Global Navigation Solution Files

Two additional files are needed to implement this solution. You will need the *topnavigation.js* and *topnavigation.css* files. The *topnavigation.js* file houses the JavaScript and jQuery code for this solution, and *topnavigation.css* houses the CSS to render the pull-downs correctly. You can download the latest version at examples.oreilly.com/9781449321000/.

SharePoint Designer

You will need to have access to SharePoint Designer and be able to edit the master pages on the sites where you want the global navigation solution to appear. When you edit a master page, add the following code between the <header> and </header> tags. This code shows a typical master page Header section, with the addition of the code for this solution, which you add at the end of the Header section. The new code is shown in bold.

```
<head runat="server">
  <meta http-equiv="X-UA-Compatible" content="IE=8"/>
  <meta name="GENERATOR" content="Microsoft SharePoint"/>
  <meta name="progid" content="SharePoint.WebPartPage.Document"/>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <meta http-equiv="Expires" content="0"/>
  <SharePoint:RobotsMetaTag runat="server"/>
  <title id="onetidTitle"><asp:ContentPlaceHolder id="PlaceHolderPageTitle"
    runat="server"/>
  </title>
  <SharePoint:CssLink runat="server" Version="4"/>
  <SharePoint:Theme runat="server"/>
  <SharePoint:ULSClientConfig runat="server"/>
  <script type="text/javascript">
    var _fv4UI = true;
  </script>
  <SharePoint:ScriptLink language="javascript" name="core.js" OnDemand="true"
    runat="server"/>
  <SharePoint:CustomJSUrl runat="server"/>
  <SharePoint:SoapDiscoveryLink runat="server"/>
  <asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server"/>
  <SharePoint:DelegateControl runat="server" ControlId="AdditionalPageHead"
    AllowMultipleControls="true"/>
  <SharePoint:SPShortcutIcon runat="server" IconUrl="/_layouts/images/
    favicon.ico"/>
  <asp:ContentPlaceHolder id="PlaceHolderBodyAreaClass" runat="server"/>
  <asp:ContentPlaceHolder id="PlaceHolderTitleAreaClass" runat="server"/>
  <SharePoint:SPPageManager runat="server"/>
  <SharePoint:SPHelpPageComponent Visible="false" runat="server"/>
  <link type="text/css" rel="stylesheet" href="http://www.bitsofsharepoint2010.com/
    SamplePoint/TopNavigation/SiteAssets/topnavigation.css" />
  <script type="text/javascript" src="http://www.bitsofsharepoint2010.com/SamplePoint/
    TopNavigation/SiteAssets/jquery-1.6.2.min.js"></script>
  <script type="text/javascript" src="http://www.bitsofsharepoint2010.com/SamplePoint/
    TopNavigation/SiteAssets/jquery.SPServices-0.6.2.min.js"></script>
  <script type="text/javascript" src="http://www.bitsofsharepoint2010.com/SamplePoint/
    TopNavigation/SiteAssets/topnavigation.js"></script>
</head>
```

You will also need to add the following code to each master page. Make sure the URLs point to where you have the files housed. They do need to be absolute URLs, as you will be adding this code across site collections, web apps, and even servers.

```
<link type="text/css" rel="stylesheet" href="http://spf-dev/Navigation/Scripts/
  topnavigation.css" />
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.js"></script>
```

```

<script type="text/javascript" src="http://yoursite/SiteCollection/LibraryName/
jquery.SPServices-latest.min.js"></script>
<script type="text/javascript" src="http://yoursite/SiteCollection/LibraryName/
topnavigation.js"></script>

```

For example, if we put *jquery-latest.js*, *jquery.SPServices-latest.min.js*, and *topnavigation.js* in the Resources site collection in the document library called ScriptsLibrary, it will look like this:

```

<link type="text/css" rel="stylesheet" href="http://spf-dev/Navigation/Scripts/
topnavigation.css" />
<script type="text/javascript" src="http://www.bitsofsharepoint.com/Resources/
ScriptsLibrary/jquery-latest.js"></script>
<script type="text/javascript" src="http://www.bitsofsharepoint.com/Resources/
ScriptsLibrary/jquery.SPServices-latest.min.js"></script>
<script type="text/javascript" src="http://www.bitsofsharepoint.com/Resources/
ScriptsLibrary/topnavigation.js"></script>

```

SharePoint Links List for Tabs and Pull-Downs

Now it is time to create the two lists where the tabs and their related pull-downs will be housed. By having these two lists, you can go to one location and make changes instantly to your global navigation. This makes it much easier to maintain and deploy. In the future, we need only to go to the Tabs list to add another tab. In the tab pull-down list, we can associate a pull-down with any of the tabs. This will make it very easy to switch a pull-down from one tab to another if needed.

Tabs list

First let's create the Tabs list. This will be a custom list and for this solution to work, it must be called Tabs. Below are the columns that must be added and have the exact same name as follows:

Title

This is the name of the tab. This column name does not need to be changed.

Pull-Down

This is a Yes/No field. When enabled, it will be a pull-down; when disabled, it will not be a pull-down and will not have the associated triangle. Deselect the box to indicate that the tab will *not* have a pull-down.

Location

This is a choice field with the options Left and Right. This specifies the location of the tab relative to the default tabs. Select either option here.

Tab Link

This is a text field for specifying the URL for the tab. Enter a URL to give the tab a link, otherwise leave blank for no link. If you add a link, ensure that it starts with *http://*.

Order Tabs

This is a choice field for ordering tabs on the left or right. Make a selection from A to K. Right orders in ascending manner and left in descending.

Next let's change the view called All Items in order to sort and group appropriately. In the Tabs list, look for the view named All Items, just above SharePoint's top navigation in the site breadcrumb. This view will dictate the order in which the tabs show up.

On the All Items view, select Modify View. For Sort, select Order Tab in ascending order. For Group By, select Location in ascending order and set to Expand.

Tab pull-down list

Next, let's add the Tab Pull-Down list. This list will be a standard links list and must be called Tab Pull Down. Add the following additional columns (for this solution to work, you must use the exact column names provided here):

Tab

This is a lookup column containing a list of items in the Tabs list (listed by Title).

Group

This column will be used to group the links together. So, for example, all Help links will be grouped together and all Features links will be grouped together. This can be a text field, or, if you know the names of all the groups, it can be a choice field.

LinkType

This column identifies is the type of link. It is a choice column with three choice types that each display the link in a different way:

- Link displays the item as a link within the group.
- Header displays the item as a header without the link.
- Header Link displays the item as a header with a link (Header and Header Link look the same but act differently).

Next, change the default All Links view as follows:

Sort

All users to order items in this view, set to No. Sort by LinkType in ascending order. Sort by URL in ascending order.

Group By

Set Group by to Tab in ascending order. Set Group by to Group in ascending order. Show grouping as Expanded.

This specifies how the items will be displayed in the pull-down for the global navigation. You can play with the view here to sort and group it as you wish.

Solution Code

In the following code or in the *topnavigation.js* file, you will need to change the following:

```
listSiteTabURL = 'http://www.bitsofsharepoint.com/topnav/';
```

You will find this code at the top. Change `http://www.bitsofsharepoint.com/topnav/` to the site on which your tab and pull-down lists are located. This is the URL of the site location, *not* the list location.

Incorrect example:

```
'http://www.intranet.com/SiteName/List/ListName/'
```

Correct example:

```
'http://www.intranet.com/SiteName/'
```

Once you make that change, you can save the file in your scripts library, and you are done with that file.

Here is the code to add to the *topnavigation.js* file:

```
$(document).ready(function(){
    listSiteTabURL = 'http://spf-dev/Navigation/';

    $.SPServices({
        operation: "GetListItems",
        webURL: listSiteTabURL,
        listName: 'Tabs',
        async: false,
        completefunc: function (xData, Status) {
            $(xData.responseXML).find("[nodeName='z:row']").each(function() {

                var tabSide = $(this).attr("ows_Location");
                var titleTabName = $(this).attr("ows_Title");
                var urlTab = $(this).attr("ows_Tab_x0020_Link");
                var tabPullDown = $(this).attr("ows_Pull_x0020_Down");
                var listName = 'Tab Pull Down';
                var listSiteURL = listSiteTabURL;
                var titleTabID = 'topNav' + titleTabName.replace(/[-' ']/g, '');
                var showTabPD = titleTabID + 'Show';
                var showTabAppend = titleTabID + 'Append';

                if (urlTab != null) {
                    var urlTabShow = 'href=' + urlTab;
                } else {
                    var urlTabShow = '';
                }

                if (tabPullDown == true) {
                    var pulldownClass = 'dynamic-children';
                    var pulldown = '<ul class="dynamic ' + showTabAppend + '" id="">
</ul>';
                } else {
                    var pulldownClass = '';
                }
            });
        }
    });
});
```

```

    var pulldown = '';
}

var tableRow = '<ul class="dd_menu root static"><li class="static ' +
pulldownClass + '"><a ' + urlTabShow + ' class="static ' + pulldownClass
+'menu-item"><span class="additional-background"><span class="menu-
item-text">' + titleTabName + '</span></span></a>' + pulldown + '</li>
</ul>';

if (tabSide == 'Left') {
    $('div.s4-toplinks').children().children().children().prepend
    (tableRow);
} else {
    $('div.s4-toplinks').children().children().children().append
    (tableRow);
}

$.SPServices({
    operation: "GetListItems",
    webURL: listSiteURL,
    listName: listName,
    async: false,
    completefunc: function (xData, Status) {
        $(xData.responseXML).find("[nodeName='z:row']").each
        (function() {
            var URLdataTN1 = $(this).attr("ows_URL").split(',');
            var linksTN1 = $(this).attr("ows_LinkType");
            var TabTN1 = $(this).attr("ows_Tab").split('#');
            var TabC = TabTN1[1];
            var notesVar = $(this).attr("ows_Comments");

            if (notesVar == null) {
                var notes = '';
            } else {
                var notes = 'title="' + notesVar + '"';
            }

            if (titleTabName == TabC) {
                if (linksTN1 == 'Header') {
                    $('ul.' + showTabAppend).append('<li class=
                    "dynamic"><a class="dynamic menu-item" ' +
                    notes + '><span class="additional-background">
                    <span class="menu-item-text"><strong>' +
                    URLdataTN1[1] + '</strong></span></span></a>
                    </li>');
                } else if (linksTN1 == 'Header Link') {
                    $('ul.' + showTabAppend).append('<li class=
                    "dynamic"><a href="' + URLdataTN1[0] + '" class=
                    "dynamic menu-item" ' + notes + '><span class=
                    "additional-background"><span class="menu-item-
                    text"><strong>' + URLdataTN1[1] + '</strong>
                    </span></span></a></li>');
                } else {
                    $('ul.' + showTabAppend).append('<li class="dynamic">
                    <a href="' + URLdataTN1[0] + '" class="dynamic

```



```
.dd_menu li:hover ul {
    left:0px;
    top:25px;
}

.dd_menu li:hover ul li {
    border:none;
}

.dd_menu li:hover ul li a {
    display:block;
}

.dd_menu li:hover ul li a:hover {
}
```

Summary

This solution allows you to create the global navigation from two lists. This solution continues to evolve, and you can find the latest updates at examples.oreilly.com/9781449321000/.

A

AND function, 171, 200–211

applications

- SharePoint Maturity Model, 6–11, 8
 - defining and standardizing, 9
 - evaluating, 11
 - lists and navigation links, 9
 - monitoring and maintaining, 10
 - personalization, 11
 - publication, 7

assets, content types in OneNote, 29

automating an all-day event with jQuery, 46

B

benefits of the SharePoint Maturity Model, 6

branched versus unbranched surveys, 55

branding, level 200—lists and navigation links, 9

business process competency, 4

C

Calendar lists, 46

CAML

- charts, 164
- profile property, 165

Cascading Style Sheets (CSS), 240

case numbers, getting in jQuery, 49

central repository, 17

CEWP (Content Editor web part), 124, 130

- content links, 157
- jQuery, 59

charts, SPJS Charts for SharePoint, 168

client-side scripts, 38

collaboration competency, 4

columns

- charts, 162
- lists, 120

composites and applications competency, 4

configuration list, SPJS Charts for SharePoint, 158

configuration management

about, 16

- maintenance, 41

configuring

- chart options, 161
- document libraries, 214–220

Content Editor Web Part (see CEWP)

content types

- about, 19, 220–224
- OneNote, 29
- templates in OneNote, 36

core competencies, 3

Core SharePoint competencies, 3

CSS (Cascading Style Sheets), 240

customizations competency, 5

D

Data View Web Part (see DVWP)

Data View Web Parts/Forms, content types, 37

datafields section, SharePoint Data View Web Part, 79

DataSources section, SharePoint Data View Web Part, 76

default text, jQuery, 52–55

Designer (see SharePoint Designer)

document libraries, 213–224

- about, viii
- background, 213
- configuring, 214–220
- content type, 220–224
- Data View web part, 107–113
- hyperlinks, 92
- shared document libraries, 158
- URLs, 88

documentation

- client-side scripts, 38
- folders, 21
- libraries, 20
- need for, 16
- process of, 17
- scripts, 32

We'd like to hear your suggestions for improving our indexes. Send email to index@oreilly.com.

- templates in OneNote, 36
 - who, what, where, why and when, 34
- downloading Tab pages, 227
- DVWP (Data View web part), 71–83
 - about, vii
 - hyperlinks, 85–115
 - creating, 95
 - URLs, 88
 - walkthroughs, 86
 - XSLT, 90–95

E

- Easy Tabs, 38
- edit chart GUI, SPJS Charts for SharePoint, 159–167
- EditForm.aspx, 55, 59, 62
- empowerment
 - and SharePoint Designer lockdowns, 14
 - configuration management, 16
 - responsibility, 42
- EUSP (EndUserSharePoint.com), v
- evaluating
 - publication maturity using the SharePoint Maturity Model, 11

F

- FALSE function, 172
- fear, and SharePoint Designer lockdowns, 15
- feedback, level 400—monitoring and maintaining, 10
- fields
 - adding to SharePoint 2007, 221
 - adding to SharePoint 2010, 221
- folders
 - libraries, 33
 - OneNote, 33
 - SharePoint Rudder, 21
- formats, charts, 162
- forms
 - Data View Web Parts/forms, 37
 - jQuery, 57
 - labeled section on default forms, 63–67
 - multiple, 214
- functions (see logical functions)

G

- global navigation, 231–241
 - CSS, 240
 - jQuery, 234
 - links list for tabs and pull-downs, 236
 - SharePoint Designer, 235
 - SharePoint resource site, 233
 - solution code, 238
 - solution files, 234

- SPServices, 234
- Google Visualization API, 155
- governance, platform legacy, 16
- GUIs, edit chart GUI for SPJS Charts for SharePoint, 159–167

H

- hyperlinks, 85–115
 - SharePoint Data View Web Part
 - about, vii
 - SharePoint Data View web part
 - creating, 95
 - modal dialog box, 113
 - moving, 105
 - SharePoint List, 96–105
 - URLs, 88
 - walkthroughs, 86
 - XSLT, 90–95

I

- IDs (see survey IDs)
- IF function, 171, 172–200
- images, content types in OneNote, 31
- infrastructure and administration competency, 5
- Initiation Form workflow, 40
- insight competency, 4
- integration competency, 4
- items, URLs, 89

J

- JavaScript, 45
- jQuery, vii, 45–67
 - automating an all-day event, 46
 - CEWP, 59
 - default text, 52–55
 - forms, 57
 - global navigation, 234
 - labeled section on default forms, 63–67
 - local copy, 157
 - requesting a review only once per user, 48–52
 - SharePoint Web Services, 57
 - Tab pages, 230
 - writing survey IDs to lists on response creation, 55–63

L

- labeled section on default forms, 63–67
- layout code, Tab pages, 230
- legacy, platform legacy, 16
- level 200—lists and navigation links, 9
- level 300—defining and standardizing, 9
- level 400—monitoring and maintaining, 10
- level 500—planning for personalization, 11

- libraries, 213–224
 - adding to jQuery, 49
 - document libraries
 - background, 213
 - configuring, 214–220
 - content type, 220–224
 - SharePoint Data View web part, 107–113
 - URLs, 88
 - XSLT List View web part hyperlinks, 92
 - folders, 33
 - shared document libraries, 158
 - SharePoint Rudder library, 19, 22
- links
 - changing survey links in jQuery, 50
 - document content type, 221
 - navigation links in SharePoint Maturity Model, 9
 - surveys, 57
 - type of, 237
- List View web part, 166
- lists
 - Calendar lists, 46
 - charts, 161
 - columns, 120
 - configuration list for SPJS Charts for SharePoint, 158
 - links list for tabs and pull-downs, 236
 - Quote of the Day Web Part list, 118
 - Quote of the Day web part list, 135
 - SharePoint List, 90
 - SharePoint Maturity Model, 9
 - URLs, 88
- logical functions, 171–211
 - about, viii
 - IF function, 172–200
 - list of, 171
 - OR and AND functions, 200–211

M

- maintenance
 - applications using the SharePoint Maturity Model, 10
 - configuration management, 41
- maturity (see SharePoint Maturity Model)
- metadata
 - level 300—defining and standardizing, 10
 - libraries OneNote, 20
 - OneNote, 30
- Microsoft Office suite, OneNote, 28
- modal dialog box, 113
- model (see SharePoint Maturity Model)
- monitoring applications using the SharePoint Maturity Model, 10
- MOSS 2007/WSS 3.0, configuring document libraries, 217

N

- navigation links, SharePoint Maturity Model, 9
- nesting IF statements, 179
- NewForm.aspx, 55, 57, 61
- NOT function, 172
- notebooks, OneNote, 28

O

- Office Web Apps, 111
- OneNote, 27–34
 - about, vii
 - content types, 29
 - documentation, 39
 - folders, 26, 33
 - forms, 37
 - linking, 39
 - notebooks, 28
 - SharePoint Rudder, 17
 - templates, 35–40
 - documentation, 36
 - SharePoint Rudder workflow, 39
 - workflow, 37
- OR function, 172, 200–211

P

- parameterbindings section, SharePoint Data View Web Part, 80
- people and communities competency, 4
- personalization, 11
- photos, 31
- platform legacy, SharePoint Designer, 16
- products and services, benefits of the SharePoint Maturity Model, 7
- profile property, charts, 165
- project triage, benefits of the SharePoint Maturity Model, 6
- publication
 - competency, 4
 - SharePoint Maturity Model, 7
- pull-down lists, 237

Q

- Quote of the Day web part, 117–153
 - building, 118–130
 - redistributing, 130–152

R

- radio buttons, jQuery, 52–55
- redistributing Quote of the Day web part, 130–152
- responsibility, empowerment, 42
- reviews

- requesting a review only once per user using jQuery, 48–52
- risk assessments, benefits of the SharePoint Maturity Model, 7
- Rudder (see SharePoint Rudder)

S

- Sandboxed Solutions, 131
- scripts
 - content types in OneNote, 31
 - templates in OneNote, 38
- search competency, 4
- server/site restore, SharePoint Designer, 15
- shared document libraries, 158
- SharePoint Data View Web Part, XSL tags, 69–83
- SharePoint Designer, 13–16
 - fear, 15
 - global navigation, 235
 - platform legacy, 16
 - power users, 14
 - server/site restore, 15
- SharePoint List
 - hyperlinks, 90
 - SharePoint Data View web part hyperlinks, 96–105
- SharePoint Maturity Model, 1–12
 - about, 2
 - applications, 6–11, 8
 - defining and standardizing, 9
 - evaluating, 11
 - lists and navigation links, 9
 - monitoring and maintaining, 10
 - personalization, 11
 - publication, 7
 - levels, 3
 - structure, 3
- SharePoint resource site, 233
- SharePoint Rudder, 17–26
 - clean up, 23
 - content types, 19
 - folders, 21
 - getting started, 18
 - putting it together, 22
- SharePoint Rudder library, 19, 22
- SharePoint Web Services, 57
- silos of knowledge, 16
- social networking, level 400—monitoring and maintaining, 10
- SPJS Charts for SharePoint, 155–169
 - charts in one page, 168
 - edit chart GUI, 159–167
 - setup, 157
 - technical overview, 155
 - web part templates, 167
- SPServices, 234

- staffing
 - benefits of the SharePoint Maturity Model, 7
 - and training competency, 5
- staging environments, SharePoint Designer, 15
- standardizing, SharePoint Maturity Model, 9
- survey IDs
 - getting the last one in jQuery, 50
 - writing to lists on response creation, 55–63
- survey links
 - changing in jQuery, 50
 - creating, 57

T

- Tab pages, 225–230
 - implementation, 226–230
 - jQuery, 230
 - layout code, 230
- Tab lists, 236
- templates
 - OneNote, 35–40
 - web part templates for SPJS Charts for SharePoint, 167
- testing scripts, 32
- training
 - basic needs, 42
 - benefits of the SharePoint Maturity Model, 7
- transformative projects, level 500—planning for personalization, 11
- triage, projects, 6
- TRUE function, 172
- truth tables, 184
- types, content types, 19

U

- unbranched versus branched surveys, 55
- URLs
 - filters, 166
 - OneNote, 28
- users
 - requesting a review only once per user using jQuery, 48–52

V

- views
 - charts, 161
 - URLs, 89

W

- web parts
 - Tab pages, 228
 - templates: SPJS Charts for SharePoint, 167
- Web Services, SharePoint, 57
- web, charts, 161

“who, what, when, where and why” in
documentation, 34

workflow

on surveys, 56

SharePoint Rudder, 39

templates in OneNote, 36

X

XSL tags, SharePoint Data View Web Part, 69–83

XSLT List View web part hyperlinks, 90–95

Z

zero values, in charts, 164

ABOUT THE AUTHORS

Mark Miller is founder and editor of EndUserSharePoint.com, and founding member of NothingbutSharePoint.com. He is currently Senior Storyteller and Community Advocate at FPWeb.net. His main expertise is in developing and building live online communities built around specific market verticals. He also speaks extensively on the SharePoint circuit.

Kerri Abraham was first introduced to SharePoint as a healthcare information worker. Passionate about organizing information and educating staff, the platform proved to be a great fit for her skill set. Promoted to a dedicated SharePoint support role in 2008, she won end-user adoption and accolades from management all with out-of-the-box SharePoint tools. However, when Kerri recently announced she was leaving SharePoint, the lack of supporting documentation was a very real threat to these now business critical resources. Her “Empower the Power User” message goes beyond the fight for access to tools; it is also a farewell warning to not underestimating the contributions of a power user.

Eric Alexander is a technology consultant working in the higher education sector. SharePoint is not his only responsibility, but it is his most passionate. He wears many hats in his SharePoint role: administration, training, consulting, and light development. He enjoys leveraging SharePoint to build solutions to solve problems across campus using out-of-the-box features and light development, utilizing data view web parts, and jQuery when needed.

Peter Allen has worked in the technology field for 15 years creating and deploying solutions in healthcare, financial, construction, municipalities, telecom, and engineering firms. He provides solutions that address site architecture, taxonomy, usability, and findability. He also speaks at SharePoint community events and blogs at his SharePoint site, Bits of SharePoint.

Marc Anderson is the cofounder and President of Sympraxis Consulting LLC, located in the Boston suburb of Newton, MA, USA. Sympraxis focuses on enabling collaboration throughout the enterprise using the SharePoint application platform. He has almost 30 years of experience in technology professional services and software development. Over a wide-ranging career in consulting as well as line manager positions, Marc has proven himself as a problemsolver and leader who can solve difficult technology problems for organizations across a wide variety of

industries and organization sizes. He was awarded the Microsoft MVP Award for SharePoint in January, 2011.

Alexander Bautz is 38 years old and lives in Norway. He has worked with SharePoint since 2007. Alexander's focus when blogging is client side customizations using JavaScript, and he feels that it's amazing how much you can do without venturing over to "the other side"—also known as the server side.

Alexander started blogging in 2008. His blog contains, more or less, fully functional solutions for this and that problem. Alexander accepted requests from his readers from the very beginning—and has learned a lot by figuring out how to answer the various questions and requests.

Sadie Van Buren has spent the past 10 years designing SharePoint solutions and leading deployments, while stealthily managing the human and cultural issues common to technology projects. She is a Senior Software Engineer at BlueMetal Architects, with a strong focus on strategy and alignment, usability, information architecture, and business process improvement. Sadie has a bachelor's degree from Wesleyan University, a Certification in Project Management from Boston University, and is a Microsoft Certified Information Technology Professional (MCITP). She is the creator of the SharePoint Maturity Model, and blogs about SharePoint and technology at amatterofdegree.typepad.com.

Jim Bob Howard has been working with web technologies since 1993, building websites, working with content management tools, and building web applications for companies large and small. Bringing all of that experience to SharePoint was a natural transition. He enjoys contributing to the SharePoint community through his popular article series, "Extending the Data View Web Part," speaking on the SharePoint circuit, and organizing SharePoint Saturday Austin. Working remotely from Austin, TX, Jim Bob is currently Senior Solution Engineer for Juniper Strategy, LLC, a dynamic consulting firm in the Washington, DC area.

Dessie Lunsford is the lead web developer/designer/everything-SharePoint-related for a community college in Western Washington. His main focus has been SharePoint solutions and end-user training. He has been playing with SharePoint since 2004. Much of his SharePoint time has been spent focusing on experimenting with creative solutions involving formulas in SharePoint calculated columns (the "Taming the Elusive Calculated Column" series), but he can also be seen moderating and answering questions on Stump the Panel and on Twitter.

Waldek Mastykarz is Microsoft SharePoint Server MVP. He shares his enthusiasm about the SharePoint platform through his [blog](#), [Twitter](#), articles published in on and offline magazines, and on MSDN SharePoint Forums. Waldek participates frequently as an expert in Ask-the-Expert community events such as SharePoint Connections, Microsoft TechEd, and DevDays. Recently Waldek became a Virtual Technology Solutions Professional for Microsoft Netherlands. In this role, he helps answer customer questions around SharePoint Web Content Management (WCM).

Laura Rogers is a Senior SharePoint Consultant at SharePoint911 and a Microsoft MVP. She has worked in SharePoint implementation, training, customization, and administration since 2004. Although her background is in server administration, her main focus is on making the most of SharePoint's out-of-the-box capabilities. She works extensively with SharePoint Designer workflows, InfoPath, and Data View Web Parts. Laura's latest books on SharePoint 2010 are *Using Microsoft InfoPath 2010 with Microsoft SharePoint 2010 Step by Step* (2011, Microsoft Press) and *Beginning SharePoint 2010: Building Business Solutions with SharePoint* (2011, Wiley Publishing, Inc.). She blogs at SharePoint911.com, and she is [@WonderLaura](https://twitter.com/WonderLaura) on Twitter.

COLOPHON

The cover image is a box of tools from Dreamstime. The cover fonts are Akzidenz Grotesk and Orator. The text font is Adobe's Meridien; the heading font is ITC Bailey.