Lotus software

Produced in collaboration with: Redbooks®

Content in this document was produced in collaboration with Lotus® and IBM® Redbooks®.

Best Practices for Building Domino 8 Web Applications

- **Key recommendations for updating existing Web applications**
- **Best Practices for refining application look and feel for the Web**
- **Common tips and techniques**

- **Bruce Lill**
- **Bruno Grange**
- **Chris Toohey**
- **Debra Landon**
- **Jennifer Heins**
- **Jimmy Minata**
- **John Noltensmeyer**
- **Joseph D'Armi**
- **Lisa Schenkewitz**
- **Louis Orenstein**

March 2008

Lotus. **and** Original content produced by: Redbooks.

Space Details

Key:	dominoappdev
Name:	Lotus Domino Web Application Development
Description:	
Creator (Creation Date):	dwblogadmin (Feb 04, 2008)
Last Modifier (Mod. Date):	dwblogadmin (Feb 04, 2008)

Available Pages

        - Navigation techniques
            - Moving past the frameset - Making Web-based applications that do not look like Lotus Notes
            - No more twisties - Using single category and a combobox to filter the view
            - View-based menus
        - Personalization
        - Searching
            - Creating custom and advanced searches using Domino
                - Customizing the search results display
                - Searching via Domino URL commands
                - Searching via FTsearch and DBSearch
            - robots.txt
            - Search engines and search engine optimization
            - SEO techniques
        - URL considerations
        - User management
        - Using interactive data and Web services
        - Working with data
            - JSON
            - RSS
            - Using query views
- 5.0 Extending rich client applications for Web clients
    - Benefits and pitfalls of extending rich client applications for Web clients
    - Data management in hybrid rich and Web client applications
    - Defining functional requirements based on client type
    - Developing hybrid rich client and Web client applications
- 6.0 Server configuration
    - Logging
    - Performance considerations
    - Security considerations
        - SSL support
            - Setting up SSL with a self-certified certificate
    - Server error handling
        - Global 404 error form
        - Web server configuration database
    - Topology
    - Working with Web site rules
        - Directory rules
        - HTTP Response Header rules
        - Overriding Session Authentication rules
        - Redirection and Substitution rules
- 7.0 Developer tools and resources
    - Domino resources
    - Web development resources

0.0 Preface

This page last changed on Apr 03, 2008 by jservais.

This section of the wiki contains a collection of best practices for building and updating Lotus® Domino® Web applications. The topics vary widely and range from key recommendations for updating existing Web applications and best practices for refining the look and feel of Domino applications for the Web, to common tips and techniques.

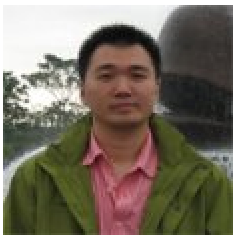
Assumptions

This information assumes that you have some basic Web development skills.

Meet the authors

Authors

	Bruce Lill (brucelill) has been providing Notes® solutions since Lotus Notes V2.0. Bruce is a certified Developer, Administrator and Instructor. Bruce He spends his time building secure Web sites for companies and state governments. His satisfaction doesn't come until the site is validated and the customer is happy. Bruce can be contacted by sending e-mail to bruce@kalechi.com.
	Bruno Grange (brunogrange) is an IBM® certified advanced application developer, instructor and system administration specialist on Lotus Notes/Domino. He works for IBM as an application developer, participating on various global delivery projects using Lotus Notes/Domino. Bruno has over eight years of experience, which includes working with several IBM business partners in Brazil as Procwork, WJ and Cyberlynxx. Bruno has in-depth experience with Web application development and implementation Web 2.0 using Lotus architecture. He is a

	coordinator of the AS Brazil Notes/Domino team, a group of developers in IBM Brazil that brainstorms new ways to promote Lotus Notes brand. He also maintains a personal Web site with information and tips about Lotus and Web application development at www.grange.com.br. Bruno can be contacted by sending e-mail to brunog@br.ibm.com.
	Chris Toohey (christoohey) is a published developer and Webmaster of DominoGuru.com, a Lotus Notes/Domino-themed "Tips & Tricks" Web site and Weblog. He is the Chief Solutions Architect for Clearframe and specializes in integrating IBM Lotus Notes/Domino with other enterprise-level solutions. Since entering the IT industry in 1998, Chris's unconventional methodologies, forward thinking and his ability to uniquely analyze and attack a given problem with award winning solutions has afforded him recognition as an expert in his field, as well as yielding many happy customers.
	**Jimmy Minata** (jminata) has over 13 years of domino development and administration experience. He specializes in developing Web applications by using CSS, DHTML, and AJAX and is an expert in integrating Domino with relational databases such as DB2®, SQL Server®, and Oracle®. As a senior consultant and a CRM Framework Manager with PSC Group, he has successfully architected and implemented CRM, Workflow, Portal, and Content Management application at various clients.
	John Noltensmeyer (jnoltensmeyer) has over 12 years experience as a Lotus Notes developer and administrator and is certified as both a Notes developer and administrator in each release from R3 to ND8. He is also a Sun™ Certified Java™ Programmer, MCSE, and holds the Certified Information Systems Security Professional (CISSP) designation. John has been developing Domino web applications since 1997 and standards-based Web design is one of his passions. You can contact John by sending e-mail to john.noltensmeyer@usa.net.
	Joseph D'Armi (josephd) has worked with Lotus Notes since version 3 and spent the last 10 years focused on Domino Web-based applications. Has architected collaborative sales, marketing and financial applications; content management systems for corporate Web sites and intranets; and commerce sites most notably the Lotus sponsored online shop for Manchester United

	Football Club.
	Lisa Schenkewitz (schenkew@us.ibm.com) has worked with Lotus Notes for 15 years since Version 3 and also has been certified in each release as a principal developer and administrator. Has done several notes project, including consulting and design reviews, and most recently an IBM Domino-based Web site for IBM manufacturing. She is an adjunct member of the Lotus Notes IBM Center of Competency (CoC).
	Louis Orenstein (lorenstein) has been supporting and troubleshooting the Lotus Domino HTTP task for both performance and interoperability issues for over four years and currently serves as the team lead for the level 2 North America support team.
	Debbie Landon (dalandon) is an IBM Certified Senior IT Specialist in the IBM ITSO, Rochester Center. Her current area of expertise is the System i collaboration products, including IBM Lotus Domino and related Lotus products, such as Sametime® and QuickPlace®. Debbie has been with IBM for 24 years working first with the S/36 and then the AS/400®, which has since evolved to the iSeries® server and is now the IBM System i™ platform. Before joining the ITSO in November of 2000, Debbie was a member of the PartnerWorld® for Developers iSeries team, supporting IBM Business Partners in the area of Domino for iSeries. You can contact Debbie by sending e-mail to dalandon@us.ibm.com.
	Jennifer Heins (heinsje) is the senior information architect and strategist for the Lotus and WebSphere® Portal family of products. She is currently driving innovation in how IBM plans, develops, and delivers technical information. Some specific goals include planning a broad spectrum of technical content deliverables for various audiences and skill levels, enabling customers, partners, and IBM to collaborate and interact with information, and fostering a knowledge sharing culture inside and outside of IBM through wikis and other technology. She also works closely with IBM and SWG groups to contribute to standards and guidelines used across IBM for technical information. Jennifer has been at IBM for 10 years working first in the WebSphere software brand and then moved to pervasive technology, which evolved into the current set of WebSphere Portal family and Lotus products. You can contact Jennifer by sending e-mail to heinsj@us.ibm.com.

Supporting contributors and reviewers

Thanks to the following people for their contributions to this project:

Mark Jourdain	IBM Lotus Domino Product Manager

Become a contributor

Join us for a two- to six-week residency program! Share your knowledge with peers in the industry and learn from others. Help create content about specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients. Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want the content in this wiki and all our wikis to be as helpful as possible. Provide us your comments in one fo the following ways:

- Use the commenting feature with in the wiki. Login and add comments, located at the bottom of each page.
- Provide feedback in the Web form located at:
  http://www-12.lotus.com/ldd/doc/cct/nextgen.nsf/feedback?OpenForm

Conventions

Navigation

Each section includes a short navigation that points to other related topics in the section.

Prerequisites

There may be prerequisites that are important to read or perform before using a topic. This is noted in a Prerequisites section.

Code samples

```
Code samples are shown in this format.
```

Copyright information

The following technologies are commonly referenced:
PHP - PHP is GNU/GPL and is maintained by the Opensource community.
ASP - Microsoft® ASP is a copyright of Microsoft corporation.
JavaScript - JavaScript™ is a copyright of Sun Microsystems.
Java - Java™ is a copyright of Sun Microsystems.

IBM copyrighted products may be referenced by using the following terms:

- *Domino* for IBM Lotus Domino
- *Domino Designer* or simply *Designer* for IBM Lotus Domino Designer
- *Notes client* or just *Notes* for IBM Lotus Notes client

Company example

Throughout this information, we reference a fictitious company, called Riverbend Coffee and Tea Company, that we use to illustrate various points.

Riverbend Coffee and Tea Company

This page last changed on Apr 03, 2008 by jservais.

Company overview

Riverbend Coffee and Tea Company (Riverbend) is a privately owned organization that consists of 45 world-wide locations, corporate and regional offices, store fronts, and distribution or manufacturing plants, and in excess of 6000 employees.

The following key information illustrates the diversity of the technology and functional requirements of the Riverbend Coffee and Tea Company:

- Riverbend has an extremely mobile sales force, who primarily work from their Blackberry mobile devices or computers when in their home or remote offices.
- Riverbend's Customer Service is located at the corporate headquarters.
- Owner and operators of Riverbend "store front" franchises use corporate-lease technology solutions (such as mobile computers and desktops) that securely connect to regional Riverbend offices.
- Riverbend Distribution and Manufacturing plants use kiosk and thin-client solutions to equip employees with e-mail and collaborative business solutions.

Technology investments

Riverbend uses the following key technologies to ensure that their employees can effectively and efficiently collaborate with each other:

- Microsoft Windows®
- IBM Lotus Notes/Domino 8.01
- IBM Sametime
- Microsoft Office 2003
- Microsoft Internet Explorer®
- Cisco Call Manager 4.2.3

1.0 Primer

This page last changed on Mar 31, 2008 by jservais.

In this section, we review a wide range of technologies, techniques, and applied methodologies that are critical to the Domino Web Application Developer. Consider this a simple-yet-indepth knowledge base for those of you who are new to Domino Web Application Development, as well as a *refresher* for those of you who are already *seasoned pros*.

As mentioned, in this section, we cover the basics, such as answering "What is HTML?", to more advanced subject matter, such as Web 2.0. The intention of this primer to give you a fundamental understanding of the same code and architecture that we use in our more advanced Domino Web Development.

Topics in this section

HTML primer

This page last changed on Apr 04, 2008 by dalandon.

What is HTML?

What exactly is HTML? It is not a programming language, but a rather a markup language for formatting and displaying Web documents and pages. It is the most commonly used markup language for Web pages today.
The more we understand both the principles and applied usage of the HyperText Markup Language (HTML) and to an extent, other markup languages such as XHTML that find their roots in HTML, the more we can use it to deliver flexible and rich functionality to our Domino Web applications. In this section, we review the fundamentals that, when mastered, give you the base tools to expand your Domino Web applications beyond previously conceived limitations.

HTML is simple text markup (Content type: text/html). It consists of a combination of various objects that Web browser clients commonly interpret based on a defined standard. These various objects consist of elements, data types, and character and entity references. As Domino Web Developers, we leverage these objects to render the display, facilitate the maintenance, and extend the functionality of our Domino Web applications.

Basic HTML Web page structure

In this section, we review the basic element and standard attribute construct of HTML for Web pages. The following example shows Web page markup.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
</head>
<body>
<p>Hello world!</p>
</body>
</html>
```

The first line in the previous example defines the DOCTYPE for this particular Web page or document. The DOCTYPE declaration is translated by the Web browser client and defines the standard specification that the document uses.

The <html> tag simply defines this Web page as an HTML document.

The <head> tag is used primarily to define *back-end* information for the document, and is not displayed to the Web Browser client screen at run time. This is an example of a *container* element, which is used to store other functional and document information such as (but not limited to) <meta>, <script>, <style>, and <link> tags and elements.

The <body> tag is the *container* element used to display the visual contents of the HTML document. This element may not only contain visual elements such as (but not limited to) <p>, <div>, <span>, and <object> tags and elements, but can additionally contain <script> and <style> tags and elements.

The <p> tag is a visual *container* element that is typically used to display text paragraphs and other such content.

Simple XHTML Web page example

In this section, we review the basic element and standard attribute construct of XHTML for Web pages. We immediately recognize the format, as XHTML is simply a more structured and flexible extension of standard HTML. The following example shows simple XHTML Web page markup.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Simple XHTML Webpage Example</title>
</head>
<body>
<p>Hello world!</p>
</body>
</html>
```

In this example, we include the XHTML required attributes to the basic DOCTYPE and <html> tags as well as included a <title> tag, which is used define the title of the document. In most Web browser clients, the contents of the <title> tag is displayed as the Window Title of the given Web browser client instance.

While there are other tags and elements, understanding how these basic tags and elements both interact with each other and render in the Web Browser client eventually gives us the ability to completely control the visual look and dynamic function of our Domino Web applications.

Elements and attributes

In this section, we review the basic tag schema for elements and their various attributes. The following example shows the standard tag and attribute schema.

```
<tagname attribute1="attributevalue1" attribute2="attributevalue2">_</tagname>_
```

An end or closing tag, which is highlighted in the above example, can be optional based on the given tag element. For example, while the <script> tag **requires** a end/closing tag, the <body> tag does not require an end/closing tag. Certain markup tags explicitly require that you do **not** use an end/closing tag, such as the <link> element.

See the Index of the HTML Elements from the W3C for a complete list of elements and the HTML /XHTML Reference from W3C Schools for more detailed information.

Basic markup

In this section, we review some of the more common tags used in an HTML page.

Common tags

You will consistently use the tags that are highlighted in the following table and therefore must be familiar with them.

HTML tag	Comments
<meta />	Meta tag used in html header. Meta data is used to describe the Web page. <meta name="Keywords" content="description" />
<a></a>	Anchor tag, used to redirect user to a different Web page. <a href="*url of web page*" >*link text displayed on page*</a>
<div></div>	Division tag, used to define a block of HTML and apply specific CSS or style attributes to it. <div id="*id of division*">*text displayed on page*</div>
<span></span>	Span tag, used to define a block of HTML and apply specific CSS or style attributes to it. Used for inline content. <span id="*id of span*">*text displayed on page*</span>
<img />	Image tag, used to include an image or image map on Web page. <img src="*image file name*" alt="*alternate text that describes image*" width="*width*" height="*height*"/>

Formatting text

As you work with HTML, format text by using a CSS stylesheet. You can find information about CSS markup in the Styles and CSS Primer in this guide. Some tags can be used in conjunction with CSS for flexibility in overriding the CSS in the marking of specific text or can be used without CSS. The following table shows a subset of formatting tags that can be useful.

HTML tag	Comments
<strong></strong>	Strong tag. Used to make text bold. <strong>*text*</strong>
<big></big>	Big tag. Used to make size of text bigger. <big>*text*</big>
<small></small>	Small tag. Used to make size of text smaller. <small>*text*</small>
<p></p>	Paragraph tag. Used to delineate and define a paragraph. <p>*text*</p>

	Line break tag. Used to define a new line.

Creating tables

The use of tables in formatting and displaying content data is invaluable on any Web site. To create a table, use the tags listed in the following table.

HTML tag	Comments
<table></table>	Table tag. Delineates the start and end of the table.
<th></th>	Table header tag. Defines the row that serves as the the table header row.
<tr></tr>	Table row tag. Defines any row that serves as a table data row.
<td></td>	Table data tag. Defines a column and the data that goes in it.

As an example, if we were to code a two row and two column table, it might look something like the following example.

```
<table>
<th>  //first row, table header
  <td>header for column 1</td>
  <td>header for column 2</td>
</th> // end table header
<tr>  // second row, table data row
  <td>data for column 1</td>
  <td>data for column 2</td>
</tr> // end table data row
</table>
```

Here is another example, with the more recent table tags.

```
<table>
<thead>
<tr>
```

```
    <td>header for column 1</td>
    <td>header for column 2</td>
  </tr>
  </thead>
  <tbody>
  <tr>
    <td>data for column 1</td>
    <td>data for column 2</td>
  </tr>
  </tbody>
  </table>
```

Functional markup

In this section, we provide a simple overview of HTML tags and elements that, when rendered in a Web browser client, have built-in functionality. In the HTML Form Element section, included as part of the form tag, we see an example of HTML functional markup: the <input>-based *Submit* button.

The following table shows several examples of functional markup.

Functional markup example	Result when rendered in Web browser client
<input type="submit" value="Submit" />	A simple button, with a label of *Submit*, that processes the user data input gathered by the HTML Form element against the HTML Form Element's *processing agent* that contains this button.
<input type="reset" value="Clear" />	A simple button, with a label of *Clear*, that blanks all field elements in the HTML Form element that contains this button.
<input type="file" name="%%File1" />	Renders a File Upload Control element object that allows the user to browse and select a single file on the local system for processing.

The HTML Form Element

The HTML Form Element is a functional element, created from the combination of the <form> tag with other elements such as (but not limited to) the <input>, <textarea>, and <select> tags. The purpose of the HTML Form Element is to gather user input and submit that information to a data processing engine via methods defined in the attributes of the given Form element. The following example shows a simple HTML Form Element.

```
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
  <html>
  <head>
  <title>HTML Form Element Example</title>
  </head>
  <body>
  <form action="_processing agent_" method="_form processing method_">
  <input type="text" name="test" value="" />
  <input type="submit" value="Submit" />
```

```
</form>
</body>
</html>
```

In the previous example, the information that is entered in the *test* <input> element by the user is submitted to the HTML Form Element's *processing agent* - defined in the <form>'s *action* attribute - after the user clicks the Submit <input> tag, which is rendered in the Web browser client as a button.

The HTML Form Element's *method* attribute, which has the value of "GET" or "POST", is the means to which the data from the HTML Form Element is submitted to the <form>'s *processing agent*, which again is defined in *action* attribute. The *method* attribute is often defined based on the requirements of the particular HTML Form Element, or the capabilities of its *processing agent*.

Understanding the basics of the HTML Form Element is absolutely vital to the Domino Web Appplication Developer. Mastery of the HTML Form Element allows you to extend the creation and maintenance capabilities of data in your Domino Applications, providing your target user audience with extended functionality that otherwise was not thought possible with Domino Web applications.

Working with HTML in Domino

Domino Designer facilitates adding HTML to your Domino Database. Working with HTML in Domino can be easy provided that you understand the basic rules for how to incorporate HTML into your database and understand how Domino renders HTML in a Web browser.

Additional topics

- Content type
- DOCTYPE
- Working with HTML in Domino

Content type

This page last changed on Apr 03, 2008 by jservais.

The content type of an HTML page specifies to the Web browser or server the type of page or resource that is being referenced or rendered. Also known as the MIME type, it is case insensitive and can be one of the types listed in the following table.

Content type	Description
text/html	The resource is an html page
image/png	The resource is a png filetype image file
image/gif	The resource is a gif filetype image file
video/mpeg	The resource is a mpeg filetype video file
audio/basic	The resource is an audio file
text/tcl	The resource is a Tool Command Language (TCL) script text file or page
text/javascript	The resource is a JavaScript text file or page
text/vbscript	The resource is a Visual Basic® text file or page
text/css	The resource is a CSS text file or page

For more information on content type, you can reference the World Wide Web Consortium (W3C) Web site for current standards and definitions.

DOCTYPE

This page last changed on Mar 31, 2008 by jservais.

In this section, we discuss the DOCTYPE declaration for an HTML page. The <!DOCTYPE> declaration is the first tag in your document. It comes before the <html> tag and tells the browser which HTML or XHTML specification the document uses.

The six current specifications are highlighted in the following table.

Type, tag syntax, and explanation
HTML or XHTML Strict DTD: Used to enforce strict HTML standards and used with CSS. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> |
HTML or XHTML Transitional DTD: Used when you need to use font and display formatting, when CSS is not feasible. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
HTML or XHTML Frameset DTD: Used when your web site or Web page uses frames. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd"> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

You have some options on configuring the DOCTYPE for your Web pages in Domino, even though Domino automatically self generates the DOCTYPE tag. See the section on working with the DOCTYPE tag in Domino for more details.

For further information about the DOCTYPE tag, you can reference the World Wide Web Consortium (W3C) Web site for current standards and definitions.

Working with HTML in Domino

This page last changed on Apr 03, 2008 by jservais.

Working with HTML in Domino can be easy if you follow a few simple guidelines. Domino Designer has four venues to allow you to add HTML to your database.

Design element type	Description
Computed text	You can create computed text as an option from the Domino Create menu while you are in a page or form. The HTML is placed as the value of the computed text. You can also highlight the computed text and turn it into passthru HTML using the Domino Designer Text menu.
Passthru HTML	You can type HTML directly into the page or form as text, highlight it, and turn it into passthru HTML using the Domino Designer Text menu.
Field formulas	HTML can be the output of field formulas.
LotusScript	HTML can be the output of any LotusScript® "print" statement.

What is passthru HTML versus regular text? Passthru HTML is passed directly to the Web browser without Domino altering any of its attributes, such as its font or style.

In addition, you should be aware of the following items before you begin:

- Domino generates its own doctype tag for each page. If you have unique requirements for your doctype tag, you are not able to override the default. There is a certain amount of control over the contents of the tag that is specified at the server level, but you as the developer are not able to override it.
- Domino generates its own <html> and <head> tags. You do not need to include these in any HTML that you place on your form.
- Domino generates its own form tag for editable forms. See adding HTML to a design for more information on this tag.

Rendering HTML

You should be aware of the rules and order in which Domino renders HTML.

HTML element	Comments
doctype	First HTML tag rendered. Domino generated
<html>	Second HTML tag rendered. Domino generated
<head>	Third HTML tag rendered. Domino generated

HTMLHeadContent field	Next piece of HTML added to the output. The HTMLHeadContent field of the Domino form is where you put the contents of the HTML head, including meta tags.
<script>	Next HTML tag rendered. Domino generated
JSHeader field from domino form	Next piece of HTML added to the output. The JSHeader field of the Domino form is where you put any JavaScript for the form.
</script>	Next HTML tag rendered. Domino generated
</head>	Next HTML tag rendered. Domino generated
<body>	Next HTML tag rendered. Domino generated. The HTMLBodyAttributes field of a domino form allows you to customize this tag.
<form>	Next HTML tag rendered. Domino generated for editable forms
content	Your masthead, navigation, content, footer
</form>	Next HTML tag rendered. Domino generated
</body>	Next HTML tag rendered. Domino generated
</html>	Next HTML tag rendered. Domino generated

Understanding the order in which HTML is presented to the browser can be of help in quickly locating and diagnosing any problems that you might face when you are presented with unpredictable rendering results.

For further information about how to code your HTML code on Domino, refer to Styling text for the Web and adding HTML to a design.

Java primer

This page last changed on Apr 04, 2008 by dalandon.

What is Java?

Java is a popular and highly useful programming language that was designed to be both powerful and simple. Developed by Sun Microsystems. It is similar in style and syntax to C, C++, and C#.

Java's similarity to C ends with its syntax. Java is object oriented, and its entire concept is a change from the traditional older programming languages such as C, Pascal, and others that are oriented towards linear processing and design. In that respect, it is more similar to C++ and C# with its use of object modeling.

Java is both a compiled language and an interpreted language. The developer compiles source code (.java files) into class files (.class). When a java program is actually run, the interpreter, known as the Java Virtual Machine (JVM™) is called to interpret the byte codes that are contained in the class file or files and run the program.

The Java environment

What is the JVM

The Java Virtual Machine is included in all Java installations, and is required to run any Java program. The JVM is responsible for interpreting the Java program .class file being run and executing the program.

While Java is portable, the JVM is unique to the software platform on which you are running. For example, the JVM for Sun Solaris™ is different from the JVM for Microsoft Windows, Linux®, and Apple. Just about all vendors provide a JVM for their operating systems, and are available for download from the internet. A good example is Sun Microsystems, which provides JVMs for its own Solaris platform as well as Microsoft Windows and Linux.

What is the Java Platform

The Java Platform is a set of classes that is included in every Java installation, and is available to every Java program. These classes provide a vast amount of functionality and include the core classes of the language itself.

All Java classes, including the Java Platform, are organized into groups called packages. As an example of the packages that are available in the Java Platform, we discuss two packages.

java.lang

The java.lang package includes the core base of the Java language. It includes the superclass called **Object** that defines all basic object functionality of the language. All classes in Java are a subclass of **Object**. The following classes of note are included in this package among others:

- **Math** - Class that provides math functionality.
- **String** - Class that provides functionality to manipulate strings.
- **Number** - Class that provides wrappers around primitive Java types.
- **System** - Class that provides an interface to system facilities.

- **Thread** - Class that provides information on a Java program thread.

java.util

The java.util package includes a set of utility classes. The classes provided in this package primarily revolve around handling and dealing with specific data structures. The following classes of note are included in this package among others:

- **Arrays** - Class that provides functionality for sorting, searching, and working with arrays.
- **Collections** - Class that provides functionality for working with java collections.
- **Hashtable** - Class that provides functionality to create and manipulate hash tables.
- **LinkedList** - Class that provides functionality to create and manipulate linked lists.

- **Stack** - Class that provides functionality to create and manipulate stacks.

These packages and classes are a small sample of what is available in the Java Platform. After you see and work with the Java development environment, you can understand how rich and robust it is.

The Java Runtime Environment

The JVM is included in the Java Runtime Environment™ (JRE™). The JRE consists of the JVM, Java core classes, and supporting files. It can be downloaded as an installable executable from an appropriate vendor website. For Windows and Linux, you can refer to the Sun Microsystems Web site for the most current installs.

One of the things to note about the JRE is that it is frequently embedded in other program installations, including Domino. Any application that uses Java as its base language, or uses Java classes in any way, needs to supply a JRE along with it at installation. Otherwise you must ensure that the JRE is installed and available separately.

You can have multiple JREs on your desktop or server, each associated with a specific application, and each isolated from each other by both the application and operating system. One of the side effects of this is that a given application can have a different level of the JRE/JVM than another. When you specifically download just the JRE from the Internet, chances are you are dealing with a unique situation in which an application or program needs it loaded separately.

The Java Development Kit

One of the applications or tools mentioned above that includes the JRE/JVM in it is the Java Development Kit (JDK™). In addition, the JDK contains all the necessary files and tools for compiling Java programs and creating your own packages.

The JDK is sometimes referred to as the Java Software Developer Kit (SDK). Like the JRE, the JDK for your OS platform can be downloaded from the proper vendor's Web site.

Learning Java

This primer is intended to be a brief and high level overview of Java and its capabilities. If you are inexperienced or are unfamiliar with Java, take the time to tour one or more of the many books or online tutorials to expand your knowledge of the language.

Reference books

Many reference books are available on Java, both by purchase on the Internet or your local bookstore. They vary in scope and depth, from the popular "Dummies" series, "Nutshell" series, and "Teach yourself..." series, to the more high level college texts and professional publications. Some of these books are available online.

- **Java in a Nutshell**, by David Flanagan. Publisher: O'Reilly Media, Incorporated
- **Java: How to Program**, by Staff of Deitel & Associates, H. M. Deitel. Publisher: Prentice Hall
- **Sams Teach Yourself Java 6 in 21 Days**, by Rogers Cadenhead, Laura Lemay. Publisher: Sams
- **Java For Dummies**, by Barry Burd. Publisher: Wiley, John & Sons, Incorporated
- **Head First Java**, by Kathy Sierra, Bert Bates. Publisher: O'Reilly Media, Incorporated

Java resources, Web sites and tutorials

There are also a multitude of Web sites with a wealth of information about Java, including a lot of reference documentation on the standard Java classes and Java language itself. In addition, for the beginner, free tutorials are available on the Internet.

Sun Microsystems' Sun Developer's Network web page should be your first resource on Java information, including downloads, and has an excellent tutorial to get you started.

Additional topics

- A simple Java program
- Introduction to applets
- Introduction to classes and objects
- Working with Java in Domino Designer

A simple Java program

This page last changed on Apr 03, 2008 by jservais.

Hello World

The most commonly taught program when learning a new programming language is Hello World!, which is shown in the following example.

```
package com.ibm.test;

import java.util.*;
import java.lang.*;

/**
 Hello world program
 */
public class hello {

        public String myName;

        public static void main(String[] args) {
                System.out.print ("Hello world!");
        }
}
```

In the previous example, the first line of code is the package statement, which identifies the package to which this class belongs. The package name is important because it is what you need to reference when importing this class into another.

Next is the import statement, which is what you use to reference classes that are in another package. In this case as an example, we have imported the Java Platform java.util package and java.lang package. In the above code example, you do not need these two import statements, because the code base never references any of the classes in either package. When compiling the above code, you might get a compiler warning that the packages are not referenced.

The class statement identifies the class name within the package. It is also important to know that the class name must match the file name of the .java file you are working in.

Within the class statement are properties and methods. The first statement marked as public is the declaration of a property of type String, whose name is "myName:. This is included as an example only and is not referenced in the code base either.

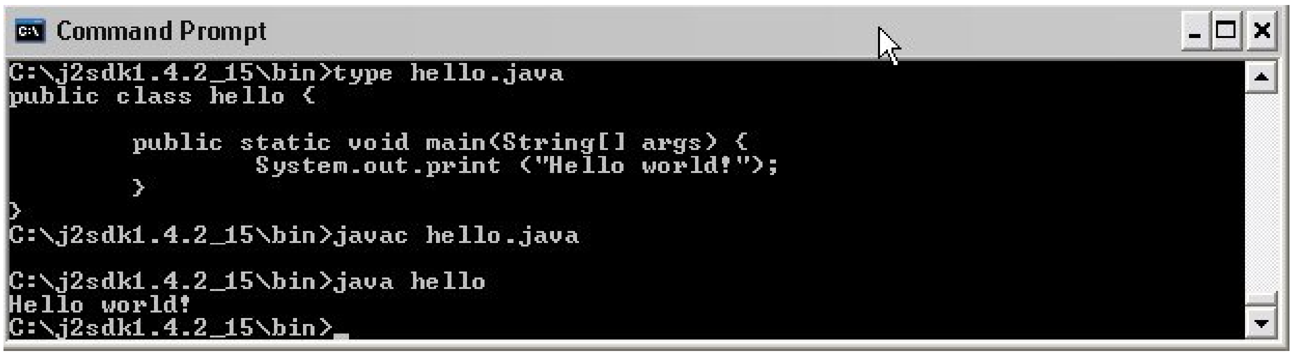
Next is the main program body, designated by the public "main" definition. All Java applications need a main function that is identified as the starting point of the application. The main statement itself has a specific structure and syntax that must be adhered to. In this case, the main function simply prints a statement to the designated system console.

Compiling and running the program

Now you are ready to compile. We remove all extra lines of code from the program and are left with the six line file that is displayed in the following example image.

The Java compiler is included in the Java SDK and is javac.exe. To invoke the compiler, you simply provide the Java source file name as its input. The JVM is manually invoked with java.exe. To run the program, simply type the name of the class file.

You're done!



```
C:\j2sdk1.4.2_15\bin>type hello.java
public class hello {

        public static void main(String[] args) {
                System.out.print ("Hello world!");
        }
}
C:\j2sdk1.4.2_15\bin>javac hello.java

C:\j2sdk1.4.2_15\bin>java hello
Hello world!
C:\j2sdk1.4.2_15\bin>_
```

Integrated development environment and other development environments

In the previous example, we use one of two methods of Java development. We downloaded the Java SDK, did all of our file creation by using standard text editors such as Notepad or Wordpad, and then manually compiled and ran the application.

The second method is the use of an integrated development environment (IDE). An IDE is a packed program or application that, when istalled, provides an entirely encapsulated and all-in-one environment for doing development work. It contains a front-end editor that allows you to easily edit, debug, build, package, and sometimes even deploy your application if it contains deployment tools within it. It frequently does not contain a JDK, as Sun releases new updates and versions of their JDK on a regular basis. You must consult the vendor documentation and Web site prior to download to determine if you need to also download a compatible JDK.

A detailed discussion on IDEs is beyond the scope of this primer, but we list a few IDEs here for your consideration:

- **Eclipse:** An open source community dedicated to open dedicated to an open development platform, the eclpse.org Ecipse IDE provides both a simple and easy to learn IDE.

- **Sun IDE:** Sun Microsystems has its own IDE environment that is downloadable from its website. The IDE comes packaged both wihout the JDK and bundled with a JDK.

- **IBM Rational® Software Architect or Rational Application Developer**: Based on Eclipse technology, the IBM robust IDE comes with a rich environment for development of enterprise applications.

Introduction to applets

This page last changed on Apr 01, 2008 by jservais.

A Java applet is essentially a small non-standalone application that can be run from a web browser. Put simply, when a user accesses and runs an applet, the Web browser downloads the applet code from a Web page to the user's machine and then runs it. Since the applet is essentially a running program after it is loaded, it can do everything that a standalone Java program can do if it were run from the user's machine.

The key here is that an applet is a non-standalone application, and it is specifically run from a Web browser.

Introduction to classes and objects

This page last changed on Apr 03, 2008 by jservais.

The concept of an object

What is an object? Put simply, an object is anything that has both specific attributes that describe it, as well as specific actions that it can perform and can have performed against it. To be more specific, we use the traditional example of a bank account.

A bank account has specific attributes that describe it. It has a balance, for example. It also is of a specific type, i.e, checking or savings. It also may have special aspects such as overdraft protection. All these things, which are considered *nouns*, that describe the account are properties of the bank account.

What can a bank account do, and what can be done with a bank account? You can make a deposit or withdrawal. You can do more specific things, such as make a transfer to a different account. You can set up overdraft protection or automatic payments. All these things, which are considered *verbs*, that describe what an account can do are methods of the bank account.

Properties

The properties of an object are nouns that describe it. In the case of the bank account, the following nouns stand out:

- Account type
- Balance
- Interest rate
- Special services

These items are all necessary to manage the account. Some do not change, such as account type, while some change on a regular basis, such as the balance of the account.

Methods

The methods of an object are actions that the object can perform, or can be performed against it. In the case of the bank account, the following verbs stand out:

- Deposit
- Withdraw
- Calculate interest rate

These are all actions that can be done with the account. You might notice that method collaboration and reuse, for example, the deposit and withdraw methods, might both call the calculate interest rate method in order to correctly calculate a final balance. The balance and interest rate properties of the account are both accessed and updated as part of executing the deposit and withdraw method.

The important point here to remember is that everything you need to manage the object is encapsulated within the object itself.

Classes

A *class* is a compiled object. All your code for a given object, including properties and methods, is placed in a single .java file, and the class name has to match the file name of the Java file. When compiled, a .class file is produced from the .java file.

Working with Java in Domino Designer

This page last changed on Apr 03, 2008 by jservais.

Domino Designer is essentially an IDE. The following example shows how HelloWorld looks in Domino.

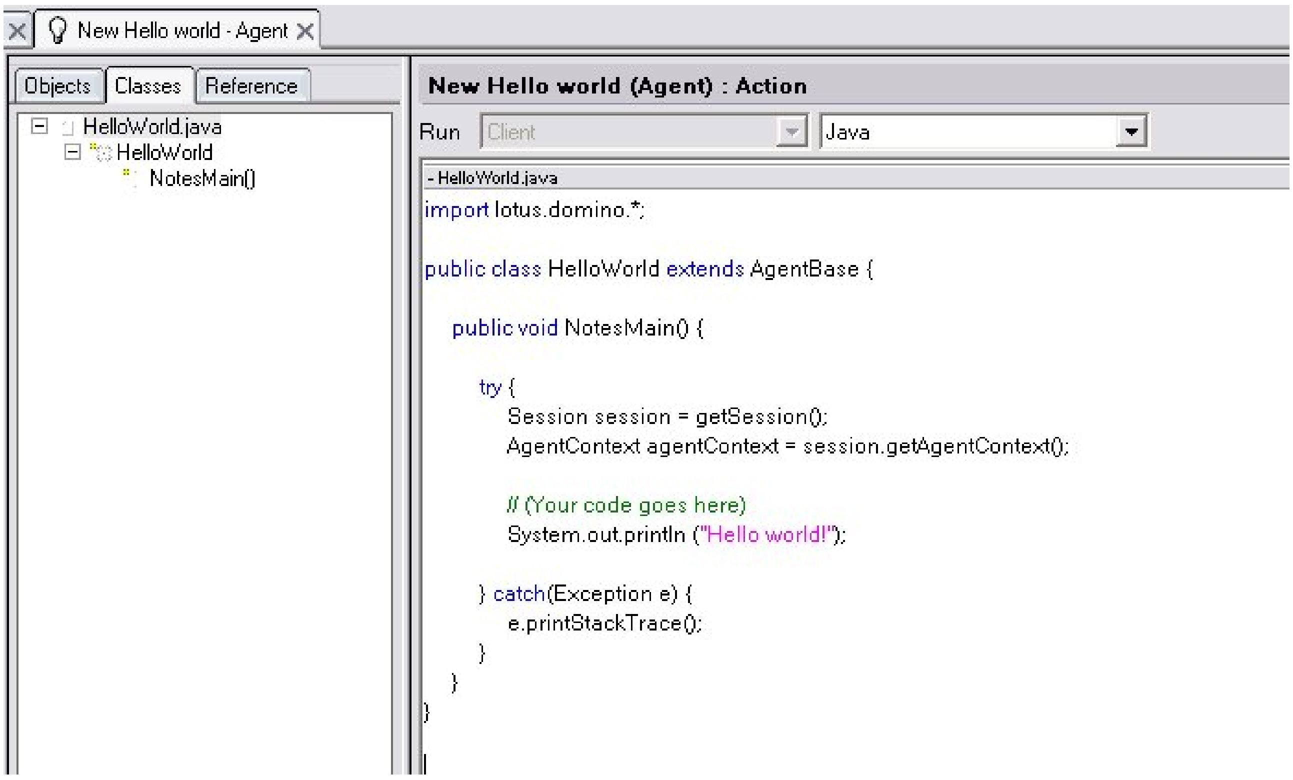
```
import lotus.domino.*;

public class HelloWorld extends AgentBase {

        public void NotesMain() {

                try {
                        Session session = getSession();
                        AgentContext agentContext = session.getAgentContext();

                        // (Your code goes here)
                 System.out.println ("Hello world!");

                } catch(Exception e) {
                        e.printStackTrace();
                }
        }
}
```

In this example, you might notice the following items:

- Domino Designer has included an import statement to include its domino classes and object model. These Domino classes essentially map out to equivalent classes in Lotus script.

- The default class name for a new Java agent is JavaAgent. In this case, we have changed it to HelloWorld. After the agent is saved, it is included in the Domino Database that has HelloWorld.java, and is suitable for export if necessary.
- The standard agent or application entry point is NotesMain
- Domino Designer inserts two lines of code to establish a Notes session to run the agent.
- A catch statement is automatically inserted to catch any errors reported to the JVM.

In the Domino Designer, our example is displayed in the agent window as shown in the following figure.



```
import lotus.domino.*;

public class HelloWorld extends AgentBase {

    public void NotesMain() {

        try {
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();

            // (Your code goes here)
            System.out.println ("Hello world!");

        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

In addition to using Java in notes agents, Java can also be included in a Domino Database as a Java Library. This give added flexibility in developing shared codes and applets.

JavaScript primer

This page last changed on Apr 04, 2008 by dalandon.

Prerequisites

- HTML primer
- Styles and CSS primer

What is JavaScript?

JavaScript is a scripting language most commonly used for client-side Web development. While sharing many of the attributes and design structures of Java, JavaScript (Content type: text/javascript) is a completely separate language. Because JavaScript executes on the client rather than a Web server, it is typically used to add interactivity to an HTML page such as:

- Manipulating the structure and style of the page
- Responding to actions taken by the user
- Form validation
- Site navigation schemes

As the Lotus Notes client has evolved, its support of JavaScript has increased, making it easier to create hybrid applications that are accessible by both Notes clients and Web browsers. As an example, you can code field validation for a Lotus Notes form using JavaScript, and it executes in both the Lotus Notes client and a Web browser. This enables you to write a single validation routine rather than having to code it twice, once in LotusScript or @formula language for Notes and a second time in JavaScript for Web browsers.

JavaScript basics

The support of JavaScript in the Lotus Notes client along with its power on the Web makes it a key component of every Lotus Notes developer's skill set. In this primer, we help you learn about key features of the JavaScript language as they relate to Domino Web programming that will hopefully motivate you to learn more.

Core JavaScript

If you are familiar with LotusScript but have never used JavaScript, there are a few key things to keep in mind. First, unlike LotusScript, JavaScript is case-sensitive. If you define a function with the name validateForm() but attempt to call ValidateForm() or validateform(), you receive an error that the function is not defined. One inadvertent error can lead to a long and frustrating debugging session.

Second, JavaScript is interpreted line-by-line as an HTML document is loaded in a browser. Consequently, it is best to put variable declarations and functions inside the <head> tag of the page or in the JS Header of a Notes form.

Third, JavaScript uses the semicolon to terminate statements. This allows you to put more than one statement on a single line as shown in the following example.

```
<script type="text/javascript">
var userName; userName = 'Jim'; alert('My name is ' + userName);
</script>
```

If the elements on a single line can logically comprise a complete JavaScript statement, but the line ends with a line break rather than a semicolon, a semicolon is implicitly inserted by the JavaScript interpreter. However, you must explicitly end each JavaScript statement with a semicolon to make your code easier to read and avoid confusion.

Note: JavaScript uses operators that are familiar to just about everyone with any programming experience. However, keep in mind that JavaScript uses a single equal sign (=) for assignment and a double equal sign (==) to test for equality.

Blocks and flow control

JavaScript uses curly braces to group statements together. For example, if more than one statement should be executed in a function, conditional statement, or loop, the statements are grouped together using curly braces as shown in the following example.

```
if (x < 10)) {
  x = x + 1;
  alert('x has been incremented by one');
} else {
  x = 0;
  alert('x has been reset to zero');
}
```

In addition to if/else statements, JavaScript also supports the following loops:

- while
- do/while
- for
- for/in

JavaScript functions

Like LotusScript, JavaScript allows you to create functions, which are reusable blocks of code that return some form of result. You define a function in JavaScript by using the function keyword, followed by a unique name for the function, a list of parameters contained in parenthesis (the parameter list can be empty), and finally a statement block surrounded by curly braces. The following example shows a simple function.

```
function riverbendGreetings() {
  alert("Welcome to River Bend Coffee!");
}
```

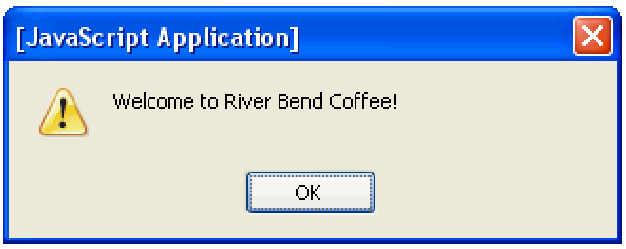
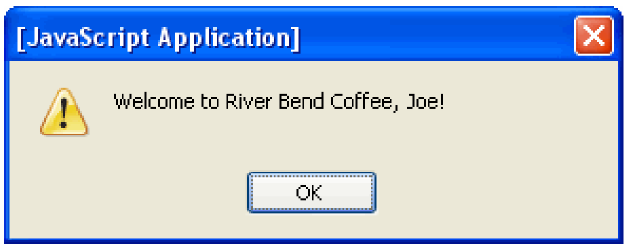
To call this function later in the script or from somewhere lower in the page, you simply use the function name as shown in the following example.

```
riverbendGreetings();
```

Typically, you want to pass data to the function so that the data can be modified or the function can determine a course of action to take. The data passed to the function is called a *parameter* or *argument*. We can expand the previous example to accept a single argument called userName as shown in the following example.

```
function riverbendGreetings(userName) {
  if (userName == null || userName == "") {
      alert("Welcome to River Bend Coffee!");
  } else {
      alert("Welcome to River Bend Coffee, " + userName + "!");
  }
}
```

If the modified riverbendGreetings() function is called without passing it an argument, the function generates the same alert as the previous example. However, if the function is passed to the parameter "Joe," then the alert changes to incorporate the parameter value as shown in the following figures.

riverbendGreetings();	**riverbendGreetings('Joe');**
 [JavaScript Application] Welcome to River Bend Coffee! OK	 [JavaScript Application] Welcome to River Bend Coffee, Joe! OK

Just as you frequently want to pass an argument to a function, you also want to get a value back from the function. In JavaScript, the return statement indicates the function should exit and return a value if one has been specified. Technically, a function always returns a value, even if you don't explicitly specify one. However, if you don't specify the value to return, the function returns a value of undefined.

Rather than generate an alert inside of the riverBendGreetings() function as we did in the previous example, the following code returns just the text used in the alert.

```
function getGreeting(userName) {
  if (userName == null || userName == "") {
      return "Welcome to River Bend Coffee!";
  } else {
      return "Welcome to River Bend Coffee, " + userName + "!";
  }
}
```

In this instance, we're interested in the return value of the getGreeting() function. Therefore, we assign it to the variable welcomeGreeting and pass welcomeGreeting as a parameter to the alert function as shown in the following example.

```
var welcomeGreeting;
welcomeGreeting = getGreeting('Joe');
alert(welcomeGreeting);
```

This results in the same "Welcome to River Bend Coffee, Joe!" alert message box pictured above. However we've gained some flexibility by using the return value of the getGreeting() function. Rather than use the return value in an alert, we could choose to include it as text on a Web page as shown in the following example.

```
var welcomeGreeting;
welcomeGreeting = getGreeting('Joe');
document.write(welcomeGreeting);
```

Including JavaScript on a Web page

There are a several different ways to include JavaScript on a Web page:

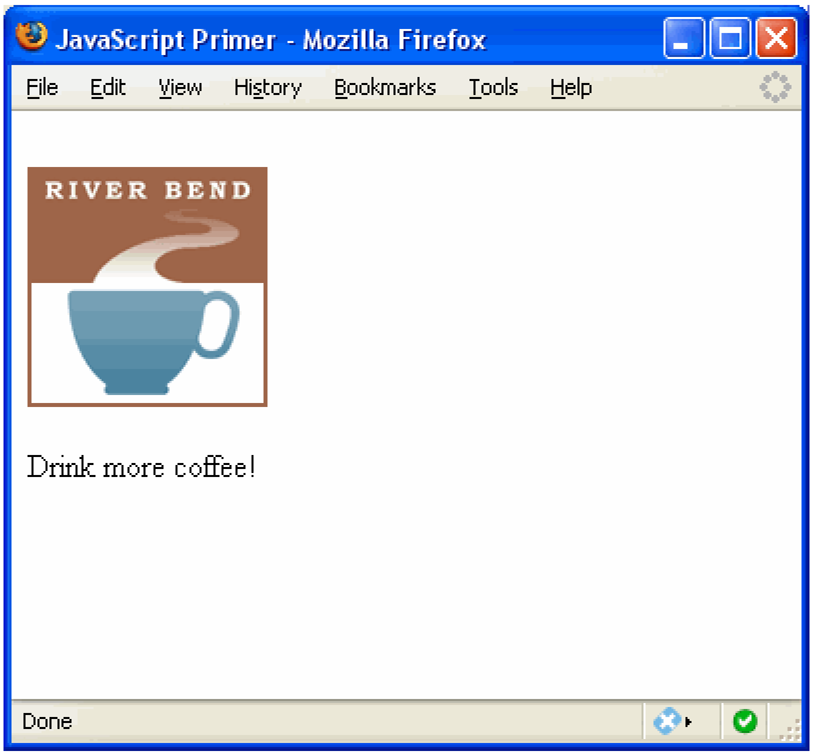
- The <script> tag
- Event handlers
- Linked scripts

The <script> tag

The first method of including JavaScript on a Web page is through the use of the <script> tag. You can place the <script> tag anywhere in the body of an HTML page and the JavaScript inside is executed as the page loads. Consider the following code for example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JavaScript Primer</title>
</head>
<body bgcolor="#ffffff" text="#000000">

<img src="images/riverbend_logo.gif" height="120" width="120">
<br />
<br />
<script type="text/javascript">
  document.write('Drink more coffee!');
</script>
<br />
</body>
</html>
```

This code yields the following results when viewed in a browser.


JavaScript Primer - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIVER BEND
Drink more coffee!
Done

Using the <script> tag to add JavaScript to a page

Event handlers

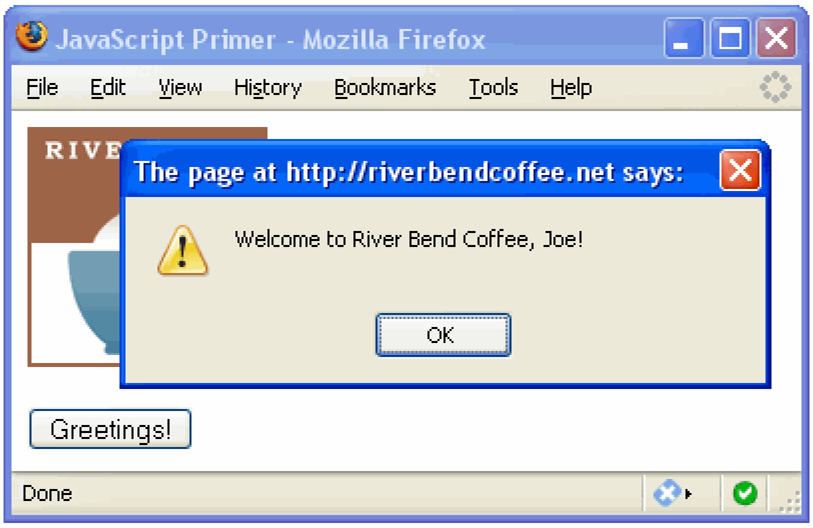
The next method of including JavaScript in a Web page is in an event handler. An *event* is something that occurs in the life of a Web page such as when the page loads or a user clicks a button or hovers over a link. Each event has a handler that allows you to determine what, if anything, happens when these events occur.

Incorporating the getGreeting() function from the JavaScript functions section, the onClick event of a button can be used to display the "Welcome to River Bend Coffee" alert as shown in the following example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JavaScript Primer</title>
<script type="text/javascript">
<!--
function getGreeting(userName) {
  if (userName == null || userName == "") {
    return "Welcome to River Bend Coffee!";
  } else {
    return "Welcome to River Bend Coffee, " + userName + "!";
  }
}
//-->
</script>
</head>
<body bgcolor="#ffffff" text="#000000">
<form action="#" method="get">
```

```
    <img src="images/riverbend_logo.gif" height="120" width="120">
    <br />
    <br />
    <input type="button" value="Greetings!" onclick="alert(getGreeting('Joe'));" />
    <br />
    </form>
    </body>
    </html>
```

Clicking the Greetings! button triggers the onClick event of the button, which in turn, causes the associated JavaScript to execute the following output.


JavaScript Primer - Mozilla Firefox
File Edit View History Bookmarks Tools Help
The page at http://riverbendcoffee.net says:
OK
Greetings!
Done

Welcome to River Bend Coffee, Joe!

Using the onClick event of a button to execute JavaScript

Linked scripts

A third method of incorporating JavaScript in a Web page is through the use of linked scripts. In the preceding code example, the getGreeting() function is defined inside the <head> tag of the HTML page. Instead, it is possible to define the function in an external file and link to the file by specifying its location via the src attribute of the <script> tag as shown in the following example.

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <title>JavaScript Primer</title>
    <!-- link to external JavaScript file -->
    <script type="text/javascript" src="scripts/riverbend.js"></script>
    </script>
    </head>
    <body bgcolor="#ffffff" text="#000000">
    ...
    </body>
    </html>
```

There are several benefits of linking to external script files:

- External script files can be used across multiple Web pages.
- The script file can be updated without editing the HTML documents that link to them.
- Browsers can cache external script files.

Consequently, you should use external script files whenever possible to improve the performance and ease of maintenance of your Web sites.

Advanced features

Now that you understand the basics of JavaScript, the following sections describe advanced features of the language:

- DHTML
- The Document Object Model
- Using JavaScript with HTML
- Working with JavaScript in Domino Designer

DHTML

This page last changed on Mar 31, 2008 by jservais.

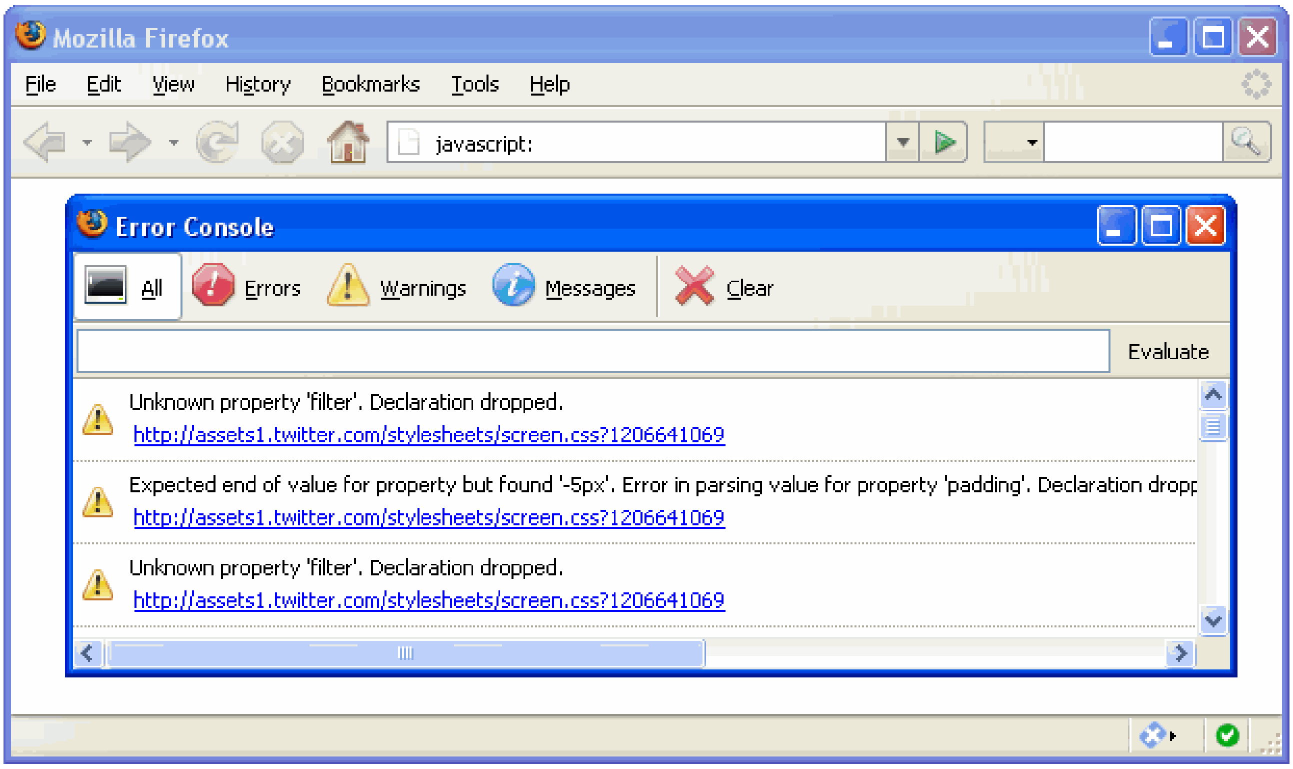
Dynamic HTML (DHTML) is the combined use of HTML, CSS, and JavaScript to create dynamic Web pages.

DHTML involves the manipulation of the HTML Document Object Model (DOM). Consequently, it's rather important your HTML document is well formed:

- Close all container tags such as <p>, <div>, and <li>.
- Always use a DOCTYPE definition.
- Use semantic rather than presentational HTML, i.e., use tags that describe meaning rather than presentation such as <strong> rather than <b>.
- Validate your HTML (the W3C has an often used validator at http://validator.w3.org).

In addition to valid HTML, ensure that you are using CSS for the presentation of your Web pages and you're using external style sheets. HTML gives Web pages structure, CSS should be used to define how they look.

The Dynamic portion of Dynamic HTML is supplied by JavaScript. If you're not familiar with JavaScript, see the JavaScript primer in this wiki. You want the ability to debug your JavaScript on the off chance there is a bug in it. If you're using Firefox, you can launch a JavaScript Console for debugging simply by typing **`javascript:`** in the address bar:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
javascript:
Error Console
All Errors Warnings Messages Clear
Evaluate

```
Unknown property 'filter'. Declaration dropped.
http://assets1.twitter.com/stylesheets/screen.css?1206641069
Expected end of value for property but found '-5px'. Error in parsing value for property 'padding'. Declaration dropp
http://assets1.twitter.com/stylesheets/screen.css?1206641069
Unknown property 'filter'. Declaration dropped.
http://assets1.twitter.com/stylesheets/screen.css?1206641069
```

The built-in JavaScript Console in Firefox

As you can see from the previous example, the JavaScript Console also informs you about errors in your CSS.

The Document Object Model

This page last changed on Mar 31, 2008 by jservais.

The Document Object Model (DOM) is intended to be a platform and language neutral interface for accessing and updating the content, structure, and style of a document. The HTML DOM, along with JavaScript and CSS, comprise Dynamic HTML (DHTML) and allow the dynamic update of a Web page. Historically, each Web browser supported its own DOM implementation, which required that DHTML code be written specifically for each browser that a site supported. The W3C published the DOM Level 3 specification in April 2004 however. It's now possible to write cross-browser DHTML code that works with most current browsers.

As stated above, the DOM provides an interface for accessing and updating the structure of a document. You will often hear the DOM referred to as a *tree*, that contains branches or nodes. The nodes are the structural tags on the page, beginning with the <html> tag. Under the <html> node are the <head> and <body> nodes. The <body> node contains the bulk of the DOM tree, with nodes for tags like <h1>, <div>, and <p>.

Using JavaScript with HTML

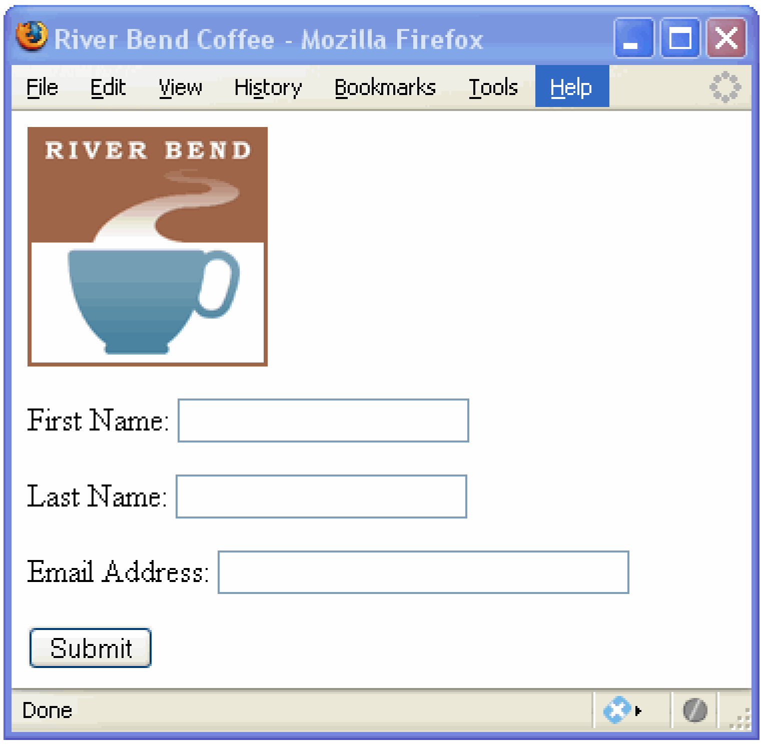
This page last changed on Mar 31, 2008 by jservais.

JavaScript is most often used in conjunction with HTML to perform client-side data validation. However, JavaScript also supplies the Dynamic portion of Dynamic HTML (DHTML), along with CSS and the HTML DOM.

Field validation

JavaScript can be used to validate data entered in an HTML form before the form is submitted to the server. Note that using JavaScript in this manner is strictly a usability enhancement of the form and should not be considered a security mechanism.

The following figure shows a simple HTML form that contains fields for user input.



First Name:

Last Name:

Email Address:

A simple HTML form with fields for user input

If the EMail Address field on the form is a required field, JavaScript can be used to ensure the user enters a value in the field. The following code example shows the markup for the simple form shown in the previous figure. When the form is submitted, the validate_form() function is called in the onSubmit event of the form. The validate_form() function in turn calls the required_field() function, both of which are defined within the <head> of the page. In this example, validate_form() is only checking to make sure the EMail Address field is not empty. However the function could also be used to check the format of text entered in the Email Address field, as well as the values of other fields on the form.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>River Bend Coffee</title>
<script type="text/javascript">
function required_field(field, alerttext) {
  with (field) {
    if (value == null || value=="") {
          alert(alerttext);return false;
        } else {
          return true
        }
  }
}

function validate_form(thisform) {
  with (thisform) {
    if(required_field(EMail, "E-mail is a required field.") == false) {
          email.focus();
          return false;
        }
  }
}
</script>
</head>

<body>
<!-- when the form is submitted, call the validate_form() function -->
<form onsubmit="return validate_form(this)" action="#" method="post">
<img src="working/JavaScript+Primer_files/riverbend_logo.gif" height="120" width="120">
<p><label for="FirstName">First Name:</label> <input type="text" name="FirstName"
id="FirstName" size="20"></p>
<p><label for="LastName">Last Name:</label> <input type="text" name="LastName" id="LastName"
size="20"></p>
<p><label for="EMail">Email Address:</label> <input type="text" name="EMail" id="EMail"
size="30"></p>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Note that the validate_form() and required_field() functions are defined within the <head> of the page only so that the example can be contained in a single file. In practice, it is better to define all of your JavaScript functions in an externally linked file. See the JavaScript primer for more details on linked scripts.

This page last changed on Mar 31, 2008 by jservais.

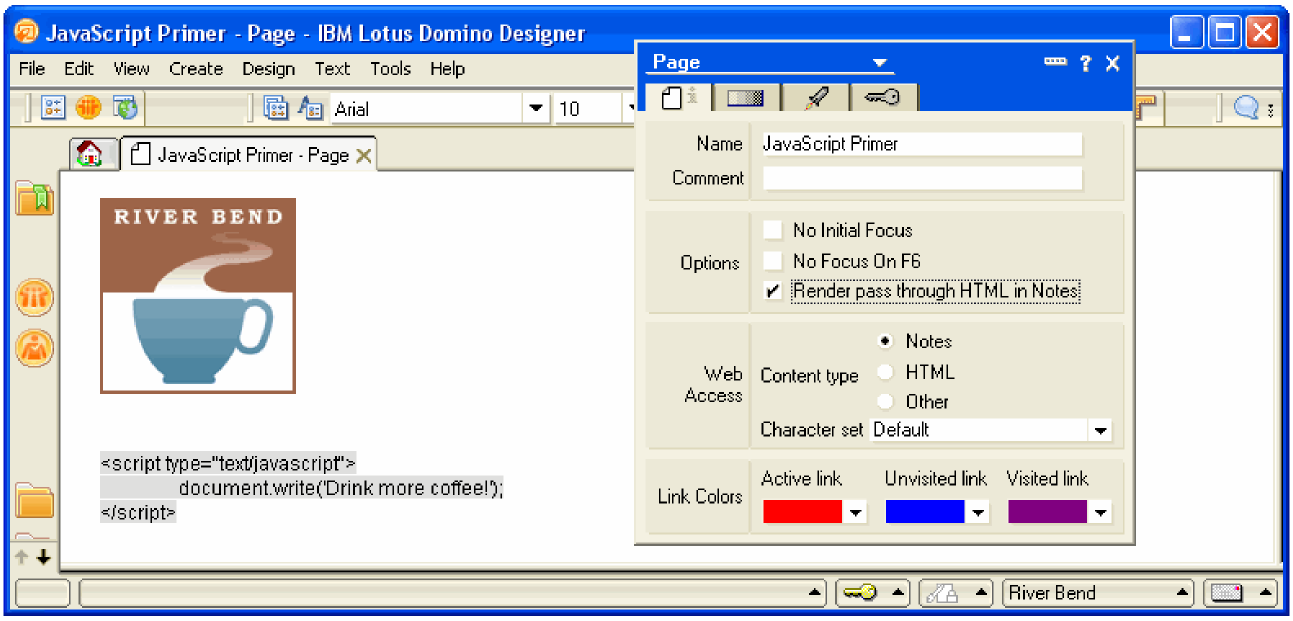
Including JavaScript on a Notes form or page

If you've never used JavaScript in a Lotus Notes application before, you can add it to a Notes form or page just as you would a Web page. As long as the JavaScript you add to the form or page is supported by the Notes client, it works just like it does in a browser.

The <script> tag

There are a several different ways to include JavaScript in a Lotus Notes form or page. One way is through the use of the <script> tag. You can place the <script> tag anywhere on the form or page and the JavaScript inside the tag will be executed as the page or form is loaded. In order for the JavaScript to be used this way to properly execute in the Notes client, the following criteria must be met:

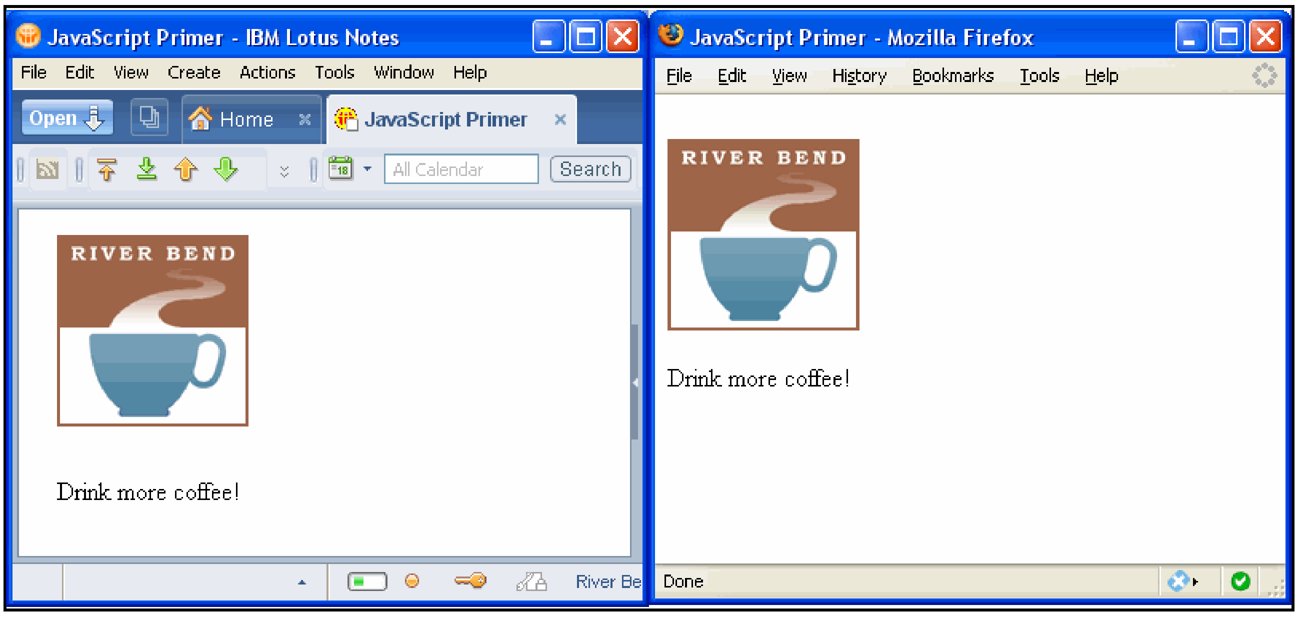
- The "Enable JavaScript" option in the Notes Client Configuration must be selected.
- The <script> tag and its contents must be marked as pass-thru HTML.
- The form that contains the <script> tag must have the "Render pass through HTML in Notes" option selected on the form or page Properties box.


JavaScript Primer - Page - IBM Lotus Domino Designer
File Edit View Create Design Text Tools Help
Arial
10
JavaScript Primer - Page
RIVER BEND
Page
River Bend

```
<script type="text/javascript">
        document.write('Drink more coffee!');
</script>
```

Name: JavaScript Primer
Comment:
Options: No Initial Focus; No Focus On F6; [✓ Render pass through HTML in Notes]
Web Access: Content type [✓ Notes] HTML Other
Character set: Default
Link Colors: Active link 🟥 Unvisited link 🟦 Visited link 🟪

The JavaScript code contained in the example above produces a similar result in both Lotus Notes and a Web browser as shown in the following figure.


JavaScript Primer - IBM Lotus Notes
File Edit View Create Actions Tools Window Help
Open
Home
JavaScript Primer
All Calendar
Search
RIVER BEND
Drink more coffee!
River Be
JavaScript Primer - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIVER BEND
Drink more coffee!
Done

Event handlers

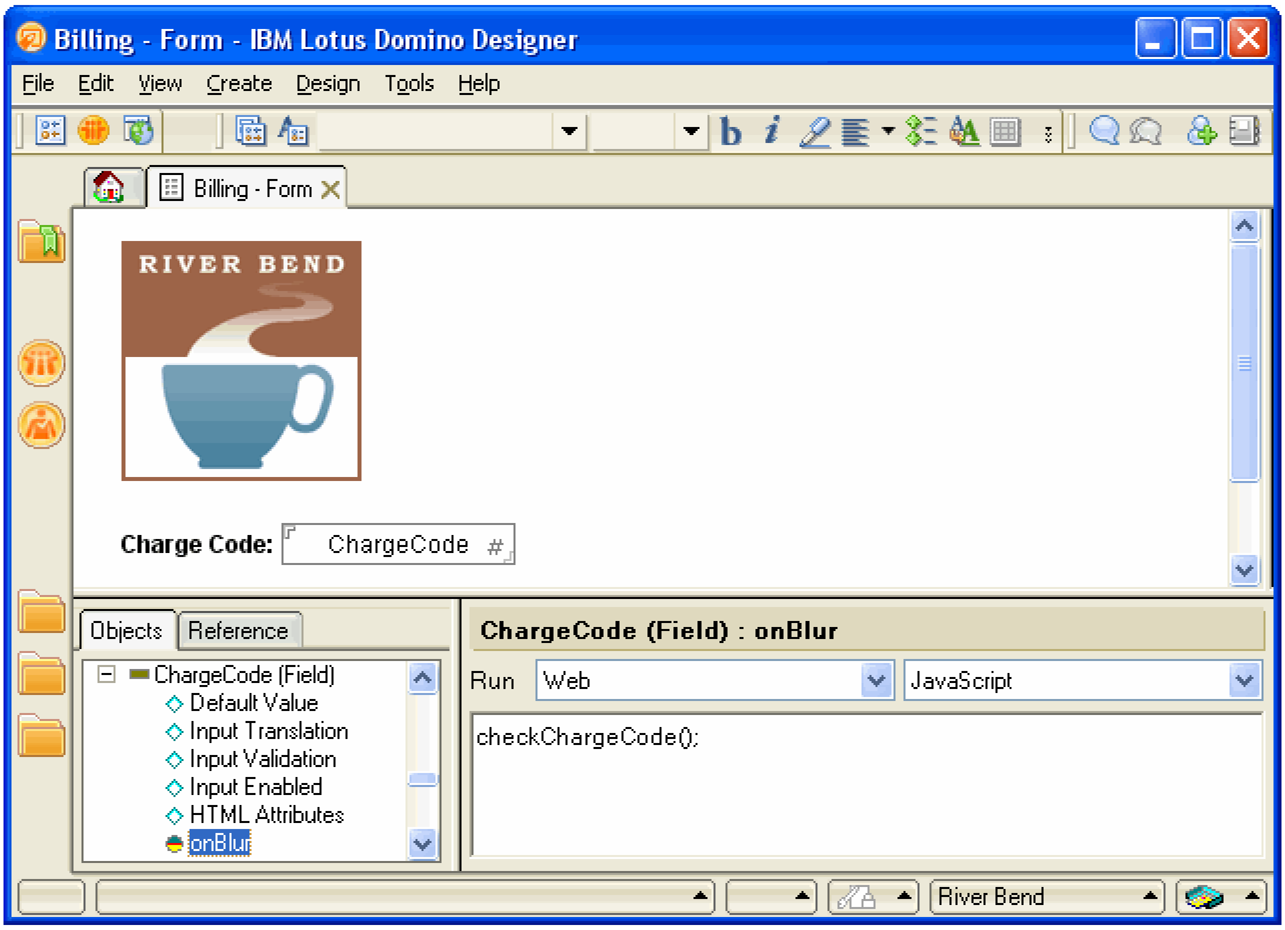
Another method of including JavaScript in a Lotus Notes form or page is in an event handler. An *event* occurs in the life of a Notes form or page such as when the page loads or a user clicks an action hotspot. Each event has a handler that allows you to determine what, if anything, happens when these events occur. Several Domino objects have JavaScript event handlers:

- Form
- Subform
- Page
- Field
- Action
- Button
- Action hotspot

If you want to execute the checkAnswer() function when leaving a field on a Web page, you add the event handler to the markup for the field as shown in the following example:

```
<input type="text" id="ChargeCode" onblur="checkChargeCode();">
```

The same functionality can be achieved in Domino Designer by selecting the onBlur event of the field in the Objects tab and adding the function to the script area as shown in the following figure.



Charge Code: ChargeCode

ChargeCode (Field) : onBlur

Run: Web, JavaScript

```
checkChargeCode();
```

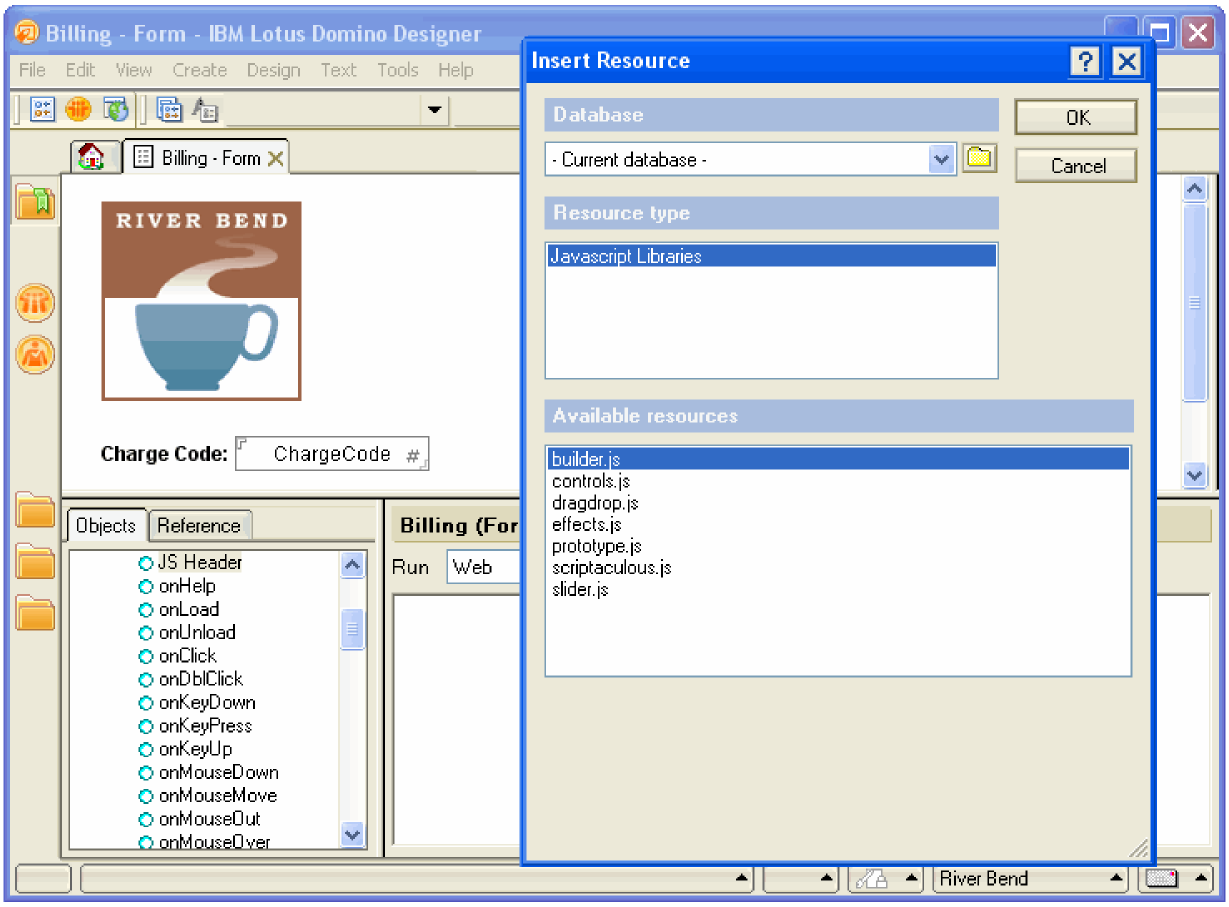
Coding the onBlur event handler for a field in Domino Designer

JavaScript library

Another method of incorporating JavaScript in a Notes form or page is through the use of a JavaScript library. The Linked Scripts section of the JavaScript primer discusses the benefits and creation of external JavaScript files. In Domino Designer, these files are referred to as *JavaScript libraries*.

A JavaScript library is created in the Script Libraries area of database in the Design pane of Domino Designer, which is the same area that LotusScript and Java libraries are created and accessed. After you create a JavaScript library that contains the variables and functions that you want to reference in your Notes page or form:

1. Open the form and place your cursor in the JS Header event.
2. Select **Create > Resource > Insert Resource** from the menu in Domino Designer.
3. In the Insert Resource dialog box, select **Javascript Libraries** under Resource type.
4. Select the JavaScript library that you want to include in the form and click **OK**.



Billing - Form - IBM Lotus Domino Designer

Insert Resource

Database: - Current database -

Resource type: Javascript Libraries

Available resources:

builder.js
controls.js
dragdrop.js
effects.js
prototype.js
scriptaculous.js
slider.js

Charge Code: ChargeCode

Selecting the JavaScript library that you want to include in the form in the Insert Resource window

Styles and CSS primer

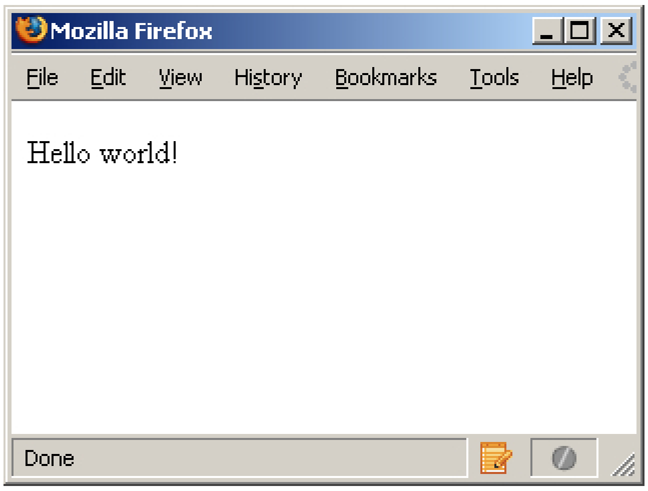
This page last changed on Apr 04, 2008 by dalandon.

What are styles and CSS?

In an effort to separate structural markup and data from user interface design, today's Web Development Standards suggest that we use styles and style sheets in our Web development efforts. Style sheets and cascading style sheets (CSS) (Content type: text/css) are documents that define the visual preferences for markup, and how such markup should be formatted when viewed via specific clients or at various media states. These pages are written in a *language* commonly referred to as *CSS*.

To properly understand how Styles effect the visual formatting of our markup, we must first understand how our standard non-styled markup renders in a Web browser client. For example, let us look at a non-styled <span> element is rendered in a Web Browser client.

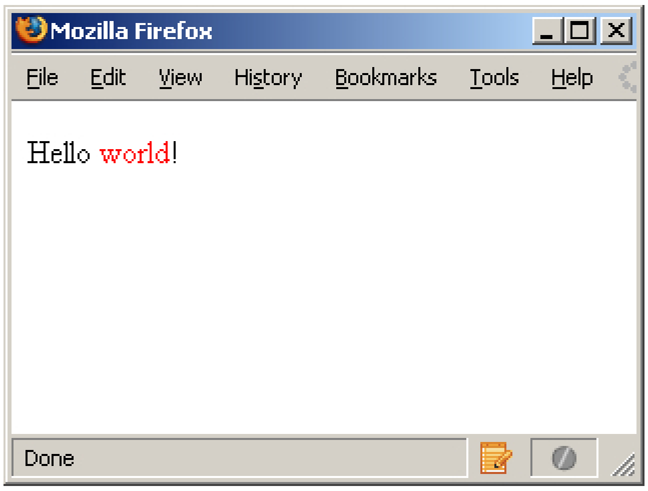
```
<\!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
 </head>
 <body>
  <p>Hello <span>world</span>!</p>
 </body>
</html>
```



Hello world!

A <span> element is designed to act as an inline content container and has no inherent visual element enhancers. With styles however, we can expand the visual element. For this example, we change the color of the <span> element by using the *style* attribute of the element.

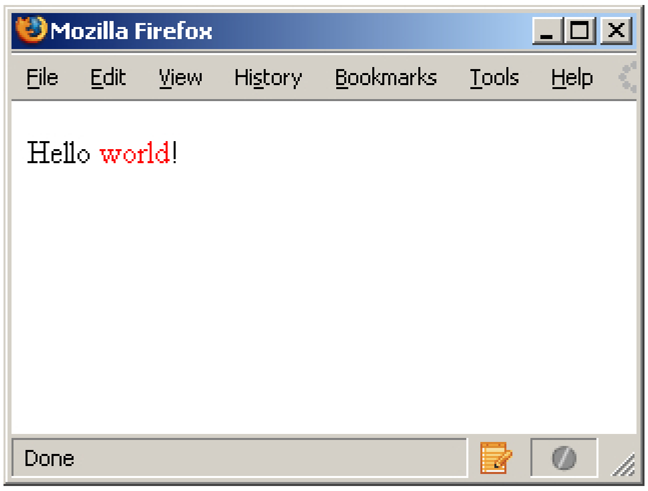
```
<\!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
 </head>
 <body>
  <p>Hello <span style="color: red;">world</span>!</p>
 </body>
</html>
```



Hello world!

The utilization of inline style attributes does not truly separate data content from the visual rendering, which is the main reason that we are looking at using styles in our development practices. Therefore, let us expand on the element styling capabilities and review the <style> tag.

```
<\!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
  <style>
   span { color: red; }
  </style>
 </head>
 <body>
  <p>Hello <span>world</span>!</p>
 </body>
</html>
```



Hello world!

While we can see no change in the rendered end result in the Web browser client, by adding the <style> element to the <head> element of our example Web page, we have set that any <span> element content will be in the color red, thus successfully separating content from the visual user interface.

Element, ID, and class assignments

In the above example, the <style> element declaration for <span> element styling would impact *all* <span> elements in the Web page. In this section, we discuss how we can pinpoint the styling of elements via element, named ID, and named class assignments.

Style selectors

A style selector is a named element, element ID, or element class that a style sheet can use in the maintenance of specific style properties. Selectors may contain both general properties and properties specific to their given element capabilities.

Selector	Selector usage examples

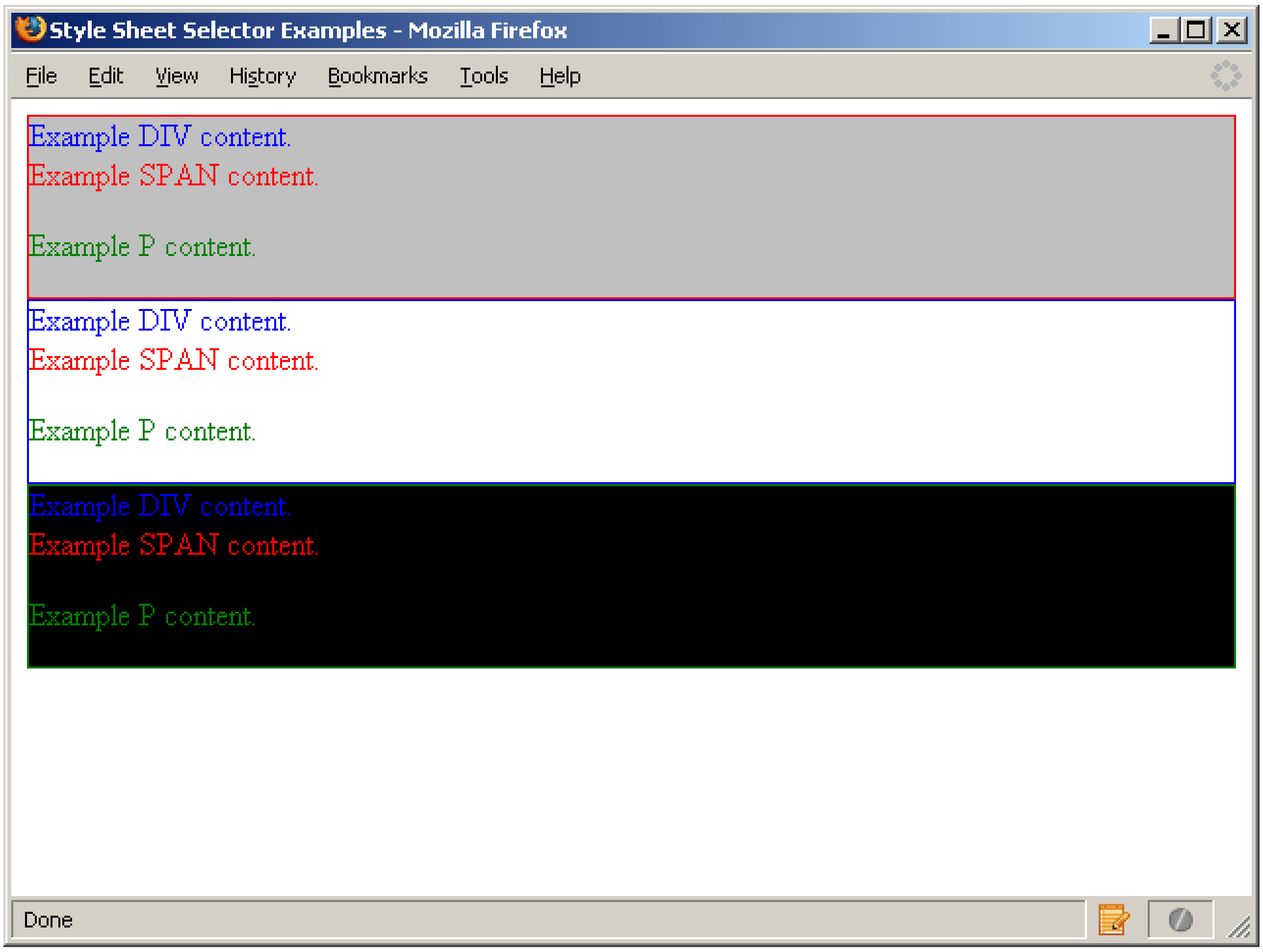
Named Element	span { color: red; }
	div { color: blue; }
	p { color: green; }

Element ID	#ID1 { background-color: silver; }
	#ID2 { background-color: white; }
	#ID3 { background-color: black; }

Element Class	.section1 { border: 1px solid red; }
	.section2 { border: 1px solid blue; }
	.section3 { border: 1px solid green; }

If we take these selector examples and create a simple XHTML page that leverages said examples (see the following code example and figure), you can see how named element, element ID, and element class selectors can be used to provide visual formatting to your markup content.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
 <title>Style Sheet Selector Examples</title>
 <style>
  span { color: red; }
  div { color: blue; }
  p { color: green; }
  #ID1 { background-color: silver; }
  #ID2 { background-color: white; }
  #ID3 { background-color: black; }
  .section1 { border: 1px solid red; }
  .section2 { border: 1px solid blue; }
  .section3 { border: 1px solid green; }
 </style>
</head>
<body>
 <div id="ID1" class="section1">
  <div>Example DIV content.</div>
  <span>Example SPAN content.</span>
  <p>Example P content.</p>
 </div>
 <div id="ID2" class="section2">
  <div>Example DIV content.</div>
  <span>Example SPAN content.</span>
  <p>Example P content.</p>
 </div>
 <div id="ID3" class="section3">
  <div>Example DIV content.</div>
  <span>Example SPAN content.</span>
  <p>Example P content.</p>
 </div>
</body>
</html>
```


Style Sheet Selector Examples - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Done

Example DIV content.
Example SPAN content.

Example P content.

Example DIV content.
Example SPAN content.

Example P content.

Example DIV content.
Example SPAN content.

Example P content.

Advanced selectors

The usage of style selectors can give you complete control over the visual representation of every markup element, one you have made a conscience effort to architect our markup to best facilitate more advanced usage of selectors. To do this, we must first understand the more advanced capabilities of selectors.

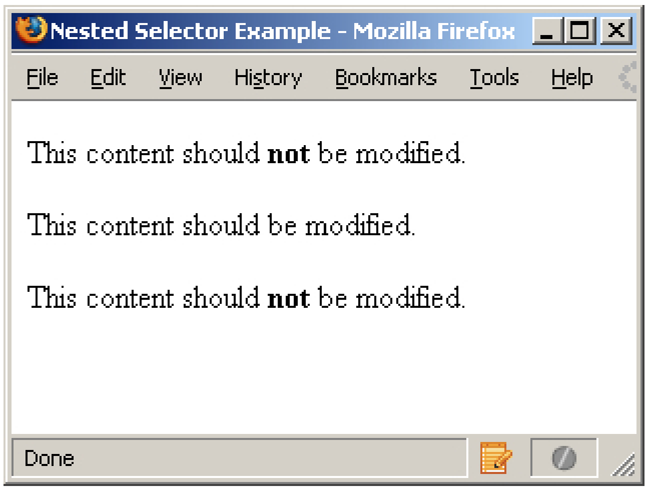
Nested selectors

In *CSS*, we have the ability to create *nested selectors*. A nested selector is a CSS declaration that uses a combination of named elements, element IDs, and/or element class names to traverse the markup to affect specific target markup elements. The following simple example shows _nested selectors_, and their impact on the target markup element.

```
div#content div.section p {
 font-weight: normal;
 margin: 0px;
 padding: 0px;
 color: red;
}
```

The above CSS is designed to only affect the paragraph (<p>) tag element that is located in the DIV (<div>) tag element with a Class Name of *section* which is located in the DIV (<div>) tag element with an ID of *content*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
 <title>Nested Selector Example</title>
 <style>
  div#content div.section p {
   font-weight: normal;
   margin: 0px;
   padding: 0px;
   color: red;
  }
 </style>
</head>
<body>
 <div id="header">
  <div class="section">
   <p>This content should <strong>not</strong> be modified.</p>
  </div>
 </div>
 <div id="content">
  <div class="section">
   <p>This content should be modified.</p>
  </div>
 </div>
 <div id="footer">
  <div class="section">
   <p>This content should <strong>not</strong> be modified.</p>
  </div>
 </div>
</body>
</html>
```

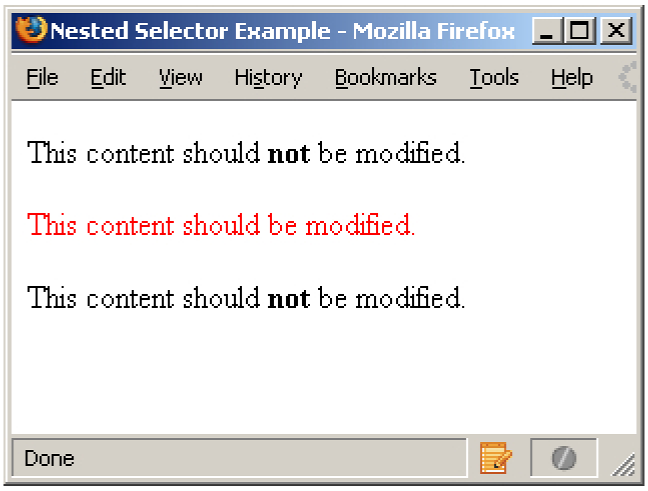

Nested Selector Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Done

This content should **not** be modified.

This content should be modified.

This content should **not** be modified.

Without CSS style


Nested Selector Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Done

This content should **not** be modified.

This content should be modified.

This content should **not** be modified.

With applied CSS style

By using named elements, element IDs, element class names, and CSS nested selectors, developers can both architect their solution in compliance with current Web standards primer and move toward giving their applications Web 2.0-class functionality.

Using media types

What are CSS media types

With CSS media types, you can specify which particular style sheet to apply to your markup based on the given access medium. The following table outlines the different media types and examples of their usage mediums.

Media type	Description
all	Global/all media types and devices
aural	Sound and speech synthesizers
braille	Braille tactile feedback devices
embossed	Braille printer media
handheld	Mobile/handheld devices
print	Printed medium
projection	Projected media and devices
screen	Computer-specific media (ie., Web browser client)
tty	Designed for fixed-pitch character grid media (ie., teletypes and terminals)
tv	Television-type devices

By understanding and properly using media types in both your markup architecture and your CSS design, you can completely control how the content of your Web pages is displayed across any media type.

Using CSS media types

There are currently two ways to use CSS media types: @Import/@Media rules or the *media* attribute of the <link> element.

The @Import rule

The following code showcases the proper usage schema for using *CSS* via the @Import rule.

```
@import url("example.css") screen;
```

The @Media rule

The following code showcases the proper usage schema for using *CSS* via the @Media rule.

```
@media screen {
 body { font-size: 10pt }
}
```

The <link> element *media* attribute

The following code showcases the proper usage schema for using *CSS* via the *media* attribute of the <link> element.

```
<link rel="stylesheet" type="text/css" media="screen" href="example.css">
```

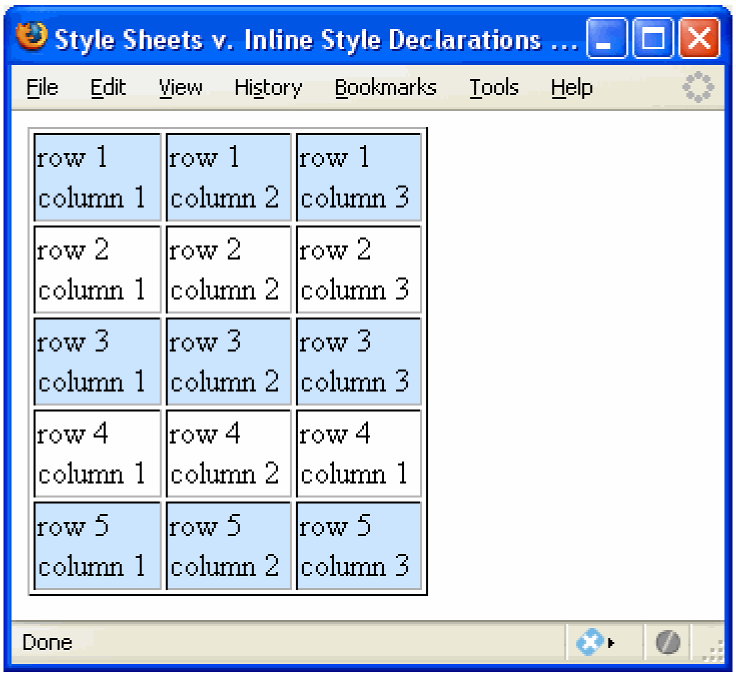
Style sheets versus inline style declarations

As was mentioned in the What are Styles and CSS? section, utilizing an inline style declaration does not separate the data content of a Web page from its design. This is bad for a number of reasons that are covered in detail in the Web standards primer primer, but chief among them is the impact on maintenance.

As a simple example, the following markup example uses inline style declarations to give an HTML table alternating row colors. The background color of the alternating rows is specified inside, or inline, of the HTML markup.

```
<table width="200" border="1">
  <tr style="background-color: #CCE5FF"> <!-- inline style -->
    <td>row 1 column 1</td>
    <td>row 1 column 2</td>
    <td>row 1 column 3</td>
  </tr>
  <tr>
    <td>row 2 column 1</td>
    <td>row 2 column 2</td>
```

```
        <td>row 2 column 3</td>
      </tr>
      <tr style="background-color: #CCE5FF"> <!-- inline style -->
        <td>row 3 column 1</td>
        <td>row 3 column 2</td>
        <td>row 3 column 3</td>
      </tr>
      <tr>
        <td>row 4 column 1</td>
        <td>row 4 column 2</td>
        <td>row 4 column 1</td>
      </tr>
       <tr style="background-color: #CCE5FF"> <!-- inline style -->
        <td>row 5 column 1</td>
        <td>row 5 column 2</td>
        <td>row 5 column 3</td>
      </tr>
    </table>
```

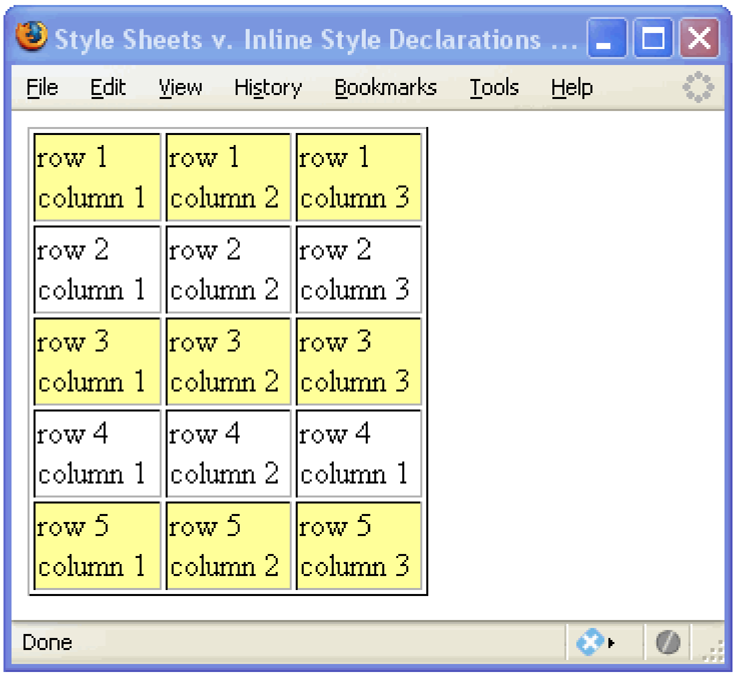

Style Sheets v. Inline Style Declarations ...
File Edit View History Bookmarks Tools Help
Done

row 1 column 1	row 1 column 2	row 1 column 3
row 2 column 1	row 2 column 2	row 2 column 3
row 3 column 1	row 3 column 2	row 3 column 3
row 4 column 1	row 4 column 2	row 4 column 1
row 5 column 1	row 5 column 2	row 5 column 3

If the alternating row color must be changed from blue to yellow, it requires editing the page in three places. In an example like this, such a task is easily accomplished. If this table was larger or there were inline styles for each <td> tag that needed to be modified, the updates would become significantly more cumbersome.

Again, as was demonstrated in the What are Styles and CSS? section, we can simplify the maintenance of this page by assigning the alternating rows of the table a class and creating an embedded style sheet in the head of the HTML document using the <style> tag. Inside the embedded style sheet, we can create a selector for the class assigned to the alternate table rows and set the background-color property of the selector to the desired color. If the color of the alternate table rows needs to be modified at some point in the future, the change now only needs to be made inside the embedded style sheet rather than to each table row.

```
    ...
    <!-- embedded style sheet -->
    <style type="text/css">
        .alternateTableRow {
            background-color: #FFFF99;
        }
    </style>
    ...
    <table width="200" border="1">
      <tr class="alternateTableRow"> <!-- inline style replaced by class -->
        <td>row 1 column 1</td>
        <td>row 1 column 2</td>
        <td>row 1 column 3</td>
      </tr>
      <tr>
        <td>row 2 column 1</td>
        <td>row 2 column 2</td>
        <td>row 2 column 3</td>
      </tr>
      <tr class="alternateTableRow"> <!-- inline style replaced by class -->
        <td>row 3 column 1</td>
        <td>row 3 column 2</td>
        <td>row 3 column 3</td>
      </tr>
      <tr>
        <td>row 4 column 1</td>
        <td>row 4 column 2</td>
        <td>row 4 column 1</td>
      </tr>
       <tr class="alternateTableRow"> <!-- inline style replaced by class -->
        <td>row 5 column 1</td>
        <td>row 5 column 2</td>
        <td>row 5 column 3</td>
      </tr>
    </table>
```



row 1 column 1	row 1 column 2	row 1 column 3
row 2 column 1	row 2 column 2	row 2 column 3
row 3 column 1	row 3 column 2	row 3 column 3
row 4 column 1	row 4 column 2	row 4 column 1
row 5 column 1	row 5 column 2	row 5 column 3

What if there are several pages in a Web site and each of them contains a table that should be styled consistently? We can take this example a step further and move the embedded style sheet to an external file. Separating HTML documents and style sheets has several benefits:

- External style sheets can be used across multiple Web pages.
- The styles can be updated without editing the HTML documents that import or link to them.
- Style sheets can be loaded based on the media type used to view the page.

Moving an embedded style sheet to an external file is as simple as cutting and pasting the content between the <style> tags into a new text document and saving it with a .css extension. The HTML document can then link to the style sheet by replacing the <style> tag with a <link> tag that specifies the path to the style sheet.

```
<link href="mystyle.css" rel="stylesheet" type="text/css">
```

By using external style sheets and semantic HTML, you can greatly simply the maintenance of your Web pages in the following ways:

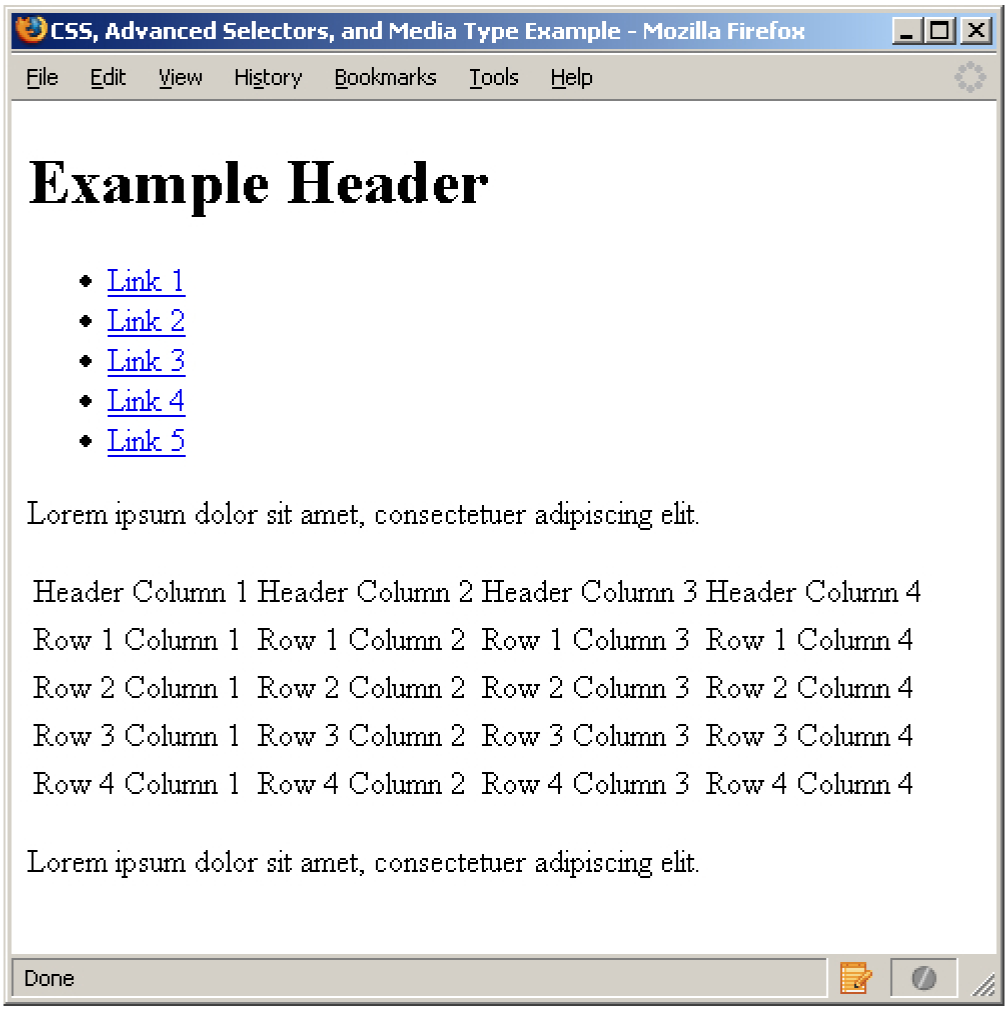
- Making HTML documents more concise and easier to read
- Consolidating all the styling and presentation of the HTML in one place for easier editing
- Allowing you to use the same style sheets across multiple HTML documents

Combining CSS, advanced selectors, and media types

In this section, we review a simple example that showcases the combination of *CSS*, advanced selectors, and media types to facilitate functional web development. The following example shows the simple non-styled Web browser client rendering example XHTML page and markup.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
 <title>CSS, Advanced Selectors, and Media Type Example</title>
</head>
<body>
 <div id="maincontainer">
  <div id="header" class="layoutsection">
   <h1>Example Header</h1>
   <ul id="menu_header">
    <li class="menuitem" id="menu_header_item1"><a href="#" id="menu_header_link1"
class="active">Link 1</a></li>
    <li class="menuitem" id="menu_header_item2"><a href="#" id="menu_header_link2">Link
2</a></li>
    <li class="menuitem" id="menu_header_item3"><a href="#" id="menu_header_link3">Link
3</a></li>
    <li class="menuitem" id="menu_header_item4"><a href="#" id="menu_header_link4" >Link
4</a></li>
    <li class="menuitem" id="menu_header_item5"><a href="#" id="menu_header_link5" >Link
5</a></li>
   </ul>
  </div>
  <div id="body" class="layoutsection">
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
   <table id="bodycontent" class="content">
    <thead>
     <tr class="odd">
      <td class="col0">Header C 1</td>
      <td class="col1">Header C 2</td>
      <td class="col2">Header C 3</td>
      <td class="col3">Header C 4</td>
     </tr>
    </thead>
    <tbody>
     <tr class="odd">
      <td class="col0">R 1 C 1</td>
      <td class="col1">R 1 C 2</td>
      <td class="col2">R 1 C 3</td>
      <td class="col3">R 1 C 4</td>
     </tr>
     <tr class="even">
```

```
          <td class="col0">R 2 C 1</td>
          <td class="col1">R 2 C 2</td>
          <td class="col2">R 2 C 3</td>
          <td class="col3">R 2 C 4</td>
         </tr>
         <tr class="odd">
          <td class="col0">R 3 C 1</td>
          <td class="col1">R 3 C 2</td>
          <td class="col2">R 3 C 3</td>
          <td class="col3">R 3 C 4</td>
         </tr>
         <tr class="even">
          <td class="col0">R 4 C 1</td>
          <td class="col1">R 4 C 2</td>
          <td class="col2">R 4 C 3</td>
          <td class="col3">R 4 C 4</td>
         </tr>
        </tbody>
       </table>
      </div>
      <div id="footer" class="layoutsection">
       <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
      </div>
     </div>
    </body>
    </html>
```


CSS, Advanced Selectors, and Media Type Example - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
Done

Example Header

- Link 1
- Link 2
- Link 3
- Link 4
- Link 5

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Header Column 1	Header Column 2	Header Column 3	Header Column 4
Row 1 Column 1	Row 1 Column 2	Row 1 Column 3	Row 1 Column 4
Row 2 Column 1	Row 2 Column 2	Row 2 Column 3	Row 2 Column 4
Row 3 Column 1	Row 3 Column 2	Row 3 Column 3	Row 3 Column 4
Row 4 Column 1	Row 4 Column 2	Row 4 Column 3	Row 4 Column 4

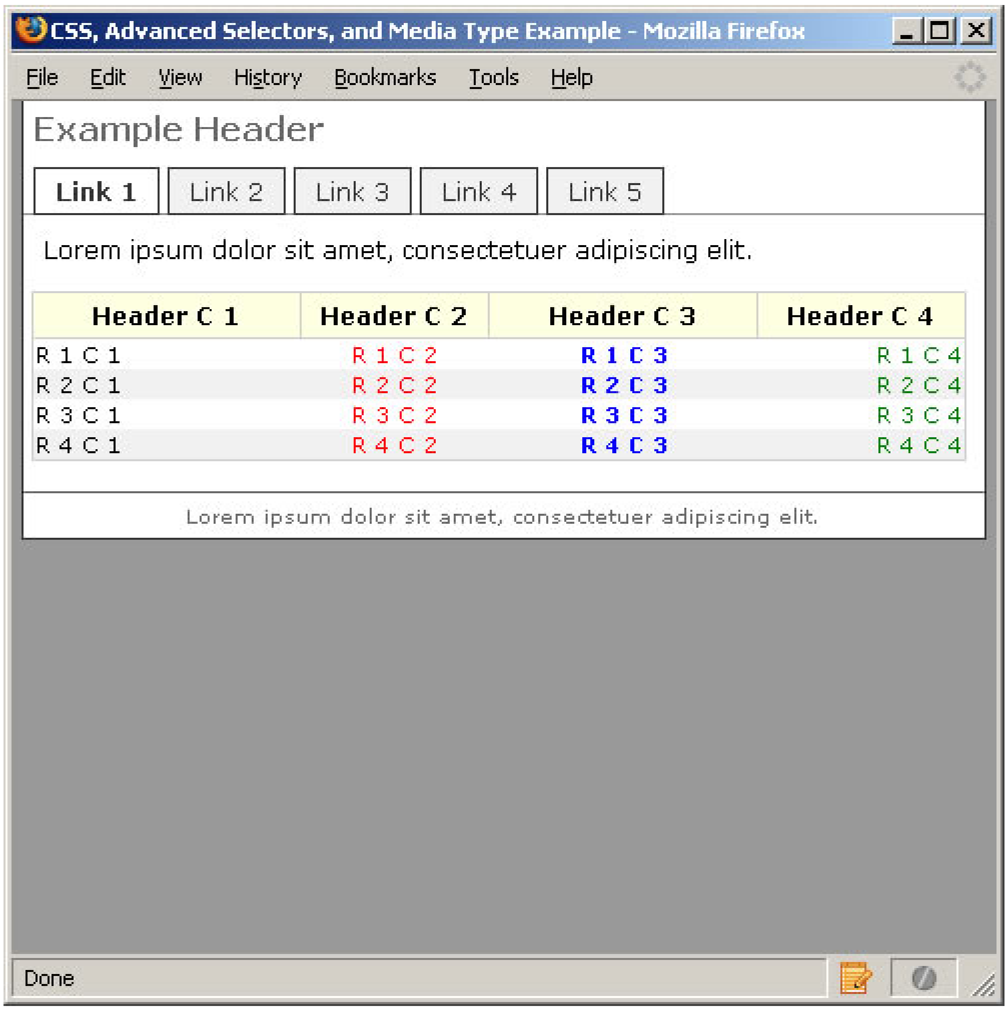
Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

We defined in the example markup explicit element IDs and element class names for all markup elements, which can help us to specifically target grouped and individual elements for style design. For this example, we define both a *screen* and *print* CSS media type. The intended result is a different design on printed materials than what a given user sees when viewing our web page via a Web browser client.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
 <title>CSS, Advanced Selectors, and Media Type Example</title>
 <style>
 @media screen {
  body {
   background-color: #999;
   text-align: center;
   margin: 0px;
   padding: 0px;
   font-size: 75%;
   font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
  }
  body div#maincontainer {
   background-color: #fff;
   text-align: left;
   margin: 0px auto;
   padding: 0px;
   border: 1px solid #333;
   border-top: none;
   width: 480px;
  }
  body div#maincontainer div#header {
   border-bottom: 1px solid #999;
   padding: 0px 0px 4px 0px;
  }
  body div#maincontainer div#header h1 {
   margin: 0px 0px 15px 5px;
   padding: 5px 0px 0px 0px;
   font-size: 12pt;
   color: #666;
  }
  body div#maincontainer div#header ul#menu_header {
   list-style: none;
   margin: 0px 0px 0px 5px;
   padding: 0px;
  }
  body div#maincontainer div#header ul#menu_header li.menuitem {
   display: inline;
  }
  body div#maincontainer div#header ul#menu_header li.menuitem a {
   text-decoration: none;
   color: #333;
   border: 1px solid #333;
   margin: 0px;
   padding: 4px 10px 4px 10px;
   text-align: center;
   background-color: #f1f1f1;
  }
  body div#maincontainer div#header ul#menu_header li.menuitem a.active {
   background-color: #fff;
   font-weight: bold;
  }
  body div#maincontainer div#body p {
   margin: 10px;
   padding: 0px;
  }
  body div#maincontainer div#body table#bodycontent {
   border-collapse: collapse;
   border: 1px solid #cfcfcf;
   margin: 5px 5px 15px 5px;
   width: auto;
  }
  body div#maincontainer div#body table#bodycontent thead tr td {
   background-color: infobackground;
   border-bottom: 1px solid #cfcfcf;
   border-right: 1px solid #cfcfcf;
   text-align: center;
   font-size: 9pt;
   font-weight: bold;
   padding: 4px;
```

```
     }
     body div#maincontainer div#body table#bodycontent tr.even td {
      background-color: #f1f1f1;
     }
     body div#maincontainer div#body table#bodycontent tbody tr td {
      font-size: 8pt;
     }
     body div#maincontainer div#body table#bodycontent td.col0 {
      width: 125px;
     }
     body div#maincontainer div#body table#bodycontent td.col1 {
      width: 85px;
     }
     body div#maincontainer div#body table#bodycontent td.col2 {
      width: 125px;
     }
     body div#maincontainer div#body table#bodycontent td.col3 {
      width: 95px;
     }
     body div#maincontainer div#body table#bodycontent tbody tr td.col1 {
      color: red;
      text-align: center;
     }
     body div#maincontainer div#body table#bodycontent tbody tr td.col2 {
      color: blue;
      font-weight: bold;
      text-align: center;
     }
     body div#maincontainer div#body table#bodycontent tbody tr td.col3 {
      color: green;
      text-align: right;
     }
     body div#maincontainer div#footer {
      border-top: 1px solid #666;
     }
     body div#maincontainer div#footer p {
      text-align: center;
      margin: 5px;
      padding: 0px;
      font-size: 7pt;
      color: #666;
     }
    }
    @media print {
     body {
      font-family: "Courier New", Courier, monospace;
      font-size: 1em;
     }
     body div#maincontainer div#header h1 {
      font-size: 12pt;
     }
     body div#maincontainer div#header ul {
      display: none;
     }
     body div#maincontainer div#body p {
      font-size: 9pt;
     }
     body div#maincontainer div#body table#bodycontent thead tr td {
      font-size: 11pt;
      font-weight: bold;
     }
     body div#maincontainer div#body table#bodycontent {
      border-collapse: collapse;
      font-size: 9pt;
      border: 1px solid #999;
      width: 100%;
     }
     body div#maincontainer div#body table#bodycontent tr td {
      border: 1px solid #999;
      border-right: none;
      border-left: none;
     }
     body div#maincontainer div#footer p {
      color: red;
      font-size: 8pt;
      text-align: center;
     }
    }
```

```
 </style>
</head>
<body>
 <div id="maincontainer">
  <div id="header" class="layoutsection">
   <h1>Example Header</h1>
   <ul id="menu_header">
    <li class="menuitem" id="menu_header_item1"><a href="#" id="menu_header_link1"
class="active">Link 1</a></li>
    <li class="menuitem" id="menu_header_item2"><a href="#" id="menu_header_link2">Link
2</a></li>
    <li class="menuitem" id="menu_header_item3"><a href="#" id="menu_header_link3">Link
3</a></li>
    <li class="menuitem" id="menu_header_item4"><a href="#" id="menu_header_link4" >Link
4</a></li>
    <li class="menuitem" id="menu_header_item5"><a href="#" id="menu_header_link5" >Link
5</a></li>
   </ul>
  </div>
  <div id="body" class="layoutsection">
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
   <table id="bodycontent" class="content">
    <thead>
     <tr class="odd">
      <td class="col0">Header C 1</td>
      <td class="col1">Header C 2</td>
      <td class="col2">Header C 3</td>
      <td class="col3">Header C 4</td>
     </tr>
    </thead>
    <tbody>
     <tr class="odd">
      <td class="col0">R 1 C 1</td>
      <td class="col1">R 1 C 2</td>
      <td class="col2">R 1 C 3</td>
      <td class="col3">R 1 C 4</td>
     </tr>
     <tr class="even">
      <td class="col0">R 2 C 1</td>
      <td class="col1">R 2 C 2</td>
      <td class="col2">R 2 C 3</td>
      <td class="col3">R 2 C 4</td>
     </tr>
     <tr class="odd">
      <td class="col0">R 3 C 1</td>
      <td class="col1">R 3 C 2</td>
      <td class="col2">R 3 C 3</td>
      <td class="col3">R 3 C 4</td>
     </tr>
     <tr class="even">
      <td class="col0">R 4 C 1</td>
      <td class="col1">R 4 C 2</td>
      <td class="col2">R 4 C 3</td>
      <td class="col3">R 4 C 4</td>
     </tr>
    </tbody>
   </table>
  </div>
  <div id="footer" class="layoutsection">
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
  </div>
 </div>
</body>
</html>
```


CSS, Advanced Selectors, and Media Type Example - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
Done

Example Header

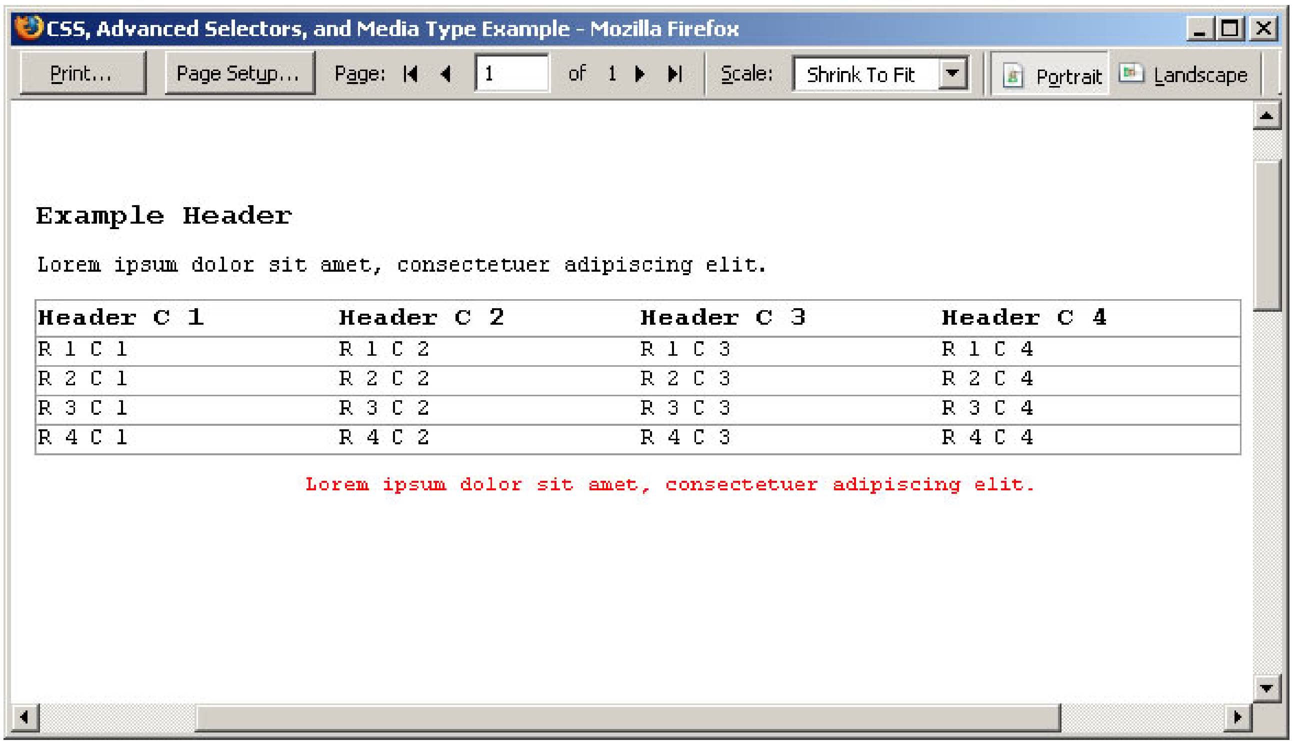
Link 1 | Link 2 | Link 3 | Link 4 | Link 5

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Header C 1	Header C 2	Header C 3	Header C 4
R 1 C 1	R 1 C 2	**R 1 C 3**	R 1 C 4
R 2 C 1	R 2 C 2	**R 2 C 3**	R 2 C 4
R 3 C 1	R 3 C 2	**R 3 C 3**	R 3 C 4
R 4 C 1	R 4 C 2	**R 4 C 3**	R 4 C 4

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Web browser client



Example Header

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Header C 1	Header C 2	Header C 3	Header C 4
R 1 C 1	R 1 C 2	R 1 C 3	R 1 C 4
R 2 C 1	R 2 C 2	R 2 C 3	R 2 C 4
R 3 C 1	R 3 C 2	R 3 C 3	R 3 C 4
R 4 C 1	R 4 C 2	R 4 C 3	R 4 C 4

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Print preview

Content size, caching, and external style sheets

As you can see in the previous examples, styling your markup elements via the in-markup <style> tag element can add more markup to the given Web page. With more markup on our Web pages, we are unnecessarily adding to the content size of the Web page, and not allowing both the HTTP server and the Web browser client to cache common style declarations.

To create a *lighter* markup and allow both the HTTP server and Web browser client to cache any common styles declarations, we can take all of the CSS markup and create two separate style sheets named *screen.css* and *print.css*. We then modify all of our Web pages that need these common style declarations, by using the <link> element method.

In the following table, we show a simple comparison of the above example (which uses the in-markup <style> tag element) and a second example Web page that uses the <link> tag element method, which charts the content size and the ratio of markup-to-displayed content (text).

Web page/markup example	**Total page size**	**Text content**	**Content percentage**
In-markup <style>	6002 Bytes	387 Bytes	6.45%
External Style Sheets	2455 Bytes	387 Bytes	15.76%

As we can see, the usage of external style sheets reduces the Total Page size by more than 50%. Despite this being a simple example, we can immediately see the benefits of using external style sheets in our development archtecture. Decreasing total page size metrics across our applications allow for not only better resource utilization and response times, but also can facilitate the expanse of said applications to low-bandwidth device types, such as mobile device clients.

Another benefit to this approach is the single-point-of-maintenance that this approach affords us when a need to modify the common style declarations across multi-Web page environments.

Web 2.0 primer

This page last changed on Apr 04, 2008 by dalandon.

Prerequisites

- HTML primer
- JavaScript primer
- XML primer
- Web services primer
- Web standards primer

What is Web 2.0?

Web 2.0 is a global community initiative to make thin-client application environments more rich, user-friendly and interactive, resulting in the creation of a Web 2.0 *Standard* and an abandoning of the more archaic Web development practices and conceptions. Instead, Web developers adhering to said *Web 2.0 Standards* are combining Dynamic HTML (DHTML), XML data streams, and new functional architecture methodologies to provide their application user community with more intuitive and fluid user interfaces while maintaining Web development standards, as well as providing the same community with collaborative tools that allow meta-communication and meta-presence.

In this section we discuss the various methodologies and technologies used to achieve what the global community has named Web 2.0, as well as review best practices examples of Web 2.0 applications.

Public or community applications of Web 2.0

Google has, for example, been a pioneer in the Web 2.0 initiative, providing the global user community with fluid rich browser applications that allow said community to collaborate via services ranging from business productivity suites, e-mail, instant messaging, blogging, and personal portal solutions. Each of these services uses technologies and methodologies that are now synonymous with Web 2.0, including REST, SOAP, and AJAX.

Other technology vendors, such as Microsoft, Yahoo, and IBM have also provided Web 2.0 services to the global user community. A perfect example of such a Web 2.0 service is the Best Practices for Building Domino 8 Web Applications wiki that are using here. This and other vendor and community Web 2.0 services allow the community to interact and collaborate in new ways.

Applied Web 2.0 technologies and methodologies can expand current and be used to *renew* existing application development initiatives in order to meet user community expectations.

- Introduction to AJAX
- Introduction to JSON

Introduction to AJAX

This page last changed on Apr 02, 2008 by jservais.

AJAX is term used to describe a group of technologies for developing interactive Web applications. Historically, Web applications have functioned in the following manner:

- A Web browser requests an entire page from a Web server.
- A user clicks a link or submits a form, causing the browser to send a new request to the server.
- The Web server responds by sending another complete page to the browser.

By using AJAX, when a user triggers an event on a page, the browser sends a request to the Web server. Rather than load a completely new page, the currently loaded page is updated with new information. This results in a much smoother experience for the user, more like that of traditional desktop applications than the page-based model of the Web.

Introduction to JSON

This page last changed on Apr 04, 2008 by dalandon.

What is JSON?

JavaScript Object Notation (JSON) is a lightweight data-interchange format that is used to communicate between a browser and a server. It has become popular because it is easy for humans and computers to read and write. It does not depend on any one programming language. Therefore, you can use your favorite languages to work with it. While the format is readable, you can make objects that are very complicated to understand. JSON is a native data format for JavaScript code. Therefore, there is no need to parse it in the browser. The data is readily accessible as objects in your JavaScript code. Retrieving values is as easy as reading from an object property in your JavaScript code.

JSON has the following drawbacks:

- You must know the JSON structure because there isn't a data definition file.
- You cannot t easily navigate or convert the data as you can with XML.

The following code sample shows an example of JSON.

```
{"addressbook": {
         "name": "Bruce Lill",
         "address": {
                 "street":"402 Main Street",
                 "city":"Lees Summit, MO",
                 "zip":"9999"
                 },
    "phoneNumbers": [
                "816 555 4444", "816 555 5555"
                 ]
        }
}
```

JSON structure

JSON can be either an Object or an Array. The *Object* is an unordered set of name/value pairs, while the *Array* is an ordered set of entries. The Object begins with **{** and end with **}** and contains name/value pares are written as name followed by **:** and the value. If there are multiple name/value pares, then they are separated with a **,** as shown in the following example.

```
{"name":"value" , "pet":"Binx"}
```

An Array begins with **[** and ends with **]** and contains values separated with a **,** as shown in the following example.

```
[Value , 1,false,"Lotus",[ 2, 3 ],{"count":23}]
```

A value can be a valid JSON data type.

JSON data types

Type	Sample object
Boolean	"active":true
String	"address": "402 3rd Street"
Positive integers	"quantity": 91
Negative integers	"rating": -123
Floats	"length":122.2344
Scientific notation	"atoms per mole":-6.023e+23
Null	"email":null
array	[value, value, value]
Object	"objname":"objvalue" , "pet":"Binx"
Methods	theMethodName

String information: In JSON, strings must be delimited by the double quotation mark characters. For a list of characters that are allowed and those that must be escaped, refer to the JSON Web site.

Using the JSON object

After you have a JSON object, you can use dot notation or array notation to access its properties.

The following example shows how to use the dot notation.

```
var cusname = JSONdata.cusname;
var homephone = JSONdata.homephone[0];
JSONdata.Method()
```

The following example shows how to use an array notation.

```
var cusname = JSONdata["cusname"];
var homephone = JSONdata["homephone"][0];
JSONdata["Method()"]
```

JSON data is received as a string, usually from an XMLHttpRequest to a server. The server has a URL that returns the request as a string in text or JSON content type. For debugging, you can return plain text so you can see it in the browser. When you set the content type to application/json, you are requested to open or save the file when it is returned from the server.

The LotusScript example is for setting the content type.

```
Print |Content-type: application/json|
OR
Print |Content-type: text/plain|
```

To convert the results from the XMLHttpRequest into an JavaScript object, use the JavaScript **eval()** function. The **eval()** function invokes the JavaScript compiler to convert the string into a Javascript object. Since JSON is a proper subset of JavaScript, the compiler correctly parses the text and produces an object structure.

```
var JSONtext = xmlhttpreq.responseText;
var data = eval("(" + JSONtext + ")");
var top = parseInt(data["@toplevelentries"]);
```

Security and the JSON parser

The Javascript **eval()** can compile and execute any JavaScript program, so there can be security issues (cross-site scripting) when using it with a site on the Internet. When security is a concern, then use a JSON parser instead of **eval()**. The JSON parser only recognizes JSON text and is much safer.

```
var JSONdata = JSON.parse(txt);
```

If you use a library such as Dojo, then use its JSON parser for security. When you use a JSON parser, remember that it expects all object names to be text with double-quotation mark delimitation.

Web services primer

This page last changed on Apr 04, 2008 by dalandon.

Prerequisites

- HTML primer
- XML primer

What are Web services?

Web services are functional predefined, self-contained applications that are available from a Web server. Another application can contact and call the Web service application remotely and receive data in return. A simple example might be a Web server that provides weather information. Another application, regardless of whether it is Web based, can call the Web service with a ZIP code and receive current weather conditions in return.

Web services provide a conduit from an application or application environment to another application or application environment via standard formats and protocols such as XML, SOAP, WSDL, and UDDI. More and more technologies are supporting Web services as today's enterprise-level business solutions require interoperability between technology solutions.

In this section, we review both the basics of Web services as well as further explore the standards-based formats and protocols that Web services employ to seamlessly integrate otherwise often non-integrated solutions.

Examples of commonly used or public Web services

Some of the most popular and commonly used examples of Web services come from Google, which provides the global community with many various Web Service-based solutions. One of the most popular is Google Maps.

The example markup illustrates a simple usage scenario of the Google Maps Web Service. In this example, the *onload* event of the <*body*> element triggers a simple JavaScript function (defined in the <*header*> element above), which makes a Web services call via the predefined Google Maps API. Data is then exchanged between the Google Maps Web Service and our example Web page, rendering a *local area map* result.

Understanding public Web services enables us to eventually architect and develop our own Web services, allowing us to integrate global community solutions and other thir-party technologies and solutions. For the most part, the only thing that is required for learning about and from these public Web services is an eagerness to learn! Most of the public Web services are free (although some may require registration), and the global community does an impressive job maintaining FAQs and tutorials which often showcase general usage examples which can be extremely helpful.

XML and Web services

XML is the basis for Web services, which uses the flexibility and definable structure of the markup language to facilitate the communication between the source and target solutions. Web services often leverage XML, XSL, and XSLT to both consume and return data result sets.

What is SOAP?

SOAP (Content Type: text/xml or application/xml) is a common and standard protocol that allows XML data to be communicated from a source to a target technology. The basic syntax of a SOAP *message* is made up of the elements listed in the following table.

Element name or type	Usage or example
<envelope>	This element identifies the XML document as a SOAP *message*.
<header>	This element contains header information regarding the specific SOAP *message*.
<body>	This element contains the SOAP *message* call and response data.
<fault>	(Optional) This element contains processing error data for the SOAP *message*.

The following example shows the basic structure of a SOAP *message*.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Header>
        </soap:Header>
        <soap:Body>
                <soap:Fault>
                </soap:Fault>
        </soap:Body>
</soap:Envelope>
```

What is WSDL?

Web Services Description Language (WSDL) is a common and standard XML-formatted language that is used for source and target technology interface. The basic syntax of a WSDL document is made up of the elements listed in the following table.

Element name or type	Usage or example
<definitions>	This element indentifies the XML document as a WSDL document.
<porttype>	This element identifies the operations performed by the Web Service.
<message>	This element identifies the message content used by the Web Service.
<types>	This element defines the Content/Data Types used by the Web Service.
<binding>	This element defines the protocols used by the Web Service.

The following example shows the basic structure of a WSDL document.

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
        <import namespace="uri" location="uri"/> *

        <wsdl:documentation .... /> ?
        <wsdl:types> ?
                <wsdl:documentation .... /> ?
                <xsd:schema .... /> *
        </wsdl:types>

        <wsdl:message name="ncname"> *
                <wsdl:documentation .... /> ?
                <part name="ncname" element="qname"? type="qname"?/> *
        </wsdl:message>

        <wsdl:portType name="ncname"> *
                <wsdl:documentation .... /> ?
                <wsdl:operation name="ncname"> *
                        <wsdl:documentation .... /> ?
                        <wsdl:input message="qname"> ?
                                        <wsdl:documentation .... /> ?
                        </wsdl:input>
                        <wsdl:output message="qname"> ?
                                        <wsdl:documentation .... /> ?
                        </wsdl:output>
                        <wsdl:fault name="ncname" message="qname"> *
                                        <wsdl:documentation .... /> ?
                        </wsdl:fault>
                </wsdl:operation>
        </wsdl:portType>

        <wsdl:serviceType name="ncname"> *
                <wsdl:portType name="qname"/> +
        </wsdl:serviceType>

        <wsdl:binding name="ncname" type="qname"> *
                <wsdl:documentation .... /> ?
                <-- binding details --> *
                <wsdl:operation name="ncname"> *
                        <wsdl:documentation .... /> ?
                        <-- binding details --> *
                        <wsdl:input> ?
                                        <wsdl:documentation .... /> ?
                                        <-- binding details -->
                        </wsdl:input>
                        <wsdl:output> ?
                                        <wsdl:documentation .... /> ?
                                        <-- binding details --> *
```

```
                            </wsdl:output>
                            <wsdl:fault name="ncname"> *
                                                    <wsdl:documentation .... /> ?
                                                    <-- binding details --> *
                            </wsdl:fault>
                    </wsdl:operation>
            </wsdl:binding>

            <wsdl:service name="ncname" serviceType="qname"> *
                    <wsdl:documentation .... /> ?
                    <wsdl:port name="ncname" binding="qname"> *
                            <wsdl:documentation .... /> ?
                            <-- address details -->
                    </wsdl:port>
            </wsdl:service>
    </wsdl:definitions>
```

What is UDDI?

Universal Description, Discovery, and Integration (UDDI) is an XML-based directory of Web services described by WSDL. Businesses and organizations can register their WSDL-based Web services in the UDDI to allow their Web services to participate in the global community and to facilitate our development to their specific WSDL standard.

UDDI business registration consists of the types that are listed in the following table.

UDDI type	Examples
White Pages	Address, contact, and known identifiers
Yellow Pages	Industrial categorizations based on standard taxonomies
Green Pages	Technical information about services exposed by the business

For more information on UDDI, visit the UDDI Home Page!

Web standards primer

This page last changed on Apr 04, 2008 by dalandon.

What are Web standards?

The term *Web standards* has generally come to refer to the initiative by the World Wide Web Consortium (W3C) and other groups to promote a common set of technical standards and best practices for Web development. The intent is to provide the greatest number of Web users with usable, accessible content in a manner that simplifies the development, maintenance and longevity of Web pages as new browsers are released and updated.

Historically, the problem has been the release of browsers that incorrectly or do not fully support W3C standards. This has given developers and designers a set of unpleasant choices:

- Create simple pages that use the lowest common subset of standards supported across most browsers.
- Create pages with extraneous markup and code so the page renders as intended across most browsers.
- Create pages for a specific browser or subset of browsers.

In turn, these choices have the following results:

- Pages that lack visual appeal and have limited functionality
- Pages bloated with layers of markup and code that are slower to download and difficult to maintain
- Pages branded with a "best viewed in" caveat (which either advertises the developer's indifference to users with alternate browsers or limited Web development skills)

Unfortunately, users are impacted most by these issues, particularly those users with special needs or disabilities.

Technical standards

The technical specifications that comprise Web standards include, but are not limited to, the following:

- World Wide Web Consortium (W3C) recommendations:
  - HTML 4.0.1
  - XHTML 1.0

    - XHTML 1.1
    - CSS Level 1
    - CSS Level 2
    - CSS Level 3
    - DOM Level 1
    - DOM Level 2
    - DOM Level 3
- European Computer Manufacturers Association (ECMA) Standards:
    - ECMAScript (standardized JavaScript)

As you can see by the version numbers on the list above, technical standards are continuously evolving. A Web page doesn't have to to use the latest specifications to be considered "standards compliant." It's also possible to create a Web page that is technically correct according to the specifications, but that violates most if not all Web design best practices.

Best practices

Just as important as the technical aspect of Web Standards are the best practices.

- Separate content from presentation
- Use a valid DOCTYPE
- Validate your documents

Separate content from presentation

Chief among the Web design practices is the separation the HTML/XHTML markup of the page from the presentation. The presentation consists of the fonts, colors, page layout, etc., and should be set with an external CSS style sheet. This best practice include the following examples among others:

- Use of the <div> tag and CSS instead of tables to layout the design of the page
- Use of the <p> tag instead of <br /> to create paragraphs
- Use of the <strong> tag instead of <b> to make text standout
- Use of the <em> tag instead of <i> to emphasize text
- Use of heading tags (h1, h2, h3, etc.) to designate headings within the document

Why is separating the content from the presentation of a page and using semantic (descriptive) markup a best practice? Web sites that are built this way are much easier to maintain. If the marketing department decides that green is the new black and wants you to update the Web site accordingly, it is much easier to modify an external CSS file than search through all the pages that comprise the site for every <font> tag and change the associated color attribute. An excellent real world example of the power of CSS-based designs is the Web site CSS Zen Garden. Each page of the site uses the same HTML file but achieves a radically different look and feel by referencing a different external CSS file.

Pages that use semantic markup and a CSS-based layout rather than a table-based layout also perform better. Semantic HTML files and their associated style sheet(s) are typically smaller and download faster than HTML files that also contain presentation markup.

Finally, pages that use semantic markup are also easier for machines and developers to read. If the heading of a page is marked up as shown in the following example, the meaning of the markup isn't as clear to a developer who is examining the page as the simple use of an <h1> tag would be.

```
<font size="14px" face="Arial, Helvetica, sans-serif" color="#9E6549" face><b>Welcome to River
Bend Coffee!</b></font>
```

It's also not as clear to search engines or screen readers.

Use a valid DOCTYPE

Another Web Standards best practice is to use a proper document type declaration (DOCTYPE) on your pages. The DOCTYPE is important for a couple of reasons. First, your page won't validate without one. Second, the DOCTYPE tells the Web browser how to render your Web page. Web browsers, with the exception of Opera, have two modes for rendering pages, standards mode and quirks mode. A missing, incomplete, or malformed DOCTYPE causes a Web browser to display a page in quirks mode. A complete, properly formed DOCTYPE triggers standards mode in a browser.

In standards mode, a browser renders a Web page as you designed it, in a standards compliant fashion (more or less). In quirks mode, the Web browser assumes that a Web page is not standards compliant and renders the page as it would have appeared in legacy browsers, before standards specifications existed and were widely adopted. Consequently, you can be less sure of how the browser will display your page when in quirks mode.

The HTML primer of this wiki details proper document type declarations and how to set them in Domino.

Validate your documents

You should always validate your HTML, XHTML, and CSS files. Validation tests markup and CSS documents against the corresponding technical specification for proper implementation and usage. Validation is often a useful method of locating potential issues and ensuring your Web pages will display properly in future browsers.

The W3C provides a HTML/XHTML validation service at http://validator.w3.org and a CSS validation service at http://jigsaw.w3.org/css-validator.

Accessibility

A key component of Web standards is accessibility, which ensures that people with visual, auditory, and physical disabilities can navigate and interact with content on the Web. In 1999, the Web Accessibility Initiative (WAI) of the W3C published the Web Content Accessibility Guidelines 1.0 (WCAG 1.0) to provide designers and developers with guidelines on making Web sites accessible.

In 2001, the W3C published the first draft of WCAG 2.0, although it's still in draft form. There has been some criticism of WCAG 2.0, including the length of time it has taken to develop the document. Consequently, a group of developers known as the WCAG Samurai published a set of errata for WCAG 1.0 for use as an alternative to WCAG 2.0.

Despite the debate regarding WCAG 2.0, much of WCAG 1.0 is considered to be the definitive source for accessibility guidelines. The following is a small subset of the guidelines in WCAG 1.0:

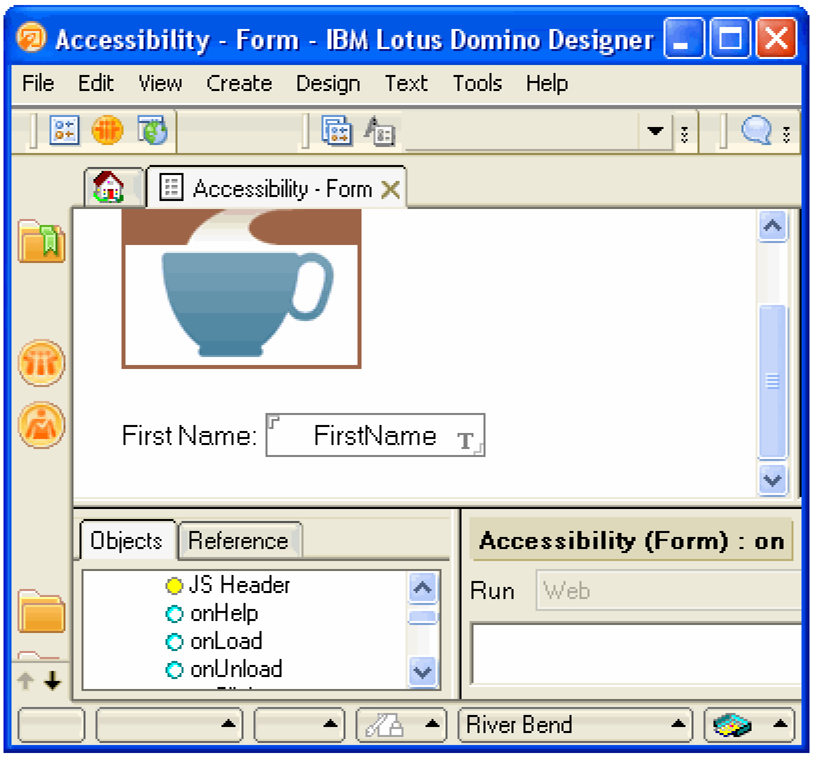
- Use the alt attribute of the <img> tag to provide a text description of the image
- Do not rely solely on color to convey information (e.g., hyper links should be underlined or somehow visually indicated other than use of an alternate color)
- Do not use tables for layout
- Ensure that background and foreground colors have sufficient contrast
- Use the <label> tag for form controls

It is important to be aware of these guidelines for several reasons. The first and obvious reason is so that Web sites you create are accessible to everyone who wants or needs to access them. The second reason is that there is increasingly a legal obligation to make Web sites accessible. A blind man in Australia successfully sued the Sydney Organization Committee of the Olympic Games in 2000, because the Sydney Olympic Games Web site wasn't adequately usable by blind people. In the United States in 2006, a blind student at UC Berkeley filed suit against the retail chain Target because its Web site was not accessible to blind users. In October 2007, the lawsuit became a nationwide class action.

The third reason to be aware of these guidelines is so that you can identify what steps you need to take to make a Domino-based Web site accessible. For example, WCAG 1.0 specifies the use of a <label> tag for form controls. The correct markup for an input field called FirstName with an associated label looks like the following example.

```
<label for="FirstName">First Name:
     <input type="text" id="FirstName" tabindex="1">
</label>
```

However, most Lotus Notes and Domino developers are used to creating fields and labels like those shown in the following figure.


Accessibility - Form - IBM Lotus Domino Designer
File Edit View Create Design Text Tools Help
Accessibility - Form
First Name: FirstName
Objects Reference
JS Header
onHelp
onLoad
onUnload
Accessibility (Form) : on
Run Web
River Bend

The following HTML is created by Domino 8.0.1 for the field on the form above.

```
First Name: <input name="FirstName" value="">
```

As you can see, the markup does not meet the WCAG 1.0 guidelines. Additionally, most Notes developers use tables to layout the design of a form, placing the field label text in one column and the field in the next. This is fine when developing for Notes clients, but if not adapted for the Web, also violates WCAG 1.0.

If you are new to designing for accessibility, Mark Pilgrim has an excellent guide to making Web sites more accessible at http://www.diveintoaccessibility.org .

Useful links: You can learn more about Web standards by visiting the following sites:

- Web Standards Group
- The Web Standards Project

Additional topics

- Application programming interface

Application programming interface

This page last changed on Apr 04, 2008 by dalandon.

What is an API?

An application programming interface (API) is a predefined collection of procedure calls that trigger specific functions in an effort to evaluate or return resultant data or functionality. Essentially, an API is the communication language to a specific technology.

Examples of commonly used or public APIs

Add your examples here.

XML primer

This page last changed on Mar 31, 2008 by jservais.

Prerequisites

- HTML primer

What is XML

Extensible Markup Language (XML) is an open standard metalanguage that developers can use to create and structure data elements. Unlike HTML, XML (Content Type: text/xml or application/xml) does not have a predefined structure of elements, but rather provides the individual developer or development community with the ability to create their own shared structure schema via a document type definition or DTD.

A subset of SGML, XML was developed by the W3C to facilitate the rendering of and cross-environment reading of data elements. The use of XML allows the global community to establish client usage standards for specific data types.

A common example of a globally-adopted standard is the XML-based RSS 2.0, or Really Simple Syndication, that we see on today's news and blogging Web sites. While the contents of each RSS *feed* may vary, developers that adhere to the global community standard for the structure of XML-based RSS can ensure that RSS Reader clients will properly read and display their content.

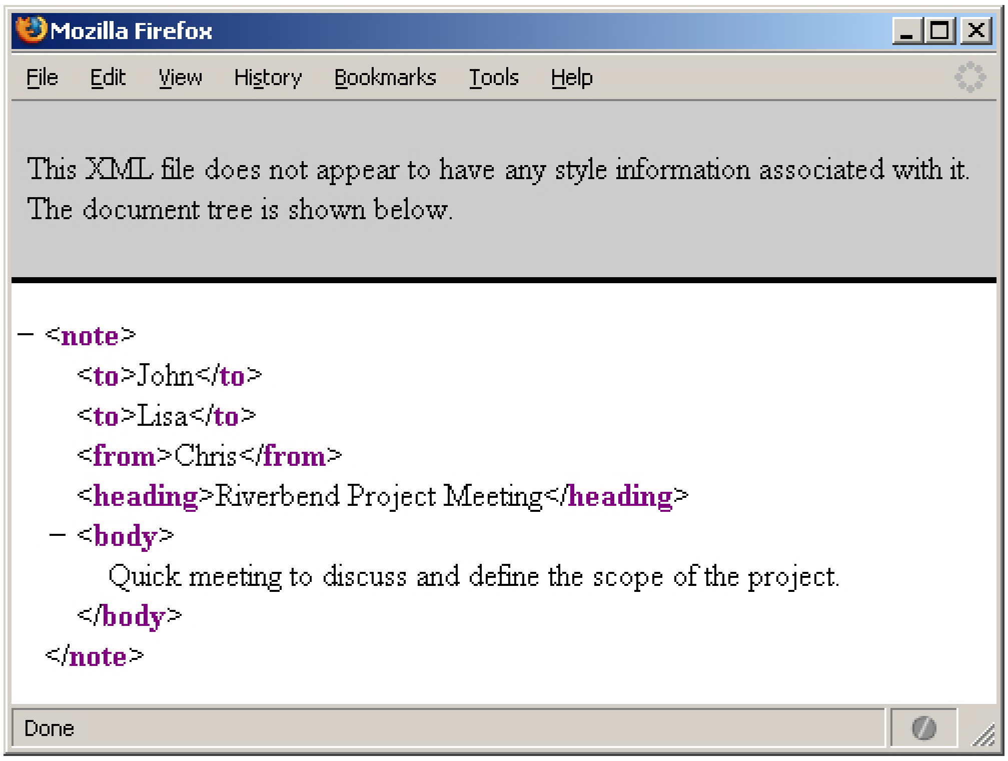
Examples of XML

In the following example, the <note> is an object that contains the elements <to>, <from>, <heading>, and <body>.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <note>
   <to>John</to>
   <to>Lisa</to>
   <from>Chris</from>
   <heading>Riverbend Project Meeting</heading>
   <body>Quick meeting to discuss and define the scope of the project.</body>
  </note>
```

Styling XML

XML is a markup language that is used to structure, and not visually style, document data. As we can see in the following figure, our XML displays a simple *document tree*when viewed via a Web browser client. We can style this XML using several globally-supported methods, which we discuss in the following sections.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <note>
    <to>John</to>
    <to>Lisa</to>
    <from>Chris</from>
    <heading>Riverbend Project Meeting</heading>
  – <body>
      Quick meeting to discuss and define the scope of the project.
    </body>
  </note>
```

Styling XML with CSS

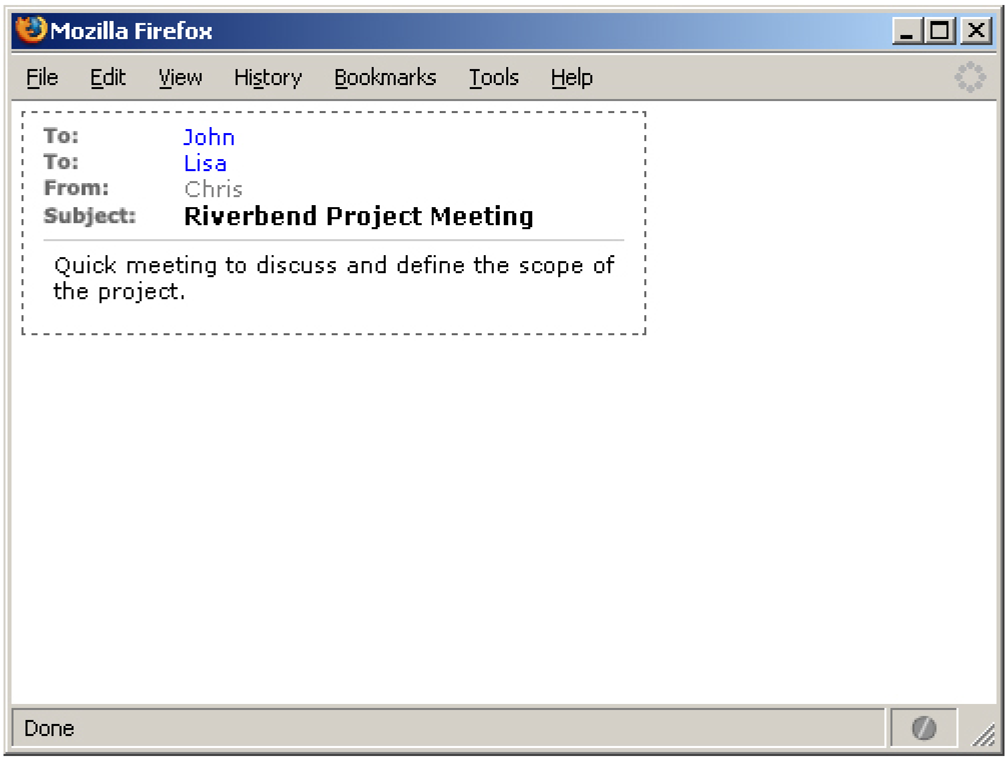
CSS can be used to modify markup to provide a visual interface with document data, and XML-based markup is ultimately no different. We can use CSS element declarations to enhance and visually structure of our XML by defining an external style sheet in our XML markup, as shown in the following example:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="example.css" type="text/css"?>
  <note>
   <to>John</to>
   <to>Lisa</to>
   <from>Chris</from>
   <heading>Riverbend Project Meeting</heading>
   <body>Quick meeting to discuss and define the scope of the project.</body>
  </note>
```

By adding the *<xml-stylesheet>* element, we have defined an external style sheet (*example.css*) that we can use to maintain the visual rendering of the XML markup as shown in the following example:

```
* {
        font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
        font-size: 85%;
}
note, to, from, heading, body {
        display: block;
        background-repeat: no-repeat;
        background-position: left center;
}
note {
        border: 1px dashed #666;
        width: 300px;
        margin: 5px;
        padding: 5px;
}
to {
        color: blue;
        background-image: url(to.jpg);
        padding-left: 75px;
}
from {
        color: Gray;
        background-image: url(from.jpg);
        padding-left: 75px;
}
heading {
        font-weight: bold;
        font-size: 9pt;
        background-image: url(heading.jpg);
        padding-left: 75px;
}
body {
        font-size: 8pt;
        border-top: 1px solid #cfcfcf;
        margin: 5px;
        padding: 5px;
        text-align: justify;
}
```

Each named element in the style sheet controls the visual rendering of its XML markup element counterpart (ie., *heading* controls the *<heading>* markup element, etc.). When the above CSS is applied to the XML markup example, we are given a drastically different visual representation of the XML data as shown in the following figure.



To: John
To: Lisa
From: Chris
Subject: **Riverbend Project Meeting**

Quick meeting to discuss and define the scope of the project.

The *media type* linked CSS attribute can be utilized to better control the visual rendering of XML markup, allowing us to specify different external style sheets for printing, Web browser clients, and handheld or mobile devices.

While the utilization of CSS to control the visual rendering of XML allows us to greatly enhance the presentation of our XML data, it requires that we take the XML markup *as-is*. For example, in order to allow for element labels in the above example while maintaining current browser and Web development standards, we were required to use a combination of *label* images and static element padding.

There is another option that facilitates both an enhancement to the visual rendering of our XML markup while allowing us to conditionally transform our markup at run time without requiring that we modify the actual XML data elements.

Styling XML with XSL, XSL-FO and XSLT

What is XSL

Extensible Stylesheet Language (XSL) (Content Type: text/xsl or application/xsl) is essentially an XML document that defines the presentation structure of an XML data document. The original XML data document uses the XSL document to transform its markup into other formats (such as HTML, XHTML, etc.). XSL was the original proposal to facilitate XML data document formatting, but has since depreciated in favor of XSL-FO and XSLT, which is the current global community standard.

What is XSL-FO and XSLT

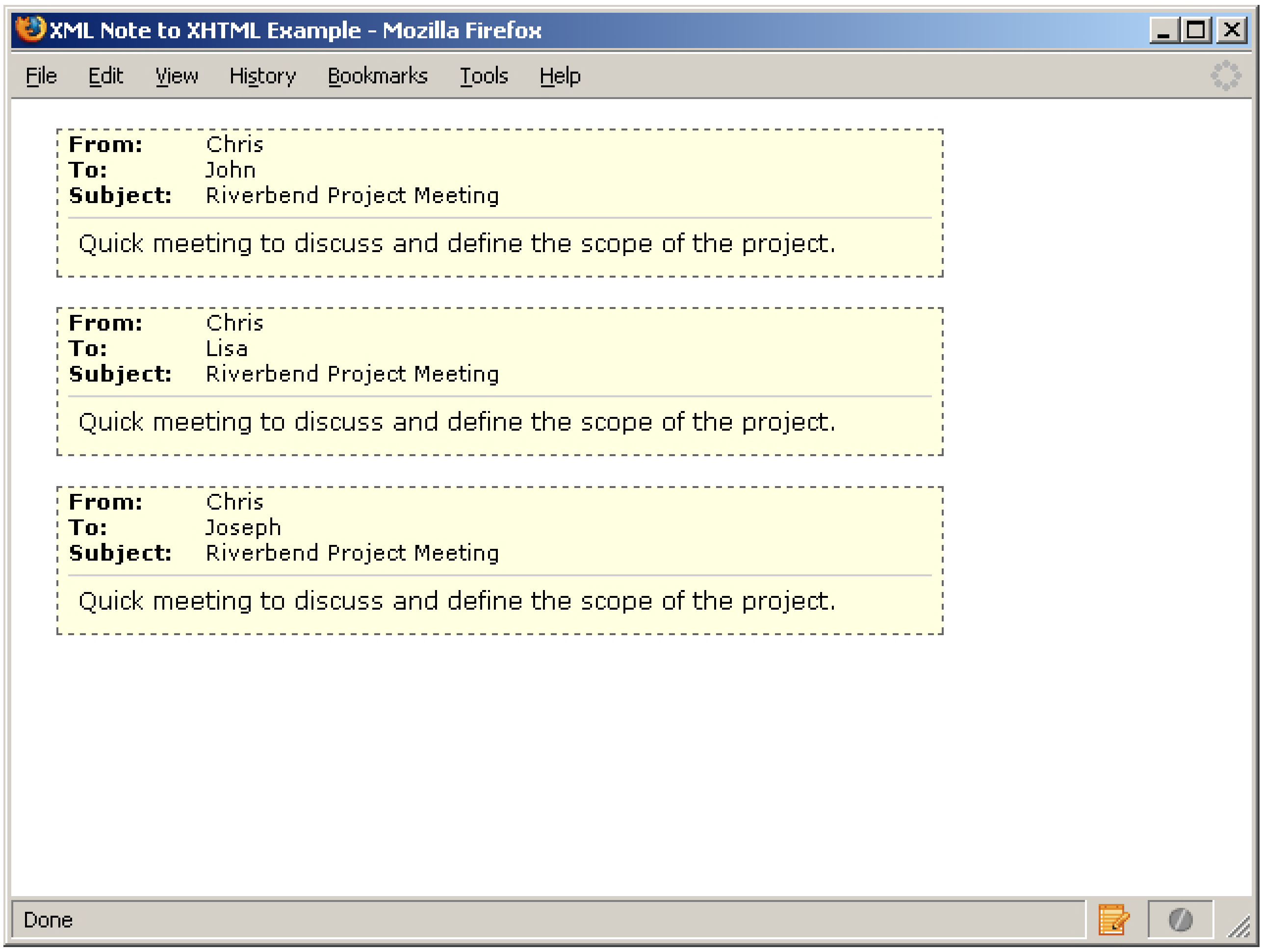
XSL Formatting Object (XSL-FO) and XSL Transformation (XSLT) documents (Content Type: text/xsl or application/xsl) allows us to transform, based on a defined *template* logic, our XML data documents to other formats such as XHTML, SVG, and other standard-based markup languages and formats. XSL-FO and XSLT also allow us to *walk the data* - extracting XML data elements and sub-elements in a manner and method similar to HTML's DOM.

The following example shows a simple XSLT document for the example XML data document from the previous sections:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
        <head>
                <title>XML Note to XHTML Example</title>
                <style>
                        body {
                                font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
                                background-color: #fff;
                        }
                        body div.note {
                                background-color: infobackground;
                                width: 450px;
                                border: 1px dashed #666;
                                margin: 15px;
                                padding: 0px;
                                font-size: 8pt;
                        }
                        body div.note label {
                                float: left;
                                display: block;
                                width: 70px;
                                font-size: 8pt;
                                font-weight: bold;
                                margin: 0px 0px 0px 5px;
                        }
                        body div.note span {
                                display: block;
                                margin-left: 75px;
                        }
                        body div.note span#body {
                                font-size: 9pt;
                                display: block;
                                clear: left;
                                border-top: 1px solid #cfcfcf;
                                margin: 5px;
                                padding: 5px;
                                text-align: justify;
                        }
                </style>
        </head>

        <body>

            <xsl:for-each select="correspondence/note">
                    <div class="note">
                        <label for="from" id="label_from">From:</label>
                                <span class="content" id="from"><xsl:value-of
select="from"/></span>
                        <label for="to" id="label_to">To:</label>
                                <span class="content" id="from"><xsl:value-of
select="to"/></span>
                        <label for="heading" id="label_heading">Subject:</label>
                                <span class="content" id="heading"><xsl:value-of
select="heading"/></span>
                        <label for="body" id="label_body"></label>
                                <span class="content" id="body"><xsl:value-of
select="body"/></span>
                    </div>
            </xsl:for-each>

        </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

To use this XSLT template, we simply define its usage in the second line of our now modified XML data document:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="example_xml_xhtml_xslt.xsl"?>
<correspondence>
        <note>
                <to>John</to>
                <from>Chris</from>
                <heading>Riverbend Project Meeting</heading>
                <body>Quick meeting to discuss and define the scope of the project.</body>
        </note>
        <note>
                <to>Lisa</to>
                <from>Chris</from>
                <heading>Riverbend Project Meeting</heading>
                <body>Quick meeting to discuss and define the scope of the project.</body>
        </note>
        <note>
                <to>Joseph</to>
                <from>Chris</from>
                <heading>Riverbend Project Meeting</heading>
                <body>Quick meeting to discuss and define the scope of the project.</body>
        </note>
</correspondence>
```


XML Note to XHTML Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Done

From: Chris
To: John
Subject: Riverbend Project Meeting

Quick meeting to discuss and define the scope of the project.

From: Chris
To: Lisa
Subject: Riverbend Project Meeting

Quick meeting to discuss and define the scope of the project.

From: Chris
To: Joseph
Subject: Riverbend Project Meeting

Quick meeting to discuss and define the scope of the project.

When the XML data document is now viewed with a Web Browser client, the XSLT template transforms the XML data elements, creating a <*DIV*> XHTML element for each *note* XML element. Tthe XSLT XHTML transformation has the *from* XML element rendering before the *to* XML element. This shows us that information contained in the XML data document can be both conditionally displayed (regardless or order) or even omitted from the transformed markup.

2.0 Getting started

This page last changed on Mar 31, 2008 by jservais.

Topics in this section

Architectural, project, and visual design considerations

This page last changed on Mar 31, 2008 by jservais.

Project considerations

If you have experience building Notes Client applications, you might find that Web projects are similar. Of course, once your users are in a Web client, expectations change dramatically. Your application must compare favorably with the navigation, page flow, and visual experience on many professionally designed Web sites. To do this, you need a good multi-discipled project team. This team usually includes a business client, project manager, and programmers depending on the size of the project.

When building Web applications, a graphics designer with Web experience should also be involved in the project. Graphic designers should not be seen as "button makers", as they so often are in technical projects. Graphics and product designers have been trained in creative exploration techniques to refine and improve user experiences, and you should do your best to take advantage of these skills.

Another key member of the team is the client representative. This could be the client, but most likely a member of the internal team who represents the client in the day-to-day project decisions. This person should not be a technical member of the team but should understand the project goals and constraints. The key responsibility of this role is to ensure that the client/user/business point of view is reinforced throughout the project and provide resource to test ideas, resolve issues and communicate with the client as needed.

If you are working independently and don't have access to these additional resources, be creative in finding others to perform these roles. You or your client may have access to design resource on a short term basis and you can ask a college to represent your clients point of view. Even the most experienced technician professional can benefit from these perspectives during the course of the project.

Visual design considerations

When you are involved in Web applications design, you may be handed a design layout at the start of the project or have the opportunity to work with a graphic designer to develop the application. Even if you are given a layout, you may need work with the designers to refine it deal with technical limitations. Designers are not always technical, but they have been trained to deal with technical constraints, so be patent and clearly explain any limitations which impacts their design, and work through it.

Because these are all inextricably connected, we cover navigation and page flow with visual design considerations in the following sections. If you are working with a designer or designing it yourself, follow the guidance presented here.

Get organized

- Use a mind-mapping tool to categorize content and help design the primary and alternate navigation.
- Use a flowchart diagramming tool to map out any processes.
- Find and isolate common processes and identify patterns.
- Prioritize elements that support both short-term and long-term user/application goals.
- Use this information to help with the following tasks:
    - Confirm your assumptions.
    - Set design goals and expectations.
    - Educate stakeholders.
    - Coordinate the development team.

Keep it simple and consistent

- Make it easy to get around:
    - Use simple menus that do not have too many levels.
    - Avoid drop-down menus, but if you need to use them keep it to two levels (to avoid mouse "gymnastics").
    - Put the next tier of menus on the page (as a second row or as a sidebar).
    - Keep end user actions together and make them easy to find.
    - Put key actions at the top and bottom of working area to avoid scrolling.
- Make it easy to figure out where you are:
    - Use clear readable page titles with enough white-space to set it apart from the rest of the page text.
    - If page titles are ambiguous or repeated more than once, precede it with the logical parent section e.g. "Customer Service - Feedback"
    - Make sure you have an HTML Title tag in the page header. It should have the site/application name and the page name e.g. "Sample.com - About this application"
    - Use breadcrumbs whenever possible, but use short page names as "crumbs." You can design your application with two titles for each page, one for the page and a shorter one for use in breadcrumbs, HTML Titles, and user friendly URLs.
- Keep relevant information handy:
    - Keep second-tier information on the page to let users move through without having to move back up.
    - Supply supporting information when gliding users through a process.
    - Think of creative ways to provide context-sensitive help (pop-ups, using title="help text" parameters on <DIV> and <SPAN> elements, etc.).
    - If you are guiding a user through a critical process, keep distracting information off the page, such as checkout on a purchasing application, or approval on a workflow application.
- Think about relevant "look and feel" requirements:
    - You may have specific requirements regarding the Web site look and feel. Be aware these needs and wishes.
    - Many organizations or companies have user interface standards that address their requirement for a common look and feel for all their web pages. Familiarize yourself with these standards and requirements to make sure they are addressed in your design.
    - To avoid visual ambiguity:
        - Keep the look and feel clear and consistent.
        - Don't crowd the page.

- Visual elements, blocks of information, margins, and so on etc. should not move around from page to page (add this to QA criteria).
- Use whitespace appropriately (not too much or too little).
- Make sure margins are consistent and line up between vertical sections.
- Use color consistently, don't mix shades, don't have clashing colors, and make sure colors have enough contast to be readable.

Remember to consider the content

Content is the key to a successful site. As noted previously, understanding the nature of the content within your site or application determines the structure and design. You must also consider the following items in regard to content:

- Where the content is generated
- The result of input into an application
- Whether it is provided by another application
- Whether it is created centrally by specific department
- Whether it is produced as part of a participative process

For more information about content, see Thoughts about content

Architectural considerations

While there is no finite checklist, any architectural planning for application development projects require that we consider seemingly unrelated variables that will ultimately influence the overall application. In the following section, we outline specific considerations and explain their importance.

Who will use this application

While it may seem a basic question to ask who will use the application, the answer we get is often the most critical to the application architecture process. For example, let's say that we are creating a Sales Force Automation application for Riverbend. This application has a target user audience that consists of sales and sales associates, customer service, and management. Understanding both how and where these particular business units work, as well as what technology they have available to them, all-but-define your application architecture.

- Riverbend has an extremely mobile salesforce, who rely on their Blackberry mobile devices and primarily keep their company-issue notebook computers in their home offices.
- Sales associates are primarily based out of one of several office locations, using a combination of notebook and desktop computers.
- Customer service is based out of Riverbend corporate headquarters and want to tie this application in with their existing Notes Client-based CS application.
- Management includes executive level and VIPs who want to review the information in a "sales pipeline" and require reports that are both functional and "visually stimulating".

In order for an application to be successful, we need to architect it to specific functional requirements of the intended user community. For Riverbend's Sales Force Automation application, this includes accounting for the mobile salesforce, the customer service application integration, and various metrics reporting.

What are the organization's current technology investments

Cataloging our organization's current (and future) technology investments allows us to understand our architecture options. Just as when we asked who will use this application, by questioning the organizations current technology investments, we can understand which technology we can use to provide our user community with a solution. Knowing which technology is (and will be) used by the organization, and possibly more importantly *how* the technology solutions portfolio will be used provides a developer with other vital information: "What technology do I *not* need to consider?"

For example, Riverbend is currently using Microsoft Office as their productivity suite. There is no intention for Riverbend to adopt a competing productivity suite, and plans are already in place for future upgrades pending future releases. Knowing this, we can utilize Microsft Office-specific functionality and integration-points in our applications without needing to consider a move to a different technology.

What is the expected life cycle of this application

Some development projects result in the creation or maintenance of applications that address business requirements, while other development projects result in _*sunset*_ applications or applications that are used to address a particular and immediate business need while a more "enterprise"-level solution is tested and deployed. By understanding the expected life cycle of our applications, we can understand both the complexity and flexibility of the application. For example, a *sunset* application may not require the built-in maintenance that a non-*sunset* application. Non-*sunset* applications often require an architecture that allow the application to be self-maintained and designed to easily allow revisioning. Alternately, a *sunset* application often requires an architecture that facilitates easy and immediate exporting of all data to the specific *target* solution.

In addition, by understanding that our application is a sunset application, we can provide feedback on the research and intended deployment of a target solution and can even alleviate the need for such a target solution.

Most sunset applications become sunset applications due to a given project champion or decision maker(s) not understanding the flexibility of their current technology investments. We have the opportunity, where and when appropriate, to advise on a better way of utilizing current technology investments that very well may, unknown to those individuals deciding on a particular second-version solution, completely address the business requirement.

When is the application deployment or deliverable expected

Understanding the expected deployment or deliverable date or dates for your development projects can greatly impact the architecture of your applications. For example, high-priority/quick-deliverable solutions often require that we refrain from trying more experimental technologies and methodologies in favor for production-ready and time-tested architectures.

Where will this application be used

Understanding the geographic locations can greatly influence and affect the architecture of our solutions. Projects in which low-bandwidth areas, for example, can create "thin-client" or even offline application architectures.

How will the user community use this application

Understanding how the user community will use your application can greatly influence your solution architecture. This is often achieved by understanding the business process and inter-department workings of your target user audience.

Additional topics

- Architectural patterns
- Thoughts about content

Architectural patterns

This page last changed on Mar 31, 2008 by jservais.

Web site content management

Share your best practices and thoughts here.

Client based

Share your best practices and thoughts here.

Browser based

Share your best practices and thoughts here.

Commerce site

Share your best practices and thoughts here.

Consumer

Share your best practices and thoughts here.

Business to Business

Share your best practices and thoughts here.

Customer relationship management

Share your best practices and thoughts here.

Internal communication (intranet)

Share your best practices and thoughts here.

Information sharing

Share your best practices and thoughts here.

Image library

Share your best practices and thoughts here.

Employee self service

Share your best practices and thoughts here.

Human resources

Share your best practices and thoughts here.

Policy documents

Share your best practices and thoughts here.

Thoughts about content

This page last changed on Mar 31, 2008 by jservais.

It is important to discover early on the type of content with which you are working so that the designer and developer can plan for the content at the start of the project. Ask the following questions about the content that you want to include:

- Has the content been written specifically for the Web?
  Text written for reports, and white papers don't always work in a browser. Sometimes its best to break this into multiple pages or summarize on the page and supply a PDF for download.

- Does the content need to edited by a copywriter?
  It may seem excessive, but you may find involving a professional writer helpful if you are trying to accomplish any of the following goals:
  - Reach an external audience.
  - Document complex processes.
  - Be persuasive or instructive.

- Does the content and imagery have the necessary rights?
  If you do not have the right to use a piece of content on the Web, you need to plan for the time it takes to purchase the rights.

- Do you have a rationale for the editorial style of the Web site?
  If you already have an audience in mind and you can define the style your are striving for, to appeal to this audience.

- How text heavy is the site?
  As well as considering the editorial style of the site, you must also consider how easy it will be to digest this information. If it is a text heavy site, you must design with this in mind.

- Are you likely to include maps, charts or graphs?
  Make sure these fit into the overall design. These may have to be re-created by a designer and will impact your budget and timeline.

Domino Web capabilities

This page last changed on Mar 31, 2008 by jservais.

- Web services
- Mail services
- Directory services
- Data services

The Lotus Domino Server contains several modules, or server tasks. These tasks perform many functions usually available as separate servers from other middleware vendors. These some of tasks can also be seen as services, in a service-oriented architecture (SOA), which can talk to other applications or integrate with your solution.

A big advantage of a Domino-based solution is how easy a Web developer can mash together a sophisticated application by using these services within the integrated application environment of Lotus Notes, which has been designed from the start for building collaborative solutions.

In the following sections, we identify the services that you should know to help architect your Domino Web application.

Web services

- **HTTP server**
  One of the aforementioned server tasks is the Web, or HTTP server with Secure Sockets Layer (SSL) support. This can be used as a simple HTTP server which can serve up HTML pages from the Domino server data directory. By default pages stored in the domino\html directory will be available as if from the root directory of an HTTP server. These defaults can be changed in Domino server administration, and multiple HTTP servers can be managed on a single Domino server.

- **Web application server**
  While the Domino HTTP server can serve up stand alone HTTP pages, it's real value is how it supports the Domino Web application server. This application server is designed to interpret Notes client based applications and translate then into HTML, JavaScript constructs and publish them to the Web through the HTTP server. see Architecture of the Domino Web Server

- **Web services provider and consumer**
  A Web service provider publishes a Web service for query and use, and a Web service consumer calls operations from the service. A Web service provider makes available a Web Services Description Language (http://www.w3.org/TR/wsdlWSDL) document that defines the service interface. Incoming requests from a consumer are passed through to the underlying code, and results are passed back to the consumer. Lotus Domino maps the WSDL interface to an agent-like Web service design element that can be coded in LotusScript or Java. This allows your Web site/application to retrieve information from internal and external services or supply information as needed, to support a service-oriented architecture (SOA) in your design.

- **Web Retriever**
The Lotus Domino Web Retriever task is responsible for retrieving Web pages on behalf of Notes client users who want to access the Web via their server. It also supports the variety of LotusScript and Java methods, such asGetDocumentByURL, so that you can obtain the HTML code of a Web page and parse through it. This is useful in "republishing" time sensitive or status information from HTML on other sites or applications (e.g. stock prices etc.)

Mail services

- **POP3 mail server and IMAP mail server**
Domino provides Post Office Protocol Version 3 (POP3) and Internet Mail Access Protocol (IMAP4rev1) servers. Both implement Internet mail protocols that supports a user running a POP3 or IMAP clients. The Domino server receives and stores mail for users, who can then connect to the server to retrieve their mail. The IMAP service differs from the POP3 service in that users are not required to download messages to a local computer to read and manipulate them. These services may be relevant if you want to support mail users using alternate clients, but both services are derived from the built in messaging capabilities of the Domino platform.

- **SMTP server**
Domino supports sending and receiving mail over Simple Mail Transfer Protocol (SMTP) by means of the SMTP listener task and SMTP Router, respectively, each of which you enable separately. The SMTP listener task handles incoming SMTP connections and delivers messages received over those connections. While not directly related to Web applications, this capability helps you to integrate SMTP mail into you overall Domino solutions

Directory services

- **LDAP Client/Server**
Domino provides a Lightweight Directory Access Protocol (LDAP) server. LDAP is a standard Internet protocol for searching and managing entries in a directory. Many Internet clients and middleware products use LDAP to share and look up directory information. Domino LDAP support is quite sophisticated. If you need LDAP services, you can review its capabilities in the Domino Administration help. Domino can also function as an LDAP client and can easily integrate external LDAP information into the Domino directory, which can be used to authenticate Web application users.

Data services

- **Database server**
Domino includes a database management system (DBMS), although it is not a relational DBMS nor an object database. Rather it is a document-centric database. It allows multiple values in items (fields), doesn't require a schema, has built-in document-level access control, and can store RichText data. This document orientation sometimes behave more like as XML document natively stored in a database, and like XML, Domino supports hierarchical relationships. While many Web application development frameworks cater to relational back-ends, the unstructured hierarchical nature of Domino makes it ideal for rapidly collecting and evolving Web-based solutions. For example, post to a Notes form with no fields defined and capture all information posted. The presentation and categorization can be programmed later.

- **DIIOP server**
  DIIOP is CORBA over IIOP for Lotus Domino. It is primarily used in Domino to allow the Domino Java applets in the web client to talk directly to the Domino without refreshing the page. This is also the way WebSphere, other IBM products and third-party applications access Domino information remotely over TCP/IP.

Planning for accessibility and compliance

This page last changed on Apr 03, 2008 by jservais.

Compliance

In short, compliance is making sure your Web site or Web application conforms to all appropriate standards that apply. This may seem simple enough, but you may find determining which standards apply daunting, and in practice, those standards may also have different interpretations. Regardless of these issues, Web standards help ensure that everyone has access to the information we are providing and makes Web development faster and easier.

Compliance is one aspect of accessibility, which makes it easier for people with special needs to use the Web, but there are also many practical reasons for developers to be concerned with Web standards. While search engines can do a better job of indexing sites, browser-specific code often doubles or triples the development effort.

Standards may seem limiting but many of the current uses of the Web would not be possible without widespread standards compliance.

The standards

- HTML 4.0 - Hypertext Markup Language (HTML) is used for adding structure to text documents.
- Extensible Markup Language (XML) is a markup language like HTML, but instead of having a single, fixed set of elements, it allows you to define your own.
- XHTML 1.0 is a reformulation of HTML as an XML application.
- HTML 4.01, and being technically stricter because of XML's influence.
- Cascading style sheets (CSS) are a mechanism for changing the appearance of HTML or XML elements.
- Document Object Model Level 1 (DOM 1) allows the full power and interactivity of a scripting language.

Compliance is just the first step in the goal of accessibility. The broad definition of compliance means compliant applications are accessible to a greater range of devices and applications. These devices and applications can reinterpret compliant information for a broader audience, for example into voice by page readers or reuse through mash ups.

Accessibility

In many countries, businesses and organizations are legally required to ensure that their services, including Web sites and other media, are accessible to everyone regardless of disability. Such disabilities may include a vision, hearing, physical movement, or reading and comprehension abilities.

The following examples of legislation have provisions requiring physical accessibility:

- In the U.S., under the Americans with Disabilities Act of 1990
- In Australia, Disability Discrimination Act 1992
- In the U.K., the Disability Discrimination Act 1995

Special guidelines have been created by the World Wide Web Consortium (W3C) to cover accessibility of Web sites. In conjunction with their guidelines, the Web Accessibility Initiative (WAI) have defined three standards of accessibility for Web sites - Single A, Double A and Triple A. Single A is the most basic standard to which all Web sites should comply.

Reasons to consider accessibility

Your business should think about accessibility for the following reasons:

- Avoid legal disputes
  If your site does not conform to legislation (see above), then you could be the target of legal action against you for not ensuring your Web site is accessible.

- Reach a wider audience
  Currently you probably unknowingly exclude some visitors to your Web site. Turn those users into customers and you will grow your business further.

- Gain the competitive edge
  If your Web site is accessible to all when your competitors' Web sites are not, you gain an immediate advantage over them.

- Enhance your image
  Negative reports have been in the press about companies who have not tried to make their products and services accessible to all. Taking positive action should have the reverse effect and enhance your image in the marketplace.

- Improvements for all
  To meet some of the required standards, coding enhancements need to be made. This has the additional advantage of improving compliance with new browsers and devices and potentially improving overall performance for all users.

Ensuring accessibility

If you already have Web sites or applications, audit them to see what actions you need to take to bring them in line with the W3C standards.

If you are developing new projects, you can refer to the Web Accessibility Initiative (WAI), which is part of the World Wide Web Consortium (W3C). This organization developed the Web Content Accessibility

Guidelines (WCAG) which explains how to make Web content accessible to people with disabilities. Web "content" generally refers to the information in a Web page or Web application, including text, images, forms, sounds, and such. The WCAG is separated into three levels of compliance, A, AA and AAA. Each level requires a stricter set of conformance guidelines, such as different versions of HTML (transitional vresus strict) and other techniques that need to be incorporated into your code before accomplishing validation.

Tools are available to help assess Web sites against these standards. See the Human Ability and Accessibility Center

Online tools exist that help developers to submit their Web site and automatically run it through the WCAG guidelines and produce a report, stating whether they conform to each level of compliance. In addition, commercial software is available with similar functionality such as the IBM Rational Policy Tester Accessibility Edition. Also many development tools such as Adobe® Dreamweaver®, IBM Rational tools and Eclipse, have facilities or plug-ins to support compliance with accessibility standards.

To comply with accessibility standards, consider including the following concepts in your design:

- (X)HTML validation from the W3C for the pages content
- CSS validation from the W3C for the pages layout
- At least WAI-AA (preferably AAA) compliance with the WAI's WCAG
- Compliance with all guidelines from Section 508 of the US Rehabilitation Act
- Use HTML Access keys
- Semantic Web Markup
- Supply a high contrast version of the site for individuals with low vision
- Provide alternatives (transcript) for any multimedia used on the site (video, flash, audio, etc.)

For more detailed information, see Web accessibility developer guidelines from IBM.

Accessible Web sites and applications should include a Web accessibility statement on the site to explain conformance, special facilities, and the commitment to accessibility.

Understanding the Web browser client environment

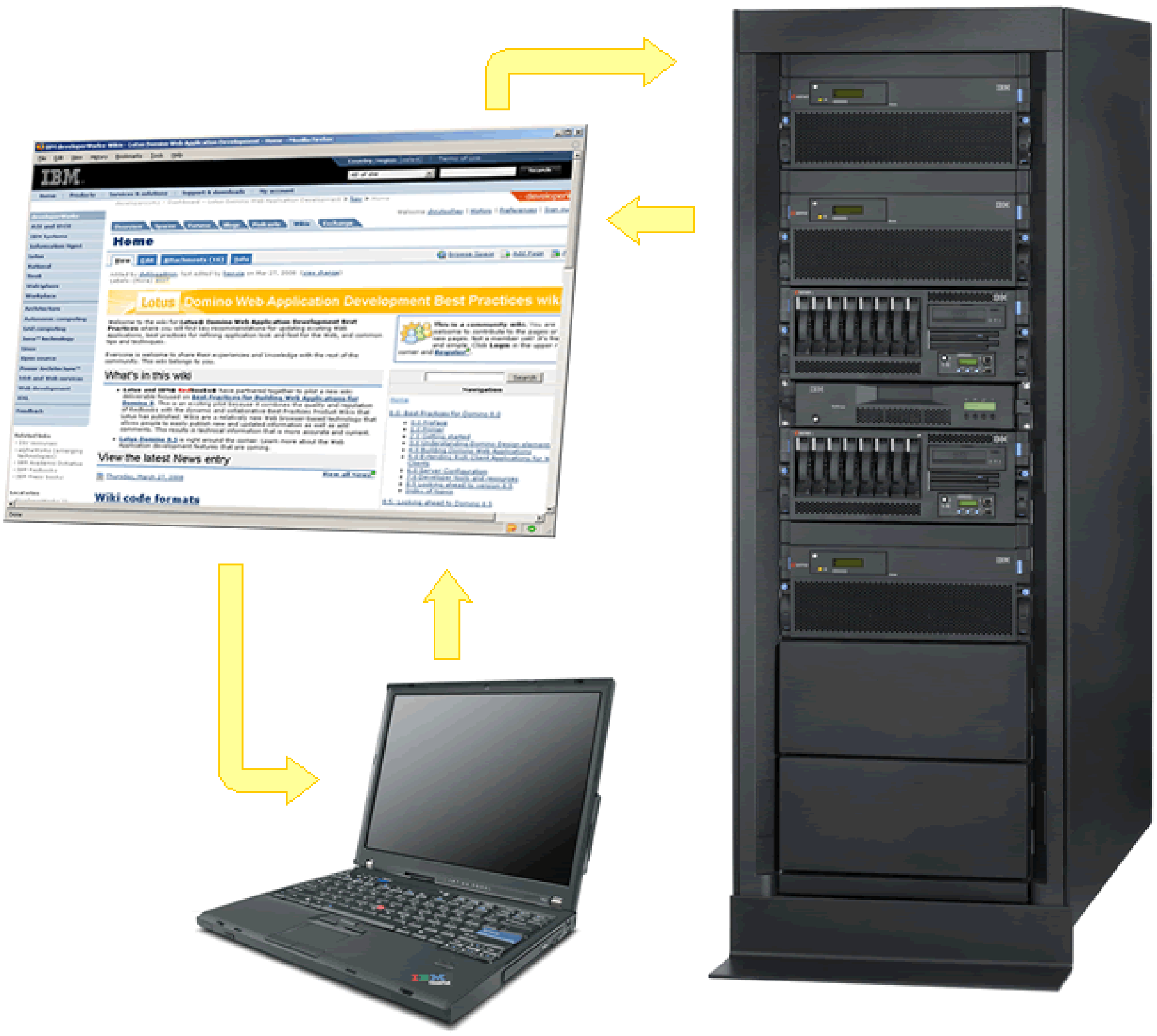
This page last changed on Apr 04, 2008 by dalandon.

At an elementary level, the Web browser client acts as the rendering device for the services provided by (in this case) the Domino Server. However, the Web browser client can offer a near-limitless functional capacity when we consider the rendering and integration capabilities that the "this client" affords the Domino Web Developer that understands the Web browser client environment. In leveraging this environment, we can provide our user community with rich applications that rival (and may even exceed in some cases) their Lotus Notes Client counterparts.

In this section, we review the Web browser client environment and how Domino Web application developers can use this environment to provide users with rich applications.

The Web browser client environment

The Web browser client environment is your current prototypical client/server environment. A "thin client" (the Web browser client in this case) communicates to a server environment via direct calls to server-generated resources and services. This explains the relationship between a Web browser client and a Domino Server in our Domino Web application environments.



The Web browser client can act as a middleware or conduit solution, facilitating server and local computer data interaction. This data interaction and the combination of local computer and server-generated resources and services can be used to create rich "thin client" solutions. By understanding the bidirectional interaction of data and using its communicated and rendered content we can mimic the Lotus Notes Client and other "rich client" functionality in our Domino Web application development practices.

Using Web browser transport layers

The Web browser client, most recently with the advent of [AJAX](), can use bidirectional data interaction via transport layers. The utilization of transport layers, the communication methods and channels that are present "between" the user interface and the underlying system architecture, can facilitate fluid and more enhanced user interfaces for our Domino Web applications.

Prior to the advent of AJAX, hidden <*iframe*> elements were used to communicate between dynamic UIs and back-end services via now standard Transport Layer/Web 2.0 development methodologies. As AJAX becomes a more standards-based approach to dynamic user interfaces and Web Service *mashups*, the utilization of transport layers to combine local client data and server-based resources and services is becoming a global user community expectation for current Web applications and services.

HTTP cookies

On of the most commonly used facilities for locally caching reusable data is the HTTP Cookie. When used properly, the HTTP Cookie can store data strings that can be used throughout an application alleviating the repeated querying of Server-maintained or user input data.

The following example shows the common syntax for HTTP cookies:

```
NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure
```

Note that in this example, all uppercase attributes are user-defined. The HTTP cookies are stored locally on the user device and are accessible throughout your applications.

For more information about HTTP cookies, see the following resources:

- http://www.quirksmode.org/js/cookies.html
- http://www.w3schools.com/js/js_cookies.asp

3.0 Understanding the Domino design elements

This page last changed on Apr 01, 2008 by jservais.

This section helps you understand the key Domino design elements.

Topics in this section

Database

This page last changed on Mar 27, 2008 by heinsje.

This section covers the following topics:

- Default Launch Elements
- Tab specific database functionality

Default Launch Elements

This page last changed on Apr 04, 2008 by dalandon.

Defining a Default Launch Element

Depending on the functional requirements of our Domino Web application, you may be required to have a specific Default Launch Element (DLE). The DLE can vary based on the specific needs of your user community or the specific function or design of our Domino Web applications. In this section, we will review several DLE methods and examples.

Application-specific and example DLEs

Application function types often drive the DLE logic because the combination of various Domino design elements are often required to achieve the results that are required by certain types of applications. When we understand the different types of DLEs, we can then properly construct our application-specific DLE. In the following sections, we discuss the various types and examples of DLEs.

Simple DLEs

Most Domino Web developers are familiar with the the simple DLEs of Web application development: Frameset design elements and Page design elements. While Page Design Element DLEs are still useful and can provide our user community with a viable DLE solution, using Frameset Design Elements have become a *depreciated* practice.

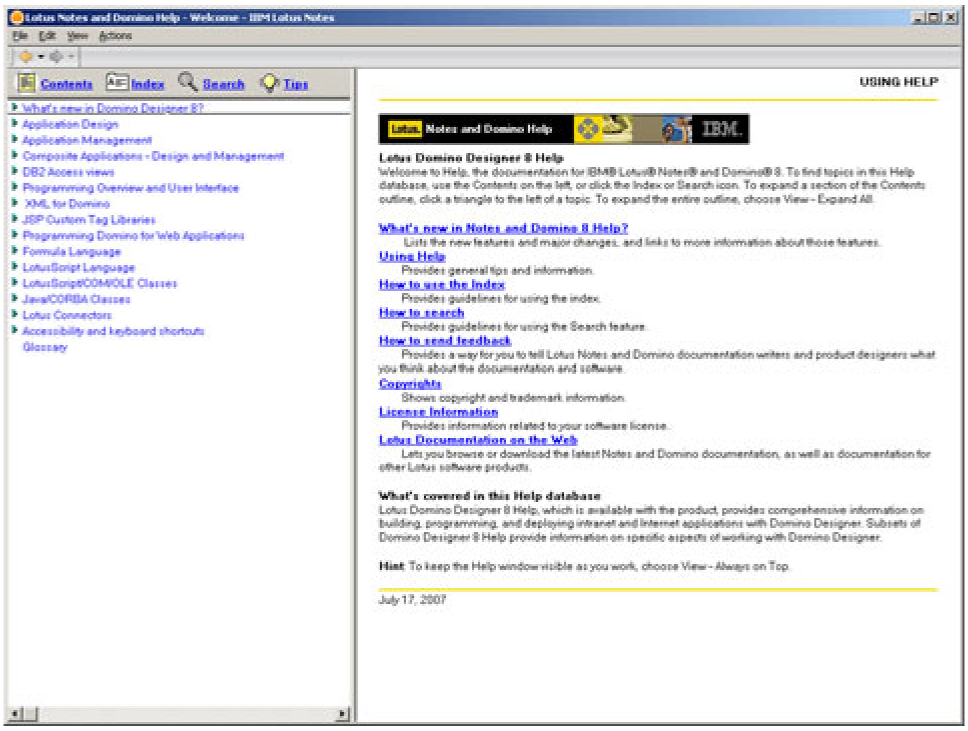
For more information about depreciated Web application development practices, see Web standards primer.

As stated, the Page Design Element DLE can provide the user community with a perfectly functional solution, but as we extend our Web application development to include external technologies and more advanced user interfaces, we discover that Form design elements often make the more logical choice when defining a Default Launch Element.

In the following sections, we discuss several methods and practices to use Form Design Element DLEs.

Index DLEs

The Lotus Notes Help databases are perfect examples of Index DLE applications. These applications launch with a Frameset-like display, typically consisting of an *index*, or navigator combined with a *content body*, or main section dedicated to the display of the highlighted/selected content.


Lotus Notes and Domino Help - Welcome - IBM Lotus Notes

File Edit View Actions

Contents Index Search Tips

- What's new in Domino Designer 8?
- Application Design
- Application Management
- Composite Applications - Design and Management
- DB2 Access views
- Programming Overview and User Interface
- XML for Domino
- JSP Custom Tag Libraries
- Programming Domino for Web Applications
- Formula Language
- LotusScript Language
- LotusScript/COM/OLE Classes
- Java/CORBA Classes
- Lotus Connectors
- Accessibility and keyboard shortcuts
- Glossary

USING HELP

Lotus Notes and Domino Help IBM.

Lotus Domino Designer 8 Help
Welcome to Help, the documentation for IBM® Lotus® Notes® and Domino® 8. To find topics in this Help database, use the Contents on the left, or click the Index or Search icon. To expand a section of the Contents outline, click a triangle to the left of a topic. To expand the entire outline, choose View - Expand All.

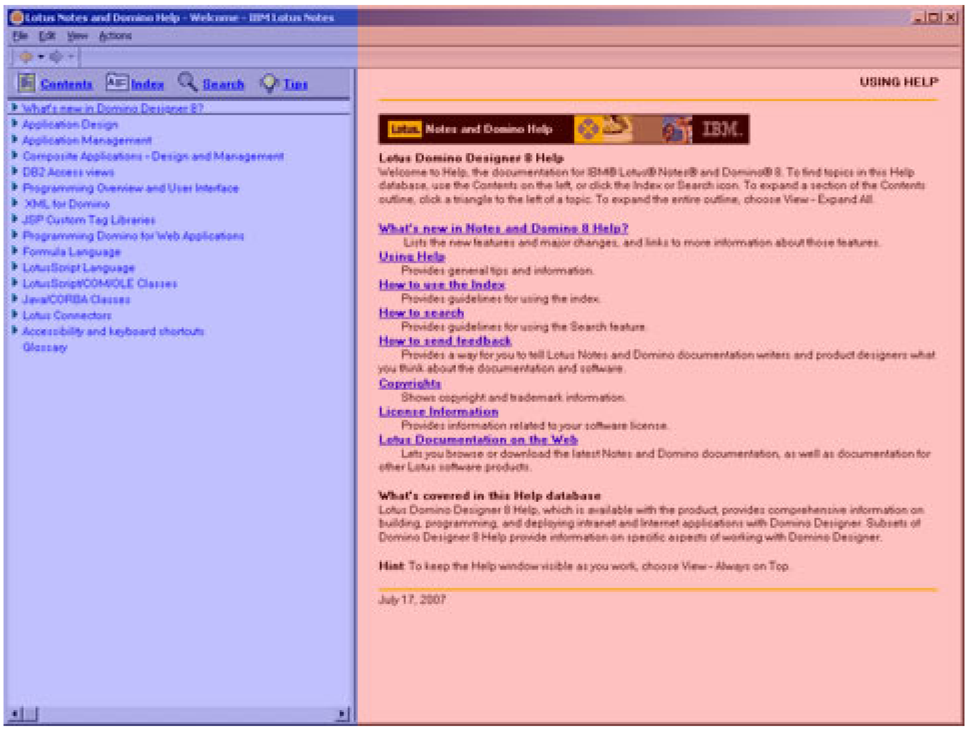
What's new in Notes and Domino 8 Help?
Lists the new features and major changes, and links to more information about those features.
Using Help
Provides general tips and information.
How to use the Index
Provides guidelines for using the index.
How to search
Provides guidelines for using the Search feature.
How to send feedback
Provides a way for you to tell Lotus Notes and Domino documentation writers and product designers what you think about the documentation and software.
Copyrights
Shows copyright and trademark information.
License Information
Provides information related to your software license.
Lotus Documentation on the Web
Lets you browse or download the latest Notes and Domino documentation, as well as documentation for other Lotus software products.

What's covered in this Help database
Lotus Domino Designer 8 Help, which is available with the product, provides comprehensive information on building, programming, and deploying intranet and Internet applications with Domino Designer. Subsets of Domino Designer 8 Help provide information on specific aspects of working with Domino Designer.

Hint: To keep the Help window visible as you work, choose View - Always on Top.

July 17, 2007

As the usage of Frameset Design Elements are depreciated, we architect our DLE toward Web Development Standards, which require the combination of several Domino design elements.


Lotus Notes and Domino Help - Welcome - IBM Lotus Notes

File Edit View Actions

Contents Index Search Tips

- What's new in Domino Designer 8?
- Application Design
- Application Management
- Composite Applications - Design and Management
- DB2 Access views
- Programming Overview and User Interface
- XML for Domino
- JSP Custom Tag Libraries
- Programming Domino for Web Applications
- Formula Language
- LotusScript Language
- LotusScript/COM/OLE Classes
- Java/CORBA Classes
- Lotus Connectors
- Accessibility and keyboard shortcuts
- Glossary

USING HELP

Lotus Notes and Domino Help IBM.

Lotus Domino Designer 8 Help
Welcome to Help, the documentation for IBM® Lotus® Notes® and Domino® 8. To find topics in this Help database, use the Contents on the left, or click the Index or Search icon. To expand a section of the Contents outline, click a triangle to the left of a topic. To expand the entire outline, choose View - Expand All.

What's new in Notes and Domino 8 Help?
Lists the new features and major changes, and links to more information about those features.
Using Help
Provides general tips and information.
How to use the Index
Provides guidelines for using the index.
How to search
Provides guidelines for using the Search feature.
How to send feedback
Provides a way for you to tell Lotus Notes and Domino documentation writers and product designers what you think about the documentation and software.
Copyrights
Shows copyright and trademark information.
License Information
Provides information related to your software license.
Lotus Documentation on the Web
Lets you browse or download the latest Notes and Domino documentation, as well as documentation for other Lotus software products.

What's covered in this Help database
Lotus Domino Designer 8 Help, which is available with the product, provides comprehensive information on building, programming, and deploying intranet and Internet applications with Domino Designer. Subsets of Domino Designer 8 Help provide information on specific aspects of working with Domino Designer.

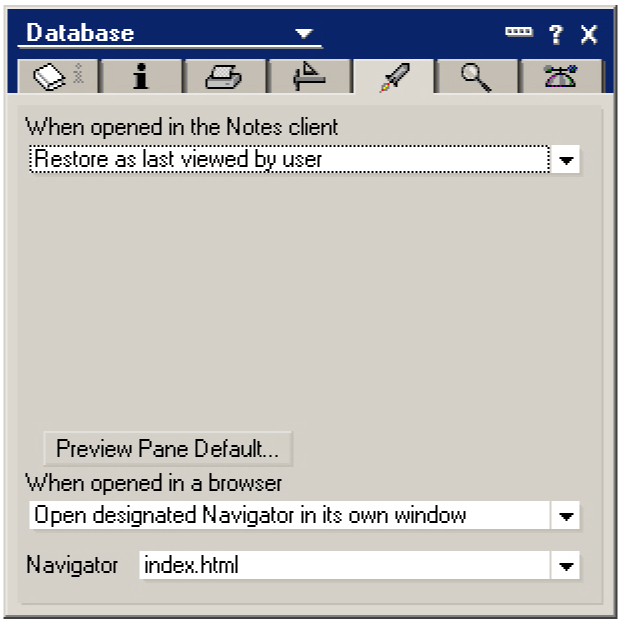
Hint: To keep the Help window visible as you work, choose View - Always on Top.

July 17, 2007

In the above example, we have an index on the left in blue and a content body on the right in red. We can achieve this same layout easily with the following design elements:

1. Create a blank Navigator Design Element named "index.html".
2. Create a Form Design Element named "$$NavigatorTemplate for index.html".

3. In the Form Design Element, add your desired user interface elements, objects, and markup.
4. Modify the Database Launch Properties to "Open designated Navigator in its own window" and select the "index.html" Navigator Design Element.


Database

When opened in the Notes client

Restore as last viewed by user

Preview Pane Default...

When opened in a browser

Open designated Navigator in its own window

Navigator index.html

For more information about how to even further extend this type of design element architecture, see Design element multi-aliasing.

Dashboard DLEs

Dashboard DLEs are becoming more and more popular with the advent of portals, *mash-ups*, and composite application development.

Tab specific database functionality

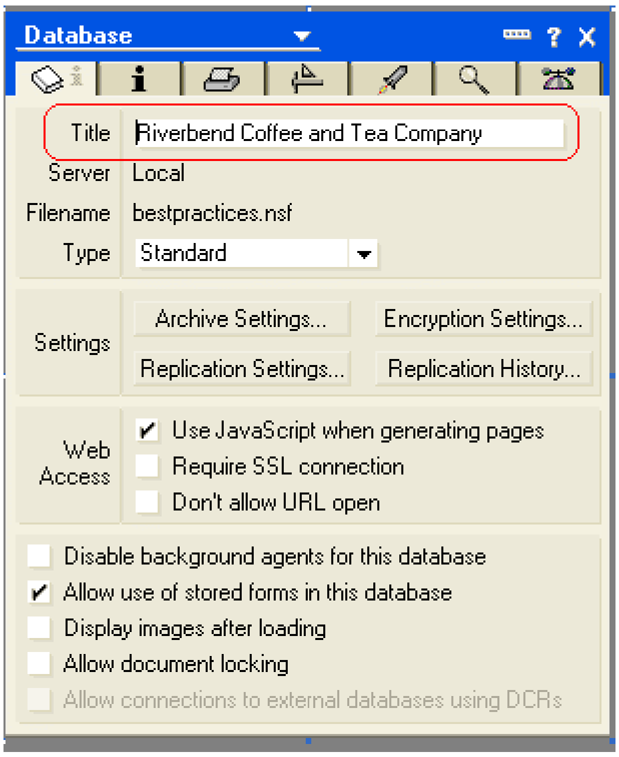
This page last changed on Apr 03, 2008 by jservais.

- Database title
- Web access
- Full-text index
- Soft delete

Database title

Having a meaningful database title is important in developing a good Web application. The title usually relates to a name of a company if you are building a Web site for a company or it could be an application name.

One of the advantages of having a meaningful database title is that you can use @DbTitle to display the database title throughout your Web application. You can use the formula on the Web site header or window title. If you change the database title, the change is reflected throughout the application.



Title: Riverbend Coffee and Tea Company
Server: Local
Filename: bestpractices.nsf
Type: Standard

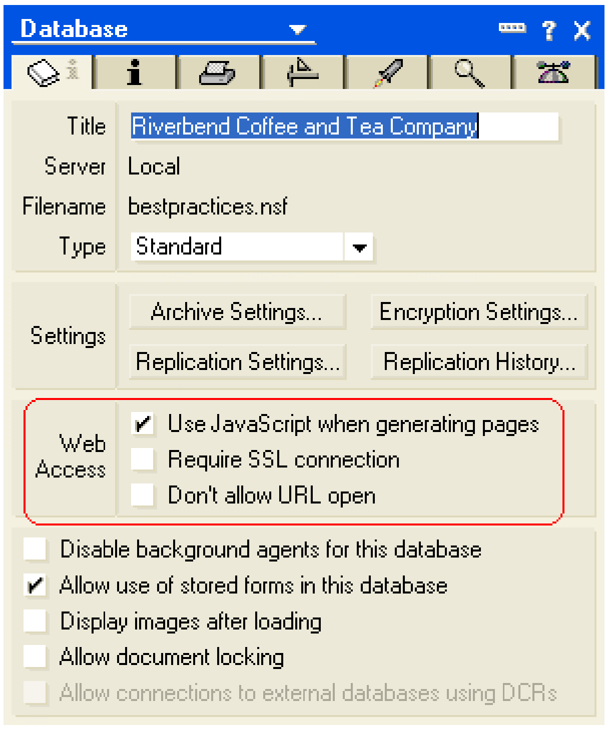
Web Access:
[✓] Use JavaScript when generating pages
[ ] Require SSL connection
[ ] Don't allow URL open

[ ] Disable background agents for this database
[✓] Allow use of stored forms in this database
[ ] Display images after loading
[ ] Allow document locking
[ ] Allow connections to external databases using DCRs

Tip

It is a good idea to have a unique window title for every form, page, or view (by using $$ViewTemplate) that you have in the database. A Web Page Tracking utility, such as Google Analytics, has an option to show the most visited pages by Window/Page Title. If you do not set the title, you must guess what the pages are by looking at the long Domino URLs.

Web access



Title	Riverbend Coffee and Tea Company
Server	Local
Filename	bestpractices.nsf
Type	Standard

Web Access
- [✓] Use JavaScript when generating pages
- [ ] Require SSL connection
- [ ] Don't allow URL open

- [ ] Disable background agents for this database
- [✓] Allow use of stored forms in this database
- [ ] Display images after loading
- [ ] Allow document locking
- [ ] Allow connections to external databases using DCRs

Use JavaScript when generating pages option

The Use JavaScript when generating pages option is always checked by default when creating a new database. We recommend that you leave this option selected when developing a Web browser client application. The main reason is that we can only have one button on our form if we deselect the option. The button is a submit button at the bottom of our form that is generated by Domino.

The following table shows the detailed effects on this option (taken from the Designer Help).

If the option is selected	If the option is not selected
Display: Documents and navigators display faster because hotspot formulas are not evaluated until users click each hotspot.	Display: Documents and navigators display more slowly because the hotspot formulas are all evaluated at the display time.

Buttons: Domino doesn't generate a Submit button automatically.
To allow users to save and close a form on the Web, you must create a button, hotspot, or action that contains these commands:
`@Command([FileSave]);`
`@Command([CloseWindow])`
	Buttons: Domino automatically generates a Submit button, at the bottom of the form.
	If there is already one or more buttons on the form, Domino converts the first button it recognizes to a Submit button automatically and ignores all other buttons on the form.
	You can have only one button, a Submit button, on a form.

You can have multiple buttons on a form.	
@Commands: The following commands are supported on the Web:	@Commands: The following commands are not supported on the Web:
`@Command([CloseWindow])` `@Command([FileSave])` `@Command([ViewRefreshFields])`	`@Command([CloseWindow])` `@Command([FileSave])` `@Command([ViewRefreshFields])`
Domino does not check the formulas before displaying pages.	Domino checks the formulas before displaying pages. Actions that contain unsupported @commands or @functions are not displayed on the Web.

Require SSL connection option

The Require SSL Connection option is not selected by default. We select this option if we have a Web application that uses an SSL connection. This option ensures that any connection to the database uses SSL (HTTPS). A typical use of SSL is when your Web application accepts, for example, credit card transactions or stores confidential information, such as a Social Security Number or salary information.

Note

The administrator can also set SSL by using either Internet site documents or a server document in the Domino Directory.

Don't allow URL open option

The Don't allow URL open option is not selected by default. If this option is selected, any URL command in the browser that uses the question mark syntax (?OpenForm, ?OpenView, etc) does not work on the database. Domino generates the following error message:

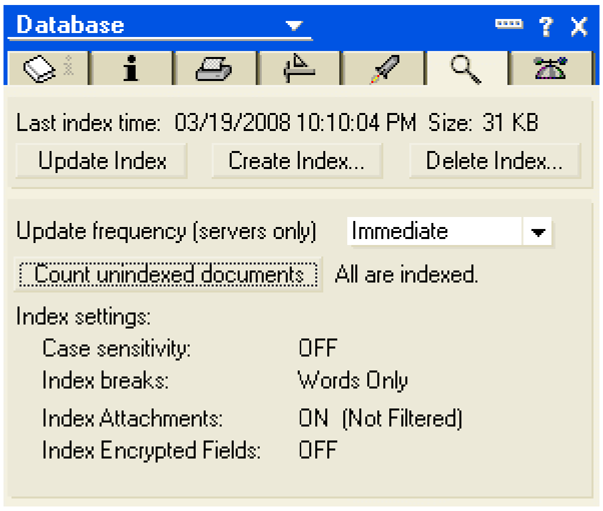
Error 500
HTTP Web Server Lotus Notes Exception - You are not authorized to access that database.

This option is rarely used because Domino developers rely on the Domino URLs when building their application. However, it is useful when you are designing a Web application that uses a servlet. The servlet issues all the Domino URLs in the back end, so that the Domino server allows it to go through.

Tip

There are other techniques that we can use to prevent users from accessing certain views in a database. You can customize the $$ViewTemplate for the views with the "You are not authorized to access this section" message, or you can customize the $$ViewTemplateDefault if you have $$ViewTemplate for all the views that you display.

Full-text index



Last index time: 03/19/2008 10:10:04 PM Size: 31 KB

Update Index | Create Index... | Delete Index...

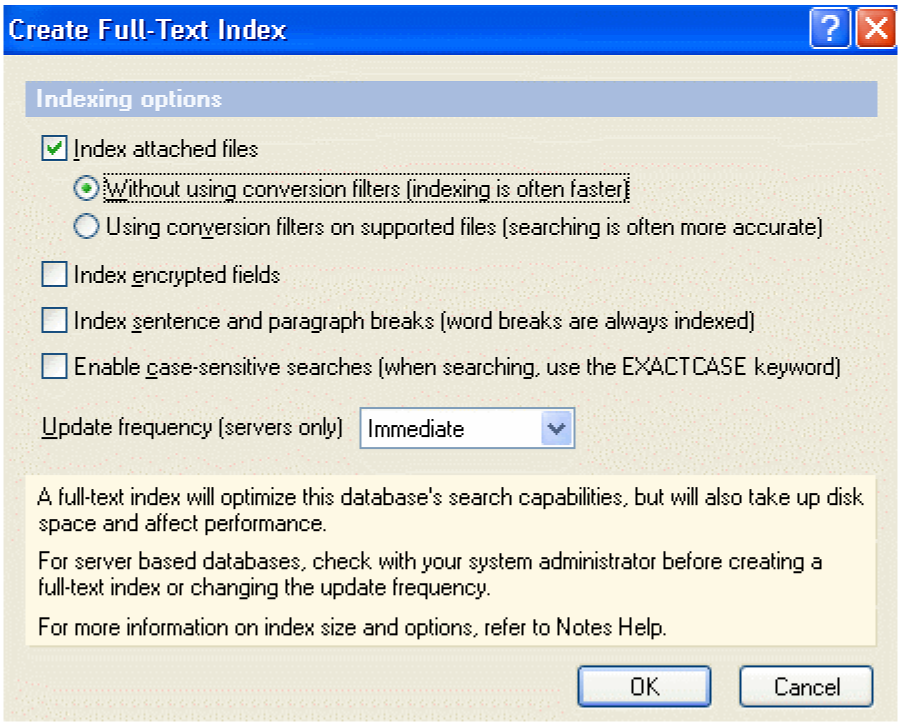
Update frequency (servers only): Immediate

Count unindexed documents | All are indexed.

Index settings:

Case sensitivity:	OFF
Index breaks:	Words Only
Index Attachments:	ON (Not Filtered)
Index Encrypted Fields:	OFF

In order to do a full-text search by using the ?SearchView Domino URL command, the full-text index must be created. When you create a full-text index, you see the options as shown in the following Create Full-Text Index window.



Indexing options

[✓] Index attached files

[●] Without using conversion filters (indexing is often faster)

[ ] Using conversion filters on supported files (searching is often more accurate)

[ ] Index encrypted fields

[ ] Index sentence and paragraph breaks (word breaks are always indexed)

[ ] Enable case-sensitive searches (when searching, use the EXACTCASE keyword)

Update frequency (servers only): Immediate

A full-text index will optimize this database's search capabilities, but will also take up disk space and affect performance.

For server based databases, check with your system administrator before creating a full-text index or changing the update frequency.

For more information on index size and options, refer to Notes Help.

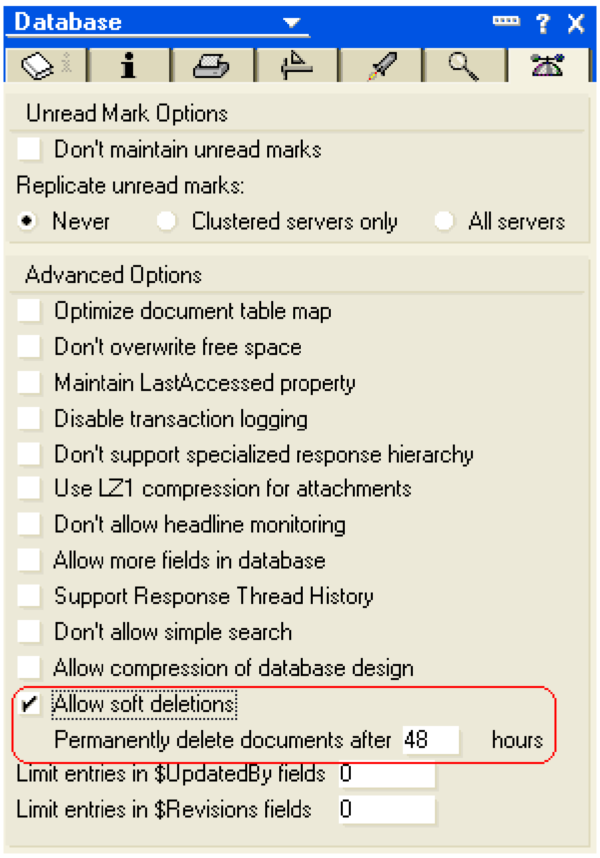
The following table shows more information about each option (taken from the Designer Help).

Option	Description
Index attached files	Searches all documents and any attachments. *Without filters, Search processes only the ASCII text of the attachments, and may not find all content. *With filters, Search may be able to process other parts of the files. **Notes:**

	• Without filters is faster than With filters but less comprehensive. Text in attachments is not highlighted. • Attachments are located in the $FILE field in a document, though a picture of them appears in the BODY field. If you specifically search the BODY field for attachments, you won't find the information.
Index encrypted fields	Searches all words in fields, including encrypted fields. **Attention:** Selecting this option compromises the security of information in encrypted fields.
Index sentence and paragraph breaks	Searches words in the same sentence or paragraph.
Enable case-sensitive searches	Searches for words by exact case match. Selecting this means that when you enter 'EXACTCASE' followed by your search terms, searches for your words exactly as you enter them. Example: If you enter "EXACTCASE Apple" in the search box, Search does not find "apple." **Note:** Selecting this option increases the size of a full-text index by 5 to 10%.
Update frequency (servers only)	Determines the frequency of automatic updates for the index. The server administrator determines the actual time or date for updating the full-text indexes of applications that use the Daily, Hourly, or Scheduled options. If you choose Immediate, the full-text index is created on the server as soon as possible. The option applies only to server-based applications and to local replicas of server-based applications (for example, a local replica of your mail application). A local replica must have this option set to Immediate if you want the full-text index updated automatically during the replication process.

For more information about full-text searching, see Searching on Section 4.0.

Soft delete



Unread Mark Options

☐ Don't maintain unread marks

Replicate unread marks:

[✓ Never] ○ Clustered servers only ○ All servers

Advanced Options

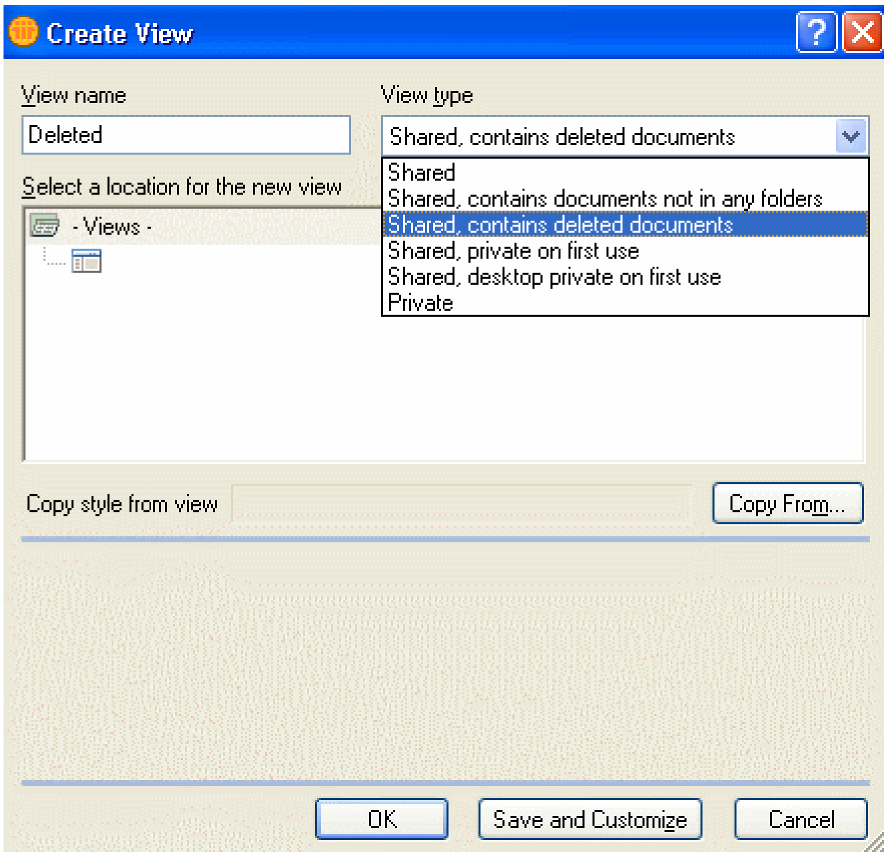
☐ Optimize document table map
☐ Don't overwrite free space
☐ Maintain LastAccessed property
☐ Disable transaction logging
☐ Don't support specialized response hierarchy
☐ Use LZ1 compression for attachments
☐ Don't allow headline monitoring
☐ Allow more fields in database
☐ Support Response Thread History
☐ Don't allow simple search
☐ Allow compression of database design
✓ Allow soft deletions

Permanently delete documents after 48 hours

Limit entries in $UpdatedBy fields 0

Limit entries in $Revisions fields 0

In certain situations, we might need to allow deletion on our Web application. However, we want the ability to review and track the deletion before the documents are gone forever. By selecting the Allow soft deletion option, documents are deleted, but they are still available in the database for the period you specify. In order to view these deleted documents, create a view and choose **Shared, contains deleted documents**. You can restore these deleted documents in Notes client by using the @UndeleteDocument formula.


Create View
View name
Deleted
View type
Shared, contains deleted documents
Shared
Shared, contains documents not in any folders
Shared, contains deleted documents
Shared, private on first use
Shared, desktop private on first use
Private
Select a location for the new view
- Views -
Copy style from view
Copy From...
OK
Save and Customize
Cancel

This is not a good solution if your company needs to comply with a certain regulations, such as Sarbanes-Oxley document retention. Having a good archive and restore process is a more appropriate solution.

Domino design elements

This page last changed on Apr 01, 2008 by jservais.

Lotus Domino was built to be a collaborative tool, focusing on fast-paced collaboration and productivity. Lotus Domino Designer enables application developers to create complex Web applications by using a unique intuitive integrated development environment (IDE). Domino offers a compact set of design elements that when combined allow developers to easily work on common Web design tasks, including pages and form creation, database records listing, and simple interface elaboration. The design elements are displayed differently according to the environment used by the user, and may vary according to other application and environment settings. On the Web, Lotus Domino design elements behave similar to their versions in Lotus Notes client with only minor differences. This design can be used programmatically and be invoked by using Domino URLs. It is only a matter of time to understand and use the available design properly on a Domino application.

Topics in this section

A design elements overview

This page last changed on Apr 01, 2008 by jservais.

The Domino application design is composed of several different types of design elements, each one with a different function over an application. In this section, we discuss the design elements that are available on the Web with Domino Designer Release 8.

Frameset

A *frameset* is an element that displays a set of frames that enables designers to improve the usage of the users' screen. A *frame* is a small portion of a frameset that is used to display a design component in a specific space of the application screen. By using framesets, designers can create relationships between frames, combining navigation, linking and data display on a same screen. The frameset design element allows designers with no experience on HTML to create Web site framesets in an intuitive tool. For more information, refer to Frameset design elements.

Pages

Pages are the best design available in Domino Designer to display information for the user. With pages, designers can store rich text data, including images and formatted text, and some programmatic components, including buttons and actions. In a Web environment, programmers can take advantage of the available rich text converter, which translates the Notes design entered on a body into HTML. For more information, refer to Page design elements.

Forms

Just like pages, forms enable designers to display information on a page. Everything that can be done with a page can be done with a form. Between forms and pages, forms allow the collection of data of the unique information from the user. To achieve data storage, forms offers a variety of fields of different data types, each of them representing a column in the Lotus Notes database model. When working on the Web, Lotus Domino creates the proper HTML tags for fields and forms on its pages automatically, enabling programmers to code applications much faster. Using forms also enables running several event driven computations. For more information, refer to Form design elements.

Views

Views are the most used way to allow users to navigate into applications' data collected on forms. They are basically used to display a selection of documents from a Lotus Notes database. Views can be sorted, customized and searched. On the Web, views can be formatted in template forms to allow user interaction, search and navigation over the database records. For more information, refer to View design elements.

Folders

Folders are containers that store documents. From the design point of view, folders are similar to views. The main difference between views and folders is that views have a selection of documents, while a folder remains empty until a set of records is moved to it. Folders can be used to display database data on the Web just like views. For more information, refer to Folder design elements.

Agents

Agents are Lotus Notes programs that can run a variety of computations. They run from several application-based events, from user triggered events to scheduled tasks. Agents can do simple tasks, from moving database data, to running complex computation like running Java programs. Agents can be called on several background tasks on the Web, including on forms events and using Domino URLs with arguments. For more information, refer to Agent design elements.

Web Services

A Web service is a self-contained, self-describing, modular application, based on XML, that can be published to and invoked from the Web. Lotus Domino Web Services are based on the WSDL language and SOAP protocol W3C standards. By using Web Services, programmers can modularize enterprise services in a standardized programming model, reducing application maintenance effort and improving reuse.

On Lotus Domino release 8, programmers can also create Web Services consumers to use available Web Services implementations from the Lotus Domino Designer IDE. For more information, refer to Web service design elements.

Outlines

Outlines are simpler to implement navigation on a Lotus Notes application. By using an outline, designers can allow user to access key functionalities from their applications on an intuitive navigation model. Outlines can have diverse entries and can be embedded in pages and forms. Outlines can be also placed in framesets, improving the usage of an application screen. On the Web, outlines help programmers promote navigation and action invoking without needing HTML experience. For more information, refer to outlines.

Shared resources

You can designate many items, such as graphics, fields, subforms, and even programs, as shared resources. Shared resources allow simple access to a set of code used repeatedly over an application, enabling reuse and reducing the application maintenance effort. In the following sections, we discuss the shared resources that are available on a notes application.

Image resources

Image resources are graphic files that can be used across a Lotus Domino application. The use of image resources helps application standardization and reduces the database file size by preventing the usage of repeated images files across an application. Image resources can be invoked on the Web by using embedding through Domino URLs. For more information, refer to Image resource design elements.

Shared fields

Shared fields have their properties defined in an unique field design, and reused across the database in any form or subform. The use of shared fields helps to reduce the effort on application maintenance. Shared fields can be used in conjunction with a variety of reserved fields on the Web to help Web-pages programming and look and feel, including <head> tags definition and current session information through CGI variables. For more information, refer to Shared field design elements.

Non-NSF file resources

Non-NSF file resources allow programmers to host non-Notes binary files in an NSF database for any kind of usage. These resources are also available on the Web through Domino URLs, by allowing programmers to host Web components, such as animations and plug-ins on a central repository, just like a common Web server file system. For more information, refer to Non\-NSF file resources.

Subforms

A *subform* is a collection of designs, such as fields, graphics, buttons, and actions that a designer may reuse in more than one form. They can be invoked programmatically depending on a document or environment property. Subforms can be used on the Web to hold different sections of a page, just like other server-side scripting languages like PHP or Microsoft ASP includes. For more information, refer to Subforms design elements.

Script libraries

A *script library* is a repository to store code that can be shared in an application by using LotusScript, JavaScript, and Java or in other applications using JavaScript and Java. Script libraries allow centralization and reuse. On Web applications, JavaScript libraries can also be accessed through Domino URLs, simplifying the access for cross-organizational routines. For more information, refer to JavaScript library design elements and LotusScript library design elements.

Shared Java files

For big Java applets using various files, it is most efficient to store some of the related files as shared resources in the database. When setting up files as shared resources, all the applets can use a single copy of a file, instead of each applet storing its own copy. Therefore, in case of an update, only one file is updated. For more information, refer to Applet design elements.

Shared actions

Shared actions can be used on forms, subforms, pages, and folders of views that have common user activated tasks. On the Web, shared actions can be displayed using HTML or Java applets. For more information, refer to shared actions.

Cascading style sheets

Cascading style sheets (CSS) can be hosted on a Domino environment for Web pages design information. For more information, refer to Styles and CSS primer.

Composite application components

Composite applications are aggregations of multiple components brought together for a business purpose. Composite applications can be used on a Domino Web application to bring "portalization", and enabling the use of "portlet" like components. For more information, refer to composite application components.

Navigators

Navigators are graphical components that help users navigate through specific parts of an application. Navigators are not a recommended design element for Web applications, since they were surpassed by other components that bring better and more efficient results. For more information, refer to navigators.

Additional topics in this section

- Adding HTML to a design
- All Domino URLs
- CGI variables
- Changing the content type of a design element
- Common design properties on Web applications
- Styling text for the Web
- Working with the DOCTYPE

Adding HTML to a design

This page last changed on Apr 03, 2008 by jservais.

Introduction

There are a number of ways you can include HTML on a page, form, or subform when you are designing. If you have existing HTML or you prefer to use HTML instead of the formatting tools IBM Lotus Domino Designer offers, you have the following options:

- Convert a page, form, or subform (or sections of the page, form, or subform) to HTML and use the HTML editor to change the HTML.
- Import HTML, thus using the source of an existing Web page or form as the base of a new page or form. Designer adds the imported HTML on the page, form, or subform already translated from HTML.
- Paste HTML directly on a page, form, or subform. The HTML stays in HTML format.
- Enter HTML directly on a page, form, or subform. The HTML stays in HTML format.

Converting pages, forms, or subforms into HTML

You can convert some or all of the contents of a page, form, or subform into HTML source code and then use the HTML editor to make changes to the HTML source code.

1. Open a page, form, or subform in Designer.
2. Select the contents of the page, form, or subform that you want to convert to HTML.
3. Choose **Edit -> Convert to HTML**. The selected contents are converted into HTML source code. Because not everything in Notes has an exact equivalent in HTML, the conversion to HTML is an approximation. You should always check your conversion results.
   If you mistakenly convert something to HTML, choose **Edit -> Undo Delete** to recover. Do not choose Edit - Convert to Notes because the conversion is not exact.
   Note Buttons in Web applications that have JavaScript associated with the Click event are converted to HTML as expected. However, buttons that do not have JavaScript associated with the Click event are not converted to HTML. For the buttons that are not converted to HTML, choose **Edit -> Undo Delete** so that the deleted button reappears on the page, form, or subform.
4. To use the HTML editor, place the cursor anywhere in the newly created HTML source code and choose **View -> HTML pane**.

The screen splits. The page, form, or subform appears in the top pane (in an embedded Internet Explorer browser control) and its HTML source code appears in the bottom pane.

5. You can edit the HTML source code in the bottom pane. Click **Refresh** to see the results in the top pane of your HTML changes.
6. Press ESC to exit from the HTML editor.
7. (Optional) To convert the HTML to Lotus Notes format, place the cursor anywhere in the HTML source code in the top pane and choose **Edit -> Convert to Notes Format**.
   Note that the conversion to Notes format is an approximation. You should check your conversion results. If you convert to HTML and then back to Notes, you may get unexpected results.

Importing HTML

To import HTML you must first save it as a file that you can access.

1. Open a page, form, or subform.
2. Choose **File -> Import**.
3. Select the file containing the HTML you want to import and click **OK**. Designer translates the HTML and then adds it to the page, form, or subform.

Pasting HTML

1. Select the content you want to paste from the source of an existing Web page, form, or subform.
2. Copy the content to the clipboard.
3. Open a page, form, or subform.
4. Choose **Edit -> Paste**.
5. Open the Properties box for the page, form, or subform (**Design -> *<design element>* Properties**).
6. On the **Info** tab, select **Render pass through HTML in Notes**.

Entering HTML directly on a page, form, or subform

1. Open a page, form, or subform.
2. Enter the HTML directly on the page, form, or subform.
3. Open the Properties box for the page, form, or subform (**Design -> *<design element>* Properties**).
4. On the Info tab, select **Render pass through HTML in Notes**.
   When you select this option, Lotus Domino passes all data on the page, form, or subform directly to the browser. Domino ignores embedded navigators and folders and any embedded views that don't have this option selected.

Including HTML on a page, form, or subform

If you do not want the entire page, form, or subform treated as HTML, you can include HTML on the page, form, or subform and mark this text as HTML. Domino Designer serves it correctly to the browser.

1. Enter or paste HTML on the page, form, or subform.
2. Select the text and choose **Text -> Pass-Thru HTML**.

Note
Pages, forms, or subforms that contain pass-thru HTML may display differently in the Notes client than on a browser. For example, if you create nested tables by using pass-thru HTML, the table may contain more white space when displayed in the Notes client than when displayed in a browser.

Notes on the form tag

Domino generates its own form tag for editable forms. If you have a unique situation where your application masthead or code has a form tag included, thereby creating a form tag within a form tag, it might not work. For example, some company mastheads have search functionality that is implemented in the HTML source by a form tag that submits to a search engine. Let us assume this case and say the embedded masthead form tag has a title of "Search". Assume also that the domino form name in question is called "Example", and therefore the Domino form tag has a name attribute of "Example". The solution is to use the following steps:

1. Close the default domino form tag for the Domino "Example" form immediately after the self generated <form> tag.
2. Create a new Domino form tag for the "Example" form after the <form> tag for "Search" has been closed with </form>.

This solution prevents embedded form tags that may produce unpredictable behavior.

All Domino URLs

This page last changed on Apr 04, 2008 by heinsje.

Domino URLs

Domino uses URLs to access servers, databases, and other components of a Web site. Knowing the URL commands lets you design links or enter commands directly into a browser to navigate a site or reach components quickly. You can use the URL commands to perform the following actions:

- Open databases and views
- Open framesets
- Open forms, navigators, and agents
- Open, edit, create, save, and delete documents
- Open documents by key name from a view
- Open pages
- Open resources
- Open attachments, image files, and OLE objects
- Open Web preferences
- Create search queries
- Require authentication
- Process SSL certificates
- Specify a character encoding

Attention
The URLs in the following sections are for example only. They are not intended to point to existing Web sites unless specifically indicated.

Quick review of URL syntax for Domino

Domino URL commands have the following syntax:

```
http://Host/Database/DominoObject?Action&Arguments
```

Where:

- *Host* is the DNS entry or IP address.
- *DominoObject* is a Domino construct (such as a database, view, document, form, navigator, agent, and so on).

URL commands for accessing DominoObjects use the following syntax:

```
http://Host/Database/DominoObject?Action&Arguments
```

Where:

- *Database* is the database in which the DominoObject resides.
- *Action* is the action you want on the specified DominoObject (for example, ?OpenDocument).
- *Arguments* are the qualifiers for the action (for example, Count=10 combined with ?OpenView limits the number of rows displayed in a view to 10).

Opening databases and views

The following commands access databases, views, About and Using documents, and database icons.

Redirect

Syntax:

```
http://Host/Database.nsf?Redirect&Name=Notesserver&Id=To=Encodedurl
```

Where:

- http: *//Host* refers to the Web server that is generating the URL.
- Name= *Notesserver* specifies a Domino server name in its common or abbreviated form. This is optional when the "By Database" setting on the server is on.
- Id= indicates the replica ID of the database to be located. This is an optional argument.
- To= *Encodedurl* specifies the rest of the URL.

Example:

```
http://www.riverbendcoffee.com/database.nsf?Redirect&Name=Mail&Id=0525666D0060ABBF&To=%FAView%3FOpenView
```

OpenDatabase

This commands opens a database.

Syntax:

```
http://Host/DatabaseFileName?OpenDatabase http://Host/_DatabaseReplicaID?OpenDatabase
```

Examples:

```
http://www.riverbendcoffee.com/leads.nsf?OpenDatabase
http://www.riverbendcoffee.com/sales/discussion.nsf?OpenDatabase
http://www.riverbendcoffee.com/_852562F3007ABFD6?OpenDatabase
```

OpenView

This command opens a view.

Syntax:

```
http://Host/Database/ViewName?OpenView
http://Host/Database/ViewUniversalID?OpenView
http://Host/Database/$defaultview?OpenView
```

Examples:

```
http://www.riverbendcoffee.com/leads.nsf/By+Salesperson?Open/View
http://www.riverbendcoffee.com/leads.nsf/DDC087A8ACE170F8852562F300702264?OpenView
http://www.riverbendcoffee.com/leads.nsf/$defaultview?OpenView
```

Optional arguments for OpenView

Append these optional arguments to refine the OpenView URL. Combine any of the following arguments for the desired result.

- **Start=n**
  Where *n* is the row number to start with when displaying the view. The row number in a hierarchical view can include sub indexes (for example, Start=3.5.1 means the view will start at the third main topic, subtopic 5, document 1).
- **Count=n**
  Where *n* is the number of rows to display.
- **ExpandView** displays the view in expanded format.
- **CollapseView** displays the view in collapsed format.
- **Expand=n**
  Where *n* is the row number to display in expanded format in a hierarchical view. Do not combine this argument with the ExpandView or CollapseView arguments.
- **Collapse=n**
  Where *n* is the row number to display in collapsed format in a hierarchical view. Do not combine this argument with the ExpandView or CollapseView arguments.
- **RestrictToCategory=category**
  Sets the category for "Show Single Category" object, where *category* is the category to be displayed in the view.
- **StartKey=string**
  Where *string* is a key to a document in the view. The view displays at that document.

Examples:

```
http://www.riverbendcoffee.com/leads.nsf/By+Category?OpenView&CollapseView
http://www.riverbendcoffee.com/leads.nsf/By+Category?OpenView&ExpandndView
http://www.riverbendcoffee.com/leads.nsf/By+Category?OpenView&RestrictToCategory=pricing
http://www.riverbendcoffee.com/leads.nsf/By+Category?OpenView&Start=3&Count=15
http://www.riverbendcoffee.com/leads.nsf/By+Category?OpenView&StartKey=F
```

OpenAbout

Use the $about?OpenAbout command to access the About This Database document.

Syntax:

```
http://Host/Database/$about?OpenAbout
```

Example:

```
http://www.riverbendcoffee.com/leads.nsf/$about?OpenAbout
```

OpenHelp

Use the $help?OpenHelp command to access the "Using This Database" document.

Syntax:

```
http://Host/Database/$help?OpenHelp
```

Example:

```
http://www.riverbendcoffee.com/leads.nsf/$help?Open/Help
```

OpenIcon

Use the $icon?OpenIcon command to access the database icon.

Syntax:

```
http://Host/Database/$icon?OpenIcon
```

Example:

```
http://www.riverbendcoffee.com/leads.nsf/$icon?OpenIcon
```

ReadViewEntries

Use this command to access view data in XML form without appearance attributes such as fonts, list separators, date formats, HTML settings, view templates and frame redirections.

Syntax:

```
http://Host/Database/ViewName?ReadViewEntries
http://Host/Database/ViewUniversalID?ReadViewEntries
```

```
http://Host/Database/$defaultview?ReadViewEntries
```

Examples:

```
http://www.riverbendcoffee.com/leads.nsf/By+Salesperson?ReadViewEntries
http://www.riverbendcoffee.com/leads.nsf/DDC087A8ACE170F8852562F300702264?ReadViewEntries
http://www.riverbendcoffee.com/leads.nsf/$defaultview?ReadViewEntries
```

Optional arguments for ReadViewEntries

Append optional arguments to refine the URL. Combine any of the following arguments for the desired result.

- **Collapse=n**
  Where *n* is the row number to display in collapsed format in a hierarchical view. Do not combine this argument with the ExpandView or CollapseView arguments.
- **CollapseView** displays the view in collapsed format.
- **Count=n**
  Where *n* is the number of rows to display
- **Expand=n**
  Where *n* is the row number to display in expanded format in a hierarchical view. Do not combine this argument with the ExpandView or CollapseView arguments.
- **ExpandView** displays the view in expanded format.
- **KeyType=textortime**
  Specifies the StartKey type of either text or time. If no argument is specified, the default is text. When you specify &KeyType=time, you can specify a time value, like ISO date time value, for both the &StartKey and &UntilKey arguments.
- **PreFormat**
  Causes all data types to be converted to text on the server. Text lists, numbers, dates and lists of numbers are converted to text before being sent. The server's locale is used for all formatting. Without this argument, the XML output stream contains information in structured, locale-neutral formats.
- **ResortAscending=column number** or **ResortDescending=column number**
  Where column number is a 0-based number of a column in a view that you want to resort either ascending or descending in alphanumeric order.
- **RestrictToCategory=category**
  Sets the category for the "Show Single Category" object, where *category* is the category to be displayed in the view Start=nWhere *n* is the row number to start with when displaying the view. The row number in a hierarchical view can include sub indexes (for example, Start=3.5.1 means the view will start at the third main topic, sub-topic 5, document 1).
- **StartKey=string**
  Where *string* is a key to a document in the view. The view displays at that document.
- **UntilKey=string**
  Displays a range of view entries that begin with the document specified by the StartKey and end with the document specified by the UntilKey. The &UntilKey argument is only valid with the &StartKey argument. You can use the &Count argument to limit the number of entries returned by the range.
- **Outputformat=JSON**
  For further support of AJAX Web applications, Lotus Domino 8 provides JavaScript Object Notation (JSON) as an output format to let you more quickly create AJAX Web applications.

Example:

```
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&CollapseView
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&ExpandView
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&KeyType=time&StartKey=20020715&UntilKey
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&KeyType=text&StartKey=Aa&UntilKey=Ab
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&PreFormat
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&ResortAscending=3
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&ResortDescending=3
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&RestrictToCategory=pricing
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&Start=3&Count=15
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&StartKey=F
http://www.riverbendcoffee.com/leads.nsf/By+Category?ReadViewEntries&Outputformat=JSON
```

Opening framesets

This command opens framesets.

OpenFrameset

Syntax:

```
http://Host/Database/FramesetName?OpenFrameset
http://Host/Database/FramesetUNID?OpenFrameset
```

Examples:

```
http://www.riverbendcoffee.com/discussion.nsf/main?OpenFrameset
http://www.riverbendcoffee.com/discussion.nsf/35AE8FBFA573336A852563D100741784?OpenFrameset
```

Opening forms, navigators, and agents

The following commands open forms, navigators, and agents in a database.

OpenForm

Syntax:

```
http://Host/Database/FormName?OpenForm
http://Host/Database/FormUniversalID?OpenForm
http://Host/Database/$defaultform?OpenForm
```

Examples:

```
http://www.riverbendcoffee.com/products.nsf/Product?Openform
http://www.riverbendcoffee.com/products.nsf/625E6111C597A11B852563DD00724CC2?OpenForm
http://www.riverbendcoffee.com/products.nsf/$defaultform?OpenForm
```

Optional arguments for OpenForm

ParentUNID =UniqueIDNumber
Where *UniqueIDNumber* is the document ID of the parent document, which is used in response forms or when the form property "Formulas inherit values from selected document" is selected.

Syntax:

```
http://Host/Database/FormUniversalID?OpenForm&ParentUNID=UniqueIDNumber
```

Example:

```
http://www.riverbendcoffee.com/products.nsf/
40aa91d55cle4c8285256363004dc9e0?OpenForm&ParentUNID=6bc72a92613fd6bf852563de001f1a25
```

OpenNavigator

Syntax:

```
http://Host/Database/NavigatorName?OpenNavigator
http://Host/Database/NavigatorUniversallID?OpenNavigator
http://Host/Database/$defaultNav?OpenNavigator
```

Examples:

```
http://www.riverbendcoffee.com/products.nsf/Main+Navigator?OpenNavigator
http://www.riverbendcoffee.com/products.nsf/7B5BC17C7DC9EB7E85256207005F8862?OpenNavigator
http://www.riverbendcoffee.com/products.nsf/$defaultnav?OpenNavigator
```

Note
$defaultnav opens the folders pane in a database.

OpenAgent

Syntax:

```
http://Host/Database/AgentName?OpenAgent
```

Example:

```
http://www.riverbendcoffee.com/sales/leads.nsf/Process+New+Leads?OpenAgent
```

Note: Agents may only be referred to by name. The use of UNID is not supported when referring to an agent.

ReadForm

Use the ReadForm command to display a form without showing its editable fields. ReadForm is useful for displaying a form as a simple Web page.

Syntax:

```
http://Host/Database/FormName?ReadForm
http://Host/Database/FormUniversalID?ReadForm
http://Host/Database/$defaultform?ReadForm
```

Examples:

```
http://www.riverbendcoffee.com/home.nsf/Welcome?ReadForm
http://www.riverbendcoffee.com/products.nsf/625E6111C597A11B852563DD00724CC2?ReadForm
http://www.riverbendcoffee.com/products.nsf/$defaultform?ReadForm
```

Creating, opening, editing, saving, and deleting documents

The following commands manipulate documents in a database. Hidden design elements are hidden from the server. You cannot use Domino URL commands to access documents in hidden views.

CreateDocument

The CreateDocument command is used as the POST action of an HTML form. When the user submits a form, Domino obtains the data entered in the form and creates a document.

Syntax:

```
http://Host/Database/Form?CreateDocument
http://Host/Database/FormID?CreateDocument
```

Examples:

```
http://www.riverbendcoffee.com/products.nsf/basketballs?CreateDocument
http://www.riverbendcoffee.com/products.nsf/b9815a87b36a85d9852563df004a9533?CreateDocument
```

OpenDocument

Syntax:

```
http://Host/Database/View/DocumentKey?OpenDocument
http://Host/Database/View/DocumentUniversalID?OpenDocument
http://Host/Database/View/$First?OpenDocument
```

Note: DocumentKey is the contents of the first sorted column in the specified view.

Examples:

```
http://www.riverbendcoffee.com/products.nsf/By+Part+Number/PC156?OpenDocument
http://www.riverbendcoffee.com/leads.nsf/By+Rep/35AE8FBFA573336A852563D100741784?OpenDocument
http://www.riverbendcoffee.com/leads.nsf/$First?OpenDocument
```

Optional arguments for OpenDocument

See the Optional outline arguments sidebar for outline arguments that apply to both OpenDocument and OpenPage.

EditDocument

Syntax:

```
http://Host/Database/View/Document/?EditDocument
```

Example:

```
http://www.riverbendcoffee.com/products.nsf/By+Part+Number/PC156?EditDocument
```

Note: Rich text fields containing hidden text will be visible to Web users with editor access to documents.

SaveDocument

The SaveDocument command is used as the POST action of a document being edited. Domino updates the document with the new data entered in the form.

Syntax:

```
http://Host/Database/View/Document?SaveDocument
```

Example:

```
http://www.riverbendcoffee.com/products.nsf/
a0cefa69d38ad9ed8525631b006582d0/4c95c7c6700160e2852563df0078cfeb?SaveDocument
```

DeleteDocument
Syntax:

```
http://Host/Database/View/Document?DeleteDocument
```

Example:

```
http://www.riverbendcoffee.com/products.nsf/By+Part+Number/PC156?DeleteDocument
```

Opening documents by key

The following commands allow you to open a document by key, or to generate a URL to link to a document by key.

Using Domino URLs to access a document

To open a document by key, create a sorted view with the sort on the first key column. Then you can use a URL to open the document.

Syntax:

```
http://Host/DatabaseName/View/DocumentName?OpenDocument
```

View is the name of the view, and DocumentName is the string, or key, that appears in the first sorted or categorized column of the view. Use this syntax to open, edit, or delete documents, and to open attached files. Domino returns the first document in the view whose column key exactly matches the DocumentName.

There may be more than one matching document. Domino always returns the first match. The key must match completely for Domino to return the document. However, the match is not case-sensitive or accent-sensitive.

Note
View can be a view UNID or view name. In addition, the implicit form of any of these commands will work when appropriate. EditDocument and DeleteDocument must be explicit commands.

Examples:

```
http://www.riverbendcoffee.com/register.nsf/Registered+Users/Jay+Street?OpenDocument
```

LDD Today uses a document key view called *Lookup*. For example, the URL for this article is:
http://www.lotus.com/ldd/today.nsf/lookup/Domino_URL_cheat_sheet?OpenDocument
To get a closer look at the Lookup view, you can download the LDD Today design template from the Sandbox here on LDD.

Using Domino URLs to access attachments

To access a file attachment by using a Domino URL, you must know the view name, the document name, and the file attachment name. Domino generates an URL for file attachments when it saves the documents to which the files are attached. These URLs end with the file name of the attachment.

Syntax:

```
http://Host/DatabaseName/View/DocumentName/$File/fileattachmentname
```

View is either the view name or the view ID, and DocumentName is the document name or ID. $File is a special identifier that indicates an attachment on a document. Fileattachmentname is the file name of the attachment.

Examples:

```
http://www.riverbendcoffee.com/products.nsf/Documents/$File/Spec_sheet.pdf
```

Opening pages

The following command opens a page element using its name, UNID, or Note ID.

OpenPage

Syntax:

```
http://Host/Database/PageName?OpenPage
http://Host/Database/PageUNID?OpenPage
```

Examples:

```
http://www.riverbendcoffee.com/discussion.nsf/products?OpenPage
http://www.riverbendcoffee.com/discussion.nsf/35AE8FBFA573336A852563D100741784?OpenPage
```

Optional arguments for OpenPage

See the Optional outline arguments sidebar for outline arguments that apply to both OpenDocument and OpenPage.

Opening resources

The following commands open image and file resources stored in an database.

OpenImageResource

Opens graphics stored as image resources in a database.

Syntax:

```
http://Host/Database/ImageResourceName?OpenImageResource
```

ImageResourceName is the file name of the image resource that you want to open.

Example:

```
http://www.riverbendcoffee.com/discussion.nsf/banner.gif?OpenImageResource
```

OpenFileResource

Opens a file resource stored in a database.

Syntax:

```
http://Host/Database/FileResourceName?OpenFileResource
```

Where FileResourceName is the name of the file that you want to open.

Example:

```
http://www.riverbendcoffee.com/discussion.nsf/index.js?OpenFileResource
```

Opening attachments, image files, and OLE objects

The ?OpenElement command opens attachments, image files, and OLE objects within a document.

Using ?OpenElement with file attachments

Syntax:

```
http://Host/Database/View/Document/$File/Filename?OpenElement
```

Example:

```
http://www.riverbendcoffee.com/lproducts.nsf/By+Part+Number/SN156/$File/spec.txt?OpenElement
```

Note: If more than one attached file has the same name, the URL includes both the "internal" file name as well as the external name. Since the internal file name is not easily determined, make sure that all attached files have unique names.

Domino treats all file attachment OpenElement commands as implicit commands, because some browsers

require that the URL end with the attached file name.
Example:

```
http://Host/Database/View?Document/$File/FileName
```

Using ?OpenElement with image files

Syntax:

```
http://Host/Database/View/Document/FieldName/FieldOffset?OpenElement&FieldElemFormat=ImageFormat
```

FieldOffset is the field number and the byte offset into the field. ImageFormat is either GIF or JPG. If the FileElemFormat is not entered, Domino assumes the image file format is GIF.

Example:

```
http://www.riverbendcoffee.com/leads.nsf/
bbe63a6b9d895dc6852567d600658601/fe5138bef254cf3a852569fc00724b69/Body/0.18AA?OpenElement&FieldElemFormat=jp
```

Using Open Element with OLE Objects

Syntax:

```
http://Host/Database/View/Document/FieldName/FieldOffset/$OLEOBJINFO/FieldOffset/obj.ods?OpenElement
```

Note: The current URL syntax for referencing images and objects in Notes documents-specifically the FieldOffset-makes it impractical to create these URLs manually. As an alternative, you may paste the actual bitmap or object in place of the reference, create URL references to files stored in the file system, or attach the files to the documents.

Opening user Web preferences

The following command opens Web preferences, a Domino feature that lets users set time zone and regional preferences.

OpenPreferences

Syntax:

```
http://Host/$Preferences.nsf?OpenPreferences&Argument
```

Where:

- Host indicates a server or a domain
- $Preferences.nsf is a virtual database that "resides" on the Domino server
- ?OpenPreferences displays the default frameset of the virtual database
- &Argument is an optional argument that you can specify to open a page instead of the frameset

The $Preferences.nsf database resides at the root of each server.

Example:

```
http://www.riverbendcoffee.com/$Preferences?OpenPreferences
```

Optional argument for OpenPreferences

You can append the following optional arguments to the ?OpenPreferences command to open a specfiied page rather than the Web preferences default frameset. PreferenceType=valueWhere *value* can be one of the following values described in the table.

Value	Description
Menu	Displays the Menu page that provides links to the Time Zone and Regional preferences page.
TimeZone	Displays the Time Zone preferences page.
Regional	Displays the Regional preferences page.

Examples:

```
http://www.riverbendcoffee.com/$Preferences?OpenPreferences&PreferenceType=Menu
http://www.riverbendcoffee.com/$Preferences?OpenPreferences&PreferenceType=TimeZone
http://www.riverbendcoffee.com/$Preferences?OpenPreferences&PreferenceType=Regional
```

Creating search queries

Search-related URLs are available for performing view, multiple-database, and domain searches. Typically you define a URL that displays an input form-either a customized search form or the default search form-to let users define their own searches, but you may also define a URL that performs text searches without user input. Both input and results forms may be customized.

SearchDomain

Use SearchDomain URLs for text searches across a domain. The search input form is opened with the OpenForm command by name or universal ID. For search results, the results template is specified as part of the URL. If no template is found, then the default template form, $$SearchDomainTemplate, is substituted. If $$SearchDomainTemplate is not found, an error will be returned. If no results are returned, the value of the $$ViewBody field remains the same.

Syntax:

```
http://Host/Database/TemplateForm?SearchDomain&ArgumentList
```

Where:

- TemplateForm is an optional argument that calls the search results form.
- ArgumentList is a list of optional arguments.

Example:

```
http://www.riverbendcoffee.com/domainsearch.nsf/SearchForm?SearchDomain
```

SearchSite

Use SearchSite URLs for text searches in multiple databases. Because the URL requires the name of a search site database, be sure to create one before using a SearchSite URL.

Syntax:

```
http://Host/Database/$SearchForm?SearchSite&ArgumentList
```

Where $SearchForm and ArgumentList are optional arguments.

Example:

```
http://www.riverbendcoffee.com/searchsite.nsf/$SearchForm?SearchSite
```

SearchView

Use SearchView URLs to limit a search to documents displayed in one database view. This URL is useful for views that display all documents (so you can have a full-database search) or for views in which you can predict what users need to see, such as all documents whose status is *Completed*.

Syntax:

```
http://Host/Database/View/$SearchForm?SearchView&ArgumentList
```

Where $SearchForm and ArgumentList are optional arguments. The special identifier $SearchForm indicates that Domino will present a search view form for search input. If this identifier is provided, the ArgumentList is ignored. If this identifier is absent, a default form is generated on the fly based on the contents of the search.htm file located on the server. The default form generated by the server does not support paged results.

Example:

```
http://www.riverbendcoffee.com/products.nsf/By+Product+Number/$SearchForm?SearchView
```

Optional arguments for SearchSite, SearchView, and SearchDomain

- **$SearchForm**
  $SearchForm is a special identifier indicating a custom search form that Domino displays. When this argument is specified, Domino ignores all arguments that follow it. If this argument is not specified, Domino displays a default search form based on the search.htm file on the server.
- **Query=string**
  Where *string* is the search string.
- **Count=n**
  Where *n* is the number of results to display on each page until the SearchMax has been reached. For example Count=10 displays 10 results per page.
- **Scope=[0,1,2]**
  Where 1=Notes databases only, 2=file system only, 0=both. The default value is 0. This argument should only be used with the SearchDomain command.
- **SearchEntry=formName**
  Where *formName* is the name of the form to use for the results of a domain search. The default argument is "ResultEntry," which supports all of the predefined results fields specified in the ArgumentList. This argument is valid for SearchDomain only and should not be used for SearchSite or SearchView.
- **SearchFuzzy=[TRUE,FALSE]**
  Indicate TRUE for fuzzy search. The default is FALSE.
- **SearchOrder=[1,2,3,4]**
  Indicate 1 to "Sort by relevance", 2 to "Sort by date ascending", 3 to "Sort by date descending." The default is 1. SearchView also supports a SearchOrder value of 4 to "Keep current order", which sorts the resulting set of documents in the order in which they appear in the view.
- **SearchMax=n**
  Where *n* is the maximum number of entries returned. The default value is determined by the server.
- **SearchWV=[TRUE, FALSE]**
  Where TRUE = include word variants in the search. The default value is FALSE.
- **Start=n**
  Where *n* is the number corresponding to the document that appears first in your list of results. For example, Start=10 begins your list of results with the 10th document found in the search. Start=0 means that paged results will not be returned.

You can use the Start and Count arguments with the SearchView or SearchSite URLs as well as with the search results page to display search results page-by-page. The Start argument specifies which result appears first in the search results list. The Count argument determines the number of results displayed on the screen. For instance, if you specify Start=1 and Count=10, the search results begin with the first result and displays the next ten results on the screen. If results extend beyond ten, you can use button or hotspots to navigate the search results pages.

For more information about creating buttons or hotspots for the Start and Count arguments, refer to Creating custom and advanced searches using Domino.

Examples:

```
http://www.riverbendcoffee.com/welcome.nsf/
?SearchSite&Query=product+info+requests&SearchOrder=2&SearchMax=30&SearchWV=TRUE&SearchEntry="myResultForm"
```

```
http://www.riverbendcoffee.com/products.nsf/By+Product+Number/
?SearchView&Query=PC156&SearchOrder=3&SearchMax=1&SearchFuzzy=TRUE&SearchWV=FALSE
```

Login

Syntax:

```
http://Host/Directory/Database?OpenDatabase&Login
```

Example:

```
http://www.riverbendcoffee.com/sales/leads.nsf?OpenDatabase&Login
```

OpenForm with SpecialAction argument

Syntax:

```
http://Host/Database/FormName?OpenForm&SpecialAction=specialActionField
```

Where *specialActionField* is the name of an editable text field on the form whose value contains a predefined command. To use the field with SSL certificates, use one of the following certificate request commands:

- SubmitCert
- ServerRequest
- ServerPickup

Examples:

```
http://www.riverbendcoffee.com/certs.nsf/UserCertificateRequest?OpenForm&SpecialAction=SubmitCert
http://www.riverbendcoffee.com/certs.nsf/ServerCertificateRequest?OpenForm&SpecialAction=ServerRequest
http://www.riverbendcoffee.com/certs.nsf/Certificate?OpenForm&SpecialAction=ServerPickup
```

SubmitCert

The SubmitCert command creates a User Certificate document in the specified database, using the form specified in the TranslateForm argument.

Syntax:

```
http://Host/Database/ResultForm?RequestCert&Command=SubmitCert&TranslateForm=TranslationFormName
```

Where:

- ResultForm is a form in the specified database that displays information about the processed request.
- TranslationFormName represents a form in the database that contains fields to hold certificate information.

Example:

```
http://www.riverbendcoffee.com/certs.nsf/
CertificateProcessed?RequestCert&Command=SubmitCert&TranslateForm=Certificate&TranslateForm=Certificate
```

Optional and required fields

The SubmitCert command requires a translation form with a field named Certificate. Domino saves information about the certificate subject and issuer in the document if the form contains fields with these names:

- CommonName
- Org
- OrgUnit
- Locality
- State
- Country
- IssuerCommonName
- IssuerOrg
- IssuerOrgUnit
- IssuerLocality
- IssuerState
- IssuerCountry

ServerRequest

The ServerRequest command creates a Server Certificate Request document in the specified database, by using the form specified in the TranslateForm argument.

Syntax:

```
http://Host/Database/MessageForm?RequestCert&Command=ServerRequest&TranslateForm=TranslationFormName
```

ResultForm is a form in the specified database that displays information about the processed request in the user's browser after a successful submission. TranslationFormName represents a form in the database that contains fields to hold certificate information.

Example:

```
http://www.riverbendcoffee.com/certs.nsf/
CertificateProcessed?RequestCert&Command=ServerRequest&TranslateForm=Certificate&TranslateForm=Certificate
```

Optional and required fields

The ServerRequest command requires a translation form with a field named Certificate. Domino saves information about the server request in the document if the form contains fields with these names:

- CommonName
- Org
- OrgUnit
- Locality
- State
- Country

Specifying character encoding

To specify character encoding for a design element, append the charset= *MIME* charset argument to the end of any URL command. You can use this argument with any design element or Notes object, including agents, folders, views, databases, and so on. This argument returns a form or page in the specified language or character set overriding the Web browser's preferred language setting as well as the $$HTMLContentLang field of a form. To use the charset=MIME charset argument, you must include it in your application. The Domino server does not generate this argument automatically.

Syntax:

```
http://Host/Form?OpenForm&charset=MIMEcharset
```

Form is either the form name or ID to open and *MIME* charset indicates the character encoding applied to the form.

Domino recognizes a limited number of character set names. If Domino does not recognize a specified character set, it defaults to the character set specified in the Server document.

Example:

```
http://www.riverbendcoffee.com/products.nsf/Product?Openform&charset=ISO-2022-JP
```

The previous example opens the Product form with a Japanese character encoding.

CGI variables

This page last changed on Apr 03, 2008 by jservais.

Using CGI in a Domino application

Common Gateway Interface (CGI) is an Internet standard for external application connection across HTTP servers. With CGI programs, programmers can add back-end processing over a Web page. To run external CGI programs on a Domino environment, you should place them in the default cgi-bin directory or in a directory that has execute access. Because Domino does not maintain access control at the file system level, scripts must include access control measures to prevent unauthorized use. For further information, refer to security considerations.

Diverse CGI variables are available to designers on Domino. When an user saves or opens an existing document on the Web for example, the Domino Web server uses CGI variables to collect information about the user, including the user's name, browser, and the user's Internet Protocol (IP) address.
To capture this information in a Web application, you can use several techniques, for example:

- Create a field with the name of a CGI variable. Example: Query_String.
- Create an agent whose script contains a CGI variable using the NotesDocument.DocumentContext property.
- Create a $$Return field on your form, and invoke a CGI variable combined with some HTML instructions to display a success message for submitted documents.

List of CGI variables on Domino

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

Field name	Returns
Auth_Type	If the server supports user authentication and the script is protected, this is the protocol-specific authentication method used to validate the user.
Content_Length	The length of the content, as given by the client.
Content_Type	For queries that have attached information, such as HTTP POST and PUT, this is the content type of the data.

Gateway_Interface	The version of the CGI spec with which the server complies.
HTTP_Accept	The MIME types that the client accepts, as specified by HTTP headers.
HTTP_Accept_language	The languages that the client accepts, as specified by HTTP headers.
HTTP_Referer	The URL of the page the user used to get here.
HTTPS	Indicates if SSL mode is enabled for the server.
HTTPS_CLIENT_CERT_COMMON_NAME	The common name on the x.509 certificate
HTTPS_CLIENT_CERT_ISSUER_COMMON_NAME	The issuer of the x.509 certificate
HTTPS_KEYSIZE	The session key during an SSL session. For example, 40-bit, 128-bit.
HTTP_User_Agent	The browser that the client is using to send the request.
Path_Info	The extra path information (from the server's root HMTL directory), as given by the client. In other words, scripts can be accessed by their virtual path name, followed by extra information that is sent as PATH_INFO.
Path_Info_Decoded	Returns the same as Path_Info, but decodes the string. For example, if a URL references a view name that contains characters that are not allowed a URL, the name is encoded. This CGI variable decodes the string. Path_Info_Decoded is available to Domino applications only.
Path_Translated	The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.
Query_String	The information that follows the question mark (?) in the URL that referenced this script. **Note**: If your Domino server is configured to allow search engines to search your Web site, Domino generates URLs with an exclamation mark (!) instead of a question mark (?). If this is the case, the Query_String CGI variable includes the information that follows the exclamation mark (!). Domino always recognizes both the question mark (?) and the exclamation mark (!), but only generates URLs with the exclamation mark (!) if your site is accessible to Web search engines. Generating URLs with an exclamation mark (!) makes them more searchable.
Query_String_Decoded	Returns the same as Query_String, but decodes the string. For example, if a URL references a view name that contains characters that are not allowed in a URL, the name is encoded. This CGI variable decodes that string. Path_Info_Decoded is

	available to Domino applications only.
Remote_Addr	The IP address of the remote host making the request.
Remote_Host	The name of the host making the request.
Remote_Ident	This variable is set to the remote user name retrieved from the server. Use this variable only for logging.
Remote_User	Authentication method that returns the authenticated user name.
Request_Content	Supported only for agents. Contains the data sent with an HTTP POST request. The data is usually "URLencoded," consisting of name=value pairs concatenated by ampersands. For example, FirstName=John&LastName=Doe
Request_Content_*nnn*	Used when the amount of data to be sent with an HTTP POST request exceeds the 64K limit. The first 64K of data is sent in Request_Content_000, the second 64K of data is sent in Request_Content_001, and so on.
Request_Method	The method used to make the request. For HTTP, this is "GET," "HEAD," "POST," and so on.
Script_Name	A virtual path to the script being executed, used for self-referencing URLs.
Server_Name	The server's host name, DNS alias, or IP address as it would appear in self-referencing URLs.
Server_Protocol	The name and revision of the information protocol accompanying this request.
Server_Port	The port to which the request was sent.
Server_Software	The name and version of the information server software running the CGI program.
Server_URL_Gateway_Interface	The version of the CGI spec with which the server complies.

For more information about CGI environment variables, refer to http://hoohoo.ncsa.uiuc.edu/cgi/env.html.

Changing the content type of a design element

This page last changed on Apr 03, 2008 by jservais.

- Changing the content type on forms and pages
- Changing the content type on agents

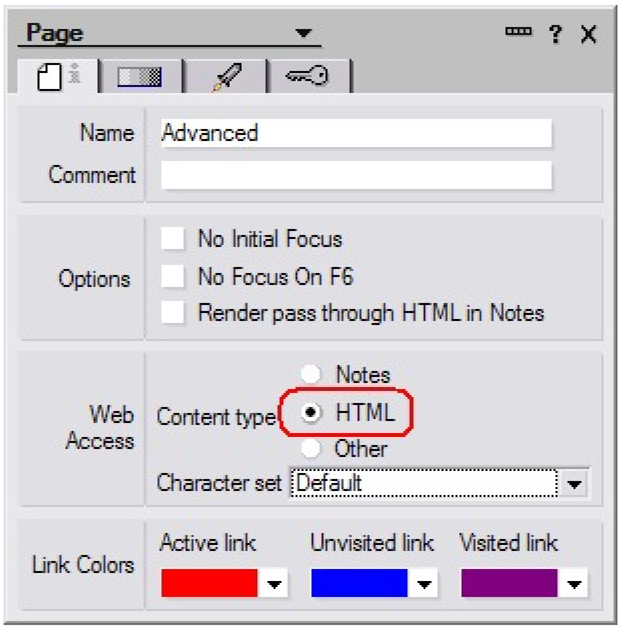
Content type, also known as MIME type, specifies the nature of a resource. The content type is used to specify the nature of a data in the body of a MIME entity, by giving media type and subtype identifiers, and by providing auxiliary information that may be required for certain media types. Examples of content types include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic". Thus, despite the fact that a file can have an .htm extension, if it does not have a "text/html" header content type, it is not rendered on a browser as a common hypertext Web page. The same can happen with images, video clips and many others.

Domino provides a way to change the content type header of some elements. In the next section, we explain how to achieve this on several design elements.

Changing the content type on forms and pages

Changing the content type to HTML

To change the content type of a form or page to HTML, in the design element Properties box, on the Basics tab, for Web Access, select **HTML** for Content type.



Page

Name: Advanced
Comment:
Options: ☐ No Initial Focus ☐ No Focus On F6 ☐ Render pass through HTML in Notes
Web Access: Content type: Notes [✓ HTML] Other
Character set: Default
Link Colors: Active link 🟥 Unvisited link 🟦 Visited link 🟪

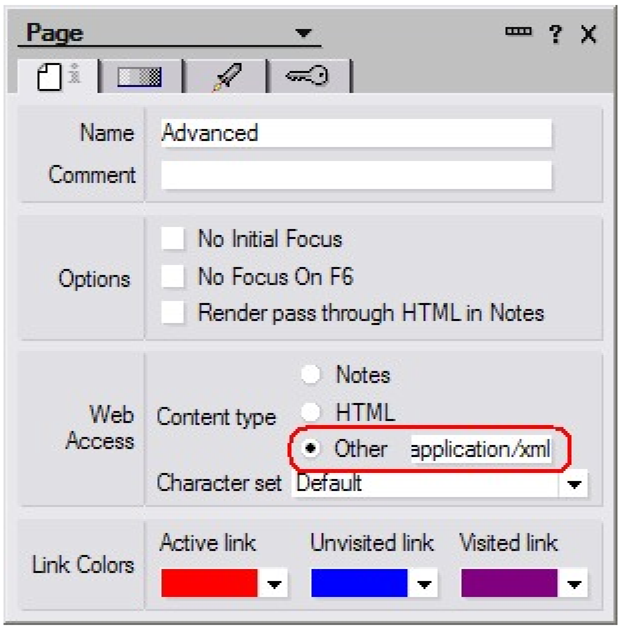
Setting the content type of a page design element to text/html

When this setting is applied, all the content of a form is rendered as is. You do not need to format text using Notes rich text or pass through HTML setting to the text.

Important
When a content type different from the default (*Notes*) is set, all the design element properties, such as window title, JS header code, and so on, are lost, and the page is treat as text. The unique portion of code evaluated by the Web browser is the plain text entered on the design element body.

Changing the content type to XML

To change the content type of a form or page to XML, in the design element Properties box, on the Basics tab, for Web Access, select Other for Content type and an enter the value "application/xml" on the input box.



Page

Name	Advanced
Comment	
Options	☐ No Initial Focus ☐ No Focus On F6 ☐ Render pass through HTML in Notes
Web Access	Content type: Notes [✓ Other application/xml] HTML Character set: Default
Link Colors	Active link 🟥 / Unvisited link 🟦 / Visited link 🟪

Setting the content type of a design to application/xml

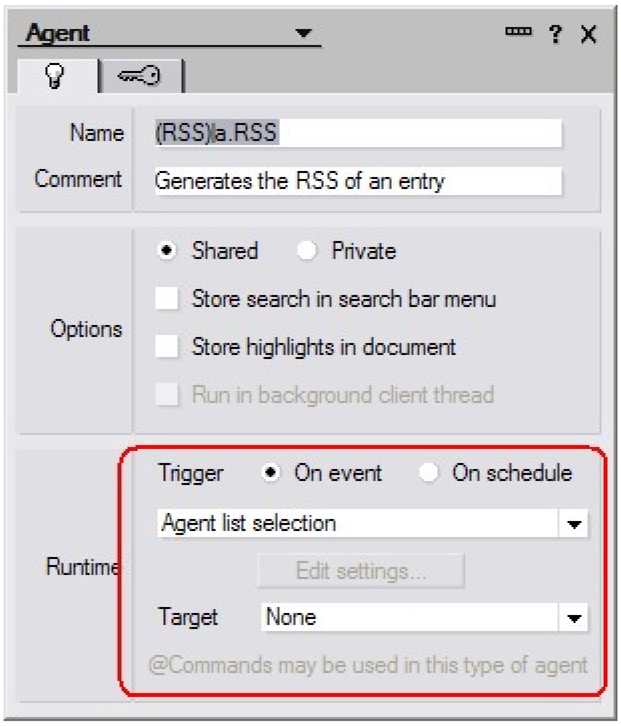
Changing the content type on agents

Agents can be used to write custom content on the Web. Designers can take advantage of this, and change the content type of an agent called from an URL to write an specific content. To achieve this, make sure that the first line "printed" from your agent is the definition of its content type (header information).

```
Print "Content-type: text/xml " & Chr(13)
```

The previous example demonstrates how to change the content type of an agent on run-time to XML using the LotusScript Print statement. This can be used to display custom XML data, like RSS feeds.

Tip
To call an agent from an URL that writes data for the user on-the-fly, in the agent properties box, on the Basics tab, for Runtime, set Trigger to **On event**. Then select **Agent list selection**, and set Target to **None**.


Agent

Name: (RSS)\a.RSS
Comment: Generates the RSS of an entry

Options: [✓ Shared] Private
☐ Store search in search bar menu
☐ Store highlights in document
☐ Run in background client thread

Runtime
Trigger: [✓ On event] On schedule
Agent list selection
Edit settings...
Target: None
@Commands may be used in this type of agent

Setting an agent to be invoked from an URL and print data

For information about how to work with the Doctype declaration on a Web page, refer to Working with the DOCTYPE.

Common design properties on Web applications

This page last changed on Apr 03, 2008 by jservais.

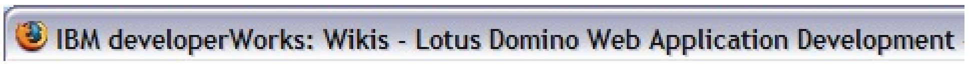
Lotus Domino has a variety of properties to help the programmer customize a Web application. Follow the information provided in the following sections for tips and best practices on how to use them to suit your application needs.

Pages and forms

Pages and forms share several common properties. In this section, we explain how to set these properties on your Web applications.

Window title

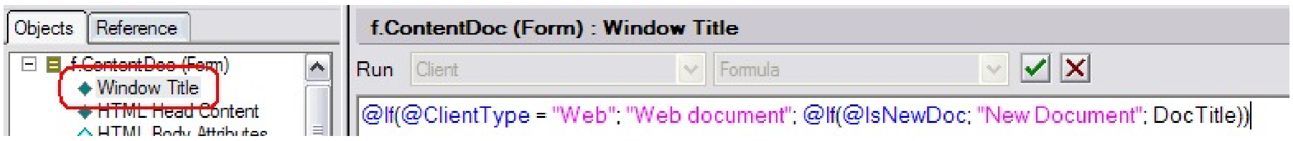
In a Web application, the window title is the text displayed in the web browser program title bar.


IBM developerWorks: Wikis - Lotus Domino Web Application Development

Firefox Web browser window title

The window title is a hybrid property, composed of an @Formula string. This string can have different values if the user is using Notes client or the Web. If you have a hybrid application, that has a form or page that is used on both Notes client and the Web, you can set the window title to display a text according to the application, using the @ClientType formula.

The following example, we illustrate how to use this property. If for example, the user is accessing the design from the example above from a Web browser, the window title is going to be "Web document". If the user is running the design Notes client, and that document was already saved, the title is going to be the value of the DocTitle variable stored in a field, or, if that is a new document, that is going to display "New document".



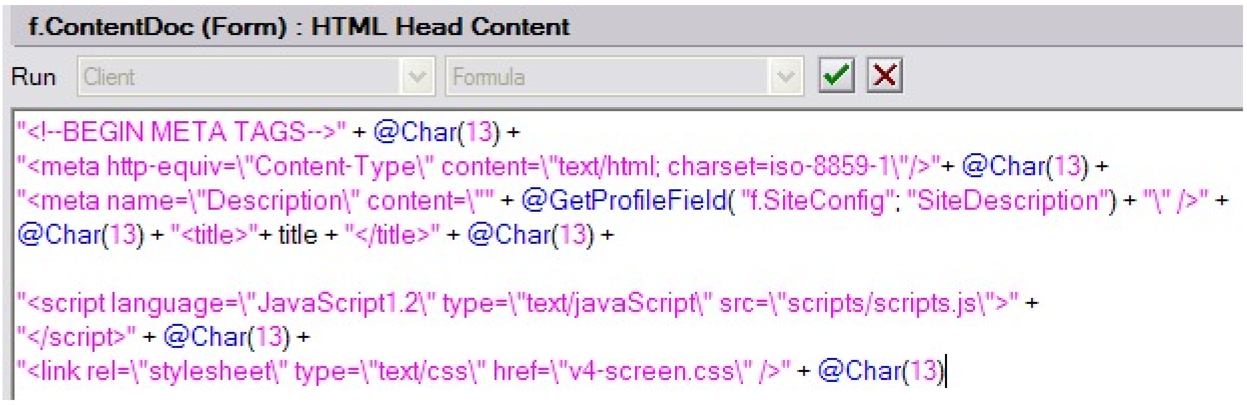
```
@If(@ClientType = "Web"; "Web document"; @If(@IsNewDoc; "New Document"; DocTitle))
```

Editing the design Window title property

Important
In a Web application, the window text set in this Window title property can have its value ignored by the Web browser. It happens when the user sets a <title> tag on the HTML head content property of a Web application.

HTML head content

The HTML head content of a form is the HTML portion that is going to be contained into the <head> tag of your web application. In this property, you can enter properties like meta tags, JavaScript code, reference stylesheets and add special page attributes, including a favicon. The following figure illustrates how to use this property.



```
"<!--BEGIN META TAGS-->" + @Char(13) +
"<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\"/>"+ @Char(13) +
"<meta name=\"Description\" content=\"" + @GetProfileField( "f.SiteConfig"; "SiteDescription") + "\" />" +
@Char(13) + "<title>"+ title + "</title>" + @Char(13) +

"<script language=\"JavaScript1.2\" type=\"text/javaScript\" src=\"scripts/scripts.js\">" +
"</script>" + @Char(13) +
"<link rel=\"stylesheet\" type=\"text/css\" href=\"v4-screen.css\" />" + @Char(13)
```

The design HTML Head content

By using this property, you may refer to design elements hosted on your Notes database. Therefore, you could have calls to a style sheet by using code such as in the following example:

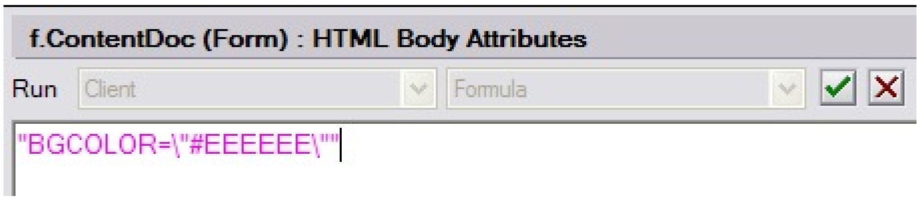
```
<link rel="stylesheet" type="text/css" href="mydb.nsf/ui.css?OpenImageResource" />
<link href="favicon.ico?OpenElement" rel="shortcut icon" type="image/x-icon" />
```

Tip
If you have several forms with similar code on the HTML head content, consider reusing this using the $$HTMLHead field. For more information, refer to the field documentation for $$HTMLHead on special reserved fields.

For more information about how to invoke Domino elements on the Web using Domino URLs, refer to All Domino URLs.

HTML body attributes

The HTML head content of a form is the HTML portion that is contained into the <body> tag of a Web application. In this form attribute, you can define properties like the page background color and some other HTML settings. The example in the following figure illustrates how to use this property.



```
"BGCOLOR=\"#EEEEEE\""
```

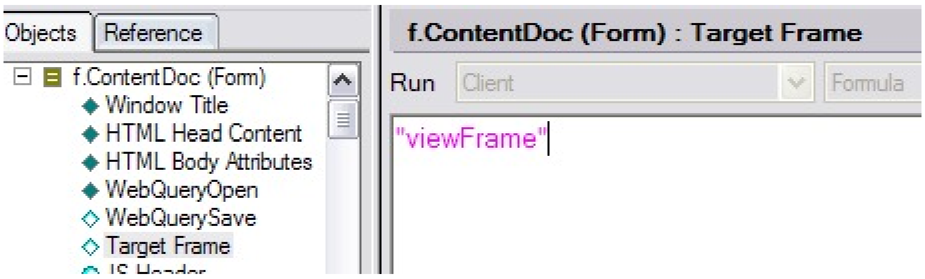
(Editing the HTML body attributes)

Tip
Set the Web page style properties using style sheets instead of using the HTML body attributes. This best practice reduces the maintenance effort over a Web application.

To set other attributes like the code that goes on the <html> tag for your page, and the code that goes before the <html> tag, refer to the field documentation for $$HTMLTagAttributes and $$HTMLFrontMatter on special reserved fields.

Target frame

In addition to adding content to a frame, you can also target a specific frame of a frameset so that data and links open in the target frame. If you have not specified a target frame anywhere in this hierarchy, the link opens in the frame that contains the link. If you specify a target frame that does not exist, the link opens in a new, top-level window. The following figure illustrates how to use this property.



```
"viewFrame"
```

Setting a value to the target frame property

Setting this property in a Web application makes all the links of a page in a frameset to be executed/opened on another frame, by the use of a baseurl. The following example shows the code appended on the <head> tag when this property is set on a design.

```
<base target="framename">
```

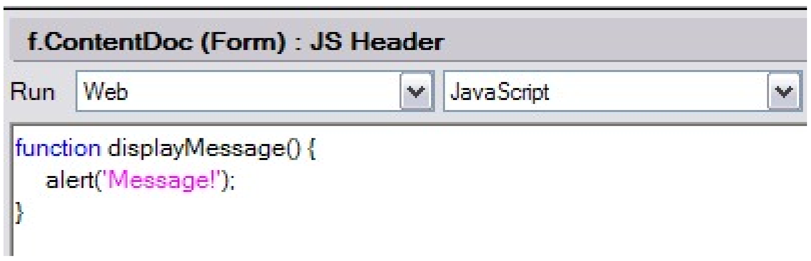
The target frame is a hybrid property, composed of an @Formula string. This string can have different values if the user is using Notes client or the web. If you have a hybrid application that has a form or page that is used on both Notes client and the Web, you can set the target frame to to a different frame of your application, using the @ClientType formula.

Important
Remember to specify a name in the Frame Properties box for any frame that is going to be a target frame.

For further information about how to use frames and frameset, refer to Frameset design elements.

JS Header

The JS Header is property used to hold the JavaScript of a design. The following figure illustrates how to use this property.


f.ContentDoc (Form) : JS Header

Run Web JavaScript

```
function displayMessage() {
    alert('Message!');
}
```

The JS header of a design

The JS Header is a hybrid property. If you have a hybrid application, you can have some JavaScript code that can work on both Notes client and the Web.

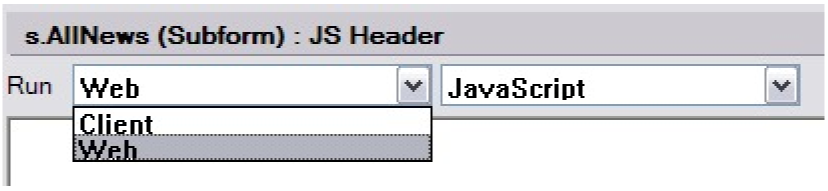
Note
Not all JavaScript functionalities are available on the Notes client.

The JavaScript header code of a design is kept inside the <head> tag of a Web page. The following example illustrates where in a Web page the JavaScript code of the JS Header is hosted.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>

<script language="JavaScript" type="text/javascript">
<!--
//HERE IS THE JS Header CODE - some JavaScript code goes just bellow
function hello() {
        alert('Hello world!');
}
// --!>
</script>

</head>
<body text="#000000" bgcolor="#FFFFFF">

</body>
</html>
```

The JS Header section of a design has a built in JavaScript text editor that helps the programmer code. The JS header is not available when you change the content type of a page from Notes to HTML. Make sure your JavaScript code is being written to run the right environment, in this case, the Web. You can set this by selecting the run property of your code or event.

The following figure illustrates how you set where a JavaScript code runs over. It is a best practice to keep the JavaScript code on subforms, JavaScript JavaScript library design elements, or even in File resources design elements accessed through Domino URLs.


s.AllNews (Subform) : JS Header

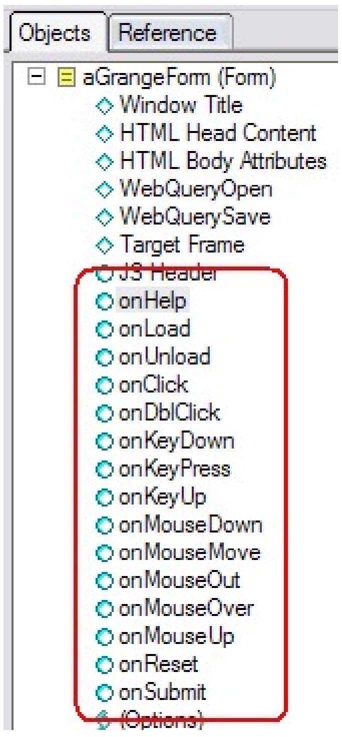
Run: Web | JavaScript

Setting were a JS code should run

Important
Since this property is also available on subforms, make sure that you do not have duplicate parts of JavaScript code on both your forms and subforms.

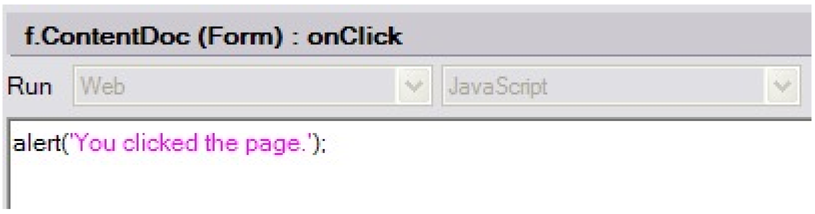
JavaScript events

An event is something that takes place over a system when an action takes place, like a mouse click or a keyboard key press. The programmer can code most of the JavaScript events of a Web page directly on the Domino Designer interface. The events are listed with round blue icons just below the JS Header property of a design. The following figure illustrates all the JavaScript events available from Domino Designer. To code an event, just select it from the list and enter the JavaScript code for that.



All the JavaScript events available at Domino Designer

The following figure illustrates where to code an event from Domino Designer. In this example, the event is onClick. Therefore, every time a user clicks the page, an alert message is displayed. Several types of eventscan be run on several elements, from forms and pages, to subforms, images and buttons.


f.ContentDoc (Form) : onClick

Run Web JavaScript

```
alert('You clicked the page.');
```

Setting the code for a JavaScript event

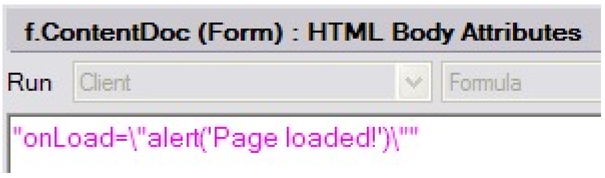
Note
Some of these events are not available on subforms.

The following JavaScript is available from the Designer interface:

- **onLoad**: Occurs immediately after a page or an image is loaded.
- **onUnload**: Occurs when a user exits a page.
- **onClick**: Occurs when an object gets clicked.
- **onDblClick**: Occurs when an object gets double-clicked.
- **onKeyPress**: Occurs when a keyboard key is pressed or held down.
- **onKeyUp**: Occurs when a keyboard key is released.
- **onMouseDown**: Occurs when a mouse button is clicked.
- **onMouseMove**: Occurs when the mouse pointer is moved.
- **onMouseOut**: Occurs when the mouse pointer moves away from a specified object.
- **onMouseOver**: Occurs when the mouse pointer moves over a specified object.
- **onMouseUp**: Occurs when a mouse button is released.
- **onReset**: Occurs when the reset button in a form is clicked.
- **onSubmit**: Occurs when the submit button in a form is clicked or when the Web forms are submitted.

Tip
You can use @Formula language to write any JavaScript event handler for your page. To achieve this, write your event code into the HTML body attributes attribute of your design.

The following figure illustrates how to use an event handler on the HTML body attributes property. We recommend this approach for small pieces of JavaScript code that needs to take advantage of the @Formula language. @Formulas string are server-side scripts, and require more server performance than client-side JavaScript plain text hosted on the JS Header, events of JavaScript libraries.


f.ContentDoc (Form) : HTML Body Attributes

Run Client Formula

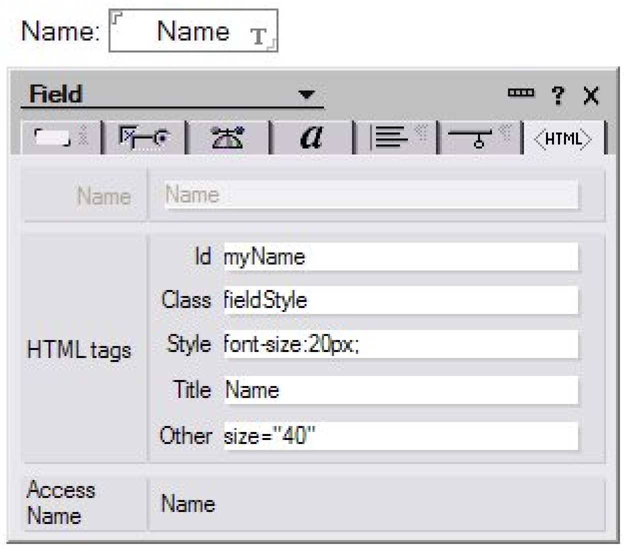
```
"onLoad=\"alert('Page loaded!')\""
```

Using HTML body attributes to set events

For further information about how to use JavaScript and its events, refer to JavaScript.

HTML properties of code elements

You can set diverse HTML properties and attributes to several code elements, like images, buttons, or fields to adapt it to your Web application needs. These attributes goes from HTML name to CSS class. To change these properties on a code element, go on its properties box, and in the HTML tab set your own properties. The following figure shows how to set diverse properties for a field on Designer.



Name: Name

Field

Name		Name
HTML tags	Id	myName
	Class	fieldStyle
	Style	font-size:20px;
	Title	Name
	Other	size="40"
Access Name		Name

Setting the HTML properties of the Name field

If the following properties are set for a field, Domino creates the following HTML for that:

```
<input name="Name" value="" id="myField" class="inputTextStyle" style="font-size:14px;"
title="Name" size="20">
```

Any attribute that is not listed on the property using a field can be entered as free text on the *Other* field (just like *size="20"* property entered in the previous example. This property is specially useful when using Web page elements using Domino resources, and not pass-through HTML.

For further information about HTML attributes, refer to HTML primer.

LotusScript and the Web

LotusScript *does not* run on the Web interface. To trap events on a Web environment, you need to use JavaScript events. If an application has some LotusScript events that need to run on an application on both client and the Web, the programmer should find a way to make the best adaptation of that code to JavaScript language. JavaScript offers several user interface features available on both Notes and the Web, but a set of the LotusScript functions does not have an equivalent on JavaScript, since some of them are related to the Notes user interface. The only way to run LotusScript on the web is using WebQueryOpen and WebQuerySave agents in back-end. For further information about WebQueryOpen and WebQuerySave agents, refer to using WebQueryOpen and WebQuerySave agents.

Styling text for the Web

This page last changed on Apr 03, 2008 by jservais.

Styling text for the Web

Lotus Domino automatically converts its text styles to HTML on the the design body when there is a corresponding HTML equivalent. Bullets, numbers, alignment (except Full Justification and No Wrap), spacing, and named styles are examples of HTML equivalents. Certain types of Notes formatting, such as indents, interline spacing, and tabs are not translated on the Web page code when viewed from a Web browser because there is no HTML corresponding format. Be aware that different browsers may display tags differently, and that not all browsers support the HTML tags that Domino generates.

Fonts

If the fonts used are not the system defaults (in Windows, Default Sans Serif and Default Serif), Domino converts font instructions to the HTML <FONT> tag and FACE= attribute to approximate the original font choice. Text may look different to a Web user than it does to a Notes user because the browser determines which fonts to use.

Tip
Domino does not use CSS or <span> tags to translate any of its content. Programmers should use pass-thru HTML if they want proper tags generated on their code.

Size

Domino maps the text size you select in Domino Designer to an HTML text size. The following table lists the text size in Lotus Notes and the corresponding HTML size. Note that Domino does not map font sizes to HTML heading tags (H1, H2, and so on).

Notes text size less than or equal to	Maps to HTML size
7	1
9	2
11	3 (default size)

13	4
17	5
23	6
greater than 23	7

Preserving spaces

To align a column of numbers or to preserve or insert spaces, use the default monospaced font. On a Windows system, the default monospaced font is Courier. Domino converts the default monospaced font to a monospaced font on the Web and preserves any spaces you enter.

Text colors

Web users see the same approximate text colors as Notes users, but the colors may not match exactly.

Working with the DOCTYPE

This page last changed on Apr 03, 2008 by jservais.

In this section, we discuss the various uses of the DOCTYPE declaration for both Standards-based Web Development and its utilization in Domino Web development best practices.

Setting DominoCompleteDoctype

You can change the DOCTYPE of the HTML pages Domino generates by changing an entry in the Domino server's notes.ini file.

Using DominoCompleteDoctype

This setting does *not* cause Domino to change the HTML it creates. It only changes the DOCTYPE line in the head section of the HTML page.

Set the DominoCompleteDoctype environment variable in notes.ini or preferably in a configuration document in the Domino Directory. It has three different values:

- 0 = <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
- 1 = <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
- 2 = <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

A value of 0 or no entry, is the default and leaves the browsers in quirks mode. A value of 1 or 2 puts the browser into standard compliance mode. If you are including Domino generated controls, then use 1, because a value of 2 causes validation errors.

Compliance mode and CSS

When the browser is in standard compliance mode, the CSS selectors become case-sensitive.

Example HTML:
<div id="frame">

The following CSS selector does not work:

```
#Frame {...}
```

The following CSS selector works:

```
#frame {...}
```

For information about how to change the content type of a design element, refer to Changing the content type of a design element

Agent design elements

This page last changed on Apr 03, 2008 by jservais.

Notes agents

Agents are the workhorse of Lotus Notes. They can be run in the foreground or backround, Notes client, Web client, or server.

When working with agents, consider the following.

- For long running agents, consider running on the server on a scheduled basis. If it is an absolute requirement to run on a Notes Client, run them in the background so the user is not locked out.
- For the Web, use agents sparingly, for the following reasons:

-
    - Agents on the Web are run using the HTTP task. Too many agents or long running agents cause a performance hit and may delay response time to Web browsers.
    - A separate thread is required for each agent, which uses up server resources.
    - An agent context must be created and destroyed each time an agent is run, further using up server resources.

WebQueryOpen

A WebQueryOpen event runs an agent via the @CommandToolsMacro command. The timing is such that the agent is called and run prior to Domino rendering the document to HTML and sending it to the Web browser.

WebQueryOpen agents only run when the user opens a form or document and are not run when the user saves a document. Be aware that any changes the WebQueryOpen agent makes to such fields are not saved when the user submits the document. You can either recalculate them in the WebQuerySave agent or set the form property "Generate HTML for all fields" to ensure you get the results you expect.

Domino ignores any output produced by an agent run by the WebQueryOpen event.

WebQuerySave

A WebQuerySave event runs an agent via the @CommandToolsMacro command. The timing is such that the agent is called after field input validation formulas are run and before the document is actually saved to disk. The document is automatically saved after the agent runs whether you explicitly save the document or not. We recommend that you *do not* explicitly save the document in the agent, because this can cause replication or save conflicts.

If you want the form to not be saved automatically, use the SaveOptions field. Create the field on the form first, and set it to 0 (false). The agent can change the value of the field if you want. Be aware that if you rely on the agent to entirely create the field, it does not work as expected.

A WebQuerySave agent can produce output to be sent back to the user via the LotusScript print statement. This statement can send a string that consists of an entire HTML page if you wish. For large strings with consistent and static HTML, such as a header, navigation or footer, you can store the such HTML in fields in your form, and then reference them while building your output string.

If the form has a $$Return field, it is ignored by Domino and the only output is from the query save agent.

Applet design elements

This page last changed on Apr 04, 2008 by dalandon.

Introduction

As stated in the primer section, an applet is a small non-standalone Java program. When an applet is loaded from a notes design element, either in the Notes client or Web client, the applet code is downloaded on the user's machine and subequently run.

An applet can be contained in several Notes design elements:

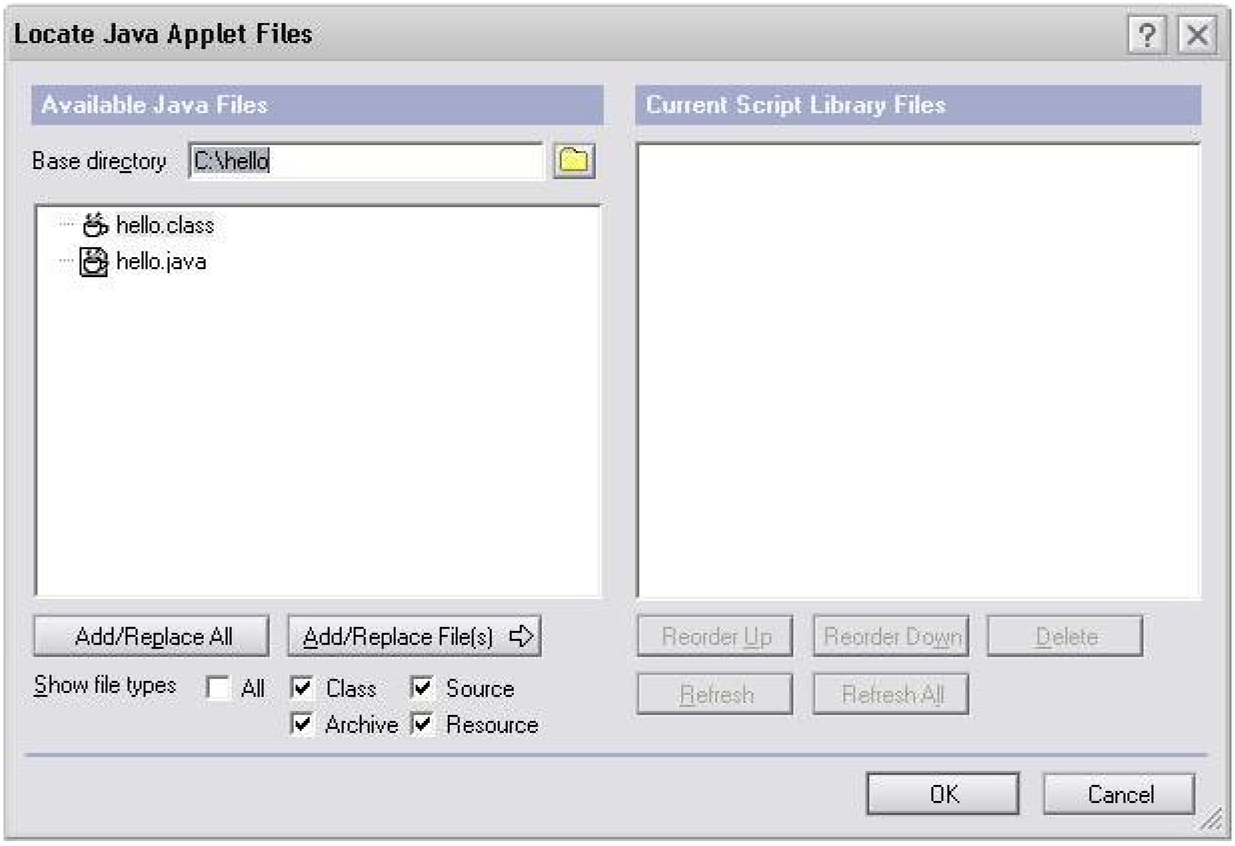
- Form: The applet is included in each document created with that form.
- Document: The applet is available only in the document.
- Page: The applet is available only in the page.

To include an applet into an application, you must first do the following actions:

- Enable Java applets on your workstation (Notes client only)
- Import the applet or link to an applet on the Web
- Set the applet parameters and attributes in the properties box of the applet.

Creating an applet

You cannot directly create applets in Domino Designer, as its Java IDE is limited in regards to this capability. You must create the applet in an external Java IDE and then import the class file or files for the applet into the Domino Database. To do the import, go to the Shared Resources section of the database in Domino Designer, and click **Applets** to open the design pane. Then click the **New Applet Resource** action button. The Locate Java Applet Files window opens (see the following figure).



After you select the directory from your file system, select the proper files and click either of the **Add/Replace** buttons to include them in the applet resource. You are prompted to give the resource a name, and then you are done.

You can use other Java resources/filetypes for your applet, for example:

- Class files (*.CLASS)
- Archive files (*.JAR, *.ZIP, *.CAB)
- Resource (*.JPG, *.JPEG, *.GIF, *.AU)
- Source (*.JAVA)

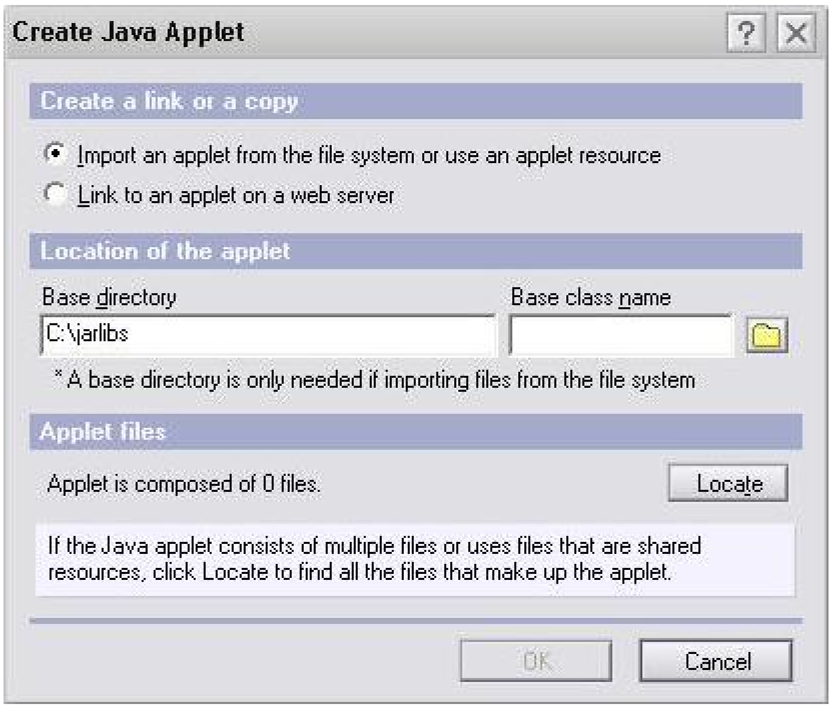
When selecting files, select the archive files, or, class file or files and resource files that are needed to execute the main applet class. Source files are not needed unless you plan to send the applet to another user who wants to export them and change the applet.

Inserting an applet

You can insert an applet into a notes design element and specify one of three sources.

- Local file system
- Domino Designer applet shared resource
- Web URL

When inserting an applet into a Notes element, in Domino Designer click **Create** followed by **Java Applet**. The Create Java Applet window opens (see the following figure).



(●) Import an applet from the file system or use an applet resource
( ) Link to an applet on a web server

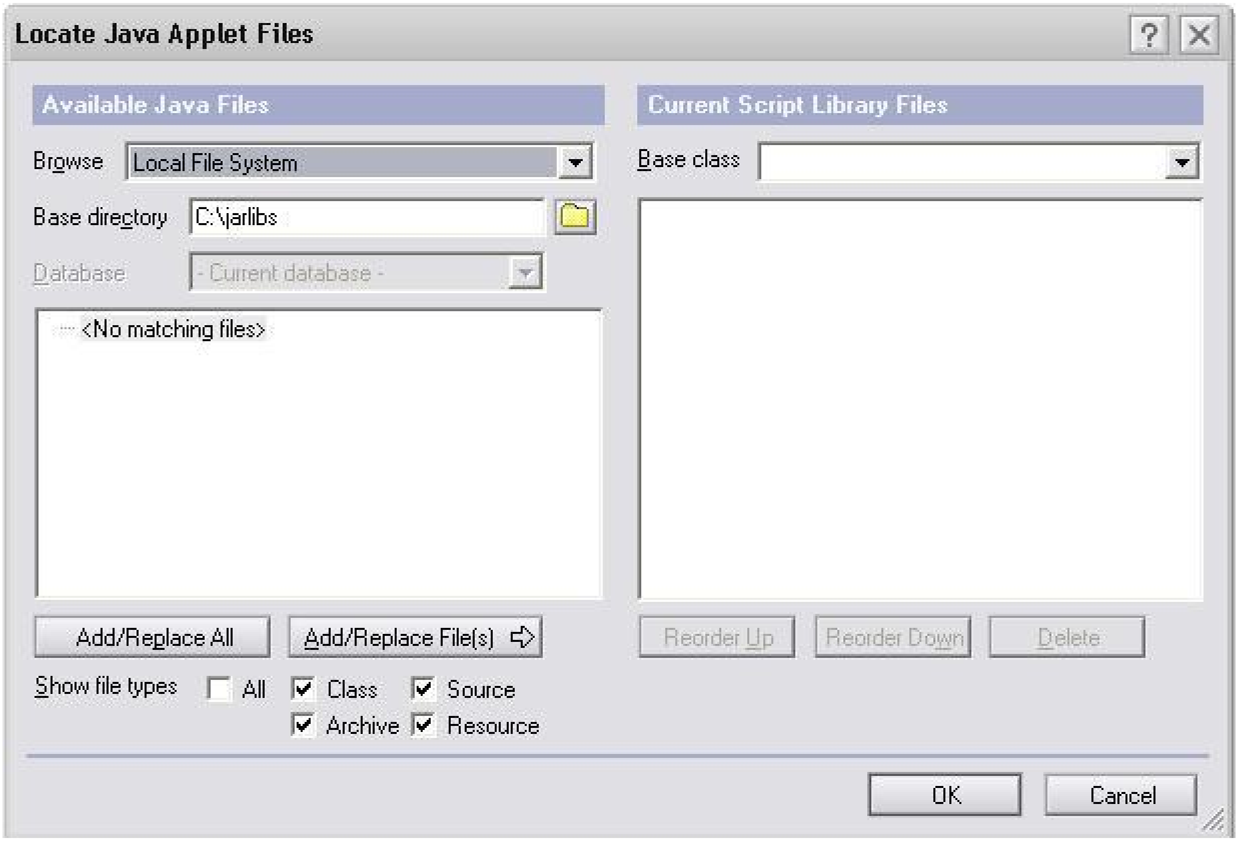
Base directory: C:\jarlibs
Base class name:

* A base directory is only needed if importing files from the file system

Applet is composed of 0 files.

If the Java applet consists of multiple files or uses files that are shared resources, click Locate to find all the files that make up the applet.

You can specify that the applet comes directly from files on the file system or a shared resources (as created previously). The first window is used to directly specify files from your file system. If you want to use a shared resource, click on **Locate**. If you click Locate, the following window opens to assist you in selecting your files, either from your file system or from a shared resource.



Browse: Local File System
Base directory: C:\jarlibs
Database: - Current database -
<No matching files>
Show file types: [ ] All [✓] Class [✓] Source [✓] Archive [✓] Resource

Inserting from the file system or shared resource

If you are inserting an applet that is packaged as a JAR file, ensure that you specify the correct class name for the main class file of the applet. Browsing for the JAR file inserts the filename with a "class" extension in the class name field. If the file name and the main applet class name are the same, this is fine, but if it is not, then you must edit the class name field to be the correct class to be loaded when the applet runs.

Inserting from the Web

In the Create Java Applet window, select **Link to an applet on a Web server**. In the base URL field, enter the URL where the applet files are stored. Do not specify the document that references the applet. Specify the location of the main applet class file as shown in the following example:

```
http://<web server url>/<applet directory>/<applet class name only, with no filetype>
```

In the Base class name field, enter the name of the main class. Note that Java is case-sensitive to file names.

Further information

Applets can provide rich visual functionality to any application. Take the time to research and sample their use, and decide whether its right for your Domino application.

Design element multi-aliasing

This page last changed on Apr 04, 2008 by dalandon.

In this section, we discuss the application of multiple aliases to the Domino design elements to both alleviate the need for duplicate like-design Domino design elements and the leveraging of more advanced common design element architectures.

What is Design Element Multi-Aliasing?

Design Element Multi-Aliasing (DEMA) is the application of additional aliases to specific Domino design elements. The Domino Designer Client allows us to enter a *vertical pipe* (|) character to separate the Common Name or Displayed Name of a design element and the design element's programmatic name. The following example shows the common naming schema for Domino Form Design Elements.

```
Form Element Display Name|programmatic_element_name
```

There is an effective and functional aspect to this naming schema. The Notes documents that are created with the *Form Element Display Name* Form Design Element store the "programmatic_element_name" as the value of the Notes Document's Form Notes Item.

We have the ability to create multiple aliases between the Display/Common Name of the design element and its programmatic name as shown in the following example:

```
Form Element Display Name|other_element_name|programmatic_element_name
```

This is an elementary example of a method that, when applied, can allow you to use individual Domino design elements far beyond their original intentions. In the next section, we discuss several *real-world* examples of DEMA.

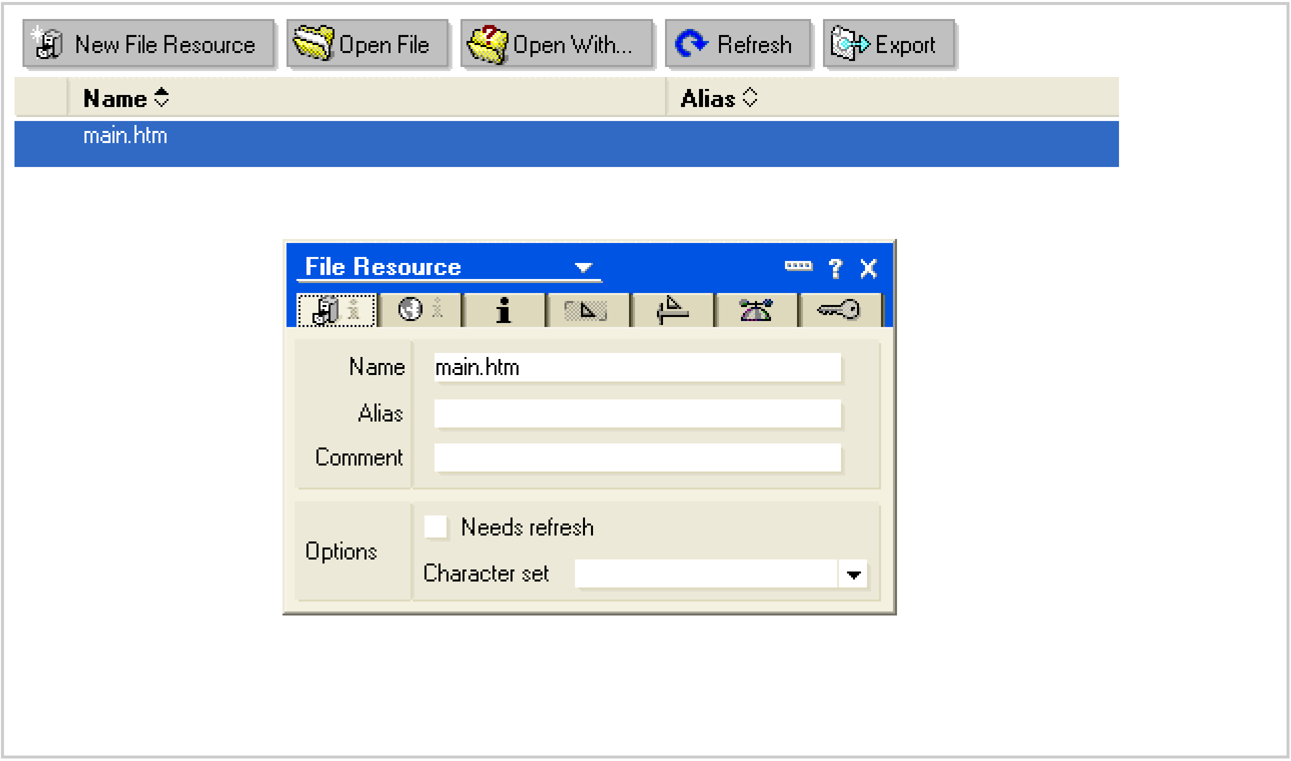
Examples of DEMA in Domino Web application development

Share your examples and best practices.

File resources design elements

This page last changed on Apr 01, 2008 by jservais.

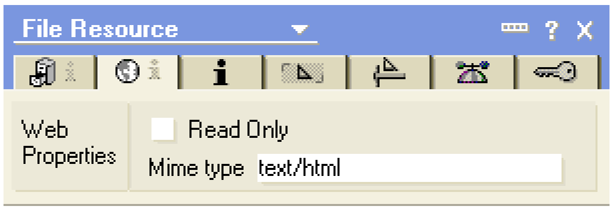
In certain situation, you might need to reference external files that are created with a non-Lotus Notes software. For example, a company might have an HTML file that somebody else maintains, but it needs to be a part of the Web application. You can use the File Resources design element to reference these type of files. After you add the external file, you can refresh the file by clicking the **Refresh** button when the external file changes.



main.htm

Name: main.htm
Alias:
Comment:
Options: Needs refresh
Character set

The second tab has options that are only applicable to Web applications.



Web Properties: Read Only
Mime type: text/html

- Read Only: Checking this option causes the selected design element(s) to be read only on the Web.
- MIME type: This refers to the MIME type of the file resource for Web clients. The Content-Type header of the HTTP response is set to this value. Domino Designer fills in this field if it recognizes the extension of the file resource (for example, a GIF image file or an EXE application file).

Folder design elements

This page last changed on Apr 01, 2008 by jservais.

In this section, we discuss the folder design element and its proper and extended usage in our development practices. We assume that you have a basic understanding of the folder design element and focus more on how we can use this design element in our Web application development.

What is a folder

The folder design element is used to create both visual and functional structures for Notes Document Collections. Unlike the View Design Element, the folder design element is a container element that stores Notes documents.

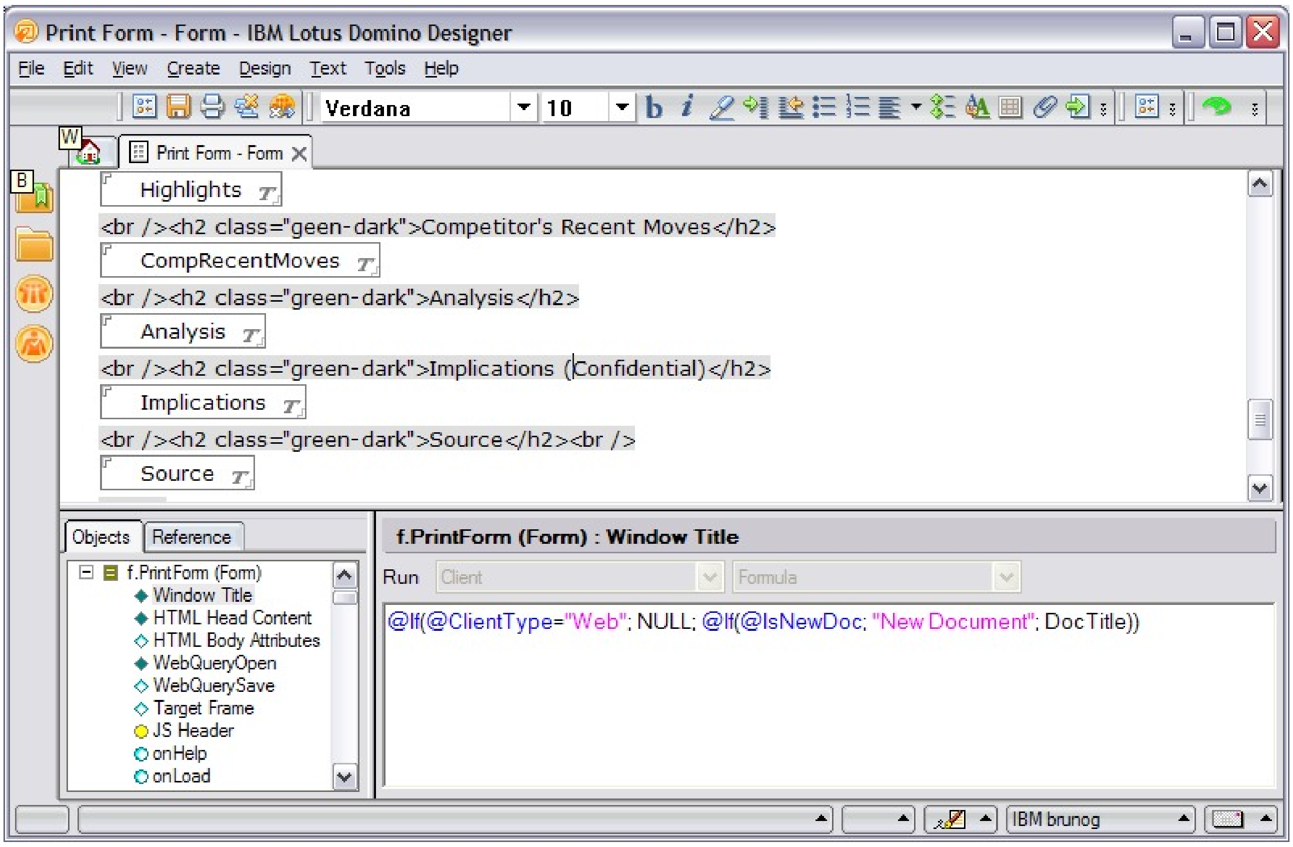
Since the folder design element is a container element, Notes documents must be *put in* or *removed from* the element. With the advent of Folder References, Notes documents can now track folder location information in the form of Notes Items (*$FolderRef*, *$FodlerRefFlags*, *$FolderRefID*) in the Notes document context.

Form design elements

This page last changed on Apr 04, 2008 by dalandon.

Introduction

Forms are design elements used to hold the information submitted into a database. A form gets rendered on the Web just as a common Web page and is used to hold content similar to the page design element. A form may also be used to control how a view gets displayed or searched on the web, using view templates or search templates. Forms can hold formatted text, graphics and embedded controls, such as outlines and applets. Different from Page design elements, forms can also hold fields and subforms. To learn how to control how your design is going to look on the Web, refer to Styling text for the Web.



Print Form - Form - IBM Lotus Domino Designer
File Edit View Create Design Text Tools Help
Verdana
10
Print Form - Form
Objects
Reference
Run
Client
Formula
IBM brunog

```
Highlights
<br /><h2 class="geen-dark">Competitor's Recent Moves</h2>
CompRecentMoves
<br /><h2 class="green-dark">Analysis</h2>
Analysis
<br /><h2 class="green-dark">Implications (Confidential)</h2>
Implications
<br /><h2 class="green-dark">Source</h2><br />
Source
```

```
f.PrintForm (Form)
    Window Title
    HTML Head Content
    HTML Body Attributes
    WebQueryOpen
    WebQuerySave
    Target Frame
    JS Header
    onHelp
    onLoad
```

f.PrintForm (Form) : Window Title

```
@If(@ClientType="Web"; NULL; @If(@IsNewDoc; "New Document"; DocTitle))
```

A form design element open in Domino Designer

The following table describes each of the Domino design elements that can be used on forms on the Web.

Elements to use on a form	Description
Actions	Actions automate tasks for the user. Add actions to the menu in the Notes client, or add actions with buttons or hotspots on a page or form. For more information, refer to actions.
Applets	Use Java applets to include small programs, such as an animated logo or a self-contained application, in a page or form. For more information about including Java applets on a page or form, refer to Applet design elements
Attachments	Attach files to a page or form so users can detach or launch files locally. For more information, refer to file attachments.
Automation	You can create form actions, buttons, or hotspots on a form, subform, or page to automate simple or complex tasks.
Computed text	Use computed text to generate dynamic text based on formula results.
Embedded elements	You can embed the following elements in a page or form: a view or folder pane, navigator, outline or instant messaging contact list. Forms can also embed the file upload, and editor. Use these elements alone or combine them to control how users navigate through your application.
Fields	Fields are the design elements that collect data. You can create fields only on forms or subforms. Each field on a form stores a single type of information. A field's field type defines the kind of information a field accepts. You can place fields anywhere on a form. For more information about fields, refer to fields.
Graphics	Place a graphic anywhere on a page or form. Use graphics to add color to a page or form or to create imagemaps.
Horizontal rules	Add horizontal rules to separate different parts of a page or form, or to make a page or form more interesting visually.
HTML	If you have existing HTML or you prefer using HTML to using the formatting tools Domino Designer offers, you can import, paste or write your own html on a page or form. You can also convert pages and forms to HTML.
Imagemaps	An image map is a graphic you enhance with programmable hotspots. Hotspots, in the form of pop-up text, actions, links, and formulas, perform an action when clicked by a user. Use imagemaps

	as navigational structures in an application.
JavaScript libraries	You can find and insert JavaScript libraries into a page, form or subform. For more information on inserting JavaScript libraries, refer to JavaScript library design elements.
Layers	Layers let you position overlapping blocks of content on a page, form, or subform. Layers give you greater design flexibility because you can control the placement, size, and content of information. For more information on layers, refer to Layers.
Links	Add links to take users to other pages, views, databases, or URLs when they click on text or a graphic.
Sections	A section is a collapsible and expandable area that can include objects, text, and graphics.
Shared resources	The following shared resources can be added to a form or subform: Image resource design elements, JavaScript libraries, Shared field design elements, subforms, style sheets and HTML Files.
Style Sheet (CSS) shared resources	You can find and insert a cascading style sheet (CSS) as a shared resource on a page, form, or subform. For more information on style sheets, refer to style sheets.
Subforms	A subform is a collection of form elements stored as a single object. A subform can be a permanent part of a form or can appear conditionally, depending on the result of a formula. Subforms save redesign time. When you change a field on a subform, every form that uses the subform changes. Common uses of subforms include adding a company logo to business documents or adding mailing label information to mail and memo forms. For more information about subforms, refer to subforms.
Tables	Use tables to summarize information, align text and graphics in rows and columns, or position elements on a page or form.
Text	Use text anywhere on a page or form and apply text attributes, such as color, size, and font styles to the text.

Tip
Layout regions are not supported on the Web. Consider using layers or div tags with CSS styles instead.

Form properties

Forms are displayed on the Web with all its fields enclosed by an HTML <form> tag. Forms have a set of properties that can impact the user experience on the Web. Refer to other web specific database properties and their effects for further information.

Differences between pages and forms

Both forms and pages are displayed on the web as web-pages (unless the user decides to change the page content type), but the main difference between them are that pages do not store data entered by the user. Fields, subforms, layout regions, and some embedded controls can only be used on forms. There are also some other differences and best practices when using pages and forms. Refer to Using forms versus pages for further information.

Form naming best practices

Forms can be named using the Design element multi-aliasing technique. Using this we can have a form called "Top rated product vendor|vendor.htm|vendor". This naming convention also brings the benefit of have an alias that works like a traditional Web server "file name" for Domino URLs, helping keeping aliases even if the main page content changes. This also helps search crawlers increase the page rank by using the search engine optimization best practice of having a small description of the page on its URL. This practice is explained more detailed on the search engine optimization section.

User input validation

There are several techniques to validate user input validation on the Web. These techniques can be applied on client side using JavaScript. We strongly recommend that you use server side validation. For further information, refer to the following topics:

- Input validation - Server side
- Input validation - Client side

Other topics

- HTMLOptions and HTMLTagAttribute fields
- Special reserved fields
- Understanding the form HTML source code
- Using forms versus pages

HTMLOptions and HTMLTagAttribute fields

This page last changed on Apr 01, 2008 by jservais.

Handling HTML code generation

With Release 8, Lotus Domino offers the possibility to change the HTML generated for a form and for its fields. These options are not related to the page HTML itself, but on how how the Domino server creates the HTML code for the field into HTML, and how that field's input is processed.

HTML options are indicated by a list of name=value pairs. The name consists of alphanumeric characters (letters, numbers, underscore, dash), and indicates the option that is being set. The value is a number, and indicates the setting of that option. The HTML options for a form are stored in a Computed for Display text field, that must be called $$HTMLOptions. If more than one HTML option/value has to be set for a form, the field must be set to allow multiple values. Field-level HTML options override form-level HTML options.

The following table list the options that are available.

Option name	Option action
DisablePassThruHTML	Disables passthru HTML, treating the HTML as plain text.
ForceSectionExpand	Forces all sections to be expanded, regardless of their expansion in the Notes rich text fields.
ForceOutlineExpand	Forces all outlines to be expanded, regardless of their expansion in the Notes rich text.
RowAtATimeTableAlt	Forces alternate formatting of tables with tabbed sections. All of the tabs are displayed at the same time, one below the other, with the tab labels included as headers.
TextExactSpacing	Preserves Notes intraline white space (spaces between characters).

Controlling the HTML tag attributes generated for a form

The HTML tag attributes are the optional attributes that are included inside the <html> tag of a Web page. On Domino Release 8, designers have the ability to define the attributes for this tag. HTML tag attributes for a form are stored in a hidden Computed for Display text field, that must be called $$HTMLTagAttributes. Specify the value of this field inside quotation marks. For example, specifying the value "lang=en" produces the following code for that <html> tab in the resulting HTML:

```
<html lang=en>
```

Controlling the HTML front matter generated for a form

The HTML front matter consists of an optional <!DOCTYPE> tag that appears before the <html> tag of a document displayed on the Web. On Domino Release 8, designers have the ability to control these attributes. HTML front matter information for a form is stored in a hidden Computed for Display text field, that must be called $$HTMLFrontMatter.
Specify the value of this field inside quotation marks, and include the statement inside an unique value. For example, this code specifies if an HTML document should be viewed with strict or transitional encoding, depending on the value of a field on the form named Mode.

```
@If(Mode="Strict";
"<!DOCTYPE HTML PUBLIC" + @NewLine + "\"-//W3C//DTD HTML 4.01\"" + @NewLine +
" \"http://www.w3.org/TR/REC-html40.strict.dtd\">" +
@NewLine; "")
```

Special reserved fields

This page last changed on Apr 01, 2008 by jservais.

Lotus Domino Designer provides predefined fields with some built-in functionalities to facilitate users. For example, to design a form with e-mail options, you add predefined mail fields such as SendTo and CopyTo to a mail form. Designer automatically recognize these special fields and offers interaction with the server to send the message.
If a reserved name field is used in a way that is different from its original intended use or redefine the field, Designer displays an error message.

Reserved names for embedded elements

Reserved field name	Contains
$$ViewBody	An embedded view.
$$ViewList	An embedded folder pane.
$$NavigatorBody	An embedded navigator.
$GroupScheduleRefreshMode	A value for refreshing an embedded group scheduling control.
$GroupScheduleShowLegend	A value of 0 or 1. If the value is 0, the color legend does not display. If the value is 1, the color legend does display. The default is 1.

Reserved fields for use in billing applications

Reserved field name	Creates a billing record when a user
$ChargeRead	Opens a document that contains this field.
$ChargeWrite	Creates, copies, edits, or saves a document that contains this field.

Reserved fields for general use

Reserved field name	Use
Categories	Categorizes documents.
$VersionOpt	Controls version tracking for documents.
FolderOptions	Puts new documents in folders.
SecretEncryptionKeys	Encrypts documents with secret, rather than public, encryption keys.
HTML	Passes HTML directly to the server.
$$HTMLHead	Passes HTML information to be hosted within the

	<HEAD> tag for a document. The passed information might be meta data (using a <META ...> tag) or JavaScript code (using a <SCRIPT ...> tag) or CSS information (using a <STYLE ...> tag).
$$Return	After Web users submit a document, Domino responds with the default confirmation "Form processed." To override the default response, add a computed text field to the form, name it $$Return, and use HTML as the computed value to create a customized confirmation.

Visit the following links for information $$HTMLOptions, $$HTMLTagAttribute, $$HTMLFrontMatter and $$Return.

Reserved fields for mail

Reserved Field name	Values	Comments
BlindCopyTo	The name(s) of a person, group, or mail-in database.	
CopyTo	The name(s) of a person, group, or mail-in database.	
DeliveryPriority	L, N, H	Values correspond to: Low, normal, or high-priority.
DeliveryReport	N, B, C, T	Values correspond to: None, Only on failure, Confirm delivery, Trace entire path
Encrypt	1, 0	Use 1 to encrypt mailed documents.
MailFormat	B, E, M, T	Enables cc:Mail users to view Notes documents in a variety of predefined formats: B = both text and encapsulated. E = encapsulated in a Notes database, which is attached to the cc:Mail memo. M = mail. Body field of document is text and pasted into cc:Mail memo. T = text. Contents of the document are rendered as text and pasted into the body of the cc:Mail memo.
MailOptions	1, 0	Use 1 for automatic mailing.
ReturnReceipt	1, 0	Use 1 to send a receipt when document is opened by the recipient.
SaveOptions	1, 0	Use 1 to save mailed documents.

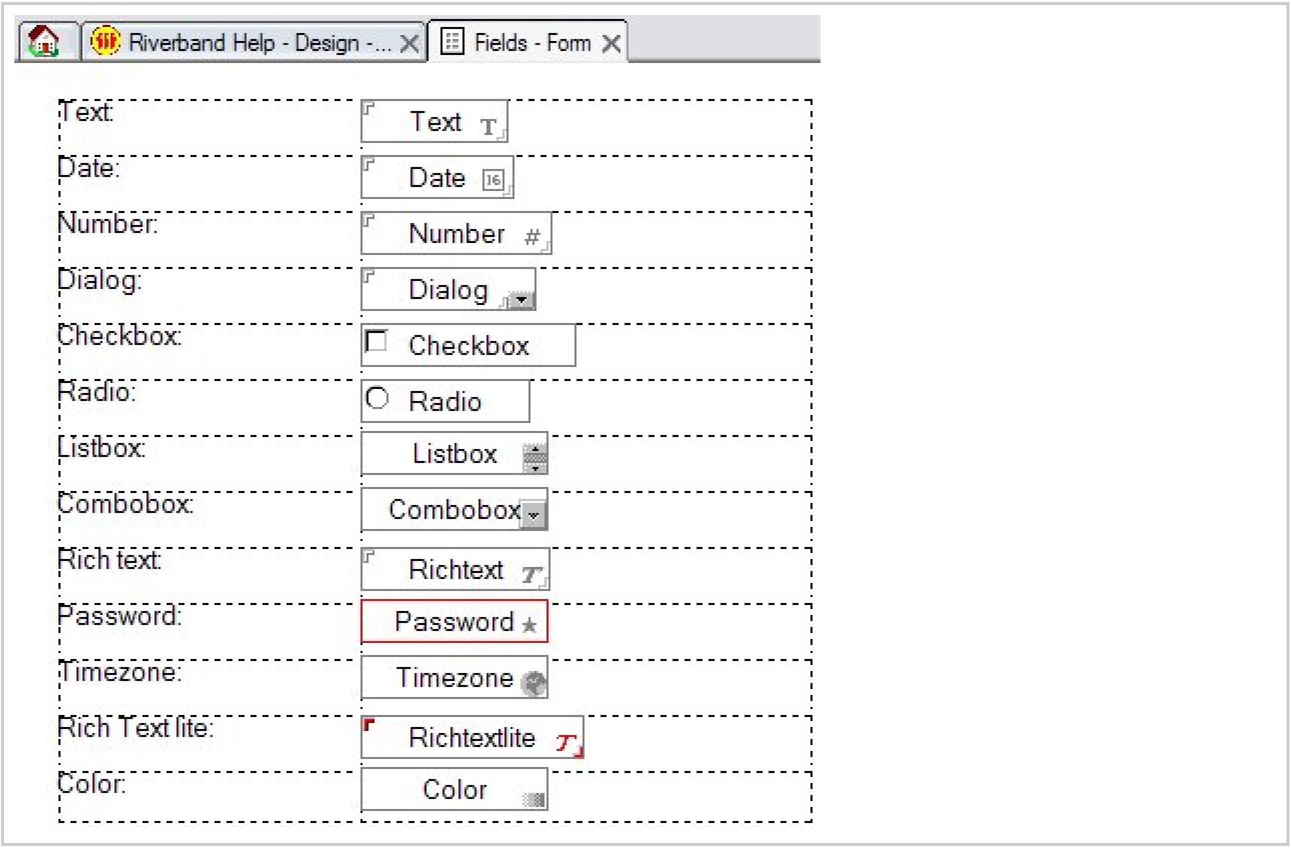
		Use 0 so that the document is not saved when mailed. prevent the document from being saved.
SendTo	The name(s) of a person, group, or mail-in database.	Required for all forms that mail documents.
Sign	1, 0	Use 1 to an add electronic signature to fields. (Only applicable if a form also contains sign-enabled fields.)

Understanding the form HTML source code

This page last changed on Apr 03, 2008 by jservais.

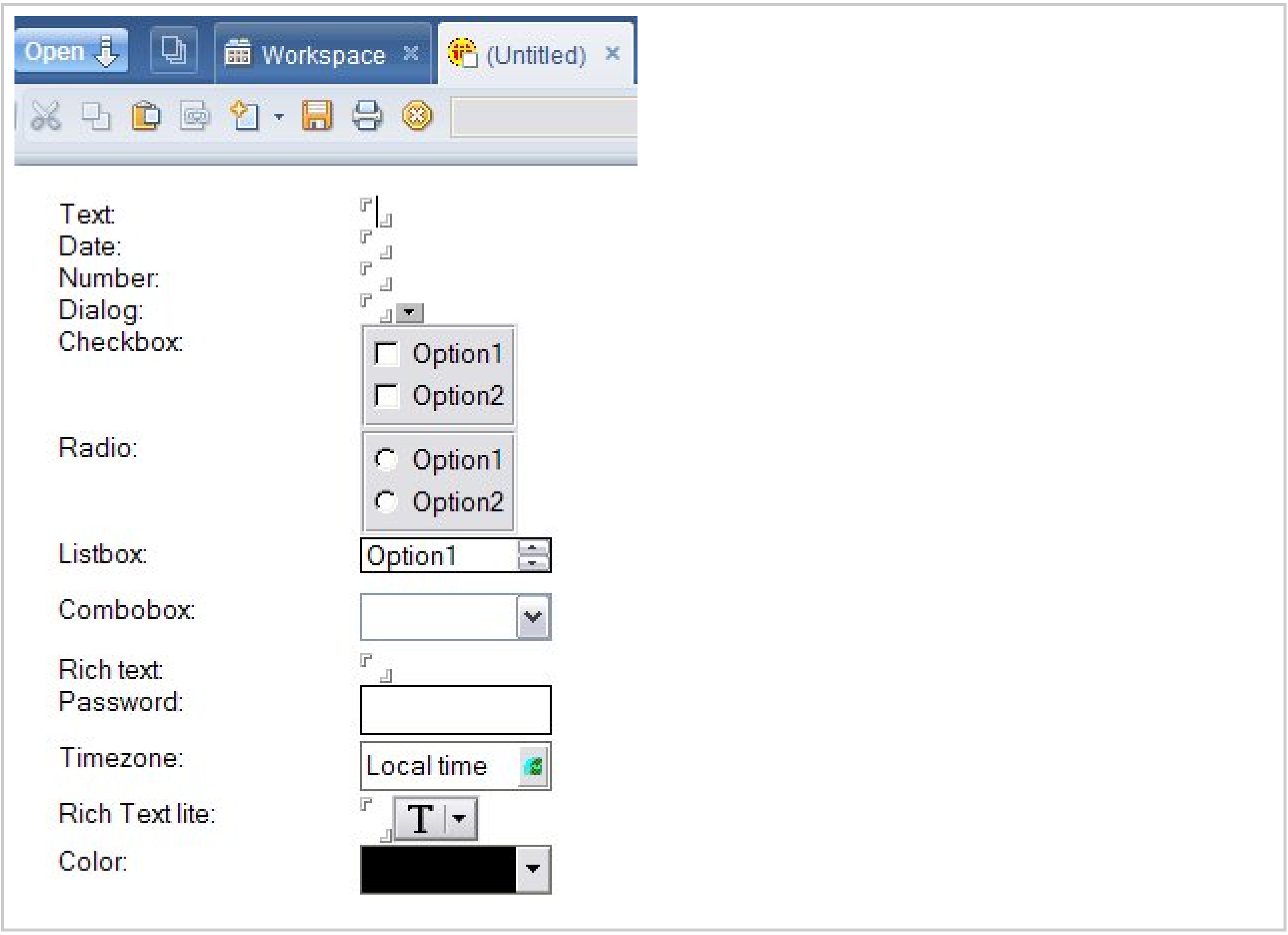
Understanding the form HTML source code

Lotus Domino forms are not identical when presented on Lotus Notes and the Web. Some minor changes appears on some field types in order to suit all types and formats to a corresponding HTML input. In this section, we go through the field types and explain its HTML translation. Refer to working with HTML in Domino for further information about HTML.



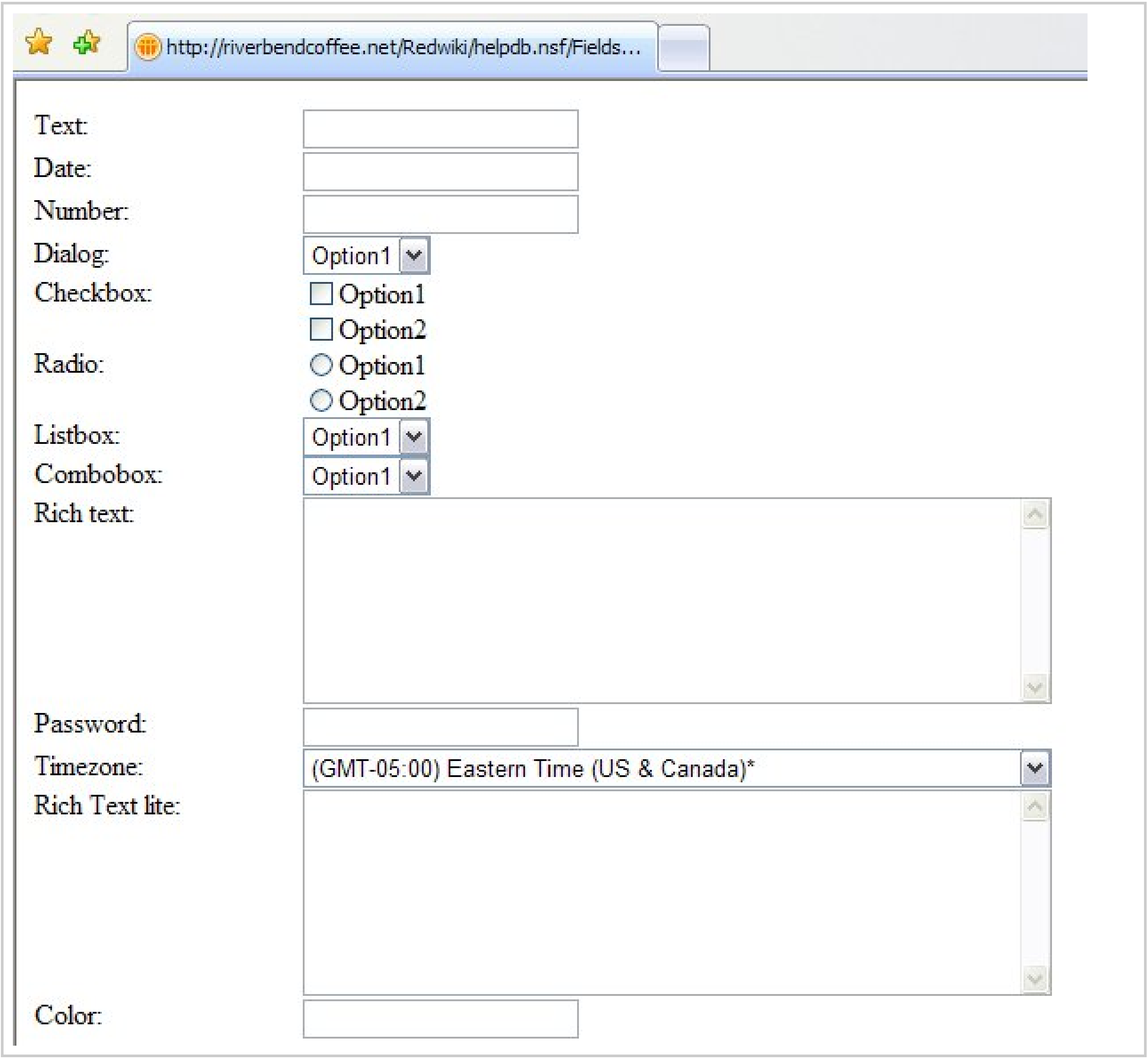
Text:	Text
Date:	Date
Number:	Number
Dialog:	Dialog
Checkbox:	Checkbox
Radio:	Radio
Listbox:	Listbox
Combobox:	Combobox
Rich text:	Richtext
Password:	Password
Timezone:	Timezone
Rich Text lite:	Richtextlite
Color:	Color

Form on Domino Designer



Text:	
Date:	
Number:	
Dialog:	
Checkbox:	☐ Option1 ☐ Option2
Radio:	○ Option1 ○ Option2
Listbox:	Option1
Combobox:	
Rich text:	
Password:	
Timezone:	Local time
Rich Text lite:	T
Color:	■

Form on Notes client



http://riverbendcoffee.net/Redwiki/helpdb.nsf/Fields...

Text:	
Date:	
Number:	
Dialog:	Option1
Checkbox:	☐ Option1 ☐ Option2
Radio:	○ Option1 ○ Option2
Listbox:	Option1
Combobox:	Option1
Rich text:	
Password:	
Timezone:	(GMT-05:00) Eastern Time (US & Canada)*
Rich Text lite:	
Color:	

Form on the Web browser

Important
The input tags that are generated by Domino are not according to W3C standards. When an HTML tag that does not have a closing tag (e.g.: <select>...</select>), the programmer must to add a closing tag to keep the mark-up "well formed" (e.g.: <input />). Since Domino generates the HTML code translation itself, there is no way to handle this issue. This issue is fixed on Domino release 8.5. Refer to W3C for further information about HTML language standards.

Text

The following example exemplifies the HTML code generated for Notes **text** fields translated to HTML on the Web:

```
<tr><td>Text: </td><td>
<input name="Text" value=""></td></tr>
```

The tag used for this field type is *<input>*, with no *type* attribute set. If the field already has a value, its contents are held on the *value* property.

Tip
Do not use New lines as a multi-value separator for this field type on the Web. That is because there is no way for the user to enter new lines on input text fields on the Web.

Date/Time

The following example exemplifies the HTML code generated for Notes date/time fields translated to HTML on the Web:

```
<tr><td>Date: </td><td>
<input name="Date" value=""></td></tr>
```

The tag used for this field type is *<input>*, with no *type* attribute set. If the field already has a value, its contents are held on the *value* property using the server date formatting. If a non-date/time value is entered, Domino indicates an error.

Tip
Do not use New lines as a multi-value separator for this field type on the Web, because there is no way for the user to enter new lines on input text fields on the Web.

Number

The following example exemplifies the HTML code generated for Notes number fields translated to HTML on the Web:

```
<tr><td>Number: </td><td>
<input name="Number" value=""></td></tr>
```

The tag used for this field type is *<input>*, with no *type* attribute set. If the field already has a value, its contents are held on the *value* property. If a non-numeric value is entered, Domino raises an error.

Tip
Do not use New lines as a multi-value separator for this field type on the Web because there is no way for the user to enter new lines on input text fields on the Web.

Dialog box

The following example exemplifies the HTML code generated for Notes dialog box fields translated to HTML on the Web:

```
<tr><td>Dialog: </td><td>
<input name="%%Surrogate_Dialog" type="hidden" value="1">
<select name="Dialog">
<option>Option1
<option>Option2</select>
</td></tr>
```

Just like a combo box, the tag used for this field type is *<select>*. The options are enclosed by the *<option>* tag. If the field already has a value, the selected option raises the *selected* property. The *value* property is only used in case aliases are used on the field options.

> **Note**
> The %%Surrogate_... input is generated automatically for this field type just to support the Domino server with internal computations.

Checkbox

The following example exemplifies the HTML code generated for Notes checkbox fields translated to HTML on the Web:

```
<tr><td>Checkbox: </td><td>
<input name="%%Surrogate_Checkbox" type="hidden" value="1">
<input type="checkbox" name="Checkbox" value="Option1">Option1<br>
<input type="checkbox" name="Checkbox" value="Option2">Option2</td></tr>
```

The tag used for this field type is *<input>*, with *type* property set to check box. The tag is repeated for every option. Check boxes can have multiple values. In case of aliases, the text enclosed by the *<input>* tag is the alias, and the *value* property holds the value. The columns option separator is the *<br>* tag. If the field already has a value, the selected option raises the *selected* property.

> **Note**
> The %%Surrogate_... input is generated automatically for this field type just to support the Domino server with internal computations.

Radio button

The following example exemplifies the HTML code generated for Notes radio button fields translated to HTML on the Web:

```
<tr><td>Radio: </td><td>
<input name="%%Surrogate_Radio" type="hidden" value="1">
<input type="radio" name="Radio" value="Option1">Option1<br>
<input type="radio" name="Radio" value="Option2">Option2</td></tr>
```

The tag used for this field type is *<input>*, with *type* property set to radio. The tag is repeated for every option. In case of aliases, the text enclosed by the *<input>* tag is the alias, and the *value* property holds the value. The columns option separator is the *<br>* tag. If the field already has a value, the selected option raises the *selected* property.

> **Note**

The %%Surrogate_... input is generated automatically for this field type just to support the Domino server with internal computations.

Listbox

The following example exemplifies the HTML code generated for Notes listbox fields translated to HTML on the Web:

```
<tr><td>Listbox: </td><td>
<input name="%%Surrogate_Listbox" type="hidden" value="1">
<select name="Listbox">
<option>Option1
<option>Option2</select>
</td></tr>
```

Just like a combo box, the tag used for this field type is *<select>*. The options are enclosed by the *<option>* tag. If the field already has a value, the selected option raises the *selected* property. The *value* property is only used in case aliases are used on the field options. List boxes can have multiple values. In this case, it is rendered with the property *multiple* set.

Note
The %%Surrogate_... input is generated automatically for this field type to support the Domino server with internal computations.

Combobox

The following example exemplifies the HTML code generated for Notes combobox fields translated to HTML on the Web:

```
<tr><td>Combobox: </td><td>
<input name="%%Surrogate_Combobox" type="hidden" value="1">
<select name="Combobox">
<option>Option1
<option>Option2</select>
</td></tr>
```

The tag used for this field type is *<select>*. The options are enclosed by the *<option>* tag. If the field already has a value, the selected option raises the *selected* property. The *value* property is only used in case aliases are used on the field options.

Note
The %%Surrogate_... input is generated automatically for this field type to support the Domino server with internal computations.

Richtext

The following example exemplifies the HTML code generated for Notes richtext fields translated to HTML on the Web:

```
<tr><td>Rich text: </td><td>
<textarea name="Richtext" rows="7" cols="50"></textarea>
</td></tr>
```

The tag used for this field type is *<textarea>*. Its text values are enclosed between the beginning and closing tag. If the field already has a value, its contents are held on the *value* property.

Important
All the rich text formats entered on a rich text field are lost if a document is saved on the Web (text is converted to plain/unformatted text). The file attachments are still held on the document, and are accessible by using the /$FILE/filename Domino URL. For further information about file attachments on the Web, refer to file attachments.

Password

The following example exemplifies the HTML code generated for Notes password fields translated to HTML on the Web:

```
<tr><td>Password: </td><td>
<input name="Password" value="" type="password"></td></tr>
```

The tag used for this field type is *<input>*, with type attribute set to password. If the field already has a value, its contents are held on the *value* property using the server date formatting. If a non-date/time value is entered, Domino raises an error.

Important
Despite the fact that Web browser masks the passwords using the operating system password field masks, the value property of this field is rendered in plain text on the source code if the document is already saved.

Timezone

The following example exemplifies the HTML code generated for Notes timezone fields translated to HTML on the Web:

```
<tr><td>Timezone: </td><td>
<select name="Timezone">
<option value="Z=12$DO=0$ZN=Dateline$ZX=129">(GMT-12:00) International Date Line West
<option value="Z=11$DO=0$ZN=Samoa$ZX=130">(GMT-11:00) Midway Island, Samoa
<option value="Z=10$DO=0$ZN=Hawaiian$ZX=131">(GMT-10:00) Hawaii
<option value="Z=9$DO=1$DL=3 2 1 11 1 1$ZN=Alaskan$ZX=132">(GMT-09:00) Alaska
<option value="Z=8$DO=1$DL=3 2 1 11 1 1$ZN=Pacific$ZX=133">(GMT-08:00) Pacific Time (US &amp;
Canada)
<option value="Z=8$DO=1$DL=4 1 1 10 -1 1$ZN=Pacific (Mexico)$ZX=134">(GMT-08:00) Tijuana, Baja
California
<option value="Z=7$DO=0$ZN=US Mountain$ZX=135">(GMT-07:00) Arizona
<option value="Z=7$DO=1$DL=4 1 1 10 -1 1$ZN=Mountain Mexico$ZX=136">(GMT-07:00) Chihuahua, La
Paz, Mazatlan
<option value="Z=7$DO=1$DL=3 2 1 11 1 1$ZN=Mountain$ZX=137">(GMT-07:00) Mountain Time (US &amp;
Canada)
<option value="Z=6$DO=0$ZN=Central America$ZX=138">(GMT-06:00) Central America, Saskatchewan
<option value="Z=6$DO=1$DL=3 2 1 11 1 1$ZN=Central$ZX=139">(GMT-06:00) Central Time (US &amp;
Canada)
```

```
<option value="Z=6$DO=1$DL=4 1 1 10 -1 1$ZN=Central Mexico$ZX=140">(GMT-06:00) Guadalajara,
Mexico City, Monterrey
<option value="Z=5$DO=0$ZN=SA Pacific$ZX=141">(GMT-05:00) Bogota, Indiana (East), Lima, Quito,
Rio Branco
<option value="Z=5$DO=1$DL=3 2 1 11 1 1$ZN=Eastern$ZX=142" selected>(GMT-05:00) Eastern Time
(US &amp; Canada)*
<option value="Z=3004$DO=0$ZN=Venezuela$ZX=143">(GMT-04:30) Caracas
<option value="Z=4$DO=1$DL=3 2 1 11 1 1$ZN=Atlantic$ZX=144">(GMT-04:00) Atlantic Time (Canada)
<option value="Z=4$DO=0$ZN=SA Western$ZX=145">(GMT-04:00) La Paz
<option value="Z=4$DO=1$DL=10 2 1 2 3 1$ZN=Central Brazilian$ZX=146">(GMT-04:00) Manaus
<option value="Z=4$DO=1$DL=10 2 7 3 2 7$ZN=Pacific SA$ZX=147">(GMT-04:00) Santiago
<option value="Z=3003$DO=1$DL=3 2 1 11 1 1$ZN=Newfoundland$ZX=148">(GMT-03:30) Newfoundland
<option value="Z=3$DO=1$DL=10 2 1 2 3 1$ZN=E. South America$ZX=149">(GMT-03:00) Brasilia
<option value="Z=3$DO=0$ZN=SA Eastern$ZX=150">(GMT-03:00) Buenos Aires, Georgetown
<option value="Z=3$DO=1$DL=4 1 1 10 -1 1$ZN=Greenland$ZX=151">(GMT-03:00) Greenland
<option value="Z=3$DO=1$DL=10 1 1 3 2 1$ZN=Montevideo$ZX=152">(GMT-03:00) Montevideo
<option value="Z=2$DO=1$DL=3 -1 1 9 -1 1$ZN=Mid-Atlantic$ZX=153">(GMT-02:00) Mid-Atlantic
<option value="Z=1$DO=1$DL=3 -1 1 10 -1 1$ZN=Azores$ZX=154">(GMT-01:00) Azores
<option value="Z=1$DO=0$ZN=Cape Verde$ZX=155">(GMT-01:00) Cape Verde Is.
<option value="Z=0$DO=1$DL=3 -1 1 10 -1 1$ZN=GMT$ZX=157">(GMT) Greenwich Mean Time : Dublin,
Edinburgh, Lisbon, London
<option value="Z=0$DO=0$ZN=Greenwich$ZX=156">(GMT) Casablanca, Monrovia, Reykjavik
<option value="Z=-1$DO=1$DL=3 -1 1 10 -1 1$ZN=W. Europe$ZX=158">(GMT+01:00) Amsterdam, Berlin,
Copenhagen, Madrid, Paris, Rome, Stockholm, Vienna, Warsaw
<option value="Z=-1$DO=0$ZN=W. Central Africa$ZX=159">(GMT+01:00) West Central Africa
<option value="Z=-2$DO=1$DL=3 -1 5 9 -1 6$ZN=Jordan$ZX=160">(GMT+02:00) Amman
<option value="Z=-2$DO=1$DL=3 -1 1 10 -1 1$ZN=GTB$ZX=161">(GMT+02:00) Athens, Bucharest,
Istanbul, Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius, Minsk
<option value="Z=-2$DO=1$DL=3 -1 1 10 -1 7$ZN=Middle East$ZX=162">(GMT+02:00) Beirut
<option value="Z=-2$DO=1$DL=4 -1 5 9 -1 5$ZN=Egypt$ZX=163">(GMT+02:00) Cairo
<option value="Z=-2$DO=0$ZN=South Africa$ZX=164">(GMT+02:00) Harare, Pretoria
<option value="Z=-2$DO=1$DL=3 -1 6 9 3 1$ZN=Israel$ZX=165">(GMT+02:00) Jerusalem
<option value="Z=-2$DO=1$DL=4 1 1 9 1 1$ZN=Namibia$ZX=166">(GMT+02:00) Windhoek
<option value="Z=-3$DO=1$DL=4 1 1 10 1 1$ZN=Arabic$ZX=167">(GMT+03:00) Baghdad
<option value="Z=-3$DO=0$ZN=Arab$ZX=168">(GMT+03:00) Kuwait, Riyadh, Nairobi, Tbilisi
<option value="Z=-3$DO=1$DL=3 -1 1 10 -1 1$ZN=Russian$ZX=169">(GMT+03:00) Moscow, St.
Petersburg, Volgograd
<option value="Z=-3003$DO=0$ZN=Iran$ZX=170">(GMT+03:30) Tehran
<option value="Z=-4$DO=0$ZN=Arabian$ZX=171">(GMT+04:00) Abu Dhabi, Muscat
<option value="Z=-4$DO=1$DL=3 -1 1 10 -1 1$ZN=Azerbaijan$ZX=172">(GMT+04:00) Baku, Yerevan
<option value="Z=-3004$DO=0$ZN=Afghanistan$ZX=173">(GMT+04:30) Kabul
<option value="Z=-5$DO=1$DL=3 -1 1 10 -1 1$ZN=Ekaterinburg$ZX=174">(GMT+05:00) Ekaterinburg
<option value="Z=-5$DO=0$ZN=West Asia$ZX=175">(GMT+05:00) Islamabad, Karachi, Tashkent
<option value="Z=-3005$DO=0$ZN=India$ZX=176">(GMT+05:30) Chennai, Kolkata, Mumbai, New Delhi,
Sri Jayawardenepura
<option value="Z=-4505$DO=0$ZN=Nepal$ZX=177">(GMT+05:45) Kathmandu
<option value="Z=-6$DO=1$DL=3 -1 1 10 -1 1$ZN=N. Central Asia$ZX=178">(GMT+06:00) Almaty,
Novosibirsk
<option value="Z=-6$DO=0$ZN=Central Asia/Sri Lanka$ZX=179">(GMT+06:00) Astana, Dhaka
<option value="Z=-3006$DO=0$ZN=Myanmar$ZX=180">(GMT+06:30) Yangon (Rangoon)
<option value="Z=-7$DO=0$ZN=SE Asia$ZX=181">(GMT+07:00) Bangkok, Hanoi, Jakarta
<option value="Z=-7$DO=1$DL=3 -1 1 10 -1 1$ZN=North Asia$ZX=182">(GMT+07:00) Krasnoyarsk
<option value="Z=-8$DO=0$ZN=China$ZX=183">(GMT+08:00) Beijing, Chongqing, Hong Kong, Urumqi,
Kuala Lumpur, Singapore, Taipei
<option value="Z=-8$DO=1$DL=3 -1 1 10 -1 1$ZN=North Asia East$ZX=184">(GMT+08:00) Irkutsk,
Ulaan Bataar
<option value="Z=-8$DO=1$DL=10 -1 1 3 -1 1$ZN=W. Australia$ZX=185">(GMT+08:00) Perth
<option value="Z=-9$DO=0$ZN=Tokyo$ZX=186">(GMT+09:00) Osaka, Sapporo, Tokyo, Seoul
<option value="Z=-9$DO=1$DL=3 -1 1 10 -1 1$ZN=Yakutsk$ZX=187">(GMT+09:00) Yakutsk
<option value="Z=-3009$DO=1$DL=10 1 1 4 1 1$ZN=Cen. Australia$ZX=188">(GMT+09:30) Adelaide
<option value="Z=-3009$DO=0$ZN=AUS Central$ZX=189">(GMT+09:30) Darwin
<option value="Z=-10$DO=0$ZN=E. Australia$ZX=190">(GMT+10:00) Brisbane, Guam, Port Moresby
<option value="Z=-10$DO=1$DL=10 1 1 4 1 1$ZN=AUS Eastern$ZX=191">(GMT+10:00) Canberra,
Melbourne, Sydney
<option value="Z=-10$DO=1$DL=10 1 1 4 1 1$ZN=Tasmania$ZX=192">(GMT+10:00) Hobart
<option value="Z=-10$DO=1$DL=3 -1 1 10 -1 1$ZN=Vladivostok$ZX=193">(GMT+10:00) Vladivostok
<option value="Z=-11$DO=0$ZN=Central Pacific$ZX=194">(GMT+11:00) Magadan, Solomon Is., New
Caledonia
<option value="Z=-12$DO=1$DL=9 -1 1 4 1 1$ZN=New Zealand$ZX=195">(GMT+12:00) Auckland,
Wellington
<option value="Z=-12$DO=0$ZN=Fiji$ZX=196">(GMT+12:00) Fiji, Kamchatka, Marchall Is.
<option value="Z=-13$DO=0$ZN=Tonga$ZX=197">(GMT+13:00) Nuku'alofa</select>
</td></tr>
```

The tag used for this field type is <*select*>. The options are enclosed by the <*option*> tag, and are predefined by the Domino server. If the field already has a value, the selected option raises the *selected* property. The *value* property is only used in case aliases are used on the field options.

Rich text lite

The following example exemplifies the HTML code generated for Notes rich text lite fields translated to HTML on the Web:

```
<tr><td>Rich Text lite: </td><td>
<textarea name="Richtextlite" rows="7" cols="50"></textarea>
</td></tr>
```

The tag used for this field type is *<textarea>*. Its text values are enclosed between the beginning and closing tag. If the field already has a value, its contents are held on the *value* property.

⚠ **Important**
All the rich text formats entered on a rich text lite field are lost if a document is saved on the Web (text is converted to plain/unformatted text). The file attachments are still held on the document and are accessible by using the /$FILE/filename Domino URL. For further information about file attachments on the Web, refer to file attachments.

Color

The following example exemplifies the HTML code generated for Notes color fields translated to HTML on the Web:

```
<tr><td>Color: </td><td>
<input name="Color" value=""></td></tr>
```

The tag used for this field type is *<input>*, with no type attribute set. If the field already has a value, its contents are held on the *value* property. If an invalid color value (non-hexadecimal or non-RGB) value is entered, Domino raises an error. At the Notes/Web tab, the user can choose from a Notes color palette or a Web color palette. Note that the Notes tab becomes the Web tab only if the user has enabled "Use Web palette" (File - Preferences - User Preferences).

Using forms versus pages

This page last changed on Apr 01, 2008 by jservais.

Both forms and pages are displayed on the Web as Web pages, unless the user decides to change the page content type. The main difference between forms and pages are that pages do not store data entered by the user. Fields, subforms, layout regions, and some embedded controls can only be used on forms.

Pages are recommended when displaying information to the user, since they have a much better performance than forms. This is because they are simpler in design than forms and they are served to the user much faster by the Domino server. However, only forms can gather information entered from a user.

There is way to use pages to display HTML forms on the Web. This allows the user to submit data using this design element. This improves performance and reduces the effort on designing application security and architecture. Following this approach might increase the maintenance effort while upgrading hybrid applications. It is also complex for users with minimal experience on HTML forms. For further information about how to use this approach, refer to using pages to submit data.

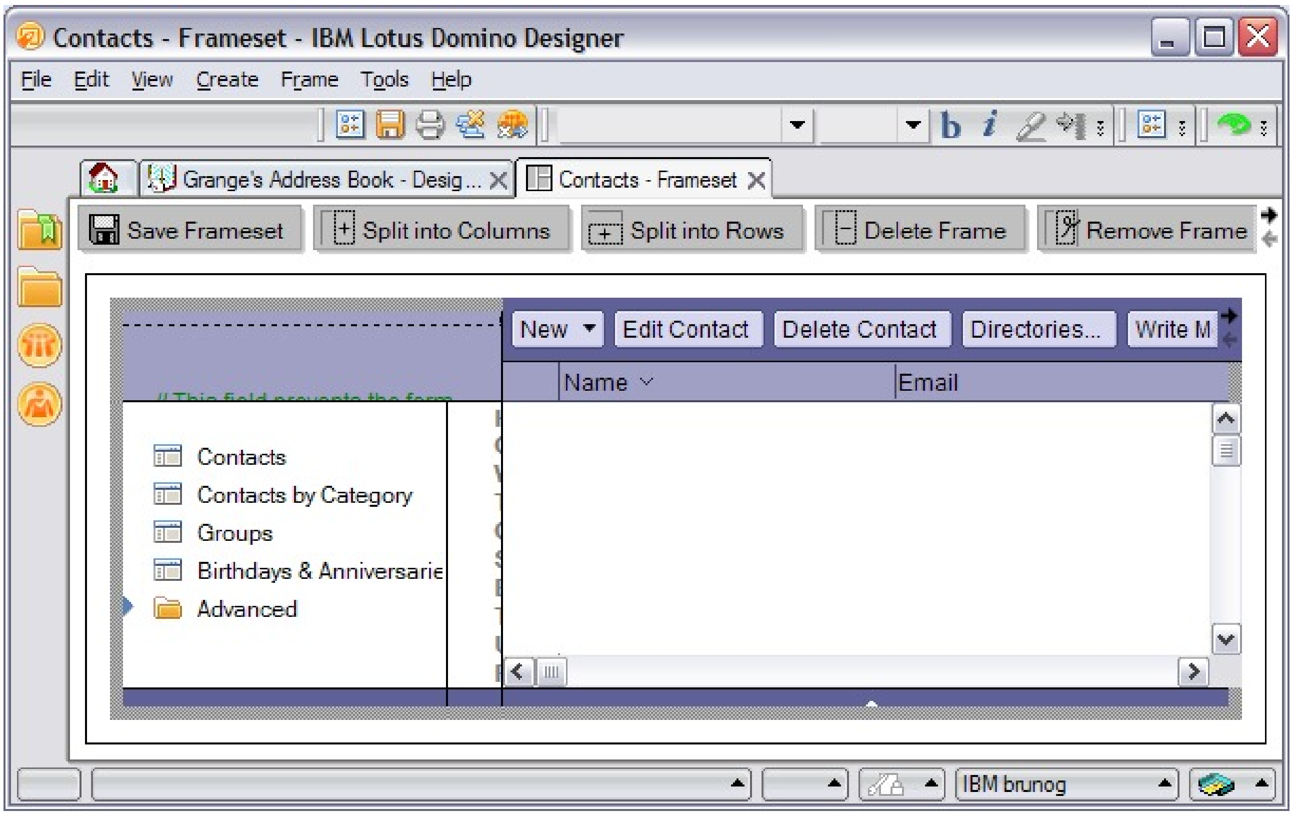
Frameset design elements

This page last changed on Apr 03, 2008 by jservais.

- Frameset and frames
- Using framesets and frames on the Web
- iframe

Frameset and frames

A *frameset* is a collection of frames used to structure a Web site or an Lotus Notes database. A *frame* is a section or pane, of the larger frameset window and is independently scrollable. In Notes, a frame can contain a form, folder, page, document, view, navigator, or another frameset. A frame can also refer to a specific URL. Framesets let you create links and relationships between frames. For example, you can have a page displayed in a page so users can link to other pages or databases in other frames.



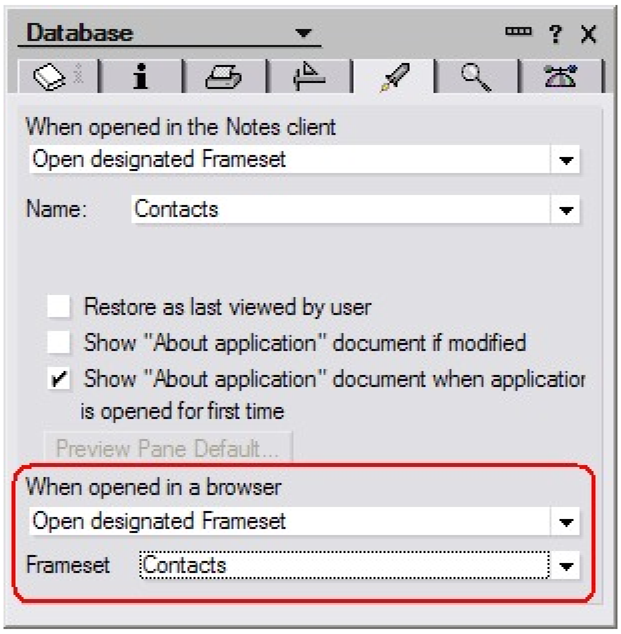
An example of the frameset design element open in Domino Designer

Frameset properties

The frameset design element does not have too may relevant aspects that can confuse a web developer.

The unique property that needs to be highlighted is the window title. Since other design properties can have a window title property set, like a form for example or another "embedded" frameset, remember that only the top frameset is displayed in the window title. Another important detail is that you can set a frameset to launch automatically when a database opens, and this frameset does not need to be the same used on the Notes client. This property is also available on the Web.

Tip
To set a frameset to be launched automatically on the Web on a Lotus Domino application, go into the application properties, select the **Launch** tab. For When opened on a web browser, select the value **Open designated Frameset**, and set your frameset on the Frameset field.



When opened in the Notes client
Open designated Frameset
Name: Contacts

Restore as last viewed by user
Show "About application" document if modified
✓ Show "About application" document when application is opened for first time

When opened in a browser
Open designated Frameset
Frameset Contacts

Setting the frameset set to be launched automatically on the Web

Besides using a frameset design element to create a Web frameset, you can also use a form or page to that. To do that, just create a page for your frameset and name it, for example, frameset.htm. Insert its content HTML content on the design body, and change its content type to HTML. For further information about this topic, refer to using pages as framesets.

Note
Despite the fact that framesets are a normal Web standard and help maintaining hybrid Notes/Web application, using framesets is getting deprecated in actual Web design trends. Refer to Navigation techniques for useful tips on this topic.

Frame properties

When a frameset is created, the text "No content" appears in each frame. To enter the content for a frame in a frameset, follow these steps:

1. Select a frame.
2. Choose **Frame -> Frame properties.**
3. In the Frame Properties box, on the Basics tab, provide a name for the frame.
   Make sure that each frame has a unique name. We do not recommend using the same frame name

for frames in different framesets. For example, if you have a frame named window1 in more than one frameset, you may have unexpected results when you set it as a target frame.

Important
You should not use HTML predefined target names to name a frame (_self, _top, _parent, and _blank).

4. At the Type field of the Basics tab, you can choose one of the following ways contents for the frame: link, named element or URL.

- - Links asks you to paste a link that you copied to the Clipboard.
  - Named elements are the application design elements. It can be be a page, form, frameset, view, folder, navigator or another frameset.

Tip
If you want to display a view or folder on the Web, consider embedding it on a page or form. You may also want to use a view template instead. For further information about this topic, refer to view templates.

- - URLs can be used to link any URL, inside or outside your website.

In addition to adding content to a frame, you can also target a specific frame of a frameset so that all the links open in the target frame. You can specify a target frame in the Frame Properties box or in the Properties boxes of many other elements (such as the properties boxes for Page design elements, Form design elements, outline entries, embedded outlines, group scheduler, and hotspot resources). If no target frame has been specified, the link opens in the same frame that contains the link. If you specify a target frame that does not exist, the link opens in a new, top-level window.

You can also set the width and height of frames, in the Frame Size tab of the Frame properties. These values may be in pixels or screen size percentage and are followed strictly on both Lotus Notes and the Web.

Using framesets and frames on the Web

If you look at the source code of a frameset Web page, you see that it is a special type of Web page that does not contain any content, but a skeleton of a framed group. A frameset tag block can only contain a couple of elements:

- frame tag (<frame>...</frame>) that defines the HTML page that is going to occupy a window. The HTML page can hold common HTML information or another frameset page that will be divided in a new sub-window with other columns or rows.
- frameset tag sub-blocks (<frameset>...</frameset>) that divides again another sub-window (in lines and columns) and can contain frame tags, and new frameset sub-blocks.

The following code example shows a frameset contained in the frameset.htm file:

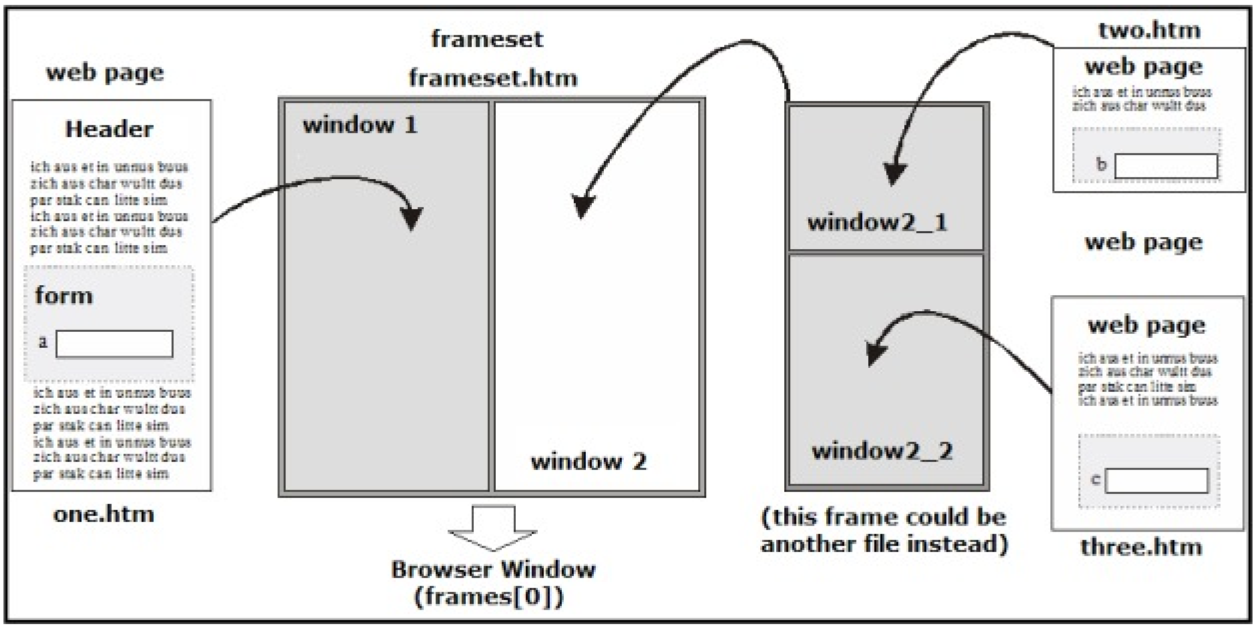
```
<html>
<head> ... </head>
```

```
<frameset cols="50%,50%">
   <frame name="window1" src="one.html">
   <frameset rows="35%,65%">
      <frame name="window2_1" src="two.html">
      <frame name="window2_2" src="three.html">
   </frameset>
</frameset>
</html>
```

Where:

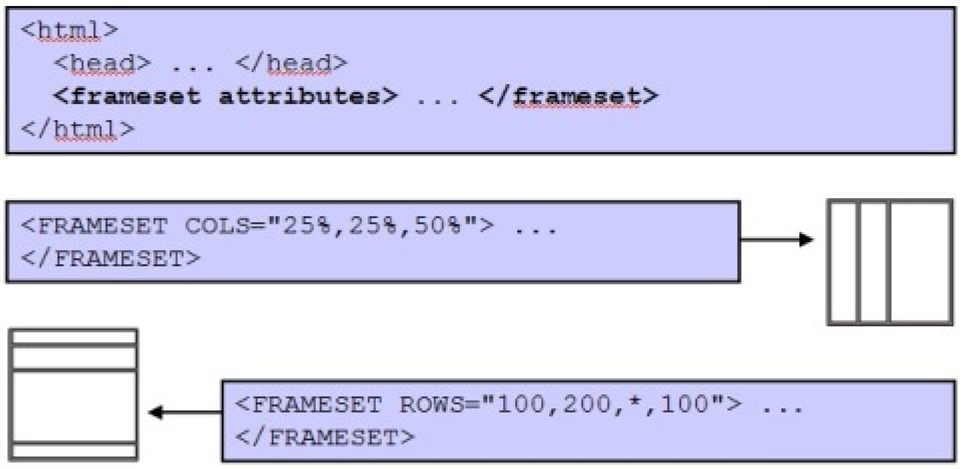
- name: The name of the window, for creating links and referencing via JavaScript.
- src: The page opened on the iframe.
- frameborder: The border of the frame, in pixels.
- scrolling: Defines the scrollbar appearance. Can be set to Yes, No, or Auto to appear only when needed.

The code in the previous example creates a special type of frameset. The first frameset tag splits the page in two columns with 50% of the screen each. The first frame can be invoked by name window_1, and has as source the file one.htm. The second frame is composed of another frameset, that divides the page in two rows: one occupying 35% of the screen and another occupying 65% of it. In this sub frameset, we can invoke the frames by the name of window2_1 and window2_2 respectivelly. These frames source files are two.htm and three.htm respectively.



An example of frameset.html disposition

The following figure shows other examples of some basic frameset structures.



```
<html>
  <head> ... </head>
  <frameset attributes> ... </frameset>
</html>
```

```
<FRAMESET COLS="25%,25%,50%"> ...
</FRAMESET>
```

```
<FRAMESET ROWS="100,200,*,100"> ...
</FRAMESET>
```

An example of a frameset structure

On the Web, a programmer can use JavaScript to set the properties of any frame in a window. It is important to keep in mind that Web framesets are not HTML JavaScript document, since they do not hold any information, but the frameset layout itself. The frames are JavaScript objects accessible through the Frame object. This object is an indexed array, ordered by the order of appearance of a frame in a frameset. The following example shows this array order:

```
<html>
<head> ... </head>
<frameset cols="50%,50%">
   <!-- frames[0] -->
   <frame name="window1" src="one.html">
   <frameset rows="35%,65%">
      <!-- frames[1] -->
      <frame name="window2_1" src="two.html">
      <!-- frames[2] -->
      <frame name="window2_2" src="three.html">
   </frameset>
</frameset>
</html>
```

Use the previous example frameset.htm page for reference in the next examples. You can access a frame in JavaScript using its index or name, as in the following example.

```
parent.frames[0].document.bgColor = "red";
parent.window1.document.bgColor = "blue";
```

Tip
The frame name property is set on the Name field of the Basics tab of the frame properties box.

Since this kind of code is being invoked from the child frames where the JavaScript code relies, we must use the JavaScript *parent* object to invoke them, as shown in the following example:

```
parent.frames[0].document.bgColor = "red";
```

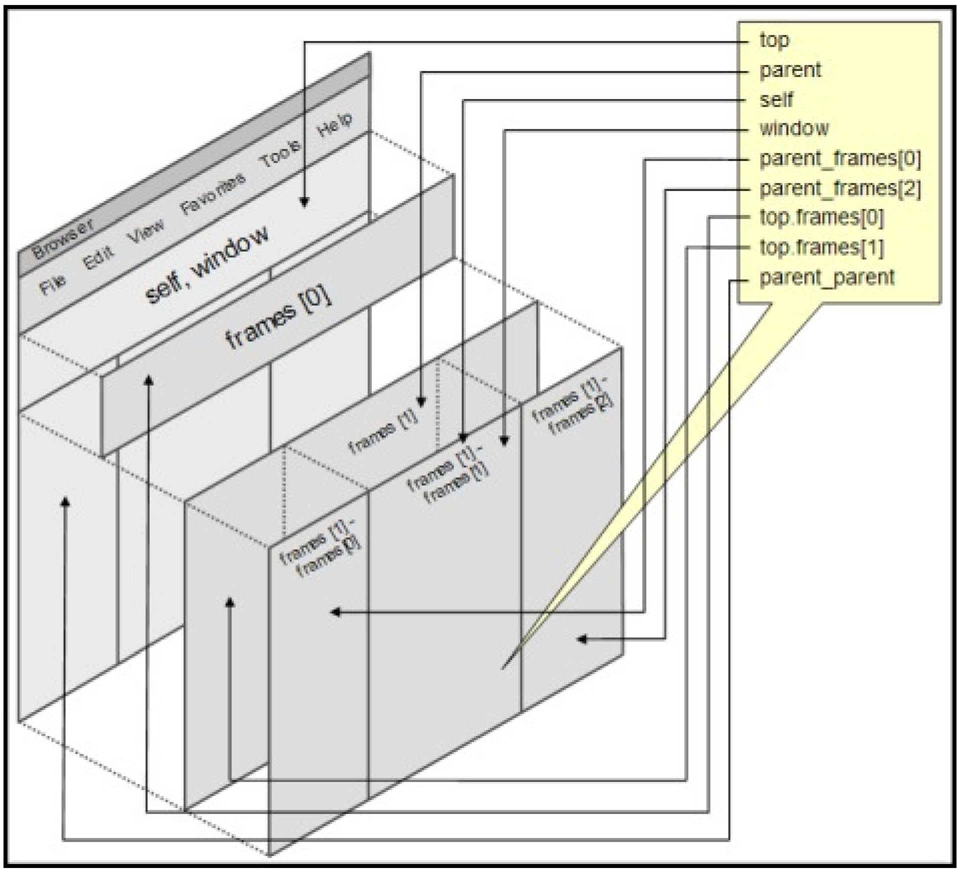
You can also use the self object to refer to frame you are working at that moment, as shown in the following example:

```
self.document.bgColor = "red";
```

Even though, the previous example would not work if they were invoked from third level frames, that is what happens in our case for the frames window2_1 and window2_2. Those frames are children of the first and second framesets respectively. To set the parent page correctly we need to call the parent object twice; or, use the *top* property, that directs to the top frame of a page, as in the following example:

```
parent.parent.frames[0].document.bgColor = "red";
//is the same as...
top.window1.document.bgColor = "red";
```

The following figure shows how JavaScript creates its Frames and Windows objects depending on the frameset disposition. It also demonstrate how a frame can invoke several routines on itself, on the top page and on the other frames of a page, to set new properties on any other document.



A JavaScript example of code called from a frame

iframe

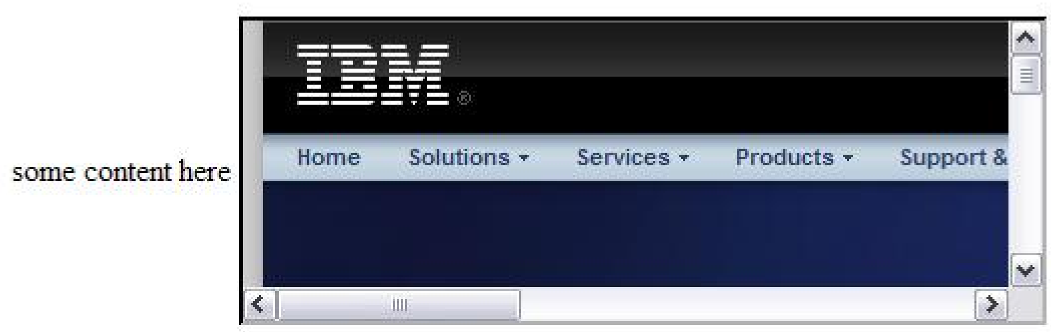
An iframe is an HTML frame that is displayed on a page just as a layer, occupying just a part of the user screen, and not a section. It is similar to the <*div*> tag, but instead of displaying and HTML content from that page, it does a reference to content in another page. The iframe is not recommended for Web sites due to it non-cross compatibility and is getting deprecated. Even though, it is even more functional than the frame itself. It is simple to use and requires just a piece of HTML.

The following example shows an HTML code example of a frame:

```
<iframe name=myiframe src="iframe?OpenForm" frameborder=0 width=400 height=150
scrolling=auto></iframe>
```

Where:

- name: The name of the window, for creating links and referencing via JavaScript.
- src: The page opened on the iframe.
- frameborder: The border of the frame, in pixels.
- width: The width of the frame, in pixels.
- height: The height of the frame, in pixels.
- scrolling: Defines the scrollbar appearance. Can be set to Yes, No, or Auto to appear only when needed.



An example of an iframe referencing ibm.com mixed with page content

Iframes can reference any design element of a Domino application or on the Web using URLs. They are referenced by the frame array object on JavaScript by the order of appearance in the code. They also good alternatives when planning to display alternate content on a web page, like a Domino view or a form in a piece of a page. Since it uses absolute values for size, use it with care to avoid horizontal browser page scrolling.

Image resource design elements

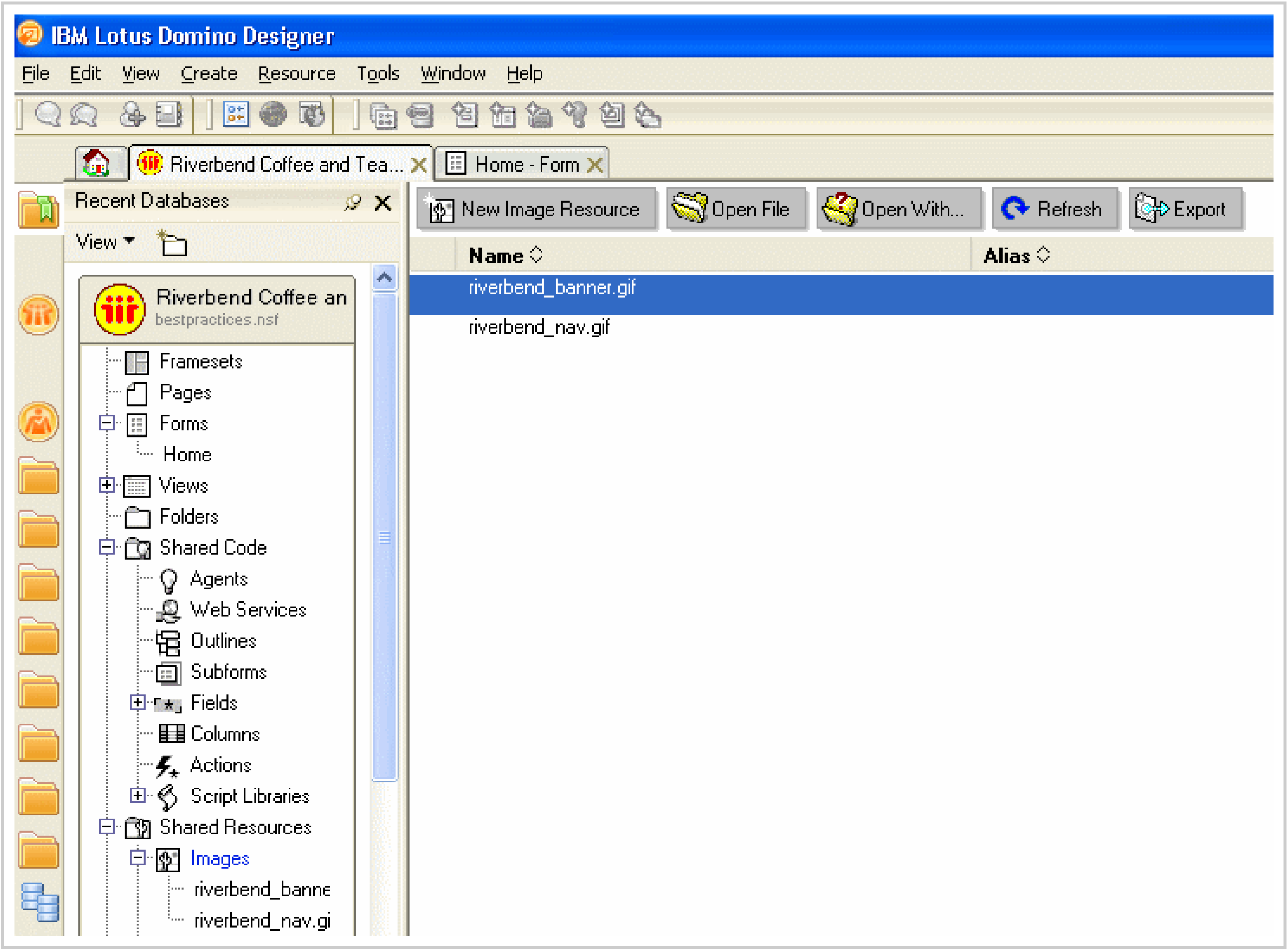
This page last changed on Apr 03, 2008 by jservais.

- Using image resources
- Adding a Web site banner
- Adding a top navigation bar

Images are essential in building a visually appealing web application. By using proper images, your Web application will look more professional.
In this section, we discuss how to use image resources in building a Web application.

Using image resources

Image resources are graphic files that can be used throughout your application. The idea is to have one place to store the graphic files, so that you only have to make changes one time. For example, if you use your company logo in many places throughout your application and the design of your logo changes, you need only change it once and the change is implemented everywhere that the image is referenced.



Note

While image resources can be in GIF, JPEG, or BMP format, they are saved in Domino as GIF or JPEG.

To use an image resource on your form or page, you can:

- Insert the image resource directly by selecting **Create -> Image Resource** from the menu bar
- Reference the image resource in pass-thru HTML

```
<img src="riverbend_banner.gif">
```

The following example shows how to open an image resource by using a Domino URL:

```
http://www.riverbendcoffee.com/riverbend_banner.gif?OpenImageResource
```

Adding a Web site banner

Most Web sites have a banner at the top of the page. A banner provides a uniform look to the Web site. In the following example, we add a banner for the Riverbend Coffee and Tea Company. The following banner (riverbend_banner.gif) was created in Adobe Photoshop. It is 800 x 160 pixels.

RIVER BEND
RIVERBEND
COFFEE AND TEA
COMPANY

The following Pass-Thru HTML was used in a form to generate a table, with the banner as a background in one of the table cells.

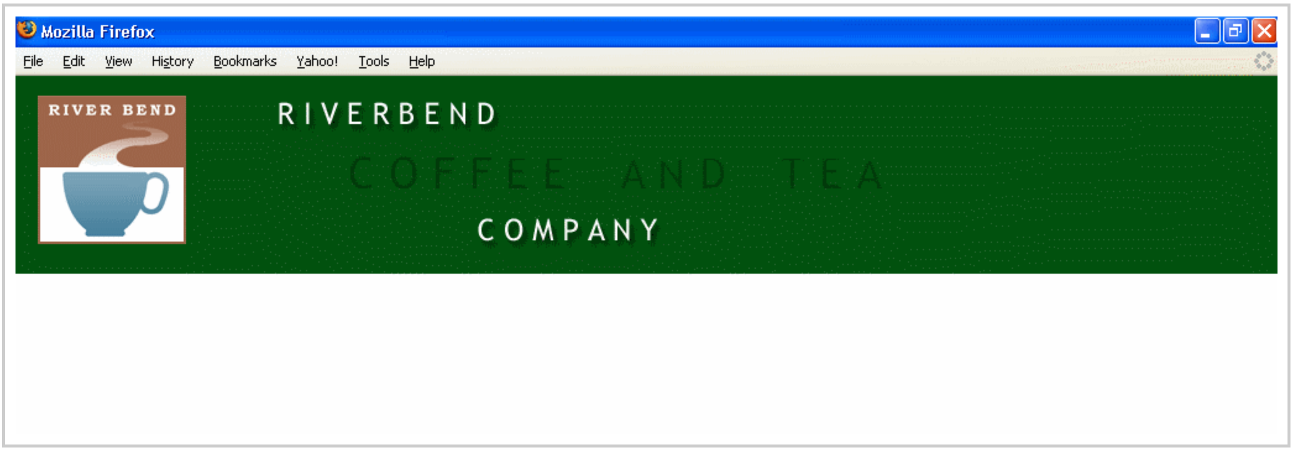
```
<table id="SiteHeaderTable" cellspacing="0">
<tr><td id="SiteHeaderBg">&nbsp;</td><td id="SiteHeaderBgEnd">&nbsp;</td></tr>
</table>
```

The following CSS is used for the form:

```
Body {
        margin: 0px;
        font-family: arial, helvetica, sans-serif;
        font-size: 10px;
}

#SiteHeaderTable {
        width: 100%;
}
```

```
#SiteHeaderTable TD {
        padding: 0px;
}

#SiteHeaderBg {
        background-image: url(riverbend_banner.gif);
        background-repeat: no-repeat;
        width: 800px;
        height: 160px;
}

#SiteHeaderBgEnd {
        background-color: #01520F;
}
```

The following figure shows how the form is displayed in the browser.



Let's look at the Pass-Thru HTML. We basically have a table with two cells. The left cell has an ID of SiteHeaderBg and the right cell has an ID of SiteHeaderBgEnd.

In the CSS, note the following points:

- Page Margin is set to 0px in the Body section. Therefore, there is no margin surrounding the banner.
- SiteHeaderBg has the background image with the the dimensions of the table cell match the image size.
- The background-repeat: no-repeat prevents your banner from tiling.
- SiteHeaderBgEnd has the same background color as the banner to ensure that the banner fills the screen regardless of the screen resolution.

Adding a top navigation bar

Top navigation bar is another common web standard that offers the following benefits:

- It makes your Web application to not look like a Lotus Notes application
- It gives you more real estate for content as opposed to the typical left navigation

We created a top navigation bar of 10 x 40 pixels with Adobe Photoshop. We used a gradient overlay to achieve the pearly effect as you can see in the following figure.

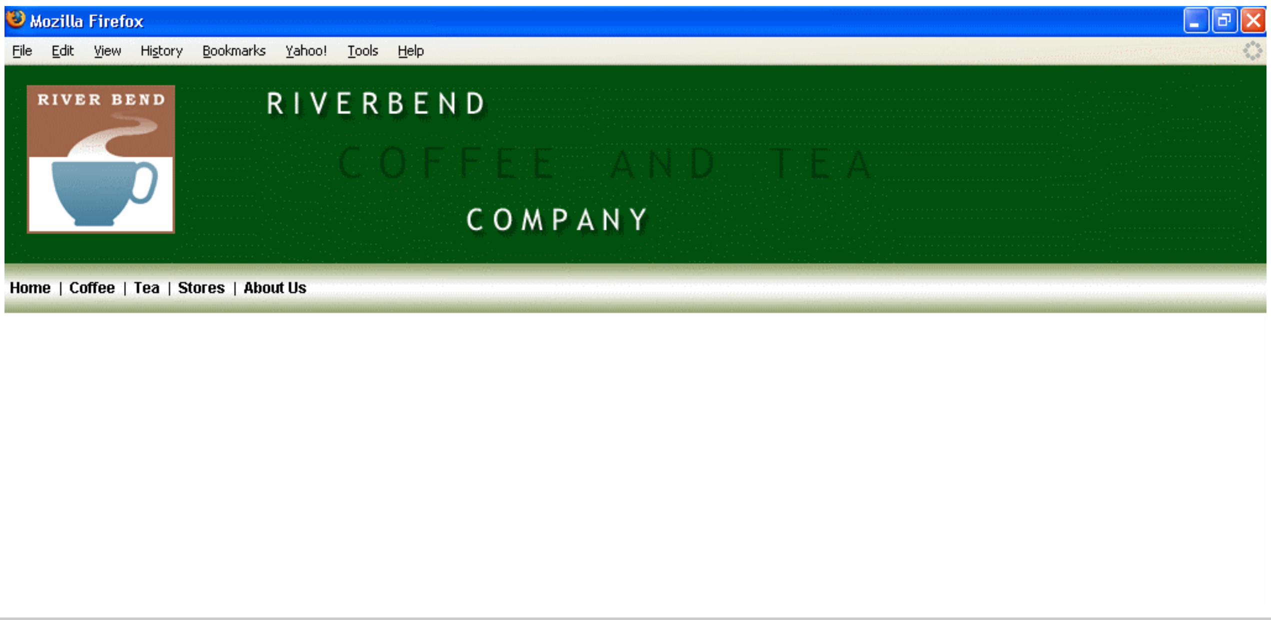
The following code example shows the pass-thru HTML:

```
<table id="SiteHeaderTable" cellspacing="0">
<tr><td id="SiteHeaderBg">&nbsp;</td><td id="SiteHeaderBgEnd">&nbsp;</td></tr>
<tr><td id="SiteNav" colspan="2">&nbsp;&nbsp;
<a href="#this" onclick="location.href="<Computed Value>">Home</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this" onclick="location.href="<Computed Value>">Coffee</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this" onclick="location.href="<Computed Value>">Tea</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this" onclick="location.href="<Computed Value>">Stores</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this" onclick="location.href="<Computed Value>">About Us</a>
</td></tr>
</table>
```

The following code example shows the style sheet:

```
#SiteNav {
        background-image: url(riverbend_nav.gif);
        height: 40px;
        font-family: arial, helvetica, sans-serif;
        font-size: 12px;
        font-weight: bold;
}

#SiteNav a {
        font-family: arial, helvetica, sans-serif;
        font-size: 12px;
        font-weight: bold;
        text-decoration: none;
        color: black;
}

#SiteNav a:hover {
        text-decoration: underline;
        color: red;
}
```

The following figure shows the end result.


Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
RIVER BEND
RIVERBEND
COFFEE AND TEA
COMPANY
Home | Coffee | Tea | Stores | About Us

Let's look back at the pass thru HTML. We added another row below our banner to show the image and anchor links such as Home, Coffee, Tea, etc. The table cell has an ID of SiteNav, which is defined in our style sheet.

In the style sheet, we reference the image to be the background image of the table cell. It also has a height of 40 pixels to make sure that the image is shown correctly. The a and a:hover are used to define how the anchor links look.

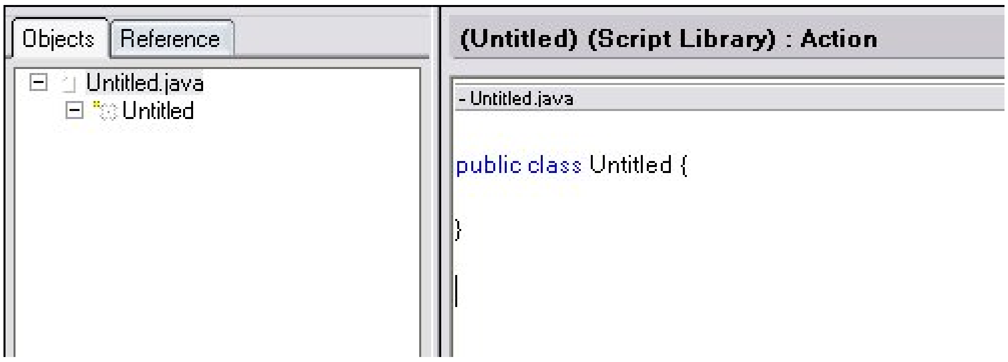
Tip

The next step after adding this navigation bar is to add a DHTML drop-down menu. You can find a lot of examples simply by searching for it on the Web.

This page last changed on Apr 01, 2008 by jservais.

How to work with Java libraries in Domino

Java can be included in the Domino Database as a shared code library. The Script Libraries window in Domino Designer has a specific action button to create a shared Java library. When the library is created, Domino pre-fills nothing except the actual class statement, as you can see in the following figure. You must to change the name of the class and save it before you can use the class.



```
public class Untitled {

}
```

Creating a new class

You can include multiple classes in a library. To include a new class, use the **New Class** action in the Domino Designer pane while editing the library. Each new class appears in the designer pane as a new Java file. In order to reference the script library, we recommend that you include a package name. The package name is placed at the beginning of the Java file, which is standard convention.

Saving your project automatically compiles the code. Alternately, you can explicitly use the **Compile...** function to compile all or individual files.

When you use the **Export** function to export your library files, only the .java source files are written to your file system. You can then include and use them in another Java IDE project. Keep in mind that if you compile such files in another development environment, you must ensure that the other development environment has notes.jar available for inclusion.

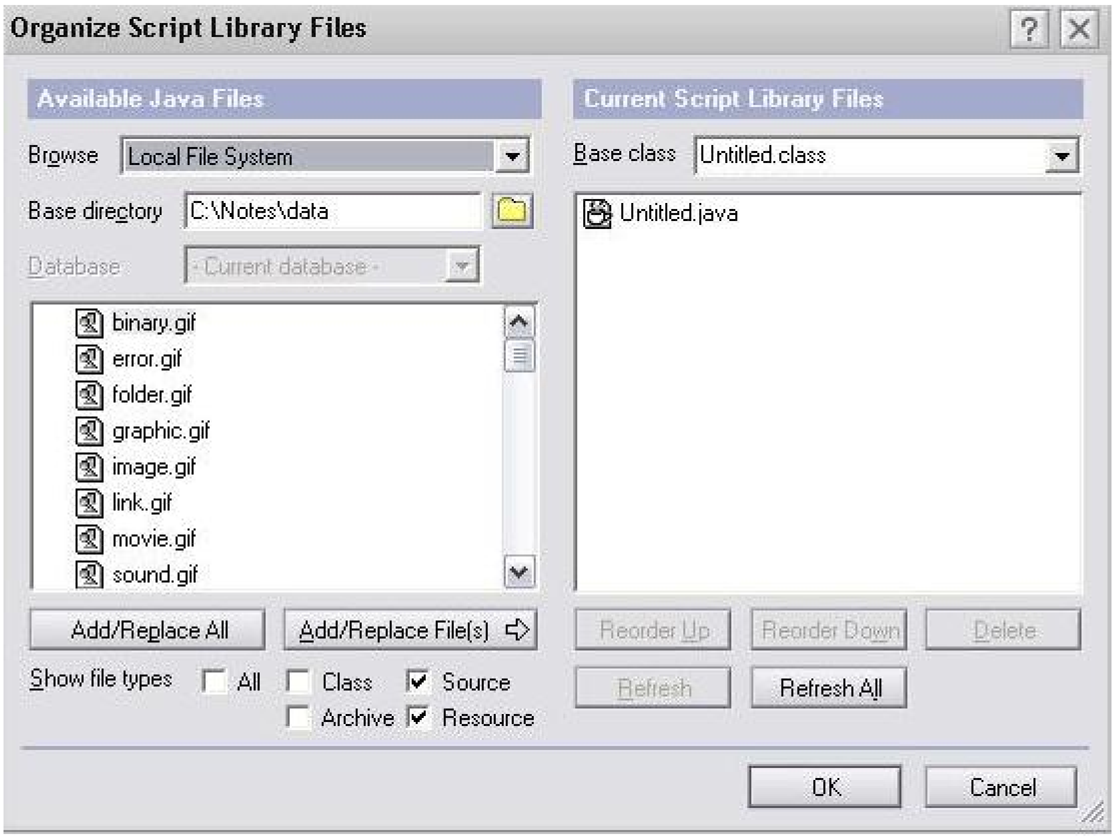
Calling your library

In order to call your Java library from an agent, you must perform the following actions:

- While editing the agent, use the **Edit Project** button at the bottom of the design window, and include the library as part of your project.

- Any agent that references your library simply includes it via an import statement.

The following figure shows a sample of the edit project window.



Browse: Local File System
Base directory: C:\Notes\data
Database: - Current database -

binary.gif
error.gif
folder.gif
graphic.gif
image.gif
link.gif
movie.gif
sound.gif

Base class: Untitled.class

Untitled.java

Show file types: [ ] All [ ] Class [✓] Source [ ] Archive [✓] Resource

There are two options to browse the available Java files. The first option is to reference your shared Java library. The second option is to reference an external JAR, class, or Java file. After you make your selection, click the appropriate **Add/Replace** button to include them in your project. Then click **OK**.

Some tips on using Java with Domino

Always recycle the underlying Domino objects, because Java's garbage collection does not recycle the mapping and underlying Domino C++ objects. To avoid this, explicitly call each Domino object's method when they are no longer referenced.

Do not attach JAR files to agents or script libraries, because the JAR files must be detached, and then loaded and unloaded every time the agent runs. This can result in low or out of memory issues. If you place the files directly on the file system (the notes external jar directory), then they are only loaded once. Your next best option is to import the source code and have everything self contained in the database.

JavaScript library design elements

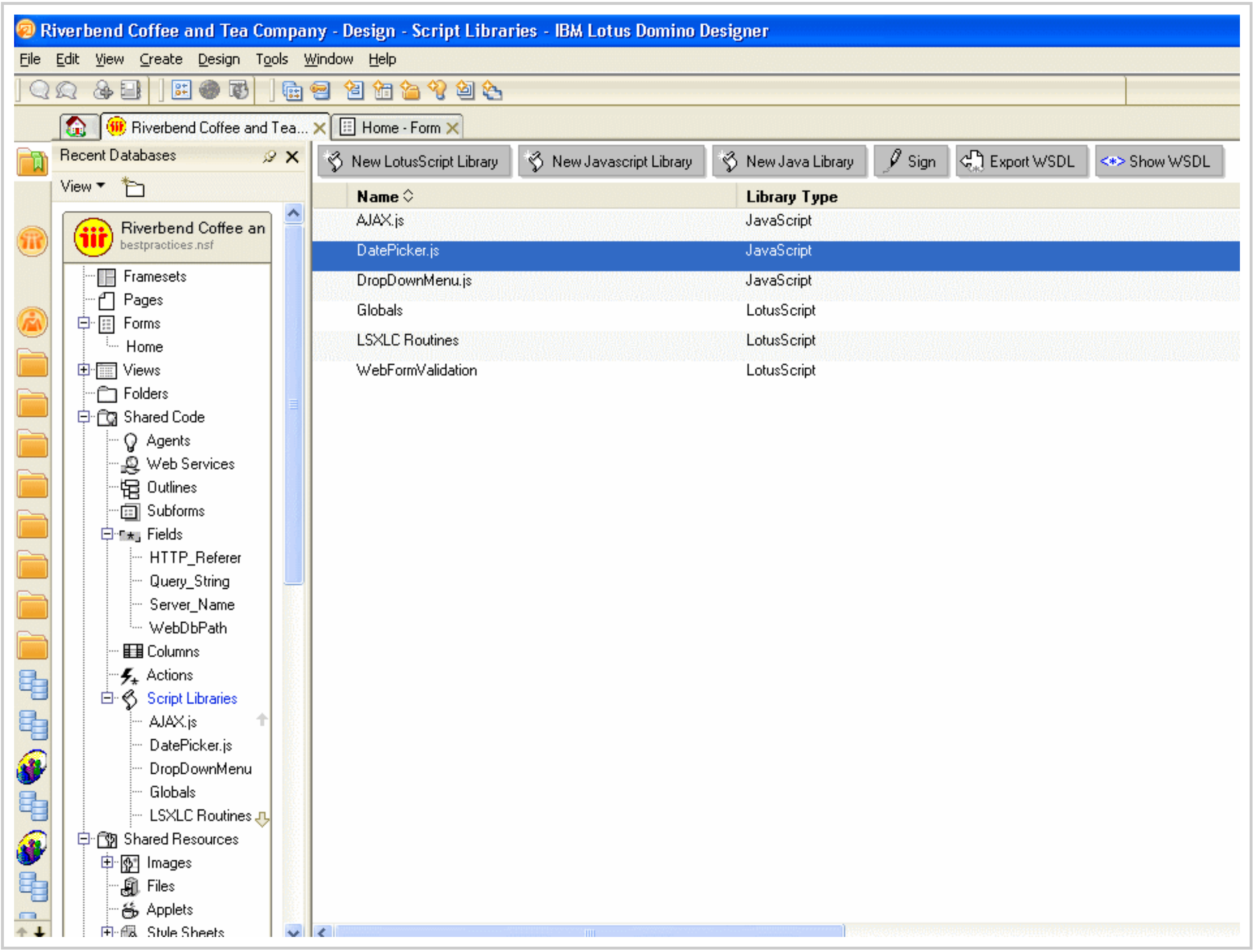
This page last changed on Apr 01, 2008 by jservais.

JavaScript libraries are a collection of JavaScript functions that are grouped by purpose or use. You can store JavaScript in a number of places for Web use. Where you store it is more a matter of personal preference.

- In-line on the form by using the HTML script tags
- On pages where the content type is set to html or text/javasctipt
- In file resources as text files (usually with a .js extention, for easy identification)
- Javascript Libraries (same place as LotusScript and Java Libraries)

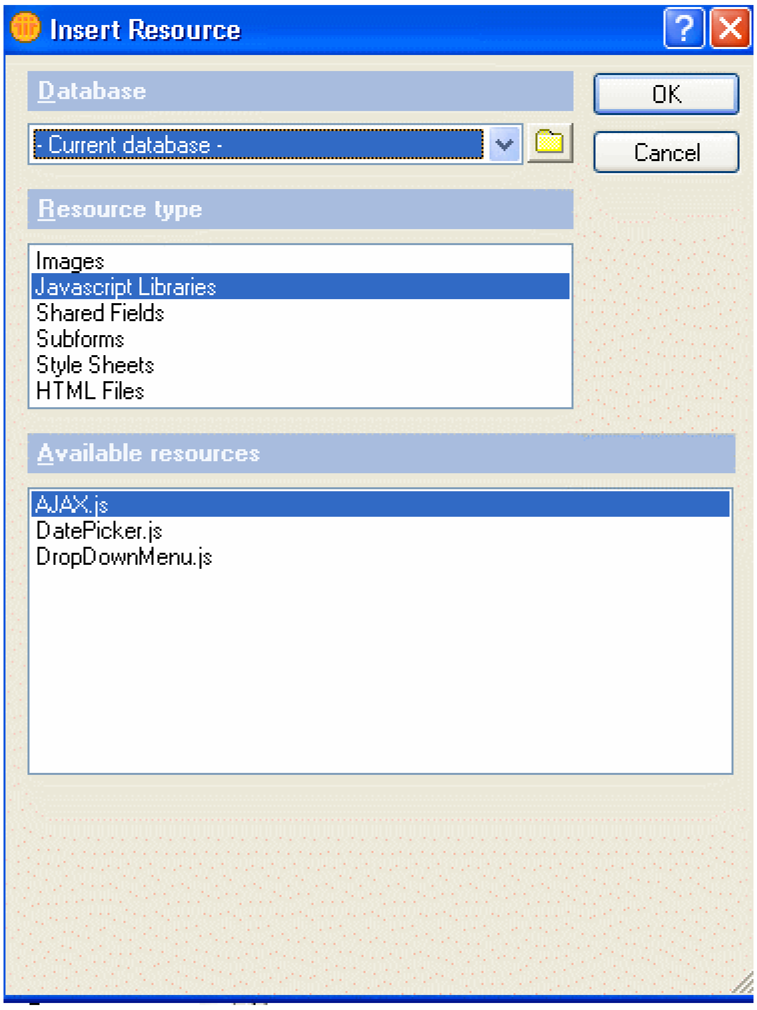
The biggest difference between the methods is that JavaScript libraries have a JavaScript validator and can be included in *JS Header* area on the form and other areas using Insert Resource. The rest must have the HTML include script tag manually entered. The disadvantage is that you cannot compress the library like you can with the text file format.

```
<script type="text/javascript" src="/lightbox.nsf/prototype.js"></script>
```

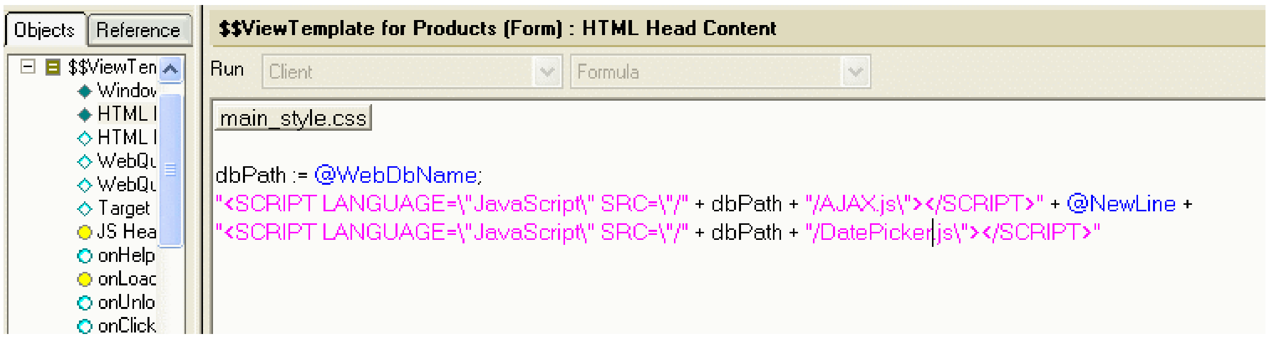


Name	Library Type
AJAX.js	JavaScript
DatePicker.js	JavaScript
DropDownMenu.js	JavaScript
Globals	LotusScript
LSXLC Routines	LotusScript
WebFormValidation	LotusScript

The following figure shows an example of referencing JavaScript libraries using Insert Resource.



The following figure shows an example of referencing JavaScript libraries in the HTML Head Content of a form.

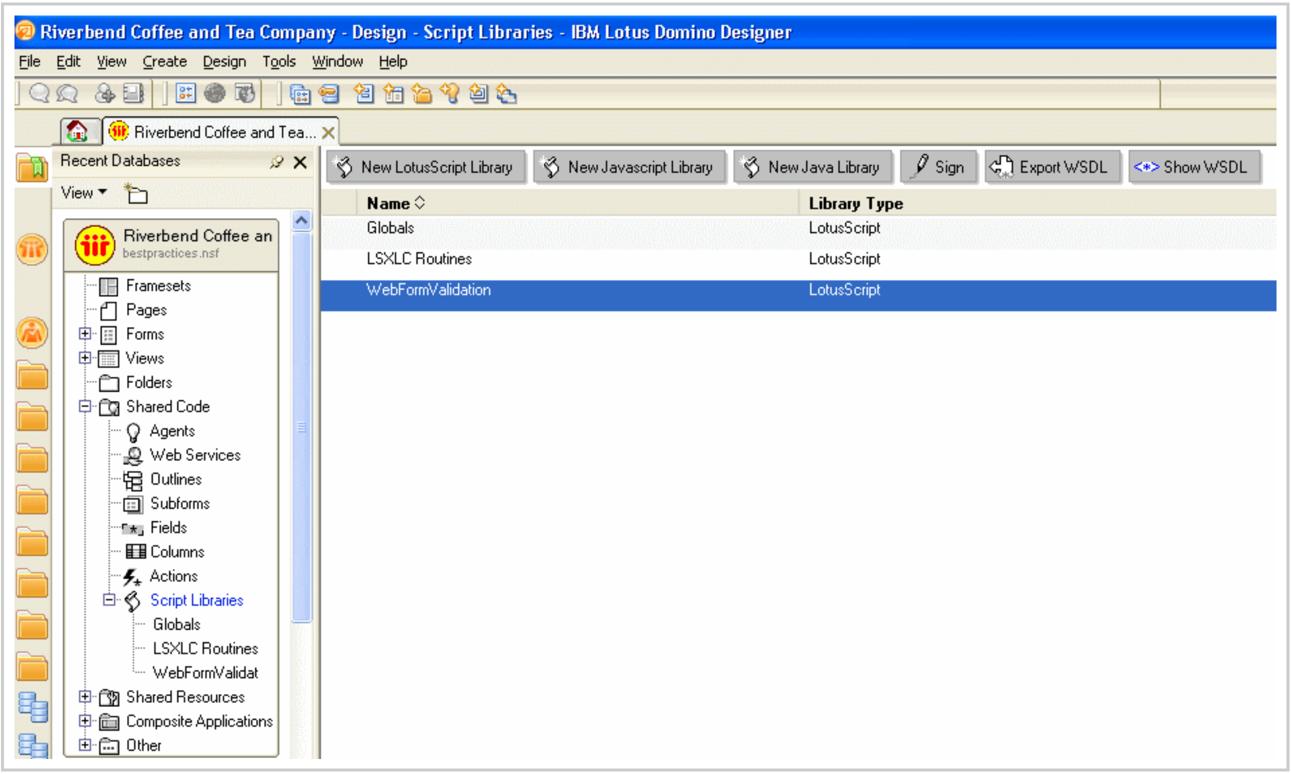


```
main_style.css

dbPath := @WebDbName;
"<SCRIPT LANGUAGE=\"JavaScript\" SRC=\"/" + dbPath + "/AJAX.js\"></SCRIPT>" + @NewLine +
"<SCRIPT LANGUAGE=\"JavaScript\" SRC=\"/" + dbPath + "/DatePicker.js\"></SCRIPT>"
```

LotusScript library design elements

This page last changed on Apr 01, 2008 by jservais.

LotusScript libraries are useful in organizing various functions and subroutines that you have in a database. For example, you might have a library called *Globals* that stores all your common code. You might also have a library called *WebFormValidation* that stores all your form validation code.



Name	Library Type
Globals	LotusScript
LSXLC Routines	LotusScript
WebFormValidation	LotusScript

When using the LotusScript library design elements, keep the following points in mind:

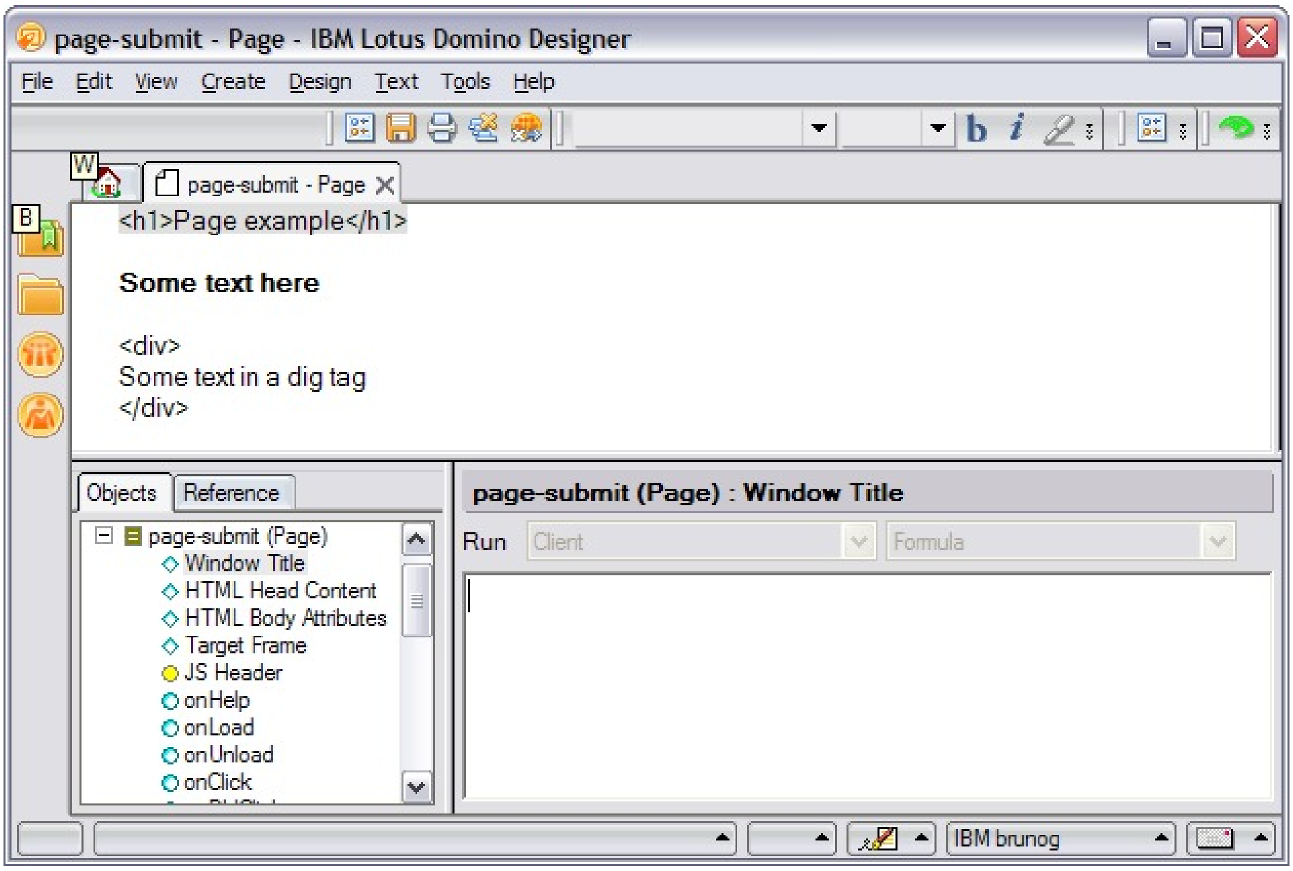
- Always use the "option declare" in the Options section of the script library. This option requires that all variables are declared, but in return, you know for sure that there will not be any typos on the variable names.
- A "Use or UseLSX" error message sometimes appear when saving a script library. Try recompiling all LotusScript (Tools -> Recompile All LotusScript)

Page design elements

This page last changed on Apr 04, 2008 by dalandon.

Page design elements

Pages and Form design elements are similar design elements, since they are used to hold similar design. These designs and are often confused among beginners Domino Designer application developers, principally when developing on Web applications. Pages are can be used to host several types of data to the user on the Web, but specially to display rich text content. Pages can hold formatted text, graphics and embedded controls, such as outlines and applets. To learn how to make sure how your design is going to look into the Web, refer to Styling text for the Web.



```
<h1>Page example</h1>

Some text here

<div>
Some text in a dig tag
</div>
```

A page design element open in Domino Designer

The following table lists the Domino design elements that can be used on pages on the Web.

Elements to use on a page	Description
Actions	Actions automate tasks for the user. Add actions to the menu in the Notes client, or add actions with buttons or hotspots on a page or form. For more information, see the topic, refer to actions.
Applets	Use Java applets to include small programs, such as an animated logo or a self-contained application, in a page or form. For more information about including Java applets on a page or form, refer to Applet design elements
Attachments	Attach files to a page or form so users can detach or launch files locally. For more information, refer to file attachments.
Computed text	Use computed text to generate dynamic text based on formula results.
Embedded elements	You can embed the following elements in a page or form: a view or folder pane, navigator, outline, date pickers or instant messaging contact list. Use these elements alone or combine them to control how users navigate through your application.
Graphics	Place a graphic anywhere on a page or form. Use graphics to add color to a page or form or to create imagemaps.
Horizontal rules	Add horizontal rules to separate different parts of a page or form, or to make a page or form more interesting visually.
HTML	If you have existing HTML or you prefer using HTML to using the formatting tools Domino Designer offers, you can import, paste or write your own html on a page or form. You can also convert pages and forms to HTML.
Imagemaps	An image map is a graphic you enhance with programmable hotspots. Hotspots, in the form of pop-up text, actions, links, and formulas, perform an action when clicked by a user. Use imagemaps as navigational structures in an application.
JavaScript libraries	You can find and insert JavaScript libraries into a page, form or subform. For more information on inserting JavaScript libraries, refer to JavaScript library design elements.
Layers	Layers let you position overlapping blocks of content on a page, form, or subform. Layers give you greater design flexibility because you can control the placement, size, and content of information. For more information on layers, refer to Layers.

Links	Add links to take users to other pages, views, databases, or URLs when they click on text or a graphic.
Sections	A section is a collapsible and expandable area that can include objects, text, and graphics.
Style Sheet (CSS) shared resources	You can find and insert a cascading style sheet (CSS) as a shared resource on a page, form, or subform. For more information on style sheets, refer to style sheets.
Tables	Use tables to summarize information, align text and graphics in rows and columns, or position elements on a page or form.
Text	Use text anywhere on a page or form and apply text attributes, such as color, size, and font styles to the text.

Page properties

Pages have a set of properties that can impact the user experience on the Web. Refer to other web specific database properties and their effects for further information.

Differences between pages and forms

Both forms and pages are displayed on the Web as Web-pages (unless the user decides to change the page content type), but the main difference between them are that pages do not store data entered by the user. Fields, subforms, layout regions, and some embedded controls can only be used on forms. There are also some other differences and best practices when using pages and forms, so refer to using forms versus pages for further reference.

Pages naming best practices

Pages can be named using the Design element multi-aliasing technique. Using this we can have a form called "Top rated product vendor|vendor.htm|vendor". This naming convention also brings the benefit of have an alias that works like a traditional Web server "file name" for Domino URLs, helping keeping aliases even if the main page content changes. This also helps search crawlers increase the page rank by using the search engine optimization best practice of having a small description of the page on its URL. This practice is explained more detailed on the search engine optimization section.

Additional topics

- Using pages to submit data

Using pages to submit data

This page last changed on Apr 04, 2008 by dalandon.

Using pages to submit data

There is way to use pages to display HTML forms on the Web. This allows the user to submit data using this design element. This improves performance and reduces the effort on designing application security and architecture. Following approach may increase the maintenance effort while upgrading hybrid applications, and is also complex for users with low experience on HTML forms. The advantage is that you have full control over the doctype. If you want an XHTML compliant form, this would work.

This approach is recommended for one-time entry only data, like Web site pools or contact forms. If it is desired to allow users to edit that data in a later time, programmer must consider using forms. The only requirements will be that the database contain a form with the same name as the HTML form, it be accessible to the Notes Client and the fields on the Notes form match the fields on the HTML form.

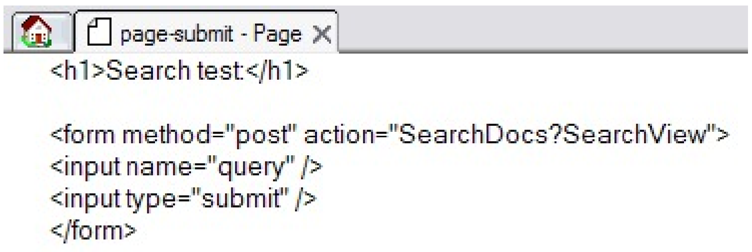
Important
To allow users to submit this data via *forms*, they must have at least depositor access on the database access control list.

Follow the example on a search input page below to understand how to submit information using pages:

1. Create a page, and set its Content type to HTML.
2. Add the following code at the beginning of your page.

```
<form method="post" action="SearchDocs?SearchView">
```

3. To allow users to submit data via this HTML form, add a code similar to the example in the following figure on your page.



```
<h1>Search test</h1>

<form method="post" action="SearchDocs?SearchView">
<input name="query" />
<input type="submit" />
</form>
```

Using a page design element to submit a query

With this code, when the user opens this Web page and clicks the Submit button, it submits the value of the *Query* input field to the SearchDocs form, that actually is a search template. This is done by using the POST HTTP method, just like using GET method, optionally used on searches via Domino URLs. Since search templates uses a field called *Query* as the search term for a search, this action triggers a search on the SearchDocs view for the value inputed on the *Query* field.

Creating a Notes document by using a page

...*Share your best practices here.*

Using QuerySave agents to receive submitted data

...*Share your best practices here.*

Request_Content_nnn

In Lotus Domino releases prior to 7.0, the maximum size of POST data that could be handled by the REQUEST_CONTENT field was 64KB. When the 64 KB threshold was exceeded, the field was not usable. This has been a problem for those applications that use the REQUEST_CONTENT field. In release 7.0.1, this limit has been removed and Domino can now handle POST data larger than 64 KB in the REQUEST_CONTENT field. This is accomplished as follows:

- If the POST data is less than 64 KB, then use REQUEST_CONTENT to access the POST data.
- If the POST data is greater than 64 KB, then use REQUEST_CONTENT_000 to access the first 64KB chunk, REQUEST_CONTENT_001 to access the second 64KB chunk, REQUEST_CONTENT_002 to access the third 64 KB chunk, and so on. Thus, a developer can use the *NotesDocument.HasItem("REQUEST_CONTENT")* call to test for the presence of the REQUEST_CONTENT field. If it exists, then there was less than 64KB of POST data.

Important
The Server Document setting, "Maximum POST data," refers to the maximum amount of POST data that Domino will accept. This field (Internet Protocols panel -> Domino Web Engine) does not affect the REQUEST_CONTENT field and how it is used.

Profile documents

This page last changed on Apr 04, 2008 by dalandon.

In this section, we will review the practice of creating and utilizing Domino Profile Documents in our Domino Application Development practices.

What is a profile document?

Profile documents are typically used to store application and user preference data in order to facilitate personalization. These documents are like typical Domino database documents, except they are excluded from the database document count and are cached when said database is opened.

Profile documents are not visible in View design elements that include all documents (via *@Select ALL* in the View Selection Formula, and are often accessed and updated programmatically.

Using profile documents

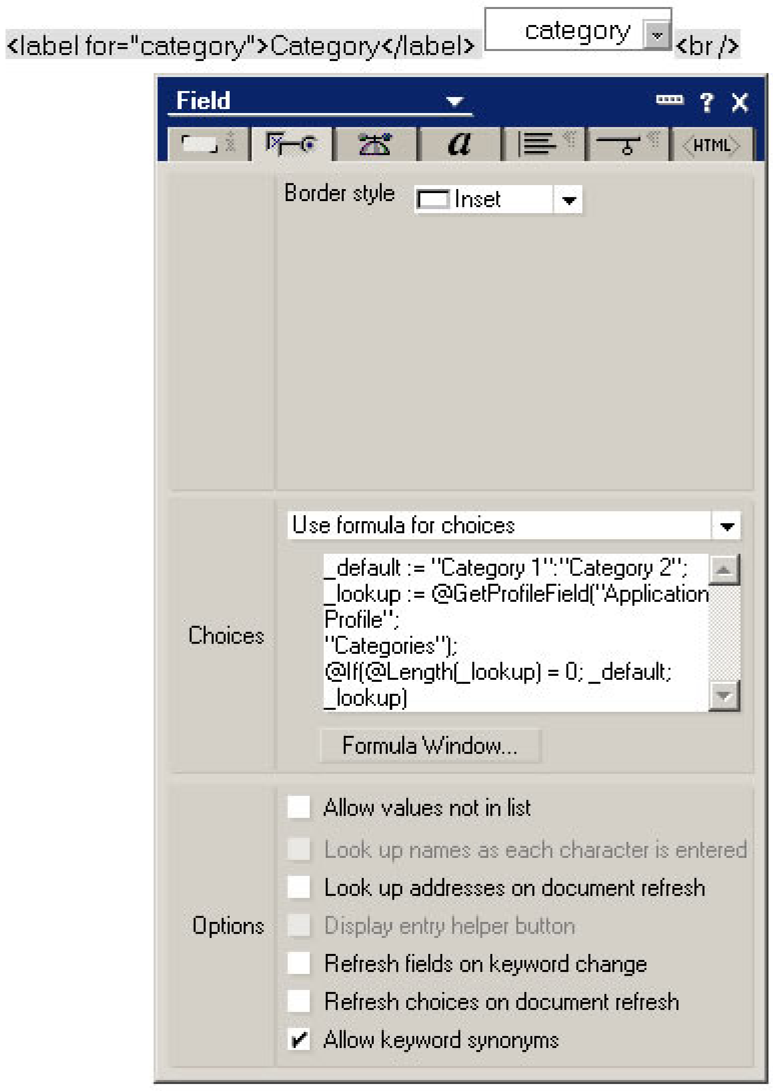
We recommend that you review the following Lotus Domino Designer Help Articles which discuss the Profile Document engine:

- Profile forms
- Creating a profile form

When you are familiar with the creation and maintenance of profile documents, you can begin to architect your Domino applications to use profile documents.

Examples of profile Documents

A simple example of the proper utilization of Profile Documents is the maintenance and utilization of a *Category* metric. Our example application has a Form Design Element named "Example Form". *Example Form* has a field called "category", which is used to categorize documents that are created/maintained with this Form Design Element. To maintain data integrity while we facilitate admin-maintenance of the categories, we create a profile document that is used to maintain the choices in the *Example Form*'s *category* field element.


```
<label for="category">Category</label> category <br />
```

Field

Border style: Inset

Choices: Use formula for choices

```
_default := "Category 1":"Category 2";
_lookup := @GetProfileField("Application Profile";
"Categories");
@If(@Length(_lookup) = 0; _default; _lookup)
```

Options:

- [ ] Allow values not in list
- [ ] Look up names as each character is entered
- [ ] Look up addresses on document refresh
- [ ] Display entry helper button
- [ ] Refresh fields on keyword change
- [ ] Refresh choices on document refresh
- [✓] Allow keyword synonyms

In the previous example, we architected the *category* field element to provide the user with a choice list that is maintained in the *Application Profile* Profile Document. Additionally, we provided a default choice list should an issue occur with the *Application Profile* Profile Document.

In using this architecture, we can easily provide application maintenance capabilities to specific *Application Administrators*.

Known issues with profile documents

When leveraging profile documents in multi-client/hybrid applications, the Domino HTTP Task does not update its cached profile documents when those documents are updated via the Notes client. This inability for the Domino HTTP Task to (immediately) recognize updates to profile documents and update its cache accordingly has resulted in the usage of *standard* Notes documents to contain typical profile document content.

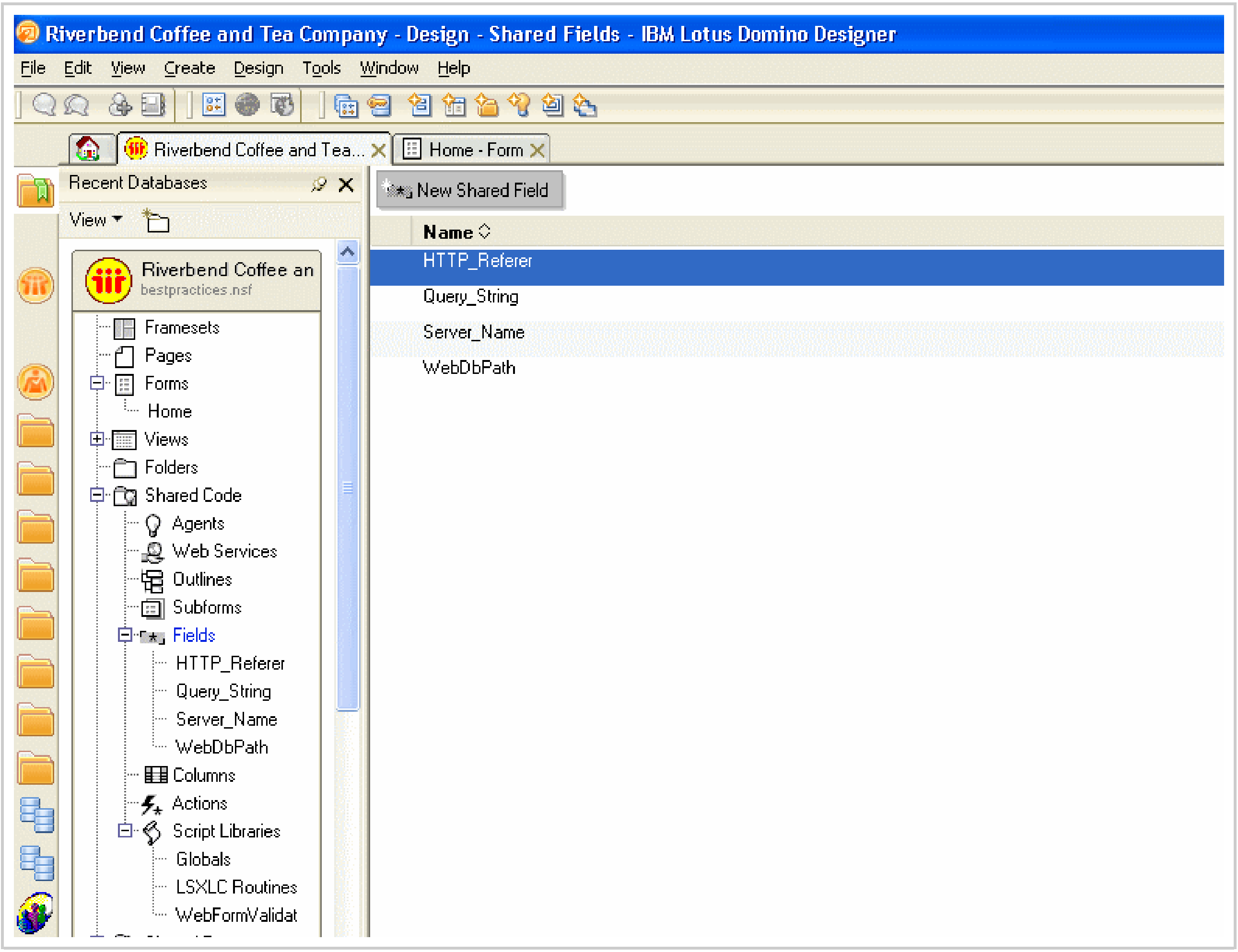
The utilization of Notes documents as profile documents, while addressing the Domino HTTP Task caching issue, adds additional documents to the Domino database document count as well as often requires utilization of other Domino Design Elements such as View design elements and Agent design elements.

Lotus Notes client applications can safely use the profile document engine. However we recommend that you consider alternatives when creating Hybrid Client Applications or Client-Type Applications that are delivered via the Domino HTTP Task.

Shared field design elements

This page last changed on Apr 01, 2008 by jservais.

Shared fields have been around for a long time. Notes developers love this design element because they only have to make changes in one place instead of touching the same field on 20 forms or subforms. Shared fields are useful to hold CGI Variable fields or the database path.



Alternatively, you can also use regular fields on a subform. It is a matter of preference.

Subforms design elements

This page last changed on Apr 03, 2008 by jservais.

A *subform* is a design element that can hold a number of form elements that are stored as a single design element. Subforms can contain the same elements as regular Form design elements, with some minor exceptions, such as the HTML Head content, the window title and the WebQueryOpen and WebQuerySave agents. Subforms also works similar to the include() method on some server side scripting languages, where the programmer can add a piece of code to the current page.

Subforms reduce the efforts with application maintenance. When you change the content of a subform, like a field formula, every form that uses the subform updates. Common uses of subforms include adding common JavaScript functionalities to form pieces or showing a Web page header. A subform can be a permanent part of a form or can appear conditionally, depending on the result of a formula. For example, you may want to hide part of a Web form if the user do not have the proper access to edit that.

Important
Field names used in the subform can not be used elsewhere on their header forms. Changes you make to a subform affect all forms and documents that use the subform.

Subform properties

Subforms have similar properties to forms. You edit them from the Subform Info tab. The following table lists the subform properties and explains when to use them.

Subform property	Use
Include in Insert Subform dialog	Lets designers see the subform name when inserting a subform. Excluding a subform from the Insert Subform dialog box is not a security measure. Users with Designer access or higher can open any subform in IBM Lotus Domino Designer and copy individual components. Note that this field does not apply to computed subforms.
Include in New Form dialog	Check this if you want the subform to appear immediately when designers choose Create - Design - Form. Note that this field does not apply to computed subforms.
Render pass through HTML in Notes	Lets you paste HTML directly into the subform. For more information on pasting in HTML, see the topic passthru HTML, html content type

Do not add field names to field index	Check this setting to prevent new field names on the subform from being saved in the field index. Checking this setting saves memory. If you do not check this setting, field names are saved to a table and then stored in memory. Storing field names in memory allows field names to appear in places such as the "Add Action" dialog box.

Example of displaying a computed subform

If in a certain application you want to display the NewWebDoc when a document is created and the SavedWebDoc when a saved document is opened. You can define to each subform contains different fields and html code. The following example shows the Insert Subform formula:

```
@If(@IsNewDoc; "NewWebDoc"; "SavedWebDoc");
```

Important
Subform formulas cannot be refreshed while the document is open.

Deleting subforms

It is important to make sure that a subform is not being used by any form before deleting it. When a user opens a document that has a reference to a deleted subform on the Web, the user will never get notified. The document will just open on the Web browser without any information from the deleted subform. When a designer opens a form that has a reference to a deleted subform, the message following message will appear on the status bar: "Subform: <subform name> not loaded". When a designer clicks the deleted subform area on the form, the message "Invalid or nonexistent document" appears and the designer cannot open the subform.

Tip
To avoid these messages, add another subform to the database and give it the same name as the deleted one.

Best practices for subform usage on Web applications

The subform design element is helpful when designing Web applications. We provide the following list of best practices of when and how to use subforms in order to achieve better design performance and reduce application maintenance:

- Use subforms to hold Web page header information using the $$HTMLHead.
- Use a combo of subforms and JavaScript library design elements to hold your application scripting code.
- Instead of using computed subforms to display data on forms that are both used on the Web and on Notes clients, use form hide when for the Web or Notes, or use views form formulas, when applicable.

- Use computed subforms for controlling different Web page display formats, like print pages.
- Consider using subforms to help you Developing hybrid rich client and Web client applications.

View design elements

This page last changed on Apr 04, 2008 by dalandon.

The view design element

In this section, we discuss the view design element and its proper and extended usage in our development practices. We assume that you have a basic understanding of the view design element, and focus more on how we can use this design element in our Web application development.

What is a view

The view design element is used to create both visual and functional structures for Notes Document Collections. Unlike Folder design elements, a view design element does not act as a container for Notes Documents, but rather acts as a filter or *mask* to show complete or subset Document Collections. A view design element can use a combination of selection formula, categorization, and sorting orders to provide an application with specific document collection content.

The view index

Repeated *runtime* queries of Notes Document Collections can impact the performance of our applications. View design elements, via the view index, allows for a caching of such queries, providing a less performance-impacting alternative to *runtime* queries. The view index is an internally stored indexing, storing the Notes Document Collection displayed in the given view design element. The view index consists of the view column and view entry content as displayed in the view design element.

Note
The Domino Designer Help file includes more information about the view index, including the variable refresh options.

View design elements on the Web

The view design element can be used to render Notes Document Collections by using several different methods provided to the Domino Web Developer. We discuss those methods in the following sections.

View content rendering options

The View Design Element allows for three primary rendering options: Domino-generated Views, Java Applet Views, and Treat Contents as HTML Views. In this section, we discuss the benefits and concerns of each of these rendering options.

Domino-generated views

By using Rapid Application Development (RAD), we can allow Domino to generate all of the HTML markup required to render the View Design Element UI.

Pros:

- Allows us to quickly create and maintain Web Browser Client-accessible view design elements
- Allows us to modify view design elements in the Domino Designer WYSIWYG View Design Element editor

Cons:

- Limited control over the rendered Content Type - Content Type: text/html
- Rendered markup (often) not Web Development Standards-compliant
- Rendered markup (often) generates inconsistant UI across different Web browser client programs and releases
- More advanced utilization requires *hacking*, including markup tags in column header labels, and other undesirable practices

Java applet views

The View Design Element, via preference setting, can be rendered as a Java applet when said element is viewed via a Web browser client. This rendering allows for category, column sorting, and other view design element-specific navigation facilities to be immediately presented to the browser client.

Pros:

- ...

Cons:

- Applets can be bandwidth intensive
- Applets are heavily dependent on currently configured and properly functioning Java Virtual Machine
- Applets do not facilitate more advanced *Web 2.0* functionality

⚠ **Important**
The use of Java Applet Views in Domino Web application development is a depreciated practice, in favor of Domino-generated Views and (ultimately) "Treat Contents as HTML" Views.

Treat contents as HTML views

The view design element, via preference setting, can be set to allow the view design element contents to be rendered as markup. The most application of this option is the generation of HTML markup to display table rows per documents in the View Notes Document Collection. This is our most flexible rendering option because it allows us to completely control the generated markup.

Pros:

- Complete control over generated markup
- Rendered markup can be Web Development Standards-compliant

Cons:

- Requires better understanding of the markup language that is used
- Modifications to generated markup often require combination for Formula and markup language

The *Treat Contents as HTML* preference can extend the intended usage of the view design elements when used in combination with a *ViewTemplate*, which we discuss in the following section.

Extending the view design element

The $$ViewTemplate

The $$ViewTemplate is a design element (often a Form Design Element) that is used as the rendered container for a View Design Element. The $$ViewTemplate can be used to structure rendered View Content, allowing us to expand the standard usage of View Design Elements. It can ultimately redefine how we architect our Domino Web applications.

The following table outlines the naming syntax for $$ViewTemplates.

Name syntax	Description	Examples
$$View Template for *View Design Element Name*	Act as the $$ViewTemplate for the View Design Element defined.	*$$ViewTemplate for people*
$$ViewTemplateDefault	Acts as the global $$ViewTemplate for views unless a specific $$ViewTemplate is defined.	*$$ViewTemplateDefault*

By using multi-aliasing of design elements, we can further extend the usage of $$ViewTemplates. For example, let's say that we the following five view design elements:

1. Example View 1|example1
2. Example View 2|example2
3. Example View 3|example3
4. Example View 4|example4
5. Example View 5|example5

Say our example Domino Web application required that all of the view design elements, except *Example View 2* and *Example View 4*, have a consistent look and function, while *Example View 2* and *Example View 4* have their own specific look and function. There are several different approaches that we can take to achieve this result. However the following example is considered the best practice for using $$ViewTemplates with multiple view design element requirements:

1. Alias your $$ReturnGeneralError with *$$ViewTemplateDefault*
2. Create a Form Design Element named *$$ViewTemplate for example1|$$ViewTemplate for example3|$$ViewTemplate for example5*
3. Create a Form Design Element named *$$ViewTemplate for example2|$$ViewTemplate for example4*

In the above example, we are using the design element name aliasing capabilities to create a single design element instance to handle commonly-designed $$ViewTemplates. We are additionally setting a *$$ViewTemplateDefault*, which do not render a view. This is a functional restriction measure to ensure that a user attempting to access (for example) *Example View 6|example6*, which may not be intended to be viewed from a Web browser client.

Tip
You can further simplify this example by adding an additional alias to the View Design Elements above. For example, *Example View 1|example|example1*. Then your $$ViewTemplate for the specific views could utilize this additional alias: *$$ViewTemplate for example*.

$$ViewTemplates can use two different approaches to display their view design element content:

- Embedded View Design Object
- *$$ViewBody* Field Object

Either of these objects acts as a placeholder for the view design element, and the content displayed in said placeholder can be electively controlled via Domino URL commands.

Domino URL commands and view design element content

Domino URL Commands can be utilized to affect rendered View Design Elements by controlling which subset of the Notes Document Collection is rendered. The following example shows the standard view design element URL command syntax:

```
viewname?command&attribute1=argument1&attribute2=argument2
```

The following table details several view design element-specific Domino URL commands and command attributes.

Domino URL command/attribute	Result/comments
?openview	Standard View Design Element rendering command
?searchview	Standard View Design Element searching command
?readviewentries	Rendering command which creates XML markup

	from View Design Element content
?*command*&count=n	Attribute to define the total number of entries to return from the View Entries
?*command*&start=n	Attribute to define the starting entry for the result set of View Entries
?*command*&restricttocategory=*category*	Attribute to define the entry result set to return from a categorized View Design Element
?*command*&startkey=*text*	Attribute to define the starting entry for the result set of a sorted View Design Element

Note
See All Domino URLs for more Domino URL commands, attributes, and arguments.

Additional topics

- Rapid application development
- SearchTemplate

Rapid application development

This page last changed on Apr 01, 2008 by jservais.

The Rapid Application Development (RAD) technique is a development methodology wherein the Domino Developer chooses to allow the Domino server to render all native markup elements by using Publish-to-Web enablement with minimal changes to the rendered output.

SearchTemplate

This page last changed on Apr 01, 2008 by jservais.

$$SearchTemplate

Add your content here...

For information about special reserved fields for Domino searching, refer to Searching via Domino URL commands.

Web service design elements

This page last changed on Apr 03, 2008 by jservais.

Web services and Domino

In today's environment of the soaring popularity, availability, and functionality of Web Services, it is important to consider whether you can or should make part of your application available as a web service, or, make use of an existing public Web service.

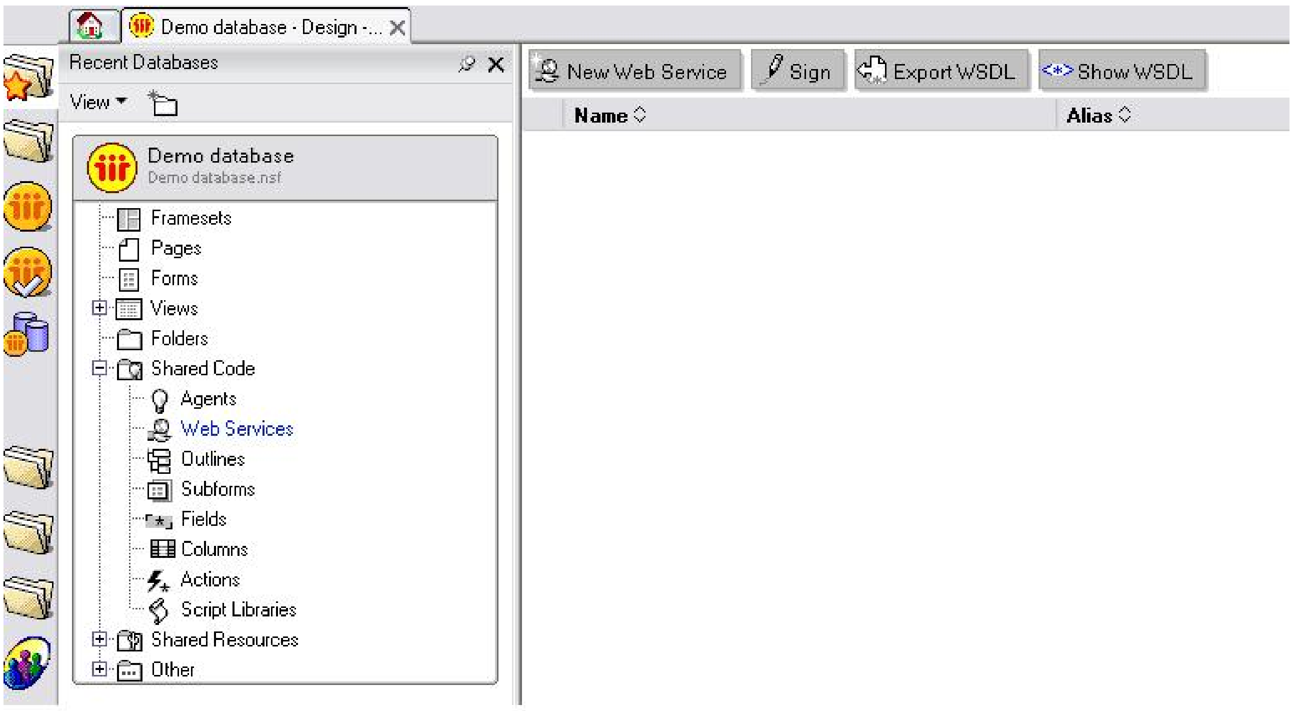
Creating a Web service in Domino

The ability to easily create a Web service was added in Domino Release 7. Included as part of Domino designer, creating a Web service involves the following basic steps:

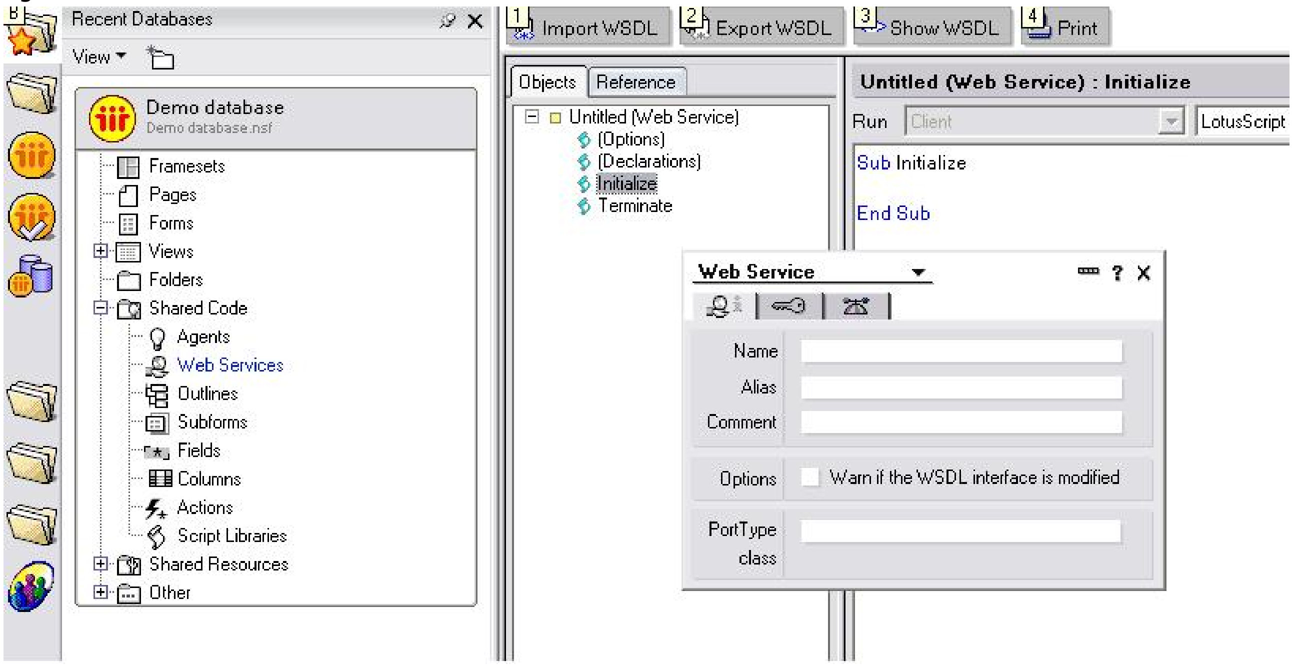
- Create the Web service and set its properties
- Code the Web service
- Export and create a WSDL if you want to make it public

Your Web service is a set of Java or LotusScript classes. In the case of LotusScript, you must use that approach and create your classes in the Declarations section.

To begin, create the Web service by clicking the **New Web Service** button in the Web Services pane as shown in the following figure.



Next, you can set the Web Service properties and then code the Web service, as shown in the following figure.



```
Sub Initialize

End Sub
```

While coding your Web service, you can also use the following actions:

- Import WSDL: If you started by creating a WSDL file from scratch, or have a WSDL file for which the entire Web service needs to be created, you can use the import function to create your code stubs.
- Export WSDL: After you create your Web service, you can use this action to export the WSDL file

and subsequently publish it.
- Show WSDL: Shows the current WSDL for the service while you are developing it.

The following table lists additional properties that you should be aware.

Property	Comments
Warn if the WSDL interface is modified	Use this! This tells the developer when a change made to the service will affect the WSDL file. After you make the Web Service publicly available, then you should not change the WSDL.
PortType class	This is the name of your Web service's class file. The service can be written as a Java or LotusScript class.
Run as web user/Run on behalf of	Equivalent to the same settings for agents.
Set runtime security level	Equivalent to the same setting for agents.
Allow Public Access users to use this web service	Equivalent to the same setting for agents.
Include operation name in SOAP action	sets the soapAction="OperationName" setting in the WSDL
Port type name	Sets the "name" attribute for <wsdl:portType> tag in the WSDL file when it is exported. This defaults to the name of the PortTypeClass specified on the Basics tab.
Service element name	Sets the "name" attribute of <wsdl:service> tag in the WSDL file when it is exported. This defaults to the name of the PortTypeClass on the basics tab followed by "Service".
Service port name	Sets the "name" attribute of <wsdl:port> tag in the WSDL file when it is exported. This defaults to "Domino".

After you code your Web service, you access it by using the following syntax, which is similar to all Domino Web URLs:

```
http://servername/databasename.nsf/webservicename?OpenWebService
```

Be aware that if you send the URL command for opening a Web service to a Domino server in an HTTP GET request (for example, by typing the command in a browser), Domino responds with the Web service's port name and the names of the operations/methods for the port. To invoke and run the Web service, an HTTP POST must be used.

To look at the WSDL, use the syntax shown in the following example:

```
http://servername/databasename.nsf/webservicename?WSDL
```

In both cases, the Web service name is the Domino design element name (or alias) of the Web service.

Consuming a Web service in Domino

Domino does not easily lend itself to consuming an external Web service. Domino Designer currently has no such tools to facilitate the the import of an external WSDL file and the code generation of its supporting functions.

However, there are two alternative suggestions, Stubby and Apache Axis.

Stubby

Stubby is an open source application that comes in the form of an NSF file. Based on Apache Axis, it is essentially a UI tool that takes as input the URL location of a WSDL file, or a WSDL file thats on your current file system, and does the following tasks:

- Stubby generate all the .java files, .class files, and a .jar file or files suitable for use and import into your notes application. You either must import the JAR file itself, or the Java files into Domino Designer (and therefore compile them) into an agent or library if you want to keep the source as a reference.

- Stubby provides a code stub for an agent that shows you how to instantiate and call the target Web service.

Some of the advantages of using Stubby is that no modifications to the client or server are needed, and there are no external library dependencies.

Stubby (as any piece of software) has been known to have some limitations. For example, its SAX parser of the Web service objects being passed back and forth could be more robust than it is. The best recommendation is to try it and see if it fills your requirements and works for you.

Apache Axis

Another alternative is to use Apache Axis. Apache Axis is essentially a downloadable package that contains Jar files and tools for generating all the files needed to call and use the target Web service. Like Stubby, the included tools take as input the physical local file location of the WSDL file of the target Web service. After you install or extract Axis, you can use a .bat file to produce your Java stubs. The batch file has to be changed to reflect the directories on your machine, and you need a Java SDK to be present.

- Apache Axis generates only the .java files, and a .jar file or files that are suitable for use and import into your notes application; However, it does not generate the compiled class files. Like Stubby, you will either need to import the jar file itself, or the Java files into Domino Designer (and therefore compile them) into an agent or library if you want to keep the source as a reference.

- Apache Axis does not provide you with a code stub to call the service from Domino.

As it turns out, the strengths and weaknesses of Stubby and Apache Axis are the opposite of each other.

One of the disadvantages of using Apache Axis is that the Jar files included in the package are external library dependencies. The Axis Jar files that were extracted onto your machine must go into a specific notes directory for external JAR files. Also, the java.policy file must exist on the Domino server (or Notes client if this is called from a Notes client) JVM and allow the proper security.

One of the advantages of using Apache Axis is that the SAX parser is more robust and can handle complex objects. There is a great writeup on using Apache Axis that is available on IBM developerWorks®.

4.0 Building Domino Web applications

This page last changed on Apr 01, 2008 by jservais.

In this section, we help you start to build Domino Web applications.

Topics in this section

Error handling

This page last changed on Apr 03, 2008 by jservais.

Introduction

When developing a Web site, you have a different way of troubleshooting problems. Unless you are a person who never writes code that could possibly fail and you have users that never do anything wrong, plan for errors. Some of the error handling is done in your code, such as field validation, and some is done by the server, such as authentication.

For troubleshooting errors, you have to resort to the server logs and some preventive coding. The server's log (log.nsf) show errors that the server detects, but it is hard to match them up. The Web server log (domlog.nsf) shows the URI that was entered and the resulting error message.

Catching coding errors

With the Notes Client, you can step through LotusScript, pop-up messages, and look at the document properties. Also the Notes Client can handle some errors like mismatched field types with just a dialog box. On the Web, this is all up to you. You can't step through the code. You can't look at the document properties. All you get is a 500 or 400 error page. You have to design the page and site with error handling in mind.

With JavaScript, you can display an alert to show what is happening and use the debuggers that are available for with Mozilla Firefox and Microsoft Internet Explorer.

LotusScript agents

You sometimes use *WebQueryOpen*, *WebQueryClose* and plain agents to process the data for the Web pages. Since you can't step through the agents by using the LotusScript debugger, you must resort to other means. Since you are working with the browser, you can't use the print statement because it sends the data to the browser. You can use a feature of Notes, the agentlog. The template for the log (alog.ntf) is on the server. You only need to create the database and use the Notes classes to write to it.

```
On Error Goto ErrorHandler

 'set to False to turn logging off
     Const LOG_ON = True
     Const LOG_VERBOSE = False
```

```
        Dim agentLog As New NotesLog(s.CurrentDatabase.Title&" - "& s.CurrentAgent.Name)
        Call agentLog.OpenNotesLog( "", "AgentLog.nsf" )
        If LOG_ON Then Call agentLog.LogAction("Agent Started" )

        '**     agent code goes here     **

        If LOG_VERBOSE Then Call agentLog.LogAction("Updated records")
        Goto TheEnd

   ErrorHandler:
        If LOG_ON then Call agentLog.LogError( Err, Error$_
            & "    Line#: "   &  Erl & |   Object: | & Lsi_info(2)  )
        Resume Next

   TheEnd:
        If LOG_ON Then
           Call agentLog.LogAction( "Agent Finished" )
           Call agentLog.Close
        end if
```

The constants, LOG_ON and LOG_VERBOSE are used to enable logging and to allow for verbose reporting that is helpful for troubleshooting.

Now you can put lines in to capture data as the agent is executing. When you done with development, set the LOG_VERBOSE to false and only the errors and agent info are captured. It's helpful to look at the log and see when an agent stopped working.

@Formula

The biggest source of errors here are mismatched field types and the dblookup/dbcolumn. For dblookup and dbcolumn, always check for an error in the return value. There is always the chance that someone deletes the keyword document or changes the key name.

```
ClassCache := "Notes":"Cache";
LookupDb := "REPID";
View := "Keywords";
Key := "DEPT";
Column := 2;
Temp := @DbLookup(ClassCache; LookupDb; View; Key; Column);
@If(@IsError(Temp); "N/A"; Temp)
```

For mismatched fields, you check the type before using it. These are normally from old data that has not been migrated correctly or an error in coding on another form or agent.

Catching server errors

Authentication, authorization, and general errors can be handled on a Web site or individual database basis. If there is error handling in the database, it's used otherwise the configuration in the Domino Web Configuration database is used for error handling. This can be set up by Web site or server. For information about the Domino Web Configuration database, see the 6.0 Server configuration section.

This handling of errors is for database usage only. Missing files on the server hard drive still cause a 404 error. You can setup an error.html pages on the server to handle these errors. This is described further in 6.0 Server configuration.

Database specific error pages

You can have forms or pages in your database handle the general errors. This way you can try to help the user correct their problem. If you build your own form, you can add a field called *MessageString*, which shows the reason for the error. By creating a form or page with a special name, you can catch the error.

Form or page name	Error
$$ReturnAuthenticationFailure	User failed to be authenticate with the server
$$ReturnAuthorizationFailure	User does not have enough access rights
$$ReturnDocumentDeleted	The document has been successfully deleted
$$ReturnGeneralError	Catch all for the rest of the errors, usually it's a design element not found

By adding a field called *MessageString* to your form, it displays the reason for the error, which could be used to give the user additional help. For example, if a user is looking for a job posting, you can tell them it's no longer there and let them search for other jobs.

For more information, see the Designer Help database. A sample of each of these forms is in the Riverbend sample database.

Input validation - Client side

This page last changed on Apr 04, 2008 by dalandon.

Why use client side input validation?

Client side user validation is used to check if data submitted by a form on a Web application is consistent with the data types expected by the Domino server for that field. You need to check, for example, if users are entering numeric data on Notes Number fields to avoid data conflict while saving this data to the Domino server database.

Attention
Client side input validation is a functional technique since it reduces the time to fill in a form and improves performance by removing round trips to server. Despite these facts, do not rely only on it while coding your application. This technique can fail for several reasons, including HTTP post failure, malicious data tampering, or the fact that clients might have JavaScript disabled on the browser. Therefore, it is important to consider using Server side user input validation as well.

Using JavaScript to validate

The simplest way to validate user input on a Domino Web application is to use JavaScript. Form input validation is on of the most common uses of the JavaScript programming. This model is fast and simple to apply, and it has the advantage of taking place on the user's browser. Therefore, it is not necessary go back to the Lotus Domino server to transmit data. If a preliminary validation of the user's inputs takes place before a form is submitted, the user does not need an answer from the server, because JavaScript validation is a instantaneous script that runs on the client browser. It does not have to transmit any data to the server. JavaScript catches any invalid data entered before it is submitted to the server. JavaScript also has the advantage of running on Lotus Notes client applications. It is interesting to consider using this language while choosing a programming language for validating form inputs.

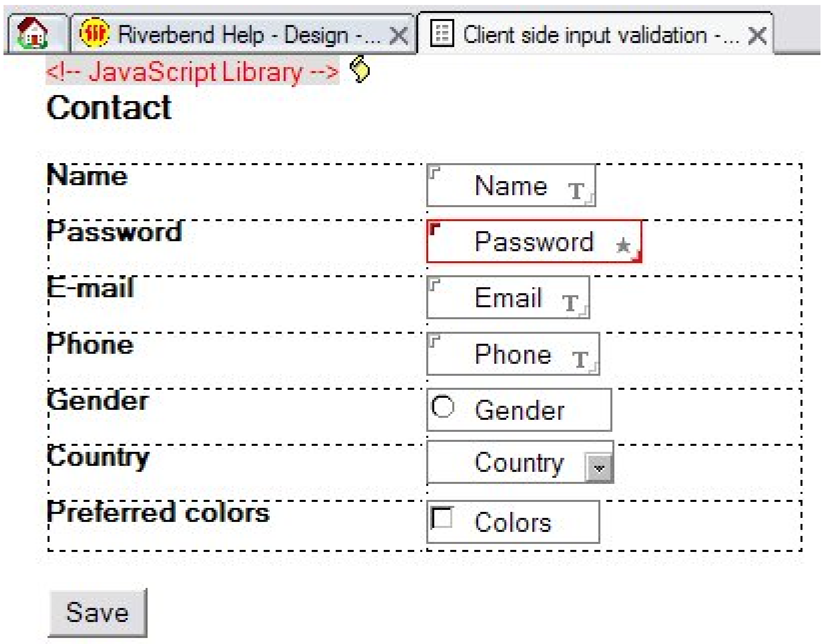
LotusScript and the Web

LotusScript *does not* run on the Web interface. To trap events on a Web environment, you need to use JavaScript events. If an application has some LotusScript events that needs to run on an application on both client and the Web, the programmer should find a way to make the best adaptation of that code to JavaScript language. JavaScript offers several user interface features available on both Notes and the Web, but a set of the LotusScript functions does not have an equivalent on JavaScript, since some of them are related to the Notes user interface. The only way to run LotusScript on the Web is to use WebQueryOpen and WebQuerySave agents in the back end. For further information about WebQueryOpen and WebQuerySave agents, refer to using WebQueryOpen and WebQuerySave agents.

Notes form validation example

There are several techniques for implementing input validations on JavaScript, from events such as onBlur, to function calls. In the following section, we use the most common and simplest technique, the onSubmit event, that runs before a page is submitted. Read through the steps in the following example to understand how to implement a client-side user input validation on a Web application.

Let us consider a Riverbend application with a form to collect a set of user information. This form is initially supposed to be available for Web users with the fields Name (Text), Password (Password), Email (Text), Phone (Text), Gender (Radio button), Country (Combobox), and Preferred colors (Checkbox).


Riverbend Help - Design -...
Client side input validation -...
<!-- JavaScript Library -->
Save

Contact

Name	Name
Password	Password
E-mail	Email
Phone	Phone
Gender	Gender
Country	Country
Preferred colors	Colors

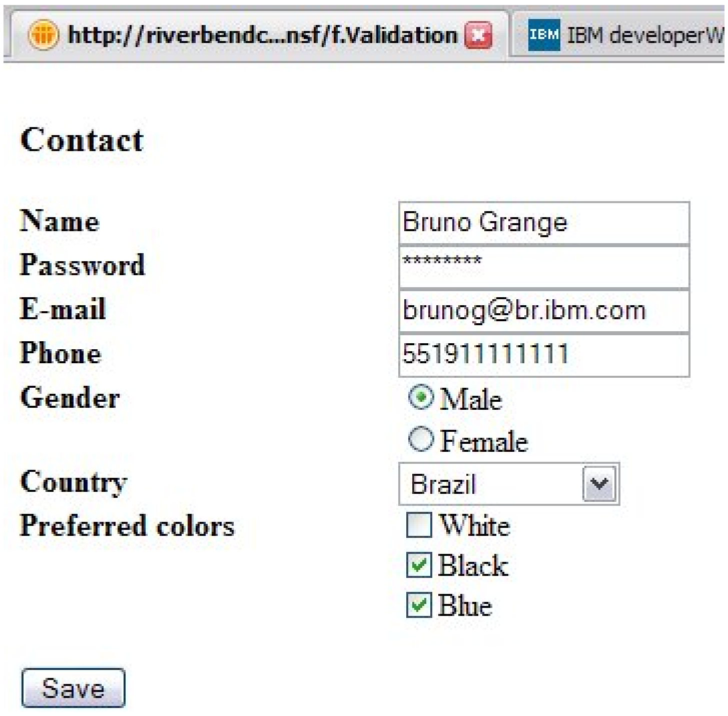
Example of the user data form on Domino Designer

This form has several data constraints that should be observed in order to populate the database with proper data from the fields. To check this, we create some JavaScript code to validate user input. We call a JavaScript function called isValidForm() on the onSubmit event of the form to check the user data input. This function returns the result from a set of validations over the user input on the form fields. To do this, include the following code on the onSubmit event of the form:

```
return isValidForm()
```

To hold the isValidForm function, we create a JavaScript library and add this script library resource to the form. In this JavaScript library, we verify whether the inputs are according to the data you expect. This function returns TRUE if data is OK, allowing the user of the onSubmit event to let the submission occur. Otherwise if data fails and returns FALSE, it prevents the document from getting saved.

The patterns that are used to validate the fields are explained as follows.



Contact

Name	Bruno Grange
Password	********
E-mail	brunog@br.ibm.com
Phone	551911111111
Gender	● Male ○ Female
Country	Brazil
Preferred colors	☐ White ✓ Black ✓ Blue

Example of the user data inserted correctly on the Web form

Creating the validation function

On the JavaScript library, we create the function, which is JavaScript function called isValidForm() as follows:

```
//starts the function
function isValidForm() {
```

The sections that follow contain pieces of code of the isValidForm() function that together make up the entire function. In these sections, we explains how to create efficient validations for several common situations.

Checking the Name field

The Name field is a text field. It should not be null. To validate this, we used the following code. Look at the code comments for an explanation of the code.

```
//sets the form to a variable
 form = document.forms[0];

        /* checks the name field */
        //sets the field object
 objField = form.Name;
        //if no data was entered on that field
 if( objField.value == "") {
                //move the cursor that field
         objField.focus();
                //displays an error message with instructions
         alert("Please enter a value for the Name field.");
                //returns a value to this function flagging this an invalid form
```

```
            return false;
           }
```

Checking the Password field

The Password field is a password type field. It must contain between 4 (minimum) and 10 (maximum) characters. It should also only be composed only of letters, numbers, and the underscore (_) sign. To validate this, we used the following code. Look at the code comments for an explanation of the code.

```
/* checks the PASSWORD field */
        //sets the field object
 objField = form.Password;
        //if name has less than 4 or more than 10 characters
 if((objField.value.length < 4) || (objField.value.length > 10)) {
                //move the cursor that field
         objField.focus();
                //displays an error message with instructions
         alert("Please enter at least 4 and at most 10 characters for the Password field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
        //sets illegal characters from a regular expression
 var expillegalChars = /\W/;
        //if name has illegal characters
 if(expillegalChars.test(objField.value)) {
                //move the cursor that field
         objField.focus();
                //displays an error message with instructions
         alert("Please use only letters, numbers or the underscore sign for the Password
field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
```

Checking the E-mail field

The e-mail field is a text field. It should be composed of some characters, followed by an at symbol (@), followed by some more characters, then a dot (.), and then two or three more characters. To validate this, we use the following code. Look at the code comments for an explanation of the code.

```
/* checks the EMAIL field */
        //sets the field object
 objField = form.Email;
        //sets illegal characters from a regular expression
 var expillegalChars = /^.+@.+\..{2,3}$/
        //if name has illegal characters
 if(!expillegalChars.test(objField.value)) {
                //move the cursor that field
         objField.focus();
                //displays an error message with instructions
         alert("Please enter a valid e-mail address for the E-mail field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
```

Checking the Phone field

The Phone field is a text field. It can contain numbers and characters such as parentheses, dashes (-), spaces ( ), and dots (.). To validate this, we used the following code. Look at the code comments for an explanation of the code.

```
/* checks the PHONE field */
        //sets the field object
 objField = form.Phone;
        //sets illegal characters from a regular expression
 var strStripped = objField.value.replace(/[\(\)\.\-\ ]/g, '');
        //if name has illegal characters
 if (isNaN(parseInt(strStripped))) {
                //move the cursor that field
         objField.focus();
                //displays an error message with instructions
         alert("Please enter a phone number with valid characters for the Phone field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
```

Checking the Gender field

The Gender field is a radio button. It should have at least one of the options selected. To validate this, we used the following code. Look at the code comments for an explanation of the code.

```
/* checks the GENDER field */
        //sets the field object
 objField = form.Gender;
        //sets a boolean before the loop
 bolFoundValue = false;
        //goes thru each value
 for (i=0; i<objField.length; i++) {
                //verifies if it was checked
         if (objField[i].checked) {
                        //if checked, stops the loop
                 bolFoundValue = true;
                        break;
                }
        }
        //checks if a value was entered
 if (!bolFoundValue) {
                //move the cursor that field
         objField[0].focus();
                //displays an error message with instructions
         alert("Please select a value for the Gender field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
```

Checking the Country field

The Country field is a combo box. It should not have the first index option (Select one...) selected. To validate this, we used the following code. Look at the code comments for an explanation of the code.

```
/* checks the COUNTRY field */
        //sets the field object
 objField = form.Country;
        //sets a boolean before the loop
 var bolFoundValue = false;
        //checks if a option selected is the first one
 if (objField.selectedIndex == 0) {
                //move the cursor that field
         objField.focus();
                //displays an error message with instructions
         alert("Please select a value for the Country field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
```

Checking the Preferred colors field

The Preferred colors field is a check box. It should have at least two values selected. To validate this, we used the following code. Look at the code comments for an explanation of the code.

```
/* checks the COLORS field */
        //sets the field object
 objField = form.Colors;
        //sets a integer before the loop
 intCountValues = 0;
        //goes thru each value
 for (i=0; i<objField.length; i++) {
                //verifies if it was checked
         if (objField[i].checked) {
                        //if checked, adds a value
                 intCountValues++;
                }
        }
        //checks if a value was entered
 if (intCountValues < 2) {
                //move the cursor that field
         objField[0].focus();
                //displays an error message with instructions
         alert("Please select at least two values for the Preferred colors field.");
                //returns a value to this function flagging this an invalid form
         return false;
        }
```

Closing the function and returning a value

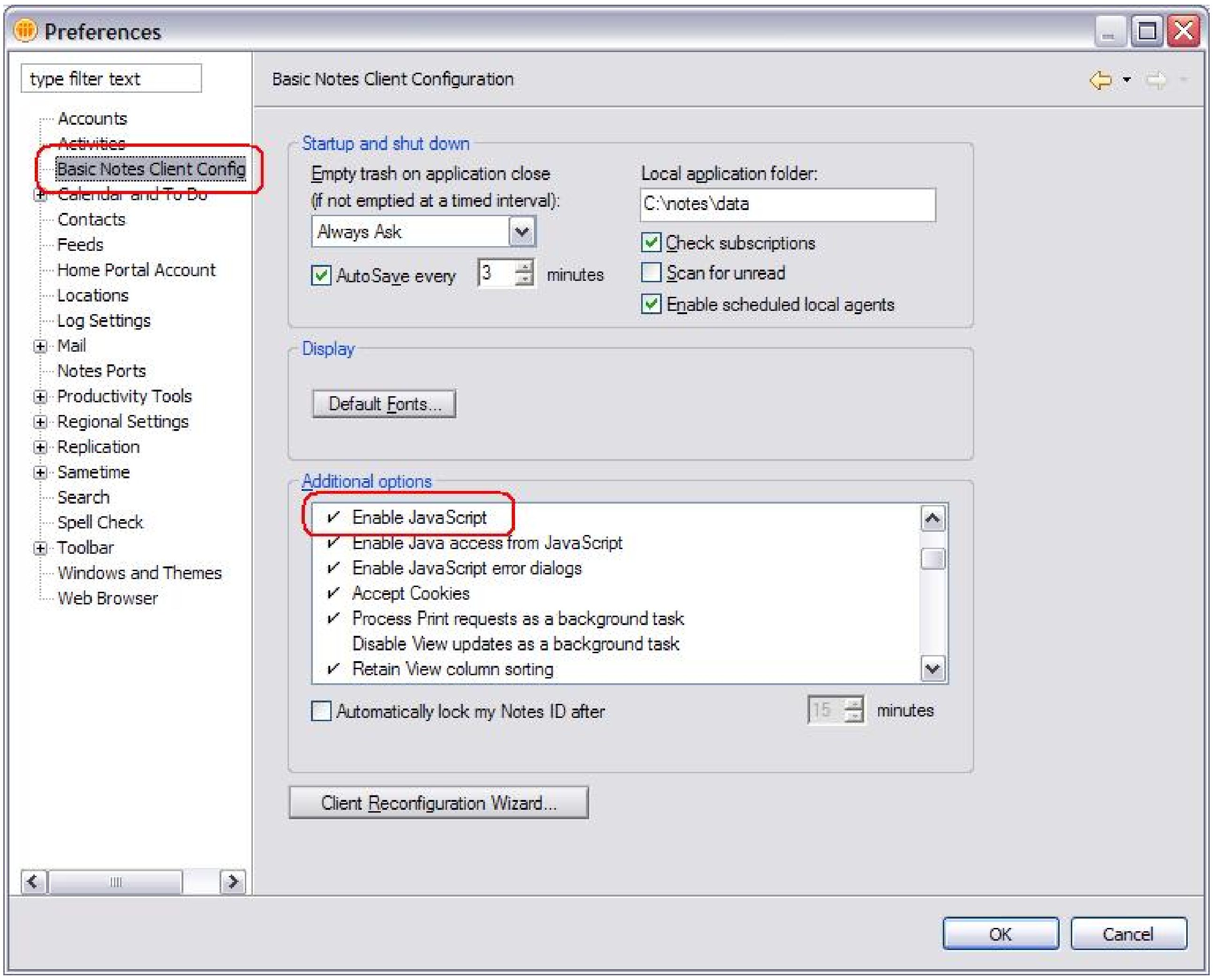
If the whole validation was done and no issues were found regarding the data, we can return true to this function to allow the onSubmit event to run.

```
//if it passed everything, the fields are ok
 return true;
//ends the function
}
```

This is just an example of a code validation. You can improve this code and make it suitable for your application based on your needs.

Reducing maintenance by using hybrid validation

Since JavaScript is also available on the Notes client interface, you may want to reduce the application maintenance by having a hybrid client side input validation by using the same JavaScript code on both the Web and Notes client. JavaScript comes enabled by default on the Notes configuration, but it is good to make sure it is enabled on the users Lotus Notes workstations. To check this, on the Notes client, click **File > Preferences**. (For Macintosh OS X users, click **Lotus Notes > Preferences**.) Click **Basic Notes Client Configuration**. Under Additional options, select **Enable JavaScript**.



Basic Notes Client Configuration

Startup and shut down

Empty trash on application close (if not emptied at a timed interval): Always Ask

Local application folder: C:\notes\data

[✓] Auto Save every 3 minutes

[✓] Check subscriptions

[ ] Scan for unread

[✓] Enable scheduled local agents

Display

Additional options

✓ Enable JavaScript
✓ Enable Java access from JavaScript
✓ Enable JavaScript error dialogs
✓ Accept Cookies
✓ Process Print requests as a background task
Disable View updates as a background task
✓ Retain View column sorting

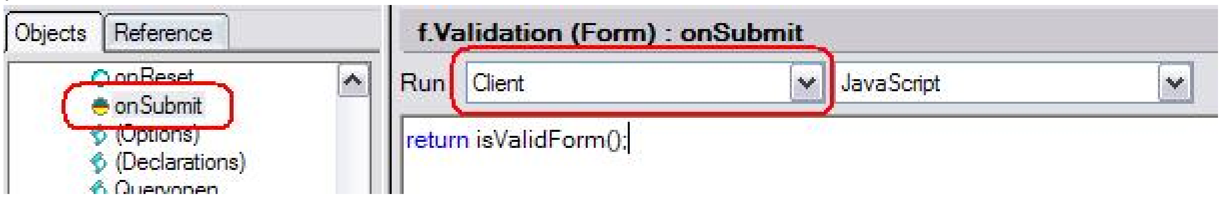
[ ] Automatically lock my Notes ID after 15 minutes

Enabling JavaScript on Notes

To do this validation on Lotus Notes, go to the form design and on the onSubmit event, enter the same code described above for the validating the form on the Web.

```
return isValidForm()
```

Now remember to set the event to run on Notes Client using JavaScript language, as shown on the picture below.



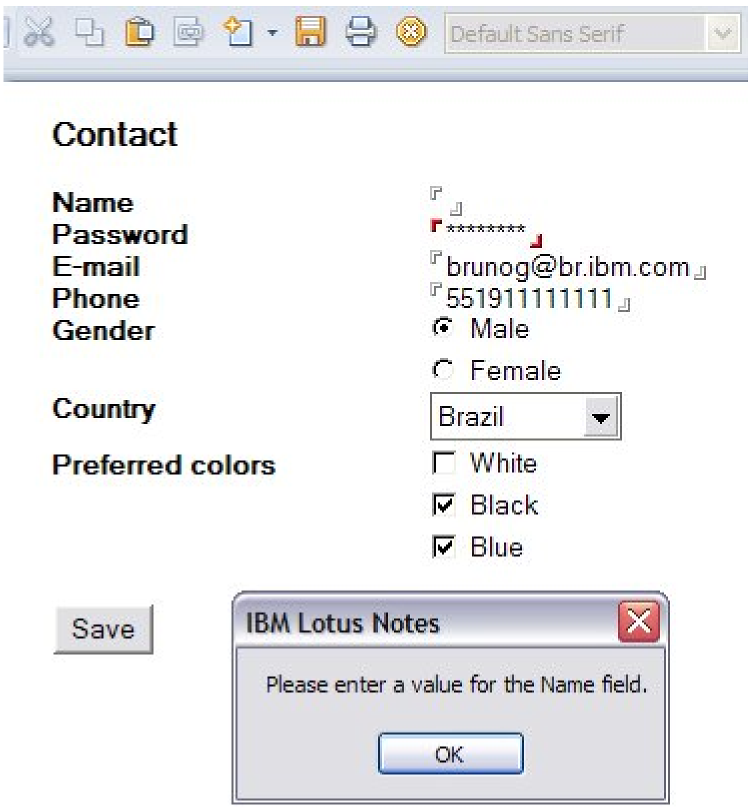
f.Validation (Form) : onSubmit

Run Client JavaScript

```
return isValidForm();
```

The onSubmit event code on selected to run on the Notes Client using JavaScript language

Since the isValidForm() function code is hosted in a script library (as described above), it can be accessed by the form on both Notes and the Web. The following figure shows the validation of the Name field occurring on Notes client.



Contact

Name	
Password	********
E-mail	brunog@br.ibm.com
Phone	551911111111
Gender	[✓ Male] Female
Country	Brazil
Preferred colors	White [✓ Black] [✓ Blue]

Save

IBM Lotus Notes

Please enter a value for the Name field.

OK

Example of the user data form on Lotus Notes client

Tip
Not all the functions that are available on JavaScript on the Web are available on the Notes interface. Consider using basic JavaScript routines on your code validation to ensure that your code can run on both environments.

The examples listed above are just a sample of how you can validate your code by using JavaScript. You should always keep in mind that JavaScript *may not suit* all your validations. In some cases, you may need to use server side validation or combo hybrid client side validations with LotusScript. Refer to Defining functional requirements based on client type for information additional resources about defying these requirements.

Input validation - Server side

This page last changed on Apr 02, 2008 by jservais.

Introduction

When working with Lotus Notes forms on Web applications, we need to make sure that data being received on the server is appropriate for its back-end computations. Depending on the level of security required by an application, a programmer may be required to validate at server level if the data submitted by a form is reliable. You may have to do small validations, just like checking if we have loss of user-submitted field data, to complex validations like data types checking, characters counting, attachment lengths and even HTTP post headers tampering.

The most common way to avoid data loss is to retrieve the values to be validated by using the hidden form fields, as well as client-side validations by using JavaScript. When there is a lot of data to be validated in a form, this can cause generated HTML pages to be very heavy, impacting the user and server performance. Additionally, it's almost impossible when several fields need to be validated against different conditions. A simpler solution is available through the use of conventional coding.

Validating by using field input validation

A simple way to validate a field is to use the input validation properties of the field. @Success and @Failure work in field input validation formulas on the Web. The @Failure path causes the message specified as the parameter to appear on a new page. In the following validation formula, if the user fails to enter a value for Name, the word "Error" in bold is displayed on a new page:

```
@If(Name = ""; @Failure("<strong>Error<\strong>"); @Success);
```

You can make the failure page more meaningful by using more extensive HTML in the error message as shown in the following example:

```
msg1 := "Please enter a value for the Name field.<br /><br >";
msg2 := "<a href=/" + @WebDbName + "/Contact?OpenForm>";
msg3 := "Click here</a> to try again.";
msg := msg1 + msg2 + msg3;
@If(@ThisValue = ""; @Failure(msg); @Success)
```

The problem with this is that you need to treat the validation entering the Input validation code field by field on the form. This practice increases the maintenance effort because the error messages gets tied to each field, while it could be more simplified using a centralized script.

Validating on WebQuerySave agents

Another practice for server side user input validation of a Web form on a Lotus Domino server is to call an agent on the WebQuerySave event of the form. This agent does all of the validations and redirects users to an error page, or prints alerts and returns to a blank form.

Generally, the WebQuerySave event is called by using the JavaScript submit command or by using the more direct "*@Command([FileSave])*". To maintain the data entered by the user, follow these steps:

1. On your WebQuerySave agent, make sure the first statement you make is to set "SaveOptions" to 0.
2. After data is submitted, validate it by using a combination of LotusScript and JavaScript, as shown on the following example:

```
Sub Initialize

        On Error Goto CommomErrHandler

        Dim session As New NotesSession
        Dim doc As NotesDocument
        Dim ndt As NotesDateTime
        Dim arrFileSizes As Variant

        'sets the current submitted document
        Set doc = session.DocumentContext

        'prevents the document from getting saved at this point
        doc.SaveOptions = "0"

                'validating a text field
        If doc.TextField(0) = "" Then
                Print |<script language="JavaScript1.2" type="text/javaScript">|
                Print |history.back();|
                Print |alert("Please enter a value for the text field");</script>|
                Exit Sub
        End If

        'validating a numeric field
        If Not Isnumeric(doc.NumberField(0)) Then
                Print |<script language="JavaScript1.2" type="text/javaScript">|
                Print |history.back();|
                Print |alert("Please enter a numeric value for the number field");</script>|
                Exit Sub
        End If

        'validates a date field
        Set ndt = New NotesDateTime( doc.DateField(0) )
        If (doc.DateField(0) = "") Or (Not ndt.IsValidDate) Then
                Print |<script language="JavaScript1.2" type="text/javaScript">|
                Print |history.back();|
                Print |alert("Please enter a date value for the date field");</script>|
                Exit Sub
        End If

        'lets the document get saved
        doc.SaveOptions = "1"
        Call doc.Save( True, False )

        'Prints a success form
        Print "[" + strDummy + "/f.Success?OpenForm&DocID=" + doc.UniversalID + "&Title=" +
doc.TitleField(0) + "]"

        Exit Sub
```

```
'treats commom errors
CommomErrHandler:
        Print "Got error on (WebQuerySaveAgent) #" + Cstr(Err) + " - " + Error$ + " on line " +
Cstr(Erl)
        Msgbox "Got error on (WebQuerySaveAgent) #" + Cstr(Err) + " - " + Error$ + " on line "
+ Cstr(Erl)
        Exit Sub
End Sub
```

There is another process to maintain data entered by a user, which we describe as follows:

1. Add one computed for display field, for example *ErrMsg*. Use the same formula you used for the Field name.
2. Add a hidden button (style='display:none") and call the agent on this button. You can call this button "btnValidate." Use the formula: "*@Command([ToolsRunMacro]; "agentName")* for calling the agent. Remove your agent call from WebQuerySave.
3. Modify the agent. Then after you set up the document, complete these tasks:
    a. Make sure the first statement you make is to set "SaveOptions" to 0 and "ErrMsg" to "" (blank).
    b. Remove any "print" statements.
    c. Make sure the messages to be displayed to the user are now set in the field "ErrMsg" (using the document handle).
    d. Remove any redirection in case of error.
    e. If no error occurs, then simply set "SaveOptions" to 1.
4. In the "Onload" event of the form, write some JavaScript to check the "ErrMsg" field and to make sure the alert value of the field is not blank. This gives an error message to the Lotus Notes user on form validation failure.
5. Make that field blank using JavaScript so that it does not alert on every page refresh after that.
6. On "btnValidate," where we call the agent, add the following formula to ensure that the form is submitted if the validations are passed and we set "SaveOptions" to 1 in our agent.

```
@If(SaveOptions="1"; @Command([FileSave]); @Return(""));
```

7. Instead of calling "submit" on the form's Save button, call "click" of "btnValidate" by using JavaScript.

Our agent is called on the Save button. Since the form is not submitted yet, this is where we have the document context live. We don't have any print statements. Therefore, it won't overwrite the document context. If validations have failed, we have messages set in our "ErrMsg" field. After the agent finishes page refreshes and causes alerts to display from the "Onload" event and all user entered data is there, there is no loss at all. If no error shows, then the next formula agent call, which checks for the "SaveOptions" value, saves the form that submits the form for saving.

Checking the attachment sizes of the submitted documents

When a Domino Web form has file upload controls to allow users to upload file attachments to their documents, you may want to limit the size of these attachments to avoid server performance issues. The easiest way to implement this kind of input validation on a Web form is to use a WebQuerySave agent to check the size of the attachments that are created on a form. A simple way to achieve this is to use the code shown in the following example:

```
Sub Initialize

        Dim session As New NotesSession
        Dim doc As NotesDocument
        Dim arrFileSizes As Variant

        'sets the current submitted document
        Set doc = session.DocumentContext

        'prevents the document from getting saved at this point
        doc.SaveOptions = "0"

        'verifies if the attachment sizes is bigger then 10MB
        arrFileSizes = Evaluate("@AttachmentLengths", doc)
        If Implode(arrFileSizes) <> "" Then
                Forall v In arrFileSizes
                        If v > (10485760) Then '10MB represented in bytes
                                doc.SaveOptions = "0"
                                Print |<script language="JavaScript1.2"
type="text/javaScript">|
                                Print |history.back();|
                                Print |alert("You can not have any attachment bigger then
10MB");</script>|
                                Exit Sub
                        End If
                End Forall
        End If

        'lets the document get saved
        doc.SaveOptions = "1"
        Call doc.Save( True, False )

End Sub
```

Interactive data (Web 2.0)

This page last changed on Mar 27, 2008 by heinsje.

Share your best practices here.

Login screens

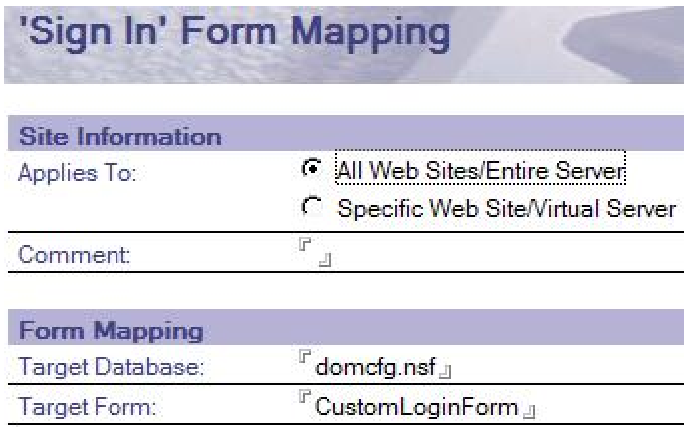
This page last changed on Apr 03, 2008 by jservais.

- Custom login screens
- Related information

Custom login screens

Custom login screens can be created for a server, so that we can have our Riverbend Coffee and Tea Company logo on the page and make other modifications if we like.

We create the custom login form in any database and then make sure the Web server configuration database is present on the server. Within the Web server configuration database, we then create the Login Form Mapping to tell the server how to find our custom login form. The Login Form mapping also tells the server when to use this custom form and lets us create multiple custom login forms if we need them for different host names.



'Sign In' Form Mapping

Site Information

Applies To:	(●) All Web Sites/Entire Server ( ) Specific Web Site/Virtual Server
Comment:	

Form Mapping

Target Database:	domcfg.nsf
Target Form:	CustomLoginForm

Related information

Web server configuration database

- Built-in forms using $$LoginUserForm
- Custom login screens using Domcfg.nsf
- Database that came with Domino R5

Built-in forms using $$LoginUserForm

This page last changed on Mar 27, 2008 by heinsje.

Share your best practices here.

Custom login screens using Domcfg.nsf

This page last changed on Apr 02, 2008 by jservais.

Share your best practices here.

Database that came with Domino R5

This page last changed on Apr 04, 2008 by dalandon.

Share your best practices here.

Navigation techniques

This page last changed on Apr 02, 2008 by jservais.

When you plan for the Web, you have to think about navigation. Users don't have a workspace to select the database they want. You must provide site-wide navigation that is consistent. Don't use outlines in one database and links in another. Some of the navigation techniques used in Notes can be used on the Web. For an application that resides in one database, you can use the outline but then changes must be done by a developer, which may be OK. The Web offers additional navigation possibilities such as horizontal and vertical menus that can have submenus. There are menus already designed and available for free and purchase on the Internet.

In this section, we explore different ways to layout the site and pages, from a view-based menu to frameless pages and ways to use embedded views.

- Moving past the frameset - Making Web-based applications that do not look like Lotus Notes
- No more twisties - Using single category and a combobox to filter the view
- View-based menus

Moving past the frameset - Making Web-based applications that do not look like Lotus Notes

This page last changed on Apr 02, 2008 by jservais.

In the Notes client, the frameset is used to set up areas for specific purposes. You have a Navigation pane on the left with a view area next to it and possibly a preview pane. It works really nicely. On the Web, it tends to have more problems then benefits. It's hard for users to print because they have to be in the frame to print it. It makes development easier since you have to make only one navigation page. It's also hard on screen readers for those with vision problems.

Since it's easy to simulate, it's slowly going the way of dial-up. It is used just where it's the only way to do it.

You can use the following ways to build pages that have the look of framesets:

- Use tables unless you are building a tabular list. The use of tables for layout is not acceptable.
- Use one form that gets a key as a URL parameter and pulls the data that is needed, so that the page doesn't change.
- Use HTML div tags and cascading style sheets (CSS) to give your page structure. By using CSS, you can adjust the pages easily and edit them live to see the changes. In addition, you have the ability to separate the content from the layout.

If you are planning for this Web site to be usable for years, easily maintained, and looking current, then plan on learning CSS. Like JavaScript libraries, the Internet provides a large number of predesigned frameworks.

Here we look at a site layout that uses a framework called Blueprint. Blueprint is a CSS framework that is laid out in a grid format to make working with similar to using tables. You set up the page with HTML div tags. Div tags let you set up the width and height using CSS, so that it looks like a framset. See the Styles and CSS primer section for more information about CSS. When you use a frameset, you define it once. With CSS, you must include style code on each form. This can be done with subforms, includes or copying the code.

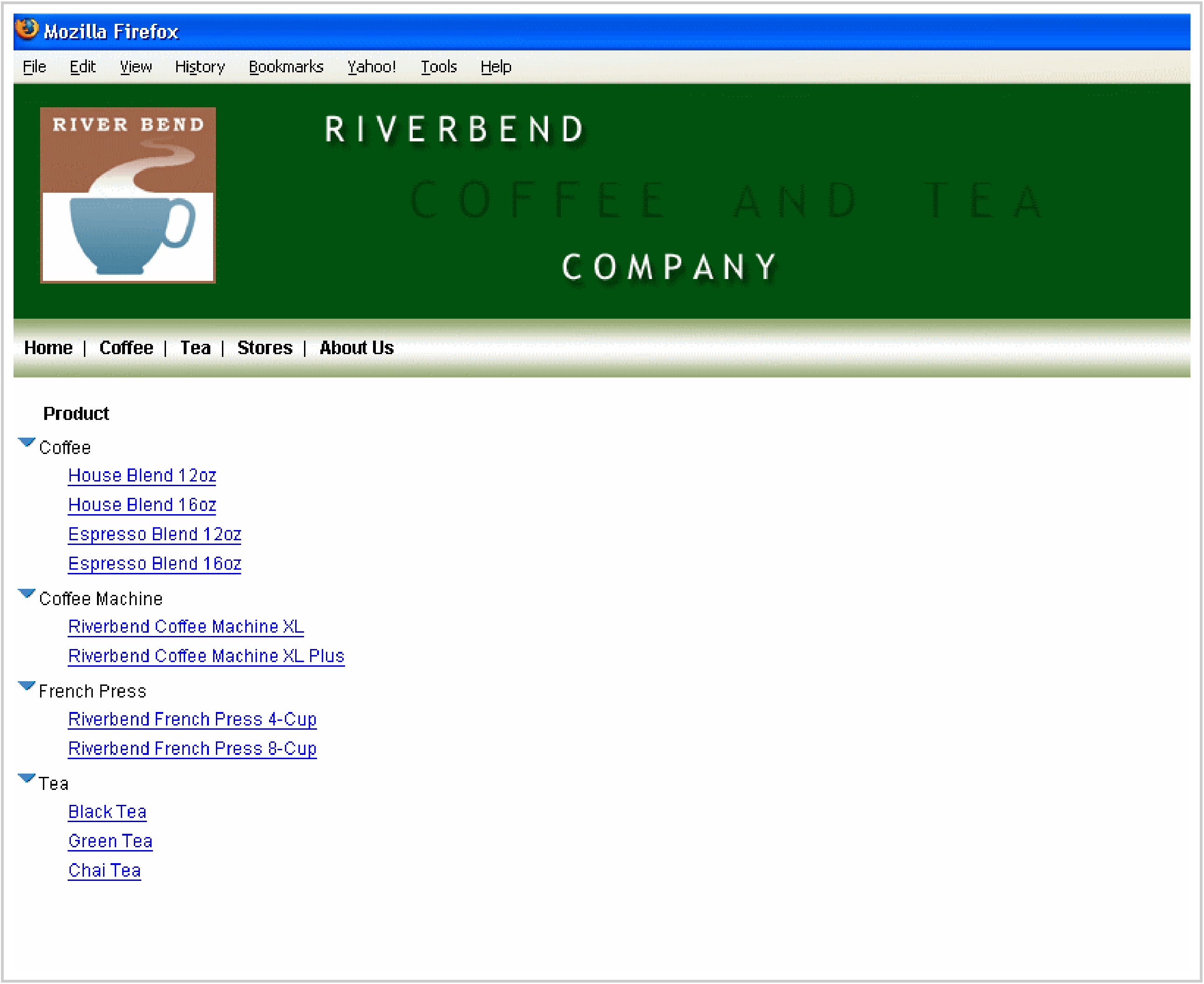
In the Riverbend sample database, there is a sample page that has a Header, Navigator, Content and footer defined.

No more twisties - Using single category and a combobox to filter the view

This page last changed on Apr 02, 2008 by jservais.

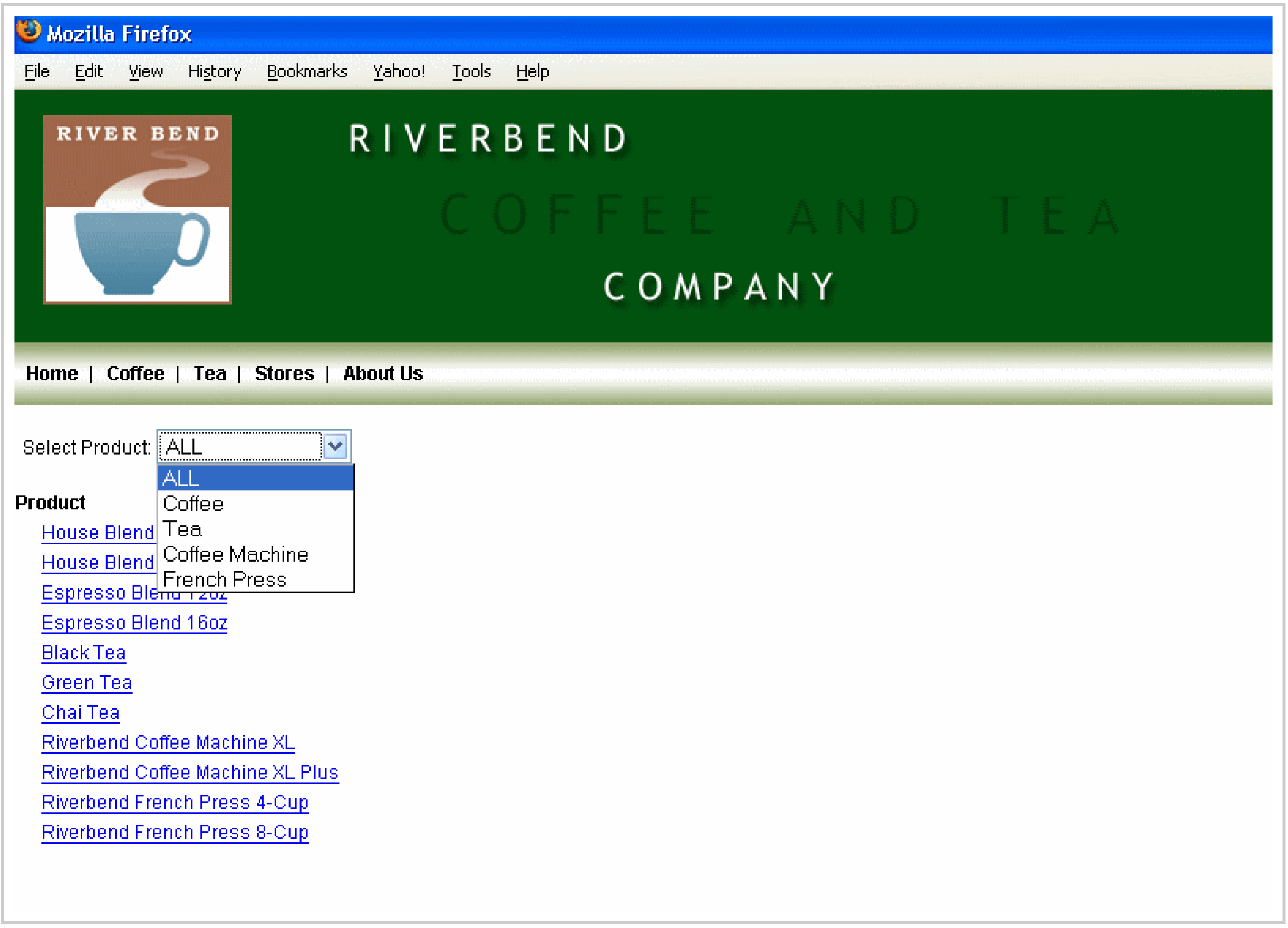
Everyone knows that categorized views are great when developing a notes application. Users can click the twistie to expand or collapse a category and find the documents they are looking for with ease. On the Web, it is difficult to navigate through a categorized view, especially if you have many categories. It also makes your Web application look too much like a Lotus Notes application that is Web-enabled from the start.

The following figure shows a categorized view for products that the Riverbend Coffee and Tea Company sells.


Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
RIVER BEND
RIVERBEND
COFFEE AND TEA
COMPANY
Home | Coffee | Tea | Stores | About Us

Product
- Coffee
  - House Blend 12oz
  - House Blend 16oz
  - Espresso Blend 12oz
  - Espresso Blend 16oz
- Coffee Machine
  - Riverbend Coffee Machine XL
  - Riverbend Coffee Machine XL Plus
- French Press
  - Riverbend French Press 4-Cup
  - Riverbend French Press 8-Cup
- Tea
  - Black Tea
  - Green Tea
  - Chai Tea

This is what we are trying to achieve. Users select a product type from the drop-down box and they see only the products that belong to the selected type.


Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
RIVER BEND
RIVERBEND
COFFEE AND TEA
COMPANY
Home | Coffee | Tea | Stores | About Us

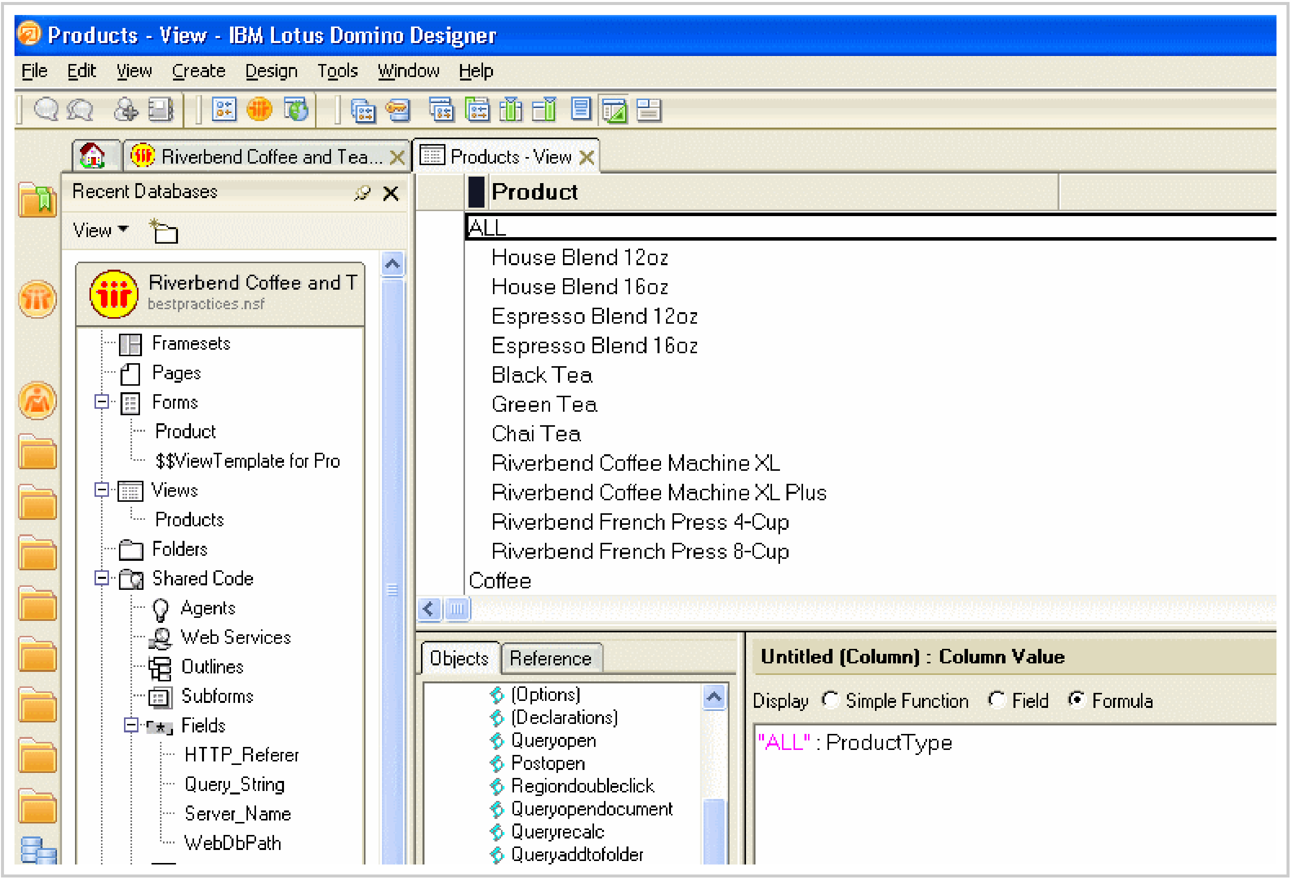
Select Product: ALL

- ALL
- Coffee
- Tea
- Coffee Machine
- French Press

Product

- House Blend
- House Blend
- Espresso Blend 12oz
- Espresso Blend 16oz
- Black Tea
- Green Tea
- Chai Tea
- Riverbend Coffee Machine XL
- Riverbend Coffee Machine XL Plus
- Riverbend French Press 4-Cup
- Riverbend French Press 8-Cup

First, we must change the Products view. We need to add "ALL" to the first column, so that users can see all products when they select "ALL".



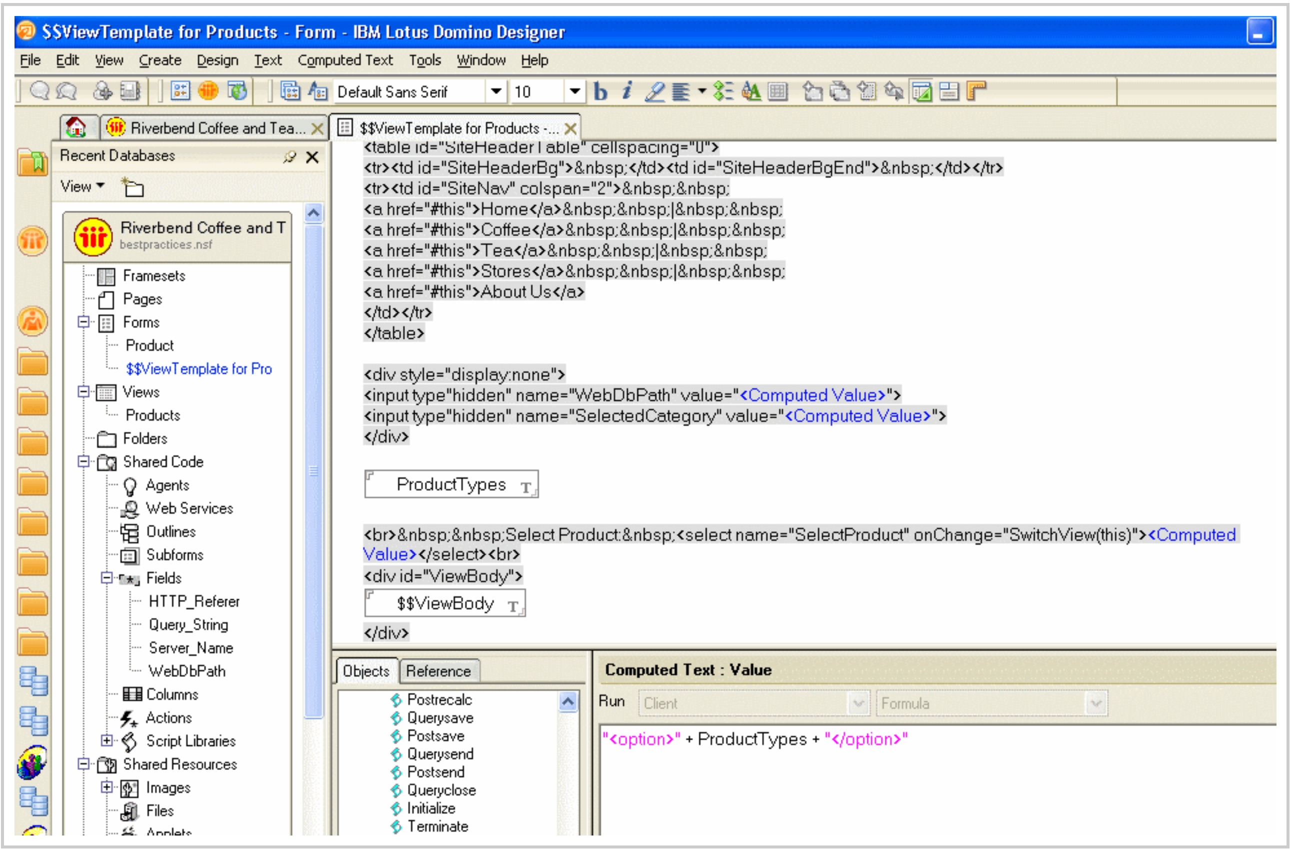
```
Product
ALL
    House Blend 12oz
    House Blend 16oz
    Espresso Blend 12oz
    Espresso Blend 16oz
    Black Tea
    Green Tea
    Chai Tea
    Riverbend Coffee Machine XL
    Riverbend Coffee Machine XL Plus
    Riverbend French Press 4-Cup
    Riverbend French Press 8-Cup
Coffee
```

Untitled (Column) : Column Value

Display ( ) Simple Function ( ) Field [✓ Formula]

```
"ALL" : ProductType
```

Next, we must add the drop-down box to the "$$ViewTemplate for Products" form.




```
<table id="SiteHeaderTable" cellspacing="0">
<tr><td id="SiteHeaderBg">&nbsp;</td><td id="SiteHeaderBgEnd">&nbsp;</td></tr>
<tr><td id="SiteNav" colspan="2">&nbsp;&nbsp;
<a href="#this">Home</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this">Coffee</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this">Tea</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this">Stores</a>&nbsp;&nbsp;|&nbsp;&nbsp;
<a href="#this">About Us</a>
</td></tr>
</table>

<div style="display:none">
<input type="hidden" name="WebDbPath" value="<Computed Value>">
<input type="hidden" name="SelectedCategory" value="<Computed Value>">
</div>

ProductTypes T

<br>&nbsp;&nbsp;Select Product:&nbsp;<select name="SelectProduct" onChange="SwitchView(this)"><Computed
Value></select><br>
<div id="ViewBody">
$$ViewBody T
</div>
```

Computed Text : Value

Run Client Formula

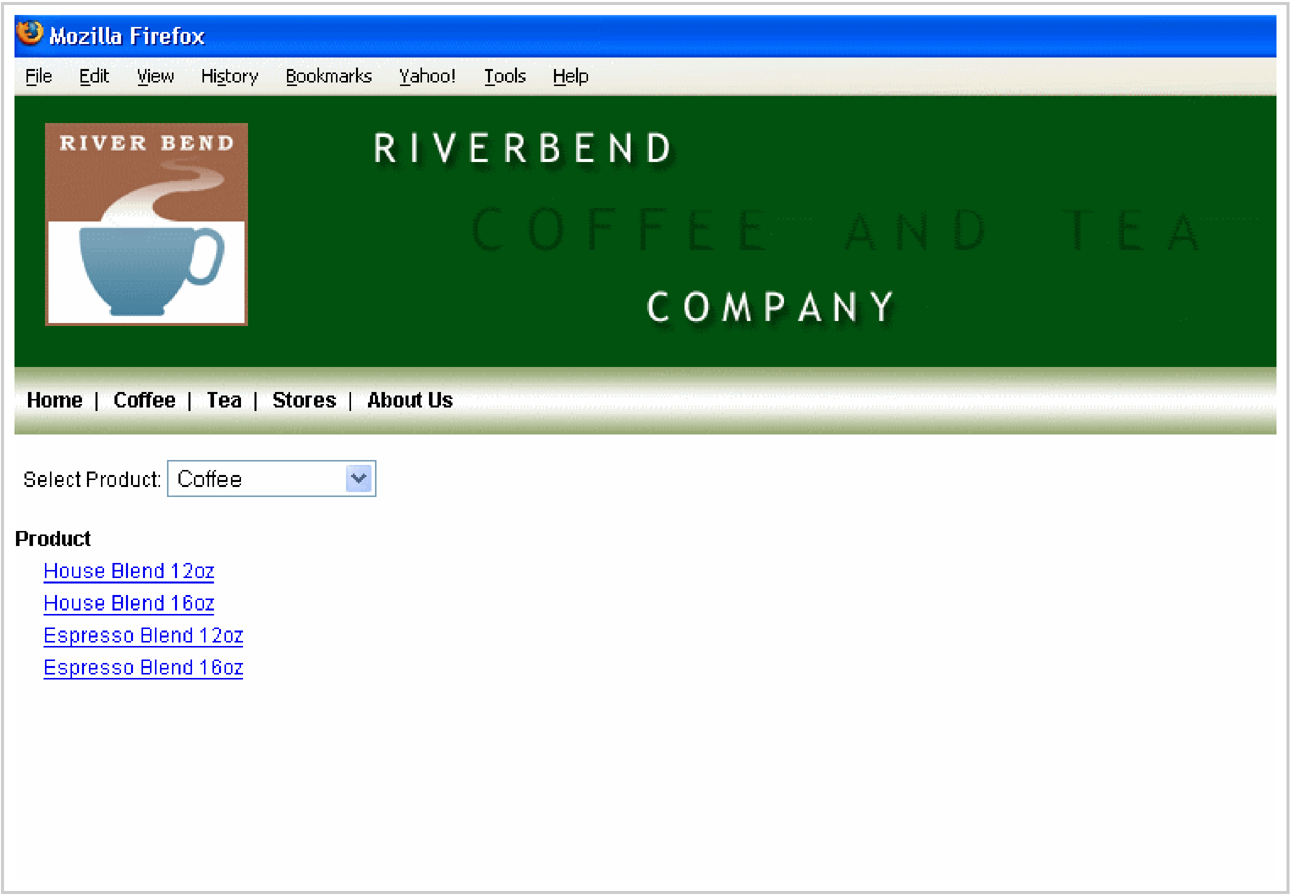
```
"<option>" + ProductTypes + "</option>"
```

In the previous example, we used pass-thru HTML to generate the drop-down box because a Notes field is not displayed when you open a view through the ?OpenView method. Ccomputed text is used to get the drop-down box values from a field or you can use @DbColumn to get the values from the categorized view.

The drop-down box calls the SwitchView function to switch the view with the RestricToCategory URL. The function uses the WebDbPath html field on the form to get the database path. It has the @WebDbName formula in it.

```
function SwitchView ( selectField ) {
        f = document.forms[0];

        selection = selectField.options[selectField.options.selectedIndex].text;

        location.href = "/" + f.WebDbPath.value + "/Products?OpenView&RestrictToCategory=" +
selection;
}
```

In addition, we want to have our selection to be chosen when switching to a new category.


Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
RIVER BEND
RIVERBEND
COFFEE AND TEA
COMPANY
Home | Coffee | Tea | Stores | About Us

Select Product: Coffee

Product

House Blend 12oz
House Blend 16oz
Espresso Blend 12oz
Espresso Blend 16oz

By adding the following code in the onLoad event of the "$$ViewTemplate for Products" form, the new category is selected. The function uses the SelectedCategory html field on the form that has the @URLQueryString ("RestrictToCategory") formula.

```
function SelectDropDown ( DropDownField ) {
        f = document.forms[0];
        if ( f.SelectedCategory.value != "" ) {
                for ( i = 0; i < DropDownField.options.length; i++ ) {
                        if ( DropDownField.options[i].value == f.SelectedCategory.value ){
                                DropDownField.options[i].selected = true
                        }
                }
        }
}
```

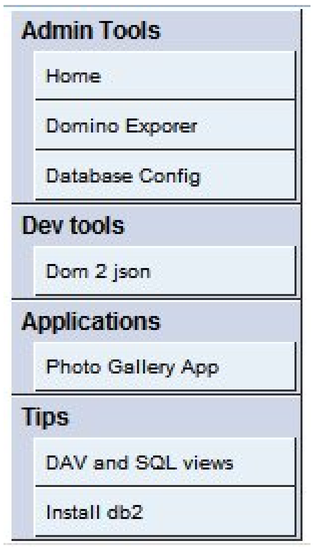
Tip

You can also use a combination of Ajax and ?ReadViewEntries to display the view. This results in a quicker response time because the page does not refresh when you switch category.

View-based menus

This page last changed on Apr 02, 2008 by jservais.

You can use a view to build a menu for the Web. This way, as new content is added, the menu is automatically updated or you can use it to build the site or navigation for the applications. By taking the area titles and the URL to them, using the HTML unordered list <ul> tags and CSS, you have a menu. If you don't have categories, you can use an embedded view that is set to HTML and put the tags in the view.



View menu on the Web

Navigation area

The following code can be placed on the form or page, where you want the navigator to be displayed:

```
<div id="navcontainer">
                        <ul>
                        <Computed Value>
                        </ul>
                        </li>
                        </ul>
                </div> <!-- ends #navcontainer-->
```

The computed text element that is shown as <Computed Value> in the previous code example contains the following formula. The formula pulls a column from a view called (nav). It formats the returned values into an HTML list with the li tags.

```
ClassCache := "Notes":"NoCache";
Db :="":"";
View:= "(nav)";
Temp := @DbColumn(ClassCache; Db; View; 3);
```

```
Navlist := @If(@IsError(Temp);"";temp);

cat := @Left(@Subset(Navlist;1);"~");
nav := "<li class=\"navlist\">"+cat+"<ul>";
cnt := @Elements(Navlist);

@For(n := 1; n <= cnt ; n := n + 1;
        tcat := @Left(@Subset(NavList[n];1);"~");
        nav := nav + @If(cat = tcat;
                     @Right(NavList[n];"~");
                    "</ul></li>" + @NewLine + "<li class=\"navlist\">"
                     + tcat + "<ul>" + @Right(NavList[n];"~"));
        cat := tcat
);
nav
```

View

The Heading field is in a format to make sorting and grouping easy. The field contains a value like "1.heading title", where the 1 ensures the correct grouping and sort order.
The sort field sorts the entries in each group.

```
col1:@TextToNumber(@Left(Heading1;"."))
col2:sort
col3 @Right(Heading1;".")+"~" +"<li>"+pURL+"</li>"
```

The following example shows the generated HTML code:

```
<div id="navcontainer">
        <ul>
                <li class="navlist">Admin Tools
                        <ul>
                                <li><a href="/homesite.nsf">Home</a></li>
                                <li><a href="/explorer.nsf">Domino Exporer</a></li>
                                <li><a
href="/homesite.nsf/all/E39B2428435A074E86256C19008195FF">Database Config</a></li>
                        </ul>
                </li>
                <li class="navlist">Dev tools
                        <ul>
                                <li><a
href="/homesite.nsf/all/4A58453D4B854DF7862573A0005E15C5">Dom 2 json</a></li>
                        </ul>
                </li>
                <li class="navlist">Applications
                        <ul>
                                <li><a
href="/homesite.nsf/all/0E46DF3DA848815B8625727A005045A7">Photo Gallery App</a></li>
                        </ul>
                </li>
                <li class="navlist">Tips
                        <ul>
                                <li><a
href="/homesite.nsf/all/359857AFE224BAD886257396000BE6CD">DAV and SQL views</a></li>
                                <li><a
href="/homesite.nsf/all/6297DD943C5736F6862573A100162189">Install db2</a></li>
                        </ul>
                </li>
        </ul>
</div> <!-- ends #navcontainer-->
```

The form, view, and CSS are available in the Riverbend sample database.

Personalization

This page last changed on Apr 02, 2008 by jservais.

We see "Personalization" in the majority of applications and services that we use today - from Google's *iGoogle* to the functional preferences of this Wiki. Application personalization can provide users with function and features specific to their needs and help facilitate ease of use. In this section, we discuss the best practices for providing user personalization in Domino Web applications. We recommend that you have a firm understanding of the Web Browser Client Environment, HTML, JavaScript, and general Domino Web development practices.

User profile and preference documents

Lotus Notes Client developers are familiar with the practice of creating a preference document, either unique to the application, user community, or a specific user, that facilitates functional personalization of the given application. This is typically accomplished via Profile documents. As we look to expand our Domino applications to the Web browser and mobile client environments, we must consider the known Domino HTTP Task caching issues that surround profile documents, and look to a more multi-client friendly personalization architectures.

To facilitate such functional requirements, we can architect our Domino applications so that they use user profile and preference documents. These documents, which are keyed to the application, user community, or to specific users, contain information that facilitates application personalization.

Client-side preference caching

Due to the known issues with profile documents and applications delivered via the Domino HTTP Task, the alternate usage of Notes documents to maintain profile document-like information can require additional resource usage and "chattiness" between (for example) the Web browser client and Domino applications.

To alleviate costly round-trips to the Domino server to confirm application and user preferences (via runtime lookups to Notes documents acting as profile documents), we can use the local client environment, via HTTP cookies, to locally access user and application preferences.

A bit of both

We can architect and design our Domino Web applications to store user profile and preference document data in HTTP cookies. These same applications can use the local HTTP cookie data throughout the session (or future sessions) with the Domino server. The lack of HTTP cookie data or otherwise missing or logically outdated data can trigger the recreation or updates of the local HTTP cookies.

By using HTTP cookies and a profile stored in a Domino database, you can get a robust tool for handling personalization. For example, you can use a WebQueryOpen Web to store a document in a profile database and store that document's unique document ID in the user's HTTP Cookie. The beauty of this approach is that you can start gathering preference information on the user, even before they have registered.

If you have a shopping basket or a favorites feature, you simply add a child document to the user's profile in the profile database, using the unique document ID from the user's HTTP cookie. When the user needs to register, you can POST this information to the user's profile in the profile database, using the unique document ID. During registration, be sure to request the user's e-mail address and a password.

If for any reason a user loses their HTTP cookie, you can ask them to log in. This would not be Domino authentication, but the "Login" form would perform the following actions:

- Look up their profile in the profile database, in a view sorted by e-mail address.
- Verify the password.
- Reset the unique document ID back in the user's HTTP cookie.

This development methodology and applied technique not only addresses the aforementioned "chattiness", but can also streamline and enhance the user experience.

We invite you to add more of Domino personalization techniques or best practices here.

Searching

This page last changed on Apr 04, 2008 by dalandon.

Searching is one of the most useful functionalities that a Web application can have. Searches allow users to easily find data and manipulate it according to their needs. To facilitate peoples' lives, Lotus Domino has a complete set of tools to allow application developers to implement simple and high quality search functionalities. These tools can be used to allow data search and to implement a search itself.

In the following sections, we present these search techniques. We provide tips on how to increase a Web site ranking on search engines by using search engine optimization (SEO) and explain how to implement the all the techniques available to search for data on a Lotus Domino Web environment.

Use the following links for search information about performance:

- IBM Redbook: *Performance Considerations for Domino Applications*, SG24-5602
- IBM developerWorks article: Database properties and document collections

Additional topics

- Creating custom and advanced searches using Domino
- robots.txt
- Search engines and search engine optimization
- SEO techniques

Creating custom and advanced searches using Domino

This page last changed on Apr 02, 2008 by jservais.

In this section, we show how to use different techniques to display search results for Domino Web applications. Refer to the following entries for further information:

- Customizing the search results display
- Searching via Domino URL commands
- Searching via FTsearch and DBSearch

Customizing the search results display

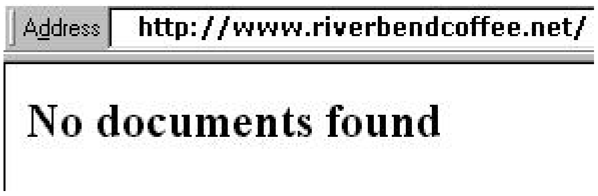
This page last changed on Apr 03, 2008 by jservais.

Introduction

On the Domino server, we can use diverse techniques to manipulate the way it displays the results of a search in a SearchTemplate form. Some of these techniques are displayed in the following sections.

Customizing the "No results found" message

Every SearchTemplate form should have an editable Rich text field called $$ViewBody. This field is used for displaying the content of views and search results over views on the Web. When a search does not produce any results that match the search criteria specified by the user, Domino generates an error message with the following statement by default on the $$ViewBody field: "No documents found", as shown in the following figure.

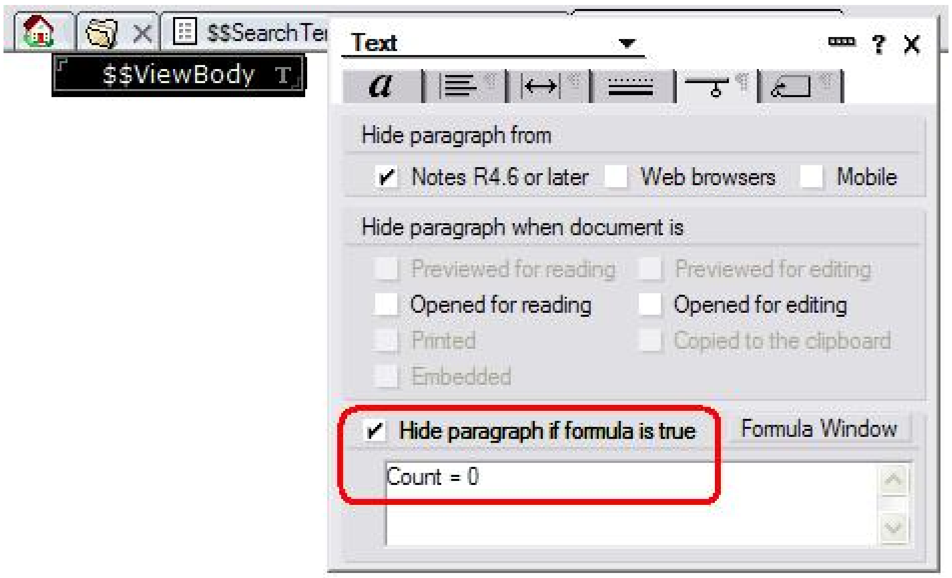


No documents found

Domino "No documents found" default message

This message can be off-context and confusing in several situations. Therefore, consider changing the resulting message. On search template forms, you may want to create a Text computed for display field called Count (value: Count). This field automatically holds the number of documents found from that search. By having the number of documents found in a search, in the paragraph that holds the view body, you can add a "hide when" formula to hide that paragraph in case documents were found.

```
@ TextToNumber(Count) = 0
```



Hide paragraph from
[✓ Notes R4.6 or later] Web browsers Mobile

Hide paragraph when document is
Previewed for reading
Previewed for editing
Opened for reading
Opened for editing
Printed
Copied to the clipboard
Embedded

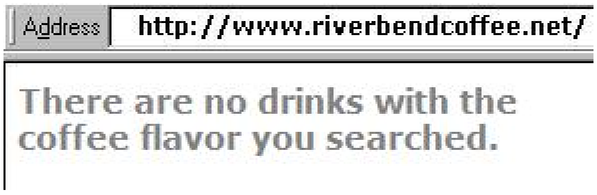
[✓ Hide paragraph if formula is true]
Count = 0

Hiding the $$ViewBody field if no documents are returned

Just below that paragraph, you add a new paragraph with your own "No documents found" custom statement that is displayed when no documents are found. It can also be in a computed for display field on computed text. The "hide when" formula for this paragraph is just the opposite of the one that is used on the $$ViewBody paragraph. It is only displayed when documents are not found.

```
@TextToNumber(Count) != 0
```

The following figure shows an example of a custom "no documents found" message for the Riverbend Coffee and Tea Company Web site. In this case, the custom message is displayed when the user searches for a coffee drink flavor that is not available.


Address http://www.riverbendcoffee.net/

There are no drinks with the coffee flavor you searched.

Customized "No documents found" message

Using the Next and Previous buttons or hotspots with Start and Count parameters

If you are using Start and Count parameters from a SearchTemplate or ViewTemplate, you can include the Next and Previous buttons or hotspots to enable users to navigate between pages of results. Both parameters must be used if you are using navigation buttons.

1. Open your customized results form and place the buttons or hotspots labeled Next and Previous where you want them to appear on the form.
2. For the button or hotpsot labeled Next, write a formula that advances the user forward one page.
3. For the button or hotspot labeled Previous, write a formula that takes the user back one page.

The following example shows a formula for a Next button or hotspot:

```
@If(Hits >= Count; @URLOpen("/" + @Subset(@DbName; \-1) + "/" + SearchView +
"?SearchView&Query=" +
@ReplaceSubstring(Query; " "; "+") + "&Start=" + @Text(Start+Hits) + "&Count=" + @Text(Count) +
"&SearchOrder="+@Text(SearchOrder) \+"&SearchWV="+@If(SearchVw =
0;"FALSE";"TRUE")+"&SearchThesaurus="+@If(SearchThesaurus = 0;"FALSE";"TRUE") +
"&SearchMax=" + @Text(SearchMax)); "")
```

The following example shows a formula for a Previous button or hotspot:

```
@If(Start > Count; @URLOpen("/" + @Subset(@DbName; \-1) + "/" \+SearchView +
"?SearchView&Query=" +
@ReplaceSubstring(Query; " "; "+") + "&Start=" + @Text(Start-Count) + "&Count=" + @Text(Count)
+
"&SearchOrder="+@Text(SearchOrder) \+"&SearchWV="+@If(SearchVw =
0;"FALSE";"TRUE")+"&SearchThesaurus="+@If(SearchThesaurus = 0;"FALSE";"TRUE") +
"&SearchMax=" + @Text(SearchMax)); "")
```

Tip
To avoid syntax errors, use @ReplaceSubstring(Query; "" ' "+") to replace all of the spaces in your query with plus signs (+).

Displaying the results by using AJAX

You can take advantage of the AJAX capabilities to build searches for a Domino Web application. The benefit of this approach is to enable users to have a page that is partially loaded, have the search results loading asynchronously in a second step, and enable such features as pagination without loading the entire page again, improving server performance.

Domino offers a URL to access view data by using XML. The format of this URL is shown on the following example:
http://www.riverbendcoffee.net/helpdb.nsf/htmlview?ReadViewEntries

You can build your a view by parsing the XML results from this URL by using AJAX. To do that, you must browse through the XML structure created by the ReadViewEntries URL. You need to get the content of the *viewentry* nodes from the XML, and print it on the page for the user. Use the following steps to create a simple AJAX view:

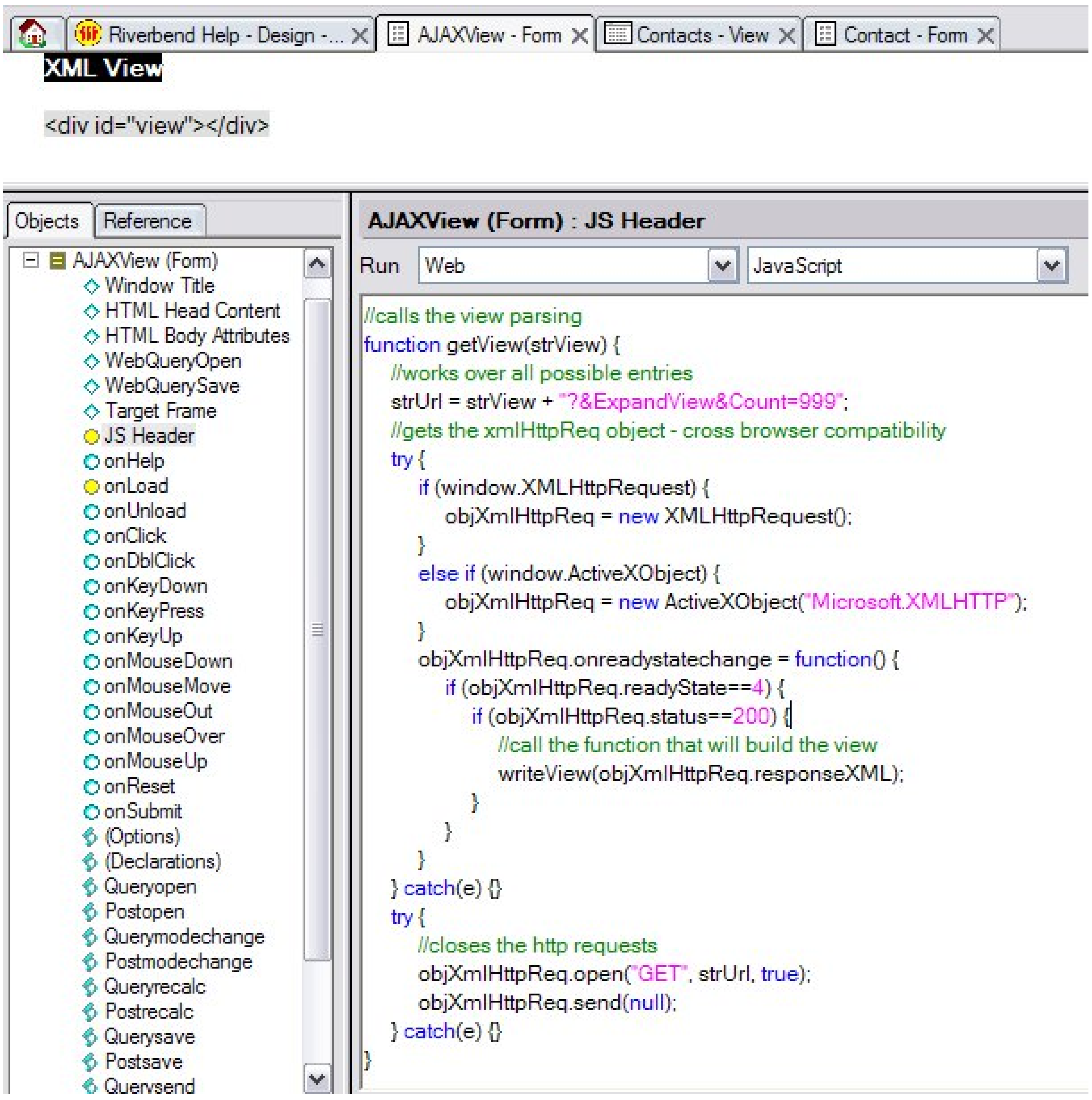
1. Create an HTML page by using a page design element, for example.
2. On this form, create a *<div>* tag with an *id* attribute, so that we can add the HTML that is going to be retrieved from the XML parsing.Refer to the following example:

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

3. On the page onLoad event, make an xmlhttp call to retrieve the XML data.Refer to the following example:

```
getView("http://www.riverbendcoffee.net/helpdb.nsf/htmlview?ReadViewEntries");
```

Your form will be similar to the one shown in the following figure.


Riverbend Help - Design -... | AJAXView - Form | Contacts - View | Contact - Form

XML View

<div id="view"></div>

Objects | Reference

AJAXView (Form)
- Window Title
- HTML Head Content
- HTML Body Attributes
- WebQueryOpen
- WebQuerySave
- Target Frame
- JS Header
- onHelp
- onLoad
- onUnload
- onClick
- onDblClick
- onKeyDown
- onKeyPress
- onKeyUp
- onMouseDown
- onMouseMove
- onMouseOut
- onMouseOver
- onMouseUp
- onReset
- onSubmit
- (Options)
- (Declarations)
- Queryopen
- Postopen
- Querymodechange
- Postmodechange
- Queryrecalc
- Postrecalc
- Querysave
- Postsave
- Querysend

AJAXView (Form) : JS Header

Run: Web | JavaScript

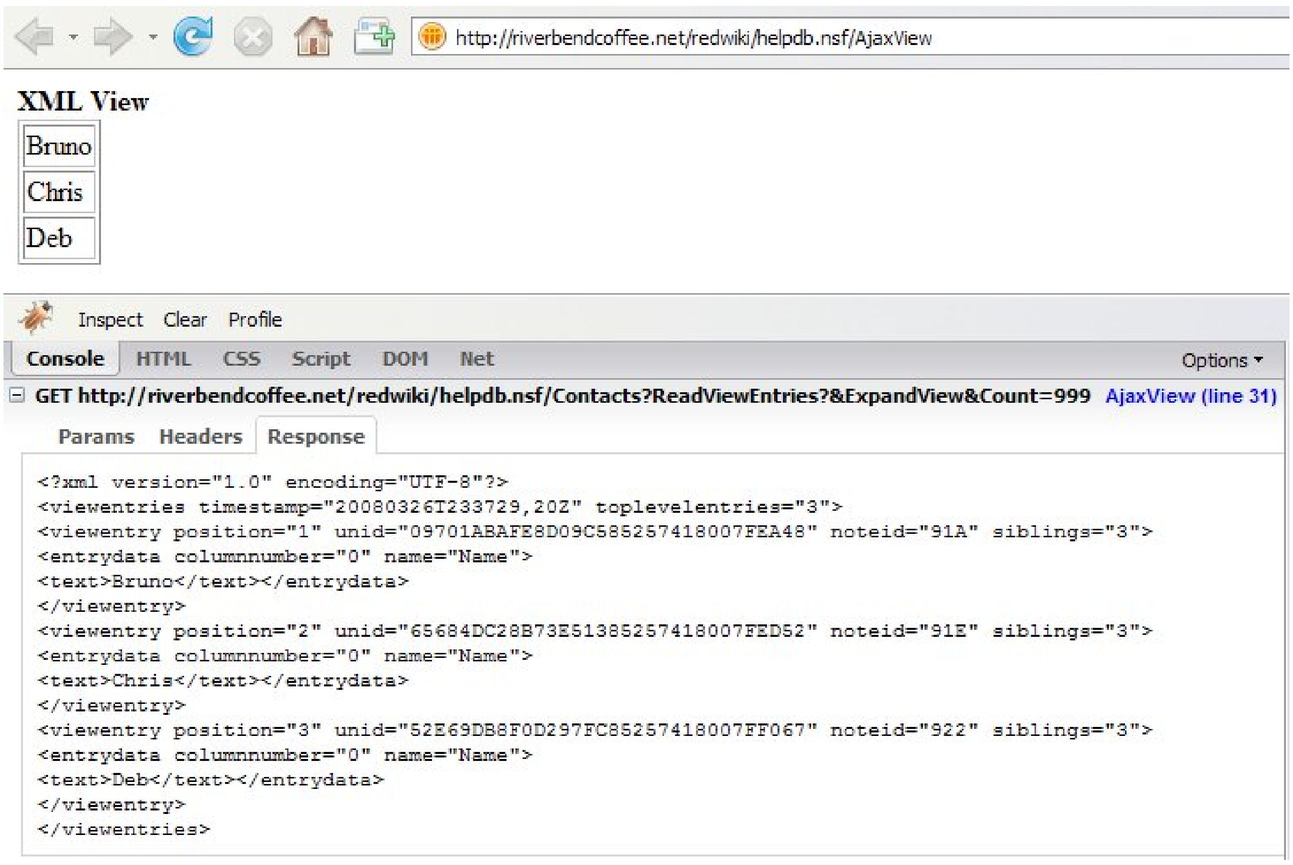
```
//calls the view parsing
function getView(strView) {
    //works over all possible entries
    strUrl = strView + "?&ExpandView&Count=999";
    //gets the xmlHttpReq object - cross browser compatibility
    try {
        if (window.XMLHttpRequest) {
            objXmlHttpReq = new XMLHttpRequest();
        }
        else if (window.ActiveXObject) {
            objXmlHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
        }
        objXmlHttpReq.onreadystatechange = function() {
            if (objXmlHttpReq.readyState==4) {
                if (objXmlHttpReq.status==200) {
                    //call the function that will build the view
                    writeView(objXmlHttpReq.responseXML);
                }
            }
        }
    } catch(e) {}
    try {
        //closes the http requests
        objXmlHttpReq.open("GET", strUrl, true);
        objXmlHttpReq.send(null);
    } catch(e) {}
}
```

Form that retrieves the XML

4. In the JS Header, add the following code:

```
//calls the view parsing
function getView(strView) {
        //works over all possible entries
 strUrl = strView + "?&ExpandView&Count=999";
        //gets the xmlHttpReq object - cross browser compatibility
 try {
                if (window.XMLHttpRequest) {
                        objXmlHttpReq = new XMLHttpRequest();
                }
                else if (window.ActiveXObject) {
                        objXmlHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                }
                objXmlHttpReq.onreadystatechange = function() {
                        if (objXmlHttpReq.readyState==4) {
                                if (objXmlHttpReq.status==200) {
                                        //call the function that will build the view
                                 writeView(objXmlHttpReq.responseXML);
                                }
                        }
                }
        } catch(e) {}
        try {
                //closes the http requests
         objXmlHttpReq.open("GET", strUrl, true);
                objXmlHttpReq.send(null);
        } catch(e) {}
}


//parses and prints the content
function writeView(xmlDoc) {
        try {
                //gets the root of the xml document
         var objXmlRoot = xmlDoc.documentElement;
                //get the toplevelentries attribute of the root - the total count of docs in
the view
         arrNodes = objXmlRoot.getElementsByTagName("viewentry");
                //start a table tag
         var strResult = '<table border="1">';
                //browser through all entries
         for(var i=0; i<arrNodes.length; i++) {
                        intNumCols = arrNodes[i].getElementsByTagName("text").length;
                        strResult += '<tr> \n';
                        //goes thru all columns getting their html cell values
                 for(var intCol=0; intCol<intNumCols; intCol++){
                                //creates a new view
                         strResult += '<td>' +
arrNodes[i].getElementsByTagName("text")[intCol].childNodes[0].nodeValue + '</td>'
                        }
                        //closes the row
                 strResult += '</tr> \n';
                }
                //closes the table
         strResult += '</table>\n';
                //removes all Domino [ and ] row tags
         strResult = strResult.split('[').join('').split(']').join('');
                document.getElementById("view").innerHTML = strResult;
        } catch(e) {}
}
```

This code results in producing a simple page that displays the content of that view on a table, row by row, as shown in the following figure.



XML View

Bruno
Chris
Deb

GET http://riverbendcoffee.net/redwiki/helpdb.nsf/Contacts?ReadViewEntries?&ExpandView&Count=999 AjaxView (line 31)

```
<?xml version="1.0" encoding="UTF-8"?>
<viewentries timestamp="20080326T233729,20Z" toplevelentries="3">
<viewentry position="1" unid="09701ABAFE8D09C585257418007FEA48" noteid="91A" siblings="3">
<entrydata columnnumber="0" name="Name">
<text>Bruno</text></entrydata>
</viewentry>
<viewentry position="2" unid="65684DC28B73E51385257418007FED52" noteid="91E" siblings="3">
<entrydata columnnumber="0" name="Name">
<text>Chris</text></entrydata>
</viewentry>
<viewentry position="3" unid="52E69DB8F0D297FC85257418007FF067" noteid="922" siblings="3">
<entrydata columnnumber="0" name="Name">
<text>Deb</text></entrydata>
</viewentry>
</viewentries>
```

Resulting view example

Important
Remember that this example uses the JavaScript xmlhttp object. Therefore, Web pages that use this technology only work if the user's browser is compatible with this technology.

We have shown an example of how to take advantage of AJAX capabilities to enhance the results display. You can customize it according to the needs of your Web site.

Searching via Domino URL commands

This page last changed on Apr 03, 2008 by jservais.

Domino URL commands for search

Search-related URLs are available to perform view, multiple-database, and domain searches. Typically you define a URL that displays an input form to allow users to define their own searches. This can be a customized search form or the default search form. A designer may also define a URL to perform text searches without user input. Both the input and the results forms may be customized.

Attention
The URLs in the following sections are for example only. They are not intended to point to existing Web sites unless specifically indicated.

SearchDomain

Use SearchDomain URLs for text searches across a domain. The search input form is opened with the OpenForm command by name or universal ID. For search results, the results template is specified as part of the URL. If no *template is found, then the default template form, $$SearchDomainTemplate, is substituted. If $$SearchDomainTemplate is not found, an error is returned. If no results are returned, the value of the $$ViewBody field remains the same.

Syntax
protocol://Host/Database/*templateForm* ?SearchDomain *ArgumentList*

Where:

- *templateForm* is an optional argument that calls the search results form.
- *ArgumentList* is a list of optional arguments.

Example
http://www.riverbendcoffee.com/mersrch.nsf/MercuryResults?SearchDomain

Redirect

The server provides a direct or redirect URL command as needed for links that are displayed on the results form if the capability has been enabled. The domain URL locates information on the server where the links are generated. The redirect command locates the correct server, and redirects a link to that server by constructing the appropriate URL. The redirect command can improve performance by resolving individual links when they are selected instead of resolving all of the links returned at once. See *Domino 5 Administration Help* for information about enabling redirect on a server.

SearchSite

Use SearchSite URLs for text searches in multiple databases. Because the URL requires the name of a search site database, be sure to create one before using a SearchSite URL.

Syntax
protocol://Host/Database/*SearchForm*?SearchSite *ArgumentList*

Where: *$SearchForm* and *ArgumentList* are optional arguments.

Example
http://www.riverbendcoffee.com/mercsrch.nsf/$SearchForm?SearchSite

SearchView

Use SearchView URLs to limit a search to documents displayed in one database view. This URL is useful for views that show all documents (so that you can have a full-database search) or for views in which you can predict what users need to see, such as all documents whose status is "Completed."

Syntax
protocol://Host/Database/View/*$SearchForm*?SearchView *ArgumentList*

Where: *$SearchForm* and *ArgumentList* are optional arguments. The special identifier *$SearchForm* indicates that Domino will present a search view form for search input. If this identifier is provided, the ArgumentList is ignored. If this identifier is absent, a default form is generated dynamically based on the contents of the search.htm file located on the server. The default form generated by the server does not support paged results.

Example
http://www.riverbendcoffee.com/products.nsf/By+Product+Number/$SearchForm?SearchView

Optional arguments for SearchSite, SearchView, and SearchDomain

$SearchForm

The special identifier *$SearchForm* indicates that Domino will present a search site form for search input. If this identifier is provided, *ArgumentList* is ignored.

ArgumentList

The *ArgumentList* must contain the Query argument. In addition, it may contain any or all of the other arguments in any order:

- Query=*string*
  Where *string* is the search string.

- Count=[n]
  Where *n* is the number of results to display on each page until the SearchMax has been reached. For example, Count=10 shows 10 results per page.

- Scope_=[1,2,3]_
  The scope of the search, where 1 = Notes databases only, 2 = file system only, 0 = both. The default value is 0. This argument should only be used with the SearchDomain command.

- SearchEntry=*formName*
  Where *formName* is the name of the form to use for the results of a domain search. The default argument is "ResultEntry," which supports all of the predefined results fields specified in the *ArgumentList*. This argument is valid for SearchDomain only and should not be used for SearchSite or SearchView.

- SearchFuzzy=[TRUE,FALSE]
  Indicate TRUE for fuzzy search. The default is FALSE.

- SearchOrder=*[1,2,3,4]*
  Indicate 1 to "Sort by relevance," 2 to "Sort by date ascending," 3 to "Sort by date descending." The default is 1. SearchView also supports a SearchOrder value of 4 to "Keep current order," which sorts the resulting set of documents in the order in which they appear in the view.

Important
Specifying SearchOrder=4 produces unexpected results if any of the following points are true:

  - 1. The Count=n argument is used with a value less than the number of documents found.
    2. The Start=n argument is used with a value other than 1.
    3. The Default Search Limit is less than the number of documents found.
    4. The Max Search Limit is less than the number of documents found.
    5. If you need to specify SearchOrder=4, observe these recommendations:
       - Never specify Count=n or Start=n
       - Always specify SearchMax=0
       - Set the Web site's Max Search Limit to a large value

- SearchMax=[n]
  Where *n* is the maximum number of entries returned. The default value is determined by the server.

- SearchWV=[TRUE, FALSE]
  Where TRUE = include word variants in the search. The default value is FALSE.

- Start=[n]
  Where *n* is the number corresponding to the document that appears first in your list of results. For example, Start=10 begins your list of results with the tenth document found in the search. Start=0 means that paged results are not returned.

Examples
http://www.riverbendcoffee.com/mercsrch.nsf/?SearchSite&Query=product+info+requests&SearchOrder=2&SearchMa
http://www.riverbendcoffee.com/products.nsf/By+Product+Number/?SearchView&Query=PC156&SearchOrder=3&Sea

URL search syntax and customized results

The following parameters for the SearchView and SearchSite URL commands allow you to display search results page-by-page and to provide buttons or hotspots to navigate between pages.

Start and Count parameters
With the Start and Count parameters, you can display search results page-by-page and include them as arguments in the SearchView or SearchSite URL commands or as items in the search results form. The Start parameter is the entry that appears first when your results are displayed. The Count parameter is the number of results that are shown on each page. For example, if Start=5 and Count=10, the search results are displayed beginning with the fifth entry and show up to 10 entries per page until the maximum number of entries is displayed. These parameters work with customized forms only.

Syntax
protocol://Host/Database/ViewName/*$SearchForm*?SearchView *ArgumentList*
protocol://Host/Database/ViewUNID/*$SearchForm*?SearchView *ArgumentList*

Where: *ArgumentList includes the Query argument and any or all of the other arguments including the Start and Count parameters, for example ?SearchView&Query=String&Start=n&Count=n&SearchOrder=n&SearchWV=TRUE or FALSE&SearchFuzzy=TRUE or FALSE&SearchMax=n.*

Examples
http://www.riverbendcoffee.com/products.nsf/ProductView?SearchView&Query=bicycles&Start=21&Count=20&Search
http://www.riverbendcoffee.com/products.nsf/F6025FD7E72456F985256540005839D3?SearchView&Query=bicycles&S

Customizing the Web results for SearchView

To customize the Web search results page for SearchView:

1. Create a form and assign it one of the form names shown in the following table.

Form name	Field required	Comments
$$SearchTemplate for viewname	$$ViewBody	Associates the form with a specific view. Domino requires the $$ViewBody field, but

		ignores the value. The form name includes viewname, the alias for the view, or, when no alias exists, the name of the view. For example, the form named "$$SearchTemplate for All Documents" associates the form with the All Documents view.
$$SearchTemplateDefault	$$ViewBody	Domino requires the $$ViewBody field, but ignores the value. This form is the default for all Web searches that are not associated with a specific form.

2. Add a field named $$ViewBody to the form.
3. If you want to display results page-by-page, add buttons or hotspots for forward and backward navigation to the form.
4. Use the Start and Count parameters in your URL command.

For more information about the URL commands and search templates, refer to All Domino URLs and $$SearchTemplate.

Using navigational buttons for paged results

Web searches over the Domain catalog can use any form through an OpenForm URL command, building and invoking a SearchDomain URL command to perform the search on catalog.nsf database, supplying arguments either as URL command arguments or through posted field values.

Single database searches over a Domino Web application can be initiated by using a $SearchForm?SearchView URL command. In this case, Domino looks in the current database for a form with the actual name or the alias name $$Search. If the form exists, Domino opens it. Otherwise, Domino displays a default search form based on the search.h file stored in the Domino\Icons directory. The $$Search form builds and invokes a SearchView URL command to perform the search, supplying arguments either as URL command arguments or using posted field values. You can also customize the default search.h form.

You may refer to the following table to customize a search form for the Web. The table provides a detailed list of the URL command arguments that are used to execute an initial search through the SearchDomain or SearchView URL. These values are available on the results page for use by buttons and hotspots on the results form. For example, you may specify &SearchOrder=2 on your initial search form. The field SearchOrder has a value of two in the results page. A Next button on the results form can use this value for the next page or override it by specifying something else. This navigation process is detailed in Customizing the search results display.

Tip
If you use editable fields on a search result form, select the option **Web Access: Use JavaScript when generating pages** in the Database properties. If selected, a URL attached to a hotspot or button is computed on the click event. If it is not selected, the URL is computed when the page is

loaded.

Although TRUE and FALSE can be specified for some of the fields, when the values are carried over onto the results page they are 1 or 0.

Field	Description
Query	Search string used
Start	Starting document number. 0 = unpaged
Count	Number of results requested for this page. 0 = unpaged
Hits	Actual number of results returned this page, which may be less than Count requested. This field is useful in determining the Start parameter for a Next button.
TotalHits	Total number of hits found by the search.
SearchMax	Maximum number of entries to return in total; 0 = no limit.
SearchWv (only for URL command)	Include word variants: 1 or 0.
SearchOrder (only for URL command)	1 = By relevance 2 = By date ascending 3 = By date descending 4 = Use view order (SearchView only)
SearchThesaurus (only for URL command)	Use thesaurus synonyms: 1 or 0.
SearchFuzzy (only for URL command)	Use fuzzy search: 1 or 0.

SortOptions
(only for Notes client)
	FT_SCORES = By relevance
	FT_DATE_ASC = By date ascending
	FT_DATE_DES = By date descending

OtherOptions
(only for Notes client)
	FT_STEMS = Include word variants
	FT_FUZZY = Use fuzzy search
	FT_DATABASE = Search databases
	FT_FILESYSTEM = Search file systems

SearchEntry (Domain Searches only)	Name of the result entry form used.
SearchView (only for SearchView URL command)	Text unique identifier of the view being searched. This identifier is useful in building subsequent SearchView URL commands.
Scope (only for SearchDomain URL command)	Scope of search: 1 = Notes databases only 2 = File system only 0 = Both

The fields in the following table are available for use with the Start and Count parameters and should be added to the results form as needed.

Field	Description
Hits	The actual number of hits returned. This field is useful in determining the Start parameter of Next.
TotalHits	The total number of hits found without regard to the number of pages.

Searching via FTsearch and DBSearch

This page last changed on Apr 02, 2008 by jservais.

The differences between FTsearch and DBSearch

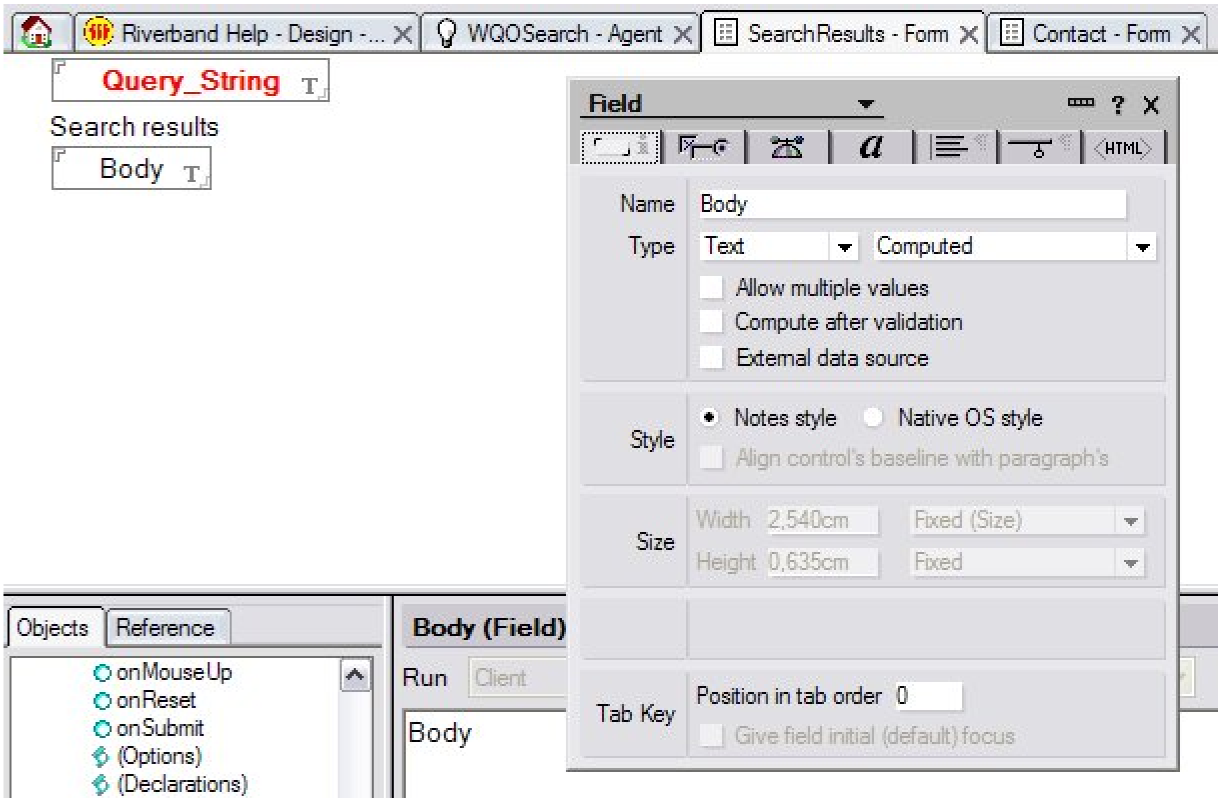
In a Domino Web application, besides searching via Domino URL commands, a designer can implement searches by using the FTSearch and Search (also known as DBSearch) methods for a word or phrase. The *DBSearch method* searches for data by using a specific query pattern and returns a set of documents based on the query. DBSearch has better performance than FTSearch. *FTSearch* uses the full text index to search for all documents in a view or database that match a query string. The database does not have to be full text indexed for FTSearch to execute properly. However the search takes longer if it is not. Just like the search method, FTSearch returns a subset of documents based on the search criteria using the NotesDocumentCollection LotusScript class.

These search techniques allow designers to implement search for documents by using conditions and operators based on user inputs that can be passed via custom URL commands. Searching on a view using Domino URL commands works best when the application has a full text index, which makes available advanced search features and faster search capability. However, using FTSearch and DBSearch methods can help find data by using more details.

Examples of search implementations

In the following example, we illustrate how to create a custom search results page with content generated by arguments passed by the URL:

1. Create a simple search results form.
2. On this form, create a hidden computed field, called *Query_String*, that has as value the CGI variable *Query_String*, that is used to get the query information about the URL.
3. Create a rich text field, called *Body*, to display your search results. Set this as a computed field with the value *Body*, and enable pass-through HTML on this field paragraph.


Riverband Help - Design - ... WQOSearch - Agent SearchResults - Form Contact - Form

Query_String

Search results

Body

Field

Name: Body
Type: Text, Computed
Allow multiple values
Compute after validation
External data source
Style: [✓ Notes style] Native OS style
Align control's baseline with paragraph's
Size: Width 2,540cm Fixed (Size)
Height 0,635cm Fixed
Tab Key: Position in tab order 0
Give field initial (default) focus

Objects | Reference
onMouseUp
onReset
onSubmit
(Options)
(Declarations)

Body (Field)
Run Client
Body

Search results form details

4. Create and set a WebQueryOpen agent to this form.
5. Set the right basic and security properties for your WebQueryOpen agent, and use the following code on the agent:

```
Sub Initialize

        On Error Goto CommomErrHandler

        Dim session As New NotesSession
        Dim db As NotesDatabase
        Dim collection As NotesDocumentCollection
        Dim doc As NotesDocument
        Dim docSearch As NotesDocument
        Dim strQuery As String
        Dim strResults As String

        '//set the current database and the current form from the WebQueryOpen property of the
form
 Set db = session.CurrentDatabase
        Set doc = session.DocumentContext

        '//gets the value to be searched from the right of the "&Query=" piece of the query
string
 strQuery = Strright(doc.Query_String(0), "&Query=")
        '//does a search on the database using the Search() method
 '//Set collection = db.Search(strQuery, Nothing, 0)
 '//does a search on the database using the FTSearch() method
 Set collection = db.FTSearch(strQuery, 0)

        '//displays an error message in case there are no documents
 If collection.Count = 0 Then
                doc.Body = "No document foud"
                Exit Sub
        End If

        '//browse through the documents, displaying the results as links
 strResults = ""
```

```
        Set docSearch = collection.GetFirstDocument
        While Not docSearch Is Nothing
                strResults = strResults + Chr(13) + |<p><a href="./0/| + docSearch.UniversalID+
|"> Unid: | + docSearch.UniversalID + |</a></p>|
                Set docSearch = collection.GetNextDocument(docSearch)
        Wend
        doc.Body = strResults

        Exit Sub
CommomErrHandler:
        doc.Body =  "Got error on agent #" + Cstr(Err) + " - " + Error$ + " on line " +
Cstr(Erl)
        Exit Sub

End Sub
```

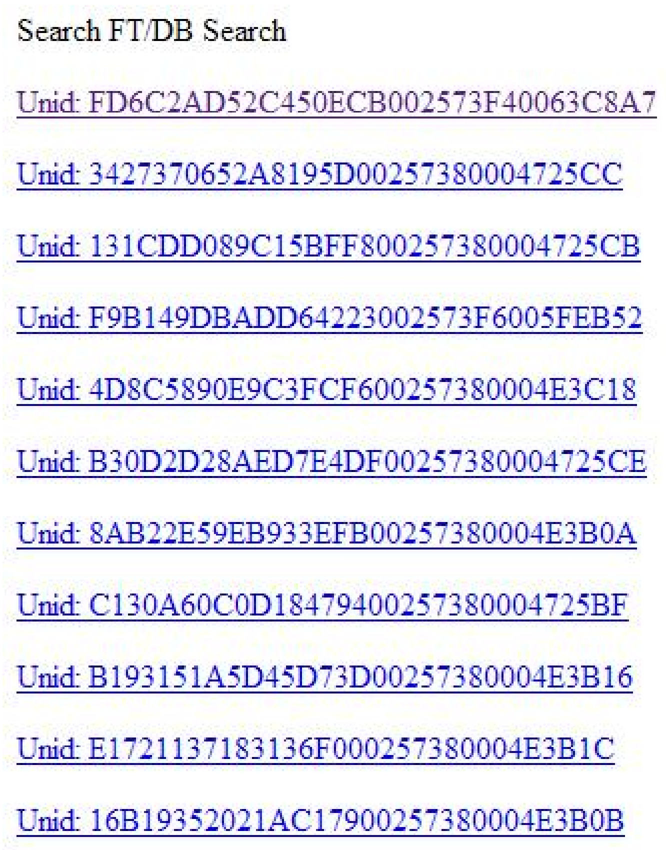
6. Create some documents to test your Search() or FTSearch() methods.
7. Invoke the agent by using a URL, for example:
http://riverbendcoffee.net/helpdb.nsf/SearchResultsForm?OpenForm&Query=SearchQuery, where *SearchQuery* is the value searched.

Explanation of the model

The previous example illustrates a custom search interface created on-the-fly for Web users. It uses a form to display links for documents found from a free text search over a database. It has a couple of fields:

- A Query_String field to retrieve the user query portion from the URL
- A rich text repository field (Body) that has as unique function to receive the HTML code generated from the search results

When that form is loaded, it expects to have a *&Query=* argument with the free text term the user wants to bring from a database search. This is what the WebQueryOpen agent does in a and simple manner, it get the query value of that argument in Query_String field of the form, and displays the results with links to the users using the *0* view, and the document universal ID.



Search FT/DB Search

Unid: FD6C2AD52C450ECB002573F40063C8A7

Unid: 3427370652A8195D00257380004725CC

Unid: 131CDD089C15BFF800257380004725CB

Unid: F9B149DBADD64223002573F6005FEB52

Unid: 4D8C5890E9C3FCF600257380004E3C18

Unid: B30D2D28AED7E4DF00257380004725CE

Unid: 8AB22E59EB933EFB00257380004E3B0A

Unid: C130A60C0D18479400257380004725BF

Unid: B193151A5D45D73D00257380004E3B16

Unid: E1721137183136F000257380004E3B1C

Unid: 16B19352021AC17900257380004E3B0B

Search results page

Thus, there could have been much more improvements depending on the application. It could have an input form where the user can use a form to submit search information via form GET, allow users to manipulate the arguments on the URL via buttons or links, display results with a better look and feel using descriptions on tables and style sheets, and much more. However, this is a simple option for search interfaces that does not use Domino URL commands and should be used according to the application needs.

Important
When choosing the methods to implement your search, keep in mind that the Search method has much better performance returning the results than the FTSearch method. Only consider using the FTSearch method in cases where a real full text search is needed. Also, remember that the query syntaxes are different.

Tip
When displaying the results on a Web page, consider sorting the collection by a specific parameter. To sort a NotesDocumentCollection collection by a specific criteria, refer to the article How do you sort a document collection?.

robots.txt

This page last changed on Apr 03, 2008 by jservais.

robots.txt

The robots.txt file is a Web standard file that is used on the whole World Wide Web to declare what search engines should not index from a Web site. This is an "old" technique, but is still helpful. By using this file, you can select which files not to index, avoiding the display of private files on search engines. This file is flexible and allows you to implement several rules in the same file to ensure distinct behavior for bots.

The robots.txt files was created around 1994 by the members of the Robots mailing list. There is no standards message or RFC for this issue. It is important to remember that robots.txt should not be used to flag what should be indexed, but to indicate what *should not be indexed*. You need robots.txt, for example, in an intranet with WWW access that has sensitive information for that company. Restrict areas and personal documents that are hosted on your server in a specific directory for backup reasons that are possible assets that you may want to prevent from getting indexed.

If you want a search engine to index your whole site, do not use robots.txt.

Using robots.txt on Domino

When creating a robots.txt file, keep the following considerations in mind:

- The robots.txt file is a text file that must be created by using plain ASCII text and *should* be saved by using the "txt" file extension.
- This file should be on the root directory of your Web site. This is the first that a spider visits on a Web site.
- The file should be written in lowercase and have the proper public read access to the world. If your Web site root directory is your NSF file, you can upload your robots.txt to your NSF database as a file resource. You can also create a page named robots.txt, insert its content on the design body, and change its content type to text. For further information about this topic, refer to Page design elements.
- Since Web crawlers consider subdirectories or subdomains as completely different Web sites, keep a new version of the robots.txt file on every subdirectory with a new site or with sensitive data. For example, if you have www.riverbandcoffee.com/ and www.riverbandcoffee.com/blog/ or www.blog.riverbandcoffee.com, in which "blog" is an important subdirectory or subdomain with some data that you do not want indexed, consider using robots.txt on that directory too. In case of

a subdirectory, you have the unique option to create a page, as described previously. You may also consider using the robots metatag approach, described later in this section.

Important
If your Web site root directory is not your NSF database, you must upload the robots.txt file to your Lotus Domino Web server root directory. For further information about this topic, refer to topology.

There are basically two rules to declare on this file: User-Agent and Disallow. The User-Agent is used to declare a specific agent. A User-Agent in this context is a search engine spider, like the Googlebot from Google.

```
User-Agent: Googlebot
```

If you want all agents (and not only the Google robot) to index the content, use an asterisk as the value, so that the search engines do not index.

```
User-agent: *
```

To block the whole site, use the root directory bar, as in the following example:

```
Disallow: /
```

To block a specific directory, enter the directory path, as in the following example:

```
Disallow: /private_directory/
```

To block a specific file, enter the file path, as in the following example:

```
Disallow: /private_file.html
```

You can use as many Disallow rules as you want. Start a new line on your file.

Important
Remember that URLs are case sensitive. Therefore, a page called Coffee.htm cannot be declared as coffee.html.

robots.txt examples

The following example prohibits any robot from indexing the whole site:

```
User-agent: *
Disallow: /
```

An asterisk indicates *everything* or that all the robots should follow that rule. A practical example is preventing indexing folders on your site from containing private information. The code in the following example prevents for directories from being indexed.

```
User-agent: *
Disallow: /cgi-bin/ #scripts e and programs
Disallow: /login/
Disallow: /tmp/ #testing area
Disallow: /private/ #corporate files
```

The number sign (#) is used for comments. You can use this sign to explain the reason for excluding the file, without impacting its usage.

If you do not have a robots.txt file, the tool indexes your site normally. It is the same as having the following robots.txt file:

```
User-agent: *
Disallow:
```

The following example shows a more complex example. In the first lines, we declare that /directory/ should not be indexed by any robot. This rule should be followed by all spiders. Then, on lines 3 to 5, we define that the Google robot, Googlebot, should not index /cgi-bin/ and /corporate/hr/ directories. Then, in lines 6 and 7, we define that Yahoo robot, Slurp, should not index /corporate/accountancy/. Then, to finish, on lines 8 and 9, we define that the MSN® robot, msnbot, should not index /msoffice_docs/ directory.

```
User-agent: *
Disallow: /private/
User-agent: Googlebot # Google (line 3)
Disallow: /cgi-bin/
Disallow: /corporate/hr/
User-agent: Slurp # Yahoo (line 6)
Disallow: / corporate/accountancy/
User-agent: msnbot # MSN (line 8)
Disallow: /msoffice_docs/
```

Tip
The robots.txt file does not affect the search results returned by Domino on the Web. If you want to see which pages are brought by a search result, you may want to review your search query and view selections, and implement security by using access control lists (ACLs) and Readers fields. For further information about how to implement security on Domino applications, refer to security considerations.

robots meta tag

If you do not have access to the robots.txt file, you can use another approach to prevent a page from getting indexed. There is an HTML meta tag, called *robots*, that prevents spiders from indexing a Web site. This tag has a property that can have a pair of values, brought by the combination of these options: index, follow, noindex, and nofollow. Index and follow are the implicit defaults for this tag. The index option allows a page to be indexed, and follow allows its links to be indexed. The noindex option prevents indexing the page, which means not to put the page in the search results. The nofollow value prohibits following the links on this page in the index. If no other pages point to the same pages as the links on this page, this can have the same effect on those pages as a noindex on those pages. However, since anyone using any Web page can deep-link to those pages, this can fail. In the following example, a robot indexes the page and follows all the links on the page:

```
<meta name="robots" content="index,follow" />
```

In the following example, a robot indexes the page, but treats it as a "dead end" and does not follow any of the links on it.:

```
<meta name="robots" content="index,nofollow" />
```

In the following example, a robot skips over the page, without indexing its content, but continues indexing all the other pages to which this page links:

```
<meta name="robots" content="noindex,follow" />
```

In the following example, an ethical robot neither indexes the page nor follows any of its links. It considers this page as nonexistent on their indexes.

```
<meta name="robots" content="noindex,nofollow" />
```

This approach has the problem of being implemented in hypertext files, preventing its use for such files as PDFs or DOCs. Thus, robots.txt have a higher scale than this approach, but both have their importance.

For further information about how to insert meta tags on your pages, refer to Common design properties on Web applications.

Important
Despite the fact that the most reliable search engine robots respect the Web site indexing rules defined on the robots.txt file, do not expose sensitive data to the World Wide Web. "Thief spiders" can crawl your Web site to search for sensitive data. To avoid this, you must implement efficient security by using such techniques as firewalls, files access control, and ACLs. For further information about how to implement security on Domino applications, refer to security considerations.

Search engines and search engine optimization

This page last changed on Apr 02, 2008 by jservais.

Search engine optimization (SEO) is the technique of preparing Web sites and Web pages to achieve better ranking on the search engine results, without infringing their rules. A search engine, also known as *crawler* or *search site*, is a Web portal where users can search for a something using natural terms. SEO is the best way to create qualified Web site traffic and is an ever-growing trend of the Internet. The term *site optimization* is related to marketing optimization, but is more specific in that the SEO wills to qualify Web site or pages to figure in more competitive searches.

SEO is a must-have in order to increase Web site traffic and reach new visitors. Search engine traffic also has the advantage of being pre-qualified. Therefore, if the business of a site is selling cars, it does not make sense to receive visitors who are interested in botanical gardening. This visitor focus increases the importance of a Web site, helping business growth.

Introduction and panorama

When the Internet first began, there were no search engines. Users had to browse through directories to find a Web site. A *directory* is a list of sites that are manually selected by people (editors) and categorized. The users had to browse by category and subcategories, until they found an interesting link. Directories have no ranking. All pages are listed under the same category. When users noticed that search engines were much easier to use and had better results than directories, they became the most common tools for finding information on the Internet.

Yahoo! is the most famous directory available and was also the most commonly used search engine. Today, Google is the most used search engine in the world, with more than 50% of the world searches (Source: Nielsen/NetRatings MegaView Search, May 2007). Google is commonly used for direct and indirect searches in other Web sites. Google was also created by academics, who have several papers regarding its structure. In addition, Google had focus on profitability and partnership, which helped it reach great success.
Google is a large scale search engine that intensively uses hypertext. It was projected to efficiently track and index the Web and produce more satisfactory results than any other existing systems.

Creating a search engine is a defying project. Search engines index dozens of millions of Web pages that contain diverse comparable search terms. They have to answer to millions of searches daily. Despite all the importance of search engines, there is not much academic research in this topic. In addition, due to fast paced Web market growth, creating a search engine today is much more difficult than it was some years ago. At this point, application designers should focus on achieving better ranking on search engines results to promote their company's business.

SEO techniques

This page last changed on Apr 03, 2008 by jservais.

Introduction

Search engine optimization (SEO) techniques are a method to improve search results ranking. These techniques involve a series of steps and may require the reformulation of an entire Web application. We provide some examples of these techniques in the following sections.

Legal techniques

Legal techniques are recommended and fair techniques that are used to improve ranking on search sites search results. We describe these techniques in the sections that follow.

What a site does

The first step to improve ranking is to think about what a Web site does. Then, define which keyword relates to the Web site. In our example of the Riverbend Coffee and Tea Company, you can have a group of words such as "Riveband Coffee," "Brazilian Blended Coffee," and "Chinese Tea."

To validate your keywords, try searching for them on a search engine. Watch how your top-rated "competitors" are disposed and focus on your target audience. Then, keep your words in mind, together with your business, and use the tips offered in the following sections.

Have a descriptive URL and domain

Have part of your keyword on your Web site URL, for example: www.riverbendcoffee.com.
To understand how to implement subdomains of your Web site, refer to Installing a Web site.

Use subdomains

Create subdomains to have the URL word and more links, for example:
www.brazilian.riverbendcoffe.com.

Have a descriptive site title

Have your keywords on the title tag. Refer to what the site or the page that the user is visiting is about. Consider the following example:

```
<title>Brazilian Coffee</title>
```

To understand how to change this tag, refer to Common design properties on Web applications.

Add a site description

The Web site description is used by robots to describe your document on search sites. This description should created by using the description meta tag. Most of the search engines restrict to something around 200 characters. Sites with big descriptions are generally discarded from searches. Therefore, consider using a small and efficient description with less than 120 characters. Consider the following example:

```
<meta name="description" content="High quality blended Brazilian coffee. Unique Chinese teas.
Wi-fi hotspots available on kiosks." />
```

To understand how to change this tag, refer to Common design properties on Web applications.

Have keywords listed

Keywords help your site to be identified on by crawlers. Therefore, be careful in choosing the right words. Some sites recommend that you use many words on the tags, but this technique deprecated due to some Web sites using illegal techniques for ranking. Today you use at most three to five words in your keywords tag. It is important to have the words match the page content. Consider the following example:

```
<meta name="keywords" content="Riverbend, Brazilian coffee, Chinese tea" />
```

To understand how to change this tag, refer to Common design properties on Web applications.

Use the content-type meta tag

Content-type is an important meta tag to search engines. This tag informs the crawler of the type of content that is hosted on that page, helping them to categorize their listing. It also shows that you are following World Wide Web Consortium (W3C) guidelines, which is an indication that your have a serious Web site. The following example shows the syntax that is used in this tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

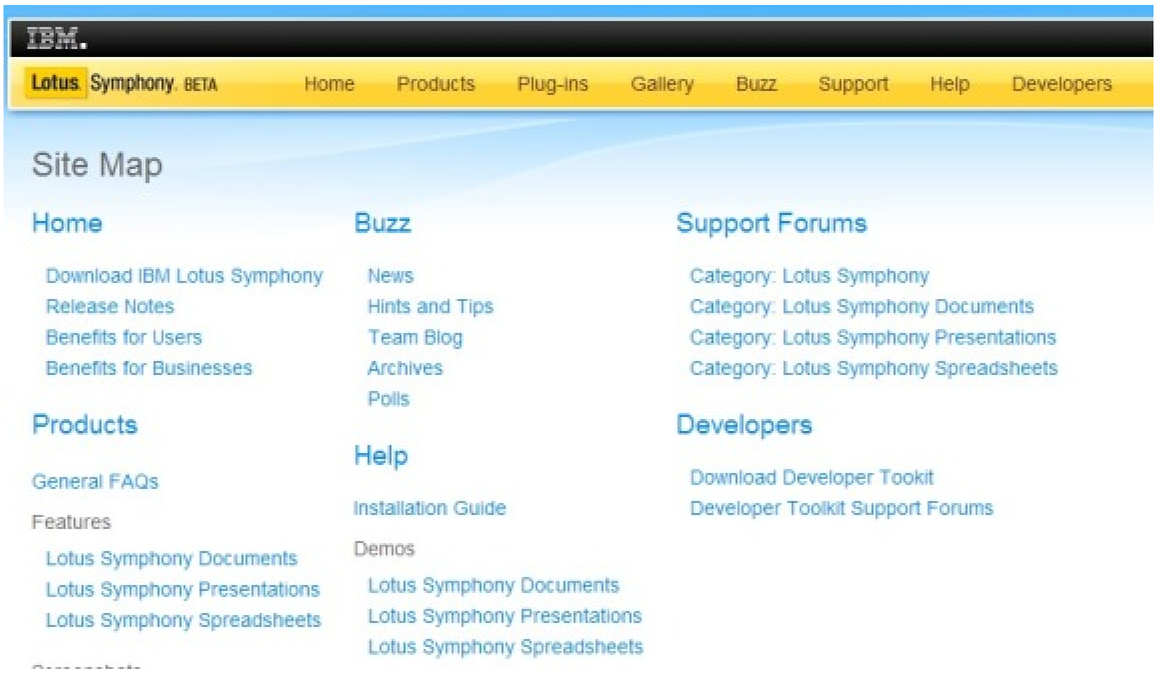
To understand how to change this tag, refer to Common design properties on Web applications. The charset property can also have several different values, according your content type. For further information about this topic, refer to Changing the content type of a design element and Working with the DOCTYPE.

Create links

Have links that point to your pages (back links). If a page is pointing with a term to your page, the search engines understand that your Web site refers to that subject. Make partnerships.

Have a site map

A site map is a page in your site with an index that shows all site pages and their subjects. When the tool craws the site map, it has all the links that comprise your site. This procedure hastens the search engines works, because the more links a crawler has to navigate deeply to find information, the less "valuable" it is going to be ranked.



A site map example from the Lotus symphony Web site

Use the proper text markup

Use the proper markup text in your page. Put header information in headers (using h1, h2, h3 tags), place text in paragraphs (using p tags), and use ordered and unordered lists (using li and ul tags) and tables (table tags) to help spiders understand what is important on a page. Use style sheets to reduce content pages size and help your work. The most important is highlighting the information by using the proper tags according to their importance.

Tip
Rich text content in Lotus Domino is not converted to any special HTML tags when translated. Notes only converts body font sizes to an HTML font size correspondent. It does not create tags such as h1 for example. Therefore, use pass-through HTML to use the proper text styles on a Web page. For further information about the translation process, refer to Styling text for the Web.

Have accessible images

Use descriptive file names for your images and the alt and title attribute of the img HTML tag. This helps indexing, is an HTML standard, and is also an accessibility must-have for people with reading disabilities. Consider the following example:

```
<img src="coffee.gif"  alt="Brazilian blended cup of coffee" title=" Brazilian blended cup of
coffee" />
```

Use intuitive files and directory names

Use intuitive files and directory naming. It is much more human understandable to use a contact page browsing through "/contact/talk+to+us.html" than to use "/cgi/contact.cgi".

Tip
On Domino, use view indexing by name instead of strange identifiers. This technique is described later in the section.

Brand, marketing and type-ins

Brand recognition is a heavy aspect. Having a popular well-known brand helps when searching for a product. When we think about an information technology company, it is easy to think about Web sites such as www.ibm.com. However, in the beginning of the Internet, such Web sites as Amazon and e-Bay invested millions in marketing to have clients knowing their services. Today one of the most common trends is to use type-ins, which are words that have good relation to a Web site business. A good example of type-in is for the domain name creditcheck.com, which was sold by US$3 million. Unless you have a famous brand or an old well-known Web site, type-ins are a highly recommended SEO technique.

Have some link exchanging

Another known method is link exchange. You send traffic to a business partner via links to their site, and your partner gives your traffic via links to your Web site. This kind of reference helps to improve the ranking of each other.

Submit your site to search engines manually

Manually submitting your site to the main search engines is a good technique and increases your Web site reliability. Many companies do not appear on search results because of indexing issues.

Create page clouds

Page clouds are visual depictions of keywords on a page. A tag cloud is composed of several words in links, with higher fonts to most important topics. Tag clouds allow users to check the most relevant topics of a page, using an intuitive interface. Search engines such as tag clouds and good clouding help to rank your Web site pages according to their relevance.

.NET advertising AI Apple attention autism barcamp blog blogs browsing
business models classifieds collaboration compensation
competition data mining decision markets democamp discovery distributed
computing doing business domain names economics
entrepreneurship events feeds free riders free speech friends fun games getting things
done grades happiness hardware how to work infrastructure innovation
intellectual Ventures intention IP iPod marketing markets measurement media
Microsoft money movies names news occupations pain patents performance
incentives personalization philosophy platforms politics power law privacy process
productivity promotion publishing racism reading recommendations regulation
religion search services technology tips torcamp Toronto trolls TVC user
interface utilities VC writing xbox

Example of a tag cloud

Refer to Web 2.0 primer for more information about this topic.

Do a semantic analysis revision

Despite the fact that search engines index each word in a text, they are not used to do semantic analysis. That is, they do not try to interpret a group of words. For example, consider a page that only has the phrase: "This page has information about Brazilian blended coffee - click here." The it could have a smaller ranking than a page that contains the text: "Brazilian blended coffee cafeterias Chinese coffee hotspots - click here." Since this second example is clearly willing to manipulate ranking through the use of word repetitions, the first example is definitely a more reliable information source.

To avoid this technique, search engines are starting to implement semantic analysis on their robots. Therefore, consider doing a semantic analysis revision for avoiding have your site banned from indexes.

Validate your code

Having Web sites that use well-formed and validated markup languages optimizes search engines results. This happens because the search engine parsers need to go into the right tags of your HTML markup code, for example, into the head of the body language of your page. If the language is not well-formed, the crawlers may skip it. The most important search engine optimization code validations are checking HTML, CSS, and ensuring that you have no broken links or images. W3C has a tool that analyzes all of these items at http://validator.w3.org/ . Having this all set will help you optimize your ranking.

Attention
Do not submit pages with sensitive corporation to any validation, translation, or validation services.

Illegal techniques

Other techniques can improve search engine ranking. These techniques are considered dangerous and unethical and are condemned from search engines. Search crawlers are banning and blacklisting Web sites that use these type of malicious techniques. Therefore, we recommend that you avoid them on your Web site. Consider checking if these techniques are being used on your site even if involuntarily. We describe some of these techniques in the following sections.

Adding invisible text

Inserting a keyword in invisible sections is considered spam. This anti-ethical technique is generally created by using text that search engines robots can read, but not people. This includes invisible layers and text with small fonts or that use the same color of the page background.

Duplicating content

Duplicating content using two URLs to same information is another spam technique.

Attention
The issue of duplicating content is involuntarily created on Domino URLs because views create links by using universal IDs. Make sure to use only one type of URL when linking your application. Remember it is better to use www.riverbendcoffe.com/home.nsf/coffe+types+view/Brazilian+Blended+Coffee than to use www.riverbendcoffe.com/home.nsf/view/E382A2533B2467F3651E1347B03C9321. You can avoid this by creating your own links or customizing the view links. For further reference on this topic, refer to View design elements.

Associating words different from the Web site content

A condemned technique is associating words to a Web site that does not match the site content. This technique is often used on the site title, keywords, or description or in the page content. This technique reduces the reliability of your site to its visitors (they will probably not visit it again), and your editors will be classified as spammers. The HTML elements, such as title, meta, h1 and other, were designed for the authors to describe the subject that is discussed. Several spammers exploit the failure by using highly searched terms to expose other media, including pornography and online gambling.

Creating random pages

Spammers create millions of random pages, with slightly variations of density and usage of headers to increase the ability of search engines to search for certain topics. This is one of the most condemned spam techniques for search engines.

Using link farms

In the early 2000s, with the ever-growing concept of link popularity, several Web developers created communities that are willing to link each other, creating link farms. Search engines are starting to cross-reference anti-ethic links and ban these sites from search results.

Domino techniques

There are several techniques, as explained in the following sections, to help your Domino Web site to get optimum results on search engines.

Make the page titles dynamic

Use relevant keywords to create dynamic page titles that display the correct content of a page. Consider using @Formula to do this on the Window title property of a design. This method increases page ranking.

Create descriptive URLs by using keywords instead of the universal ID

Notes documents are accessible on the Web by URLs with their keys in a view, such as ./viewname/key?OpenDocument. In most applications, the key that is used is the document universal ID, which is bad for the following reasons:

- If a user replaces the content of a page by another Notes document, any links to that Web page made by search engines are removed. If a page is referenced by the universal ID and that ID changes, any external links to that link become broken. Since search engines give pages a better ranking based on links, your page will also lose ranking with the search engines if you use IDs on URLs.
- Search engines give a better ranking to pages with keywords in the URL. Which of the following URLs do you think is better for buying blended coffee?
    - www.riverbendcoffe.com/home.nsf/coffe+types+view/Brazilian+Blended+Coffee
    - www.riverbendcoffe.com/home.nsf/view/E382A2533B2467F3651E1347B03C9321

Remember that when linking by using keys, the keys must be unique. If keywords do not help you because of the dynamics of your Web site, try to make your URL IDs permanent by using @Unique.

When calling design elements, consider using the Design element multi-aliasing technique. With this technique, you have a form called "Top rated product vendor|vendor.htm|vendor". This naming convention also provides the benefit of having an alias that works like a traditional Web server "file name" for Domino URLs, helping to keep aliases even if the main page content changes. This convention also helps search crawlers to increase the page rank by using the search engine optimization best practice of having a small description of the page on its URL. When linking, make the links to only one instance of the design to avoid duplicating content.

Create your own links by using HTML for views linking

Notes views use Universal ID linking to document linking by default. Change this code to use your own view IDs instead.

Use "!" instead of "?" on Domino URLs commands to open documents

The part that comes after the question mark ? on a URL is called the *query string*. A CGI variable is available to compute this portion of the URL. When a URL has a query string, search engines consider that page as having content that is generated on-the-fly that could cause the search engine to may fall into an infinite loop while indexing that page.
The problem is that the Domino server publishes all the pages by using the ? as the separator for URL commands. Since some search engines have heuristics to avoid these pages because of malicious scripting or infinite looping, Domino provides a mechanism to replace the ? with an exclamation point (!), such as ...!OpenDocument. To set this on your application, you must adjust the Web site configuration document in the Domino Administrator to give access to search engine site crawlers. This setting does not help hard-coded URLs.

✓ **Tip**
You can also remove the "OpenDocument" URL command without impacting your application, since it is optional. If any code parses query strings because of scripting, make sure that it is compatible with this change.

Avoid using JavaScript functions for linking

Avoid using JavaScript for page linking. Most crawlers do not interpret JavaScript as a Web user and can't navigate deeper on a Web site's structure while crawling. Use HTML <a> tags for menus and links and to submit pages. Use JavaScript only to improve a user's experience on a Web site, but not to build core linking mechanisms.

Create an HTML image resources reference instead of rich text

Since Domino does not properly use SEO references to Notes rich text images, consider using the <img> tag with the alt attribute by using a pass-through HTML to reference them. This method helps the crawler understand to where that link image will point.

Place subforms in order of importance

Search engines consider links that are displayed at the top of a page more relevant than the others listed to the bottom of the page. Therefore, consider using CSS positioning to block like menus or navigators, placing the more relevant code near the top of a Web page. CSS also reduces the total size of a Web page itself, since crawlers do not use spider style sheets.

Complex search engine techniques

In the following sections, we present complex search engine techniques to improve ranking.

Page rank

A *page rank* is a link analysis algorithm that assigns a grade to each element of a group of pages with the purpose of "measuring" its importance within the set. This page rank denotes a site's importance. There are two types of best practices to increase a page page rank by using keywords. One uses votes (links), and the other depends on how the Web site was built.

- Best practices without voting:
    - Use the site keyword in the URL.
    - Use the site keyword on the title.
    - Have no hidden text.
    - Use alts on images.
    - Use static URLs instead if dynamic URLs. Use Domino URLs for view indexing instead or using ID indexing.
    - Create sites with many pages, but with low content (from 700 to 1500 words).
    - Use a good variety of words on the site pages, but with a focus on your business.
    - Update your site regularly. The more a Web site gets updates, the more the robots index your pages.

- Best practices with voting:
    - Have references from other Web sites.
    - Have links from sites that are top ranked with the same words.
    - Have links from one page to another on the same site with the same keyword.
    - Add your site to directories.
    - Reach votes from important Web sites.

sitemap.xml

Search engines want to ensure that users receive the most relevant results as possible. There are programs that help ensure that pages get cached in their indexes as fast as possible. They call this program a site map. A site map is used by the most used search engines available today. Several sitemap.xml file generators are available on the Internet. In Domino, you create your own sitemap.xml file by using a page. Create a page and call it sitemap.xml. Then, change the content type to XML and add content according to your site information. For further help about this topic, refer to Page design elements.

You also can get help in creating your own sitemap.xml file for your Web site at the following address:

http://www.google.com/webmasters/sitemaps/docs/en/about.html.

URL considerations

This page last changed on Apr 02, 2008 by jservais.

- User friendly URLs
- Web site rules

User friendly URLs

It is important to have easier-to-understand Web addresses that make it easy to communicate, remember, and pass on by word of mouth, e-mail and so on. A friendly Web address should be short enough to be pasted in an e-mail without wrapping, which can break it, and should also be easy to type in a browser or mobile device.

By default, Domino produces Web addresses that are lengthy, hard to remember, and can be difficult to type. By using a bit of creativity and Domino Web site rules, you can eliminate most of these URL issues. It's not just Domino, many application frameworks use a query based Model-view-controller (MVC) architectural pattern, which makes the URL look more like machine code. To be fair, by default, Domino builds URLs it knows are unique and can always be resolved. The unique document ID of the document plays a big part, even though 32 character hexadecimal is not easy to work with.

Let's start with an original simple HTML URL such as http://astrology.yahoo.com/chinese

```
http://www.riverbendcoffee.com/directoryname/filename
```

Thanks to Dominos document access URLs

Using Domino URLs to access a document

To open a document by key, create a sorted view with the sort on the first key column. Then you can use a URL to open the document by using the following syntax:

```
http://Host/DatabaseName/View/DocumentName?OpenDocument
```

Where *View* is the name of the view, and DocumentName is the string, or key, that is displayed in the first sorted or categorized column of the view. Use this syntax to open, edit, or delete documents, and to open attached files. Domino returns the first document in the view whose column key exactly matches the DocumentName.

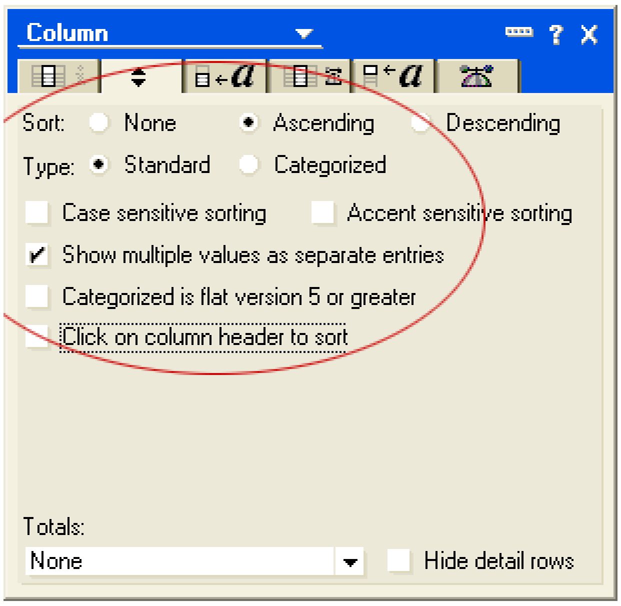
There may be more than one matching document. Domino always returns the first match. The key must match completely for Domino to return the document. However, the match is not case-sensitive nor accent-sensitive.

How does this help and what do I do now

You do not need the ?OpenDocument. Do not use it if you can avoid it. Use the database Replica ID instead of the DatabaseName. Database names, such as directory/help.nsf, are confusing and break if you move the database. The problem is not with links within a database. You can use (Replacesubstring()Subset(@DbName;-1);" ";"/")

How do you find the database name for a companion database you are linking to? Replica IDs can be stored in keyword documents and looked up by the database keyname, for example helpDB. Use @ReplicaID when working internally in Notes for @lookup (Replacesubstring()ReplicaID;":";"") and use in place of the DatabaseName.

Create a simple view for all published documents. Call this view something simple such as "pages", , or create an alias for it. In the View properties box, select **Show multiple values as separate entries** so that alias page names can be created.



Sort: None [✓ Ascending] Descending

Type: [✓ Standard] Categorized

☐ Case sensitive sorting ☐ Accent sensitive sorting

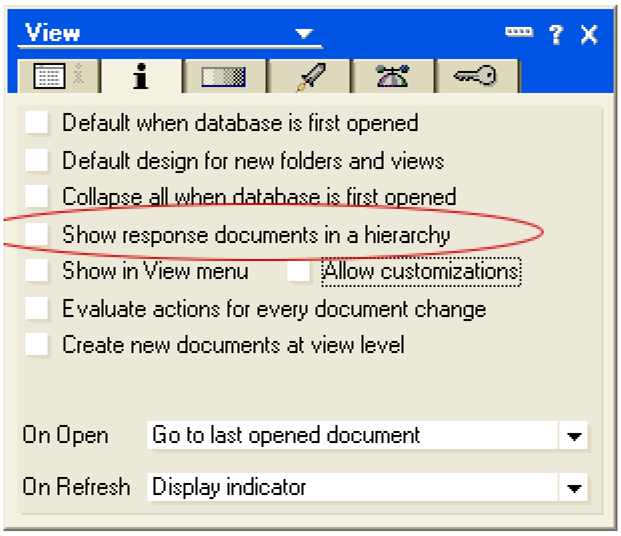
☑ Show multiple values as separate entries

☐ Categorized is flat version 5 or greater

☐ Click on column header to sort

Totals: None ☐ Hide detail rows

Column 1 contains the page name and is sorted in ascending order. Deselect **Show response documents in a hierarchy** to create a "flat" view.



☐ Default when database is first opened

☐ Default design for new folders and views

☐ Collapse all when database is first opened

☐ Show response documents in a hierarchy

☐ Show in View menu ☐ Allow customizations

☐ Evaluate actions for every document change

☐ Create new documents at view level

On Open: Go to last opened document

On Refresh: Display indicator

The page name can be derived from the page title or a page name entered by the user. The page name needs to be unique and meaningful to a user. If your information is hierarchical, you can concatenate the page names, for example, products-accessories-warmers. This makes it easier to keep a page name unique. You can have alias page names by creating lists for example,Trim()Unique(pagename:(pagename+".html"):(pagename+".htm"):anothername).

We now have something that looks like the following example:

```
http://www.riverbendcoffee.com/8025741C007AAC09/pages/products-accessories-warmers.htm
```

Column 2 contains an alternate sort. By using reader fields and duplicate page names, you can present a different view for different users. Since this method always uses the first one it finds in the view, you can stack potentially hidden documents but "fall through" to public documents. One use could be for multiple language support. For example, with the English pages alternate sort, put them on the bottom of the stack. Then non-English speakers can find the document with their group in the reader fields, and people with no language group assigned arrive at the English page, which has no restrictions.

Column 3 contains the unique document ID. This is convenient if you are checking for duplicate page names. A lookup on the pagename can return a unique document ID. If nothing is found or it returns the document ID of the document you are in, then you have no duplicates. Otherwise you have duplicate.

Web site rules

Use Web site rules to hide the Replica ID. In the Domino Directory under **Configuration-> Web-> Internet Sites**, you can set-up rules for your sites. Use the following table to map /pages to your pages view.

Basics	
Description:	Pages
Type of rule:	Substitution
Incoming URL pattern:	/pages
Replacement pattern:	/8025741C007AAC09/pages

We now have something that looks like the following example:

```
http://www.riverbendcoffee.com/pages/products-accessories-warmers.htm
```

You can use Web site rules to deal with the required ?EditDocument or ?OpenForm. Use the following table to map /edit to your pages view so that you can open the document in edit mode.

Basics	
Description:	Edit
Type of rule:	Substitution
Incoming URL pattern:	/pages/*
Replacement pattern:	/8025741C007AAC09/pages/*?EditDocument

We now have something that looks like the following example:

```
http://www.riverbendcoffee.com/pages/products-accessories-warmers.htm
```

Alternatively, you can open a form by name only.

Basics	
Description:	Forms
Type of rule:	Substitution
Incoming URL pattern:	/forms/*
Replacement pattern:	/*?OpenForm

We now have something that looks like the following example:

```
http://www.riverbendcoffee.com/forms/survey
```

You can also use this to allow site developers or administrators to create constant user-friendly URLs and map them to specific documents or views.

For example, consider the following URL:
http://www.riverbendcoffee.com/E7A291052E43215E852567240071E2AD/2A6E67721D53C50D8025741C007BA6E1?O

It can be mapped to this URL:
http://www.riverbendcoffee.com/sale

For more information about Web site rules, see the great DeveloperWorks article Building Web applications in Domino 6: Web site rules

Add more of your URL mapping best practices here.

User management

This page last changed on Apr 03, 2008 by jservais.

Introduction

Most Web applications have different types of users. You might have a technical Web site that has dealers, distributors, and content editors or a CRM Web application that has salespeople, department managers, and executives. Each user group sees a different menu, content, and has a different access level.

Notes developers have been using groups and roles for security (authors/readers field). We set up different groups, add users to members of the groups, and assign the groups specific roles. In addition to security, we also use them to show/hide certain functionality in our application.

Domino administrators would surely benefit from a system where users can register themselves, and once they are registered, they are automatically added to the proper groups. Moreover, the users can change their password and reset their password (if they forget their password).

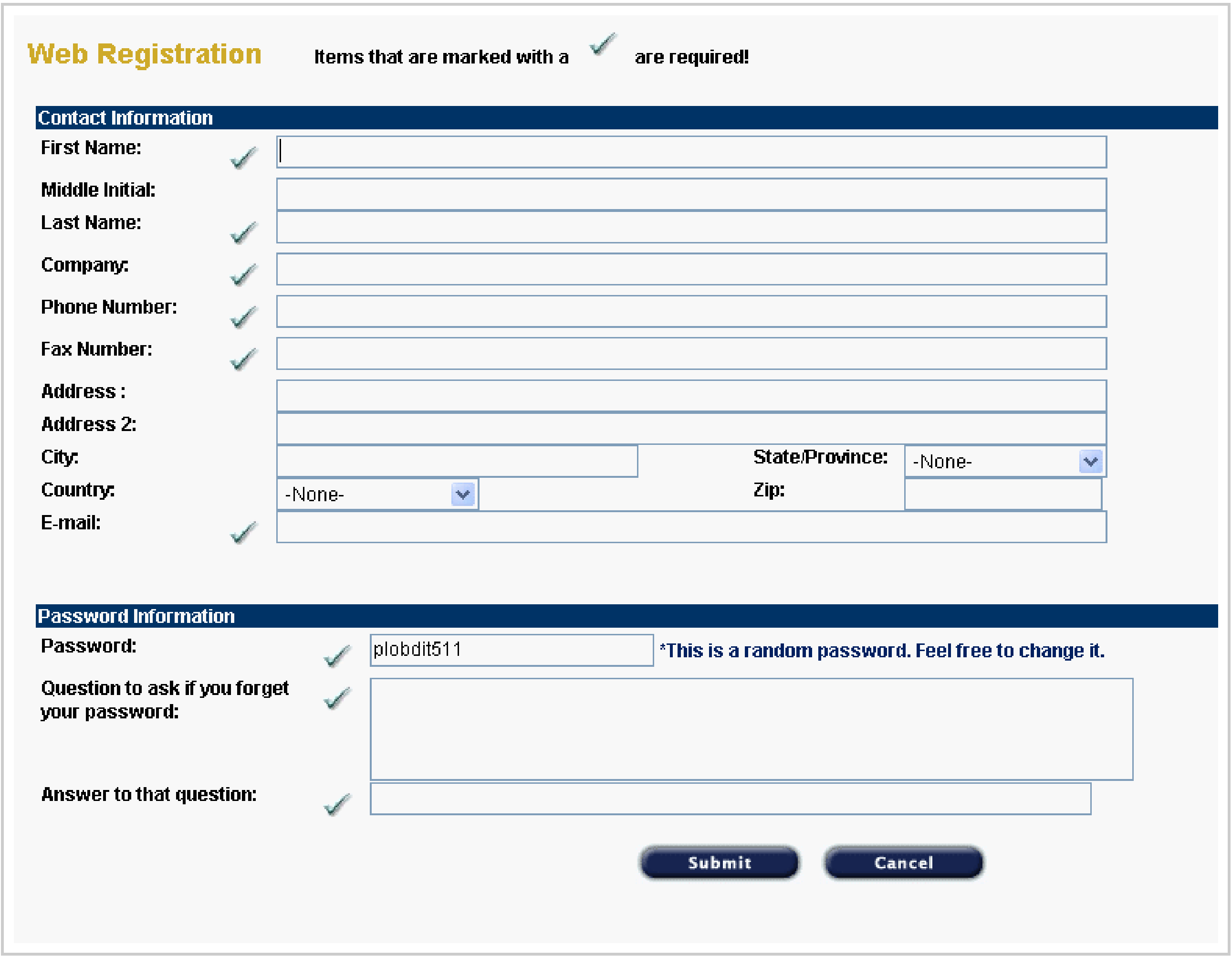
Registration database

The registration database store registration documents and has the following options available to users:

- New registration
- Change password
- Forgot password

New registration

Before users can access the Web site, they must complete a registration form as shown in the following example.



Web Registration   Items that are marked with a ✓ are required!

Contact Information

First Name:	✓		
Middle Initial:			
Last Name:	✓		
Company:	✓		
Phone Number:	✓		
Fax Number:	✓		
Address :			
Address 2:			
City:			State/Province: -None-
Country:		-None-	Zip:
E-mail:	✓		

Password Information

Password:	✓	plobdit511 *This is a random password. Feel free to change it.
Question to ask if you forget your password:	✓	
Answer to that question:	✓	

The form has basic contact and password information. The security question and answer are required for the Forgot Password feature.

You have several options after the registration is submitted depending on your needs. Here are the most common scenarios:

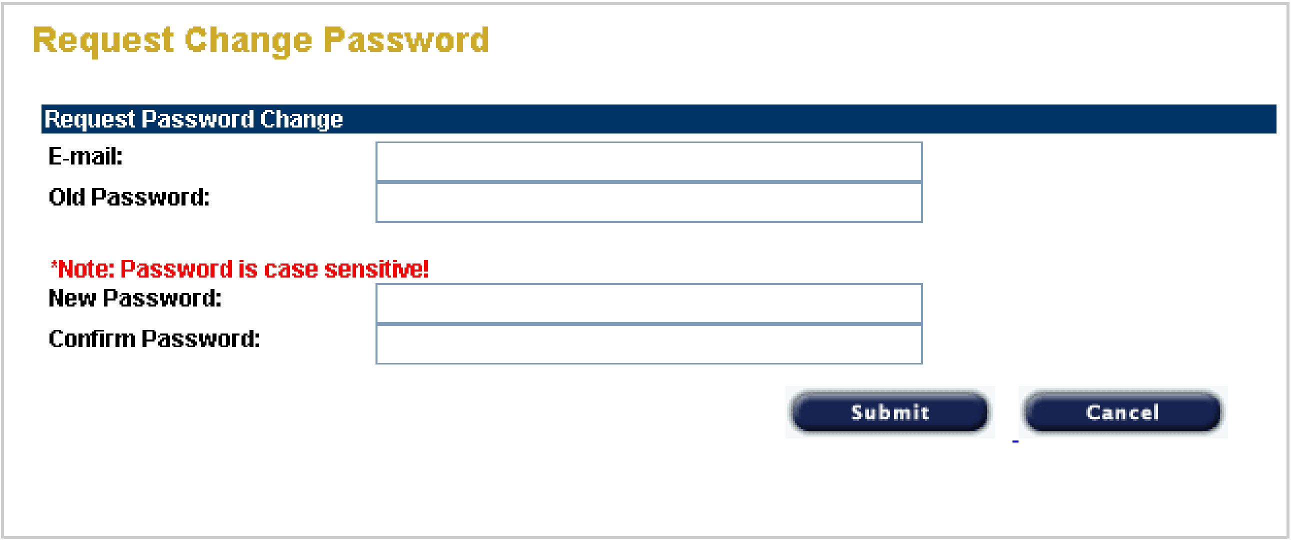
- Immediately add the user to a group in the address book.
- Notify the web application administrators to approve the registration. Once approved, the user is added to a group in the address book.
- Push the registration to a workflow system and route it to the appropriate people. The responsible people can select into which group the user needs to be placed.

In any case, you need a routine to add a user to a group. The following sample code can do that. It takes into account the 15K (around 1000 members) text list limitation (which may be increased in Lotus Notes 8) by creating subgroups under the main group. For example, if the main group is called Dealers, the subgroups are Dealers 1, Dealers 2, Dealers 3, and so on. The Dealers group will contain Dealers 1, Dealers 2, Dealers 3, and so on.

```
Sub AddUserToGroup( Byval fullname As String, Byval group As String, nab As NotesDatabase )

     ' Load up our static groups view

        Dim groups As NotesView
        Set groups = nab.GetView( "($VIMGroups)" )

        Dim groupMainDoc As NotesDocument
        Set groupMainDoc = groups.GetDocumentByKey( group )
        Dim saveGroupMainDoc As Integer
        saveGroupMainDoc = False

        If groupMainDoc Is Nothing Then
                Set groupMainDoc = New NotesDocument( nab )
                groupMainDoc.Form = "Group"
                groupMainDoc.ListName = group
                groupMainDoc.Members = group & " 1"
                groupMainDoc.GroupType = "0"
                Call groupMainDoc.ComputeWithForm( False, False )
                saveGroupMainDoc = True
        End If

        Dim groupMainMembers As NotesItem
        Set groupMainMembers = groupMainDoc.GetFirstItem( "Members" )

     ' Find last subgroup entry in the members list

        Dim subGroup As String
        subGroup = ""
        Forall s In groupMainMembers.Values
                If Left$( s, Len( group ) ) = group Then
                        subGroup = s
                End If
        End Forall

     ' Open the subgroup, and keep trying until we find one with room

        Dim groupNum As Integer
        groupNum = 0

     ' Which subgroup was the last one
        If subGroup <> "" Then
                groupNum = Val( Right( subGroup, Len( subGroup ) - Len( group ) - 1 ) )
        Else
                groupNum = 1
                subGroup = group & " 1"
        End If

        Dim groupSubDoc As NotesDocument

        Do
                Set groupSubDoc = groups.GetDocumentByKey( subGroup )

                If groupSubDoc Is Nothing Then
               ' Create a new subgroup document
                        Set groupSubDoc = New NotesDocument( nab )
                        groupSubDoc.Form = "Group"
                        groupSubDoc.ListName = subGroup
                        groupSubDoc.GroupType = "0"
                        Call groupSubDoc.ComputeWithForm( False, False )

               ' Add it to the main group if needed
                        If Not groupMainMembers.Contains( subGroup ) Then
                                Call groupMainMembers.AppendToTextList( subGroup )
                                saveGroupMainDoc = True
                        End If
                End If

          ' See if the subgroup still has room, if so, we've found our subgroup

                Dim groupSubMembers As NotesItem
                Set groupSubMembers = groupSubDoc.GetFirstItem( "Members" )

                If groupSubMembers.ValueLength < 10000 Then
                        Exit Do
                End If

          ' If no room, try the next one

                groupNum = groupNum + 1
                subGroup = group & " " & groupNum
        Loop
     ' Finally: add the user to the subgroup
        Call groupSubMembers.AppendToTextList( fullname )
        Call groupSubDoc.Save( False, True )

        If saveGroupMainDoc Then
                Call groupMainDoc.Save( False, True )
        End If

End Sub
```

Change password

The following figure shows a sample Change Password form. In this example, we use the e-mail address that is specified to look up the user. If you enable the Change Password feature when the user is logged in, you do not need the e-mail address.



Request Change Password

Request Password Change

E-mail:

Old Password:

*Note: Password is case sensitive!

New Password:

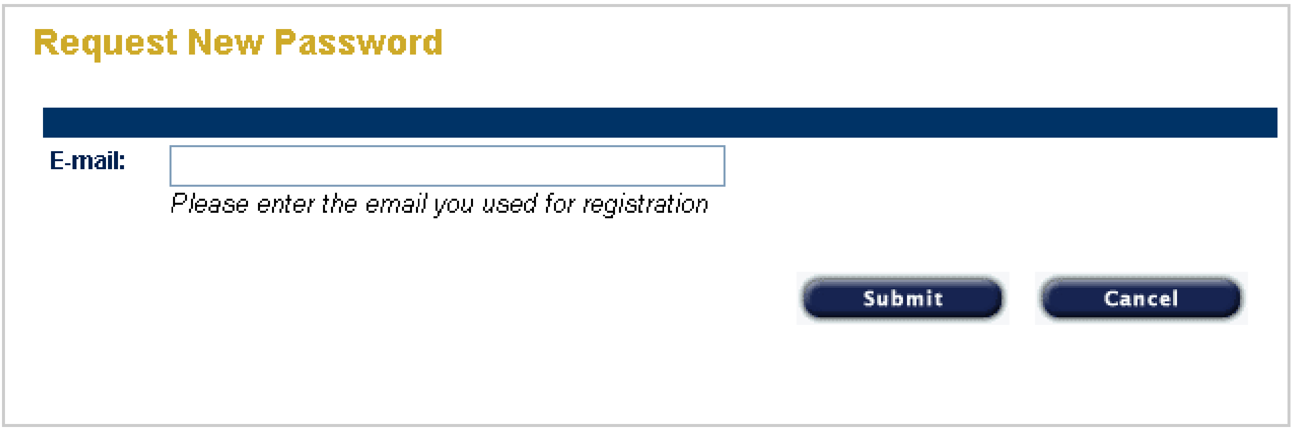
Confirm Password:

Tip

Since the password in the person document is encrypted, we cannot directly compare the old password with the password stored in the person document. However, you can use the @Password formula to encrypt the old password that is typed by the user and compare the value with the password (in the HTTPPassword field) that is stored in the person document.

Forgot password

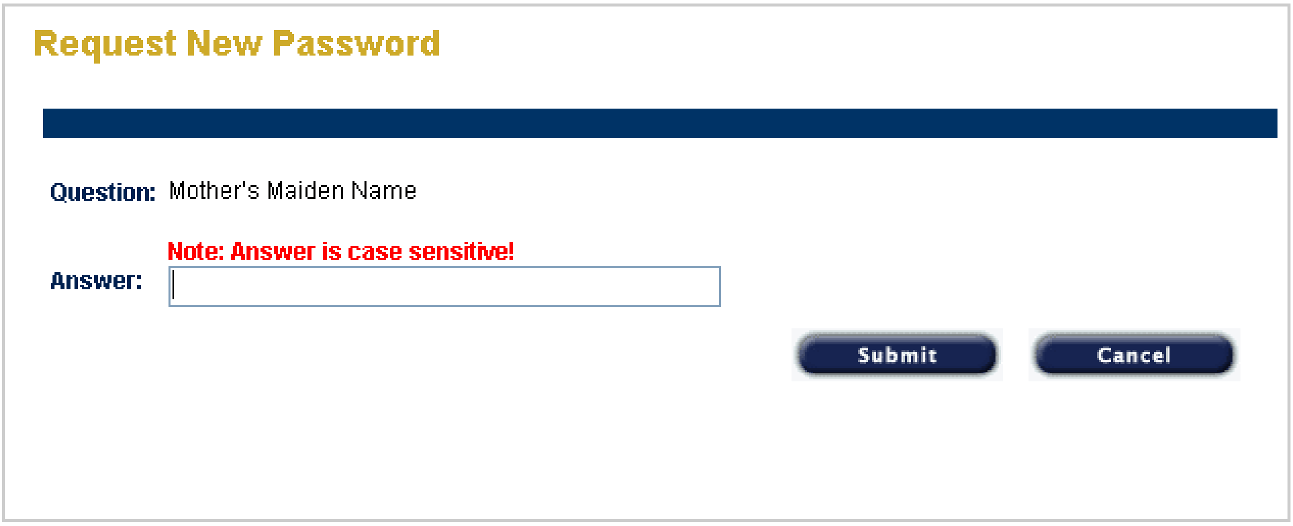
This feature is usually placed on the login form. In the following figure, we use the e-mail address that is specified to make sure the user exists.


Request New Password

E-mail:

Please enter the email you used for registration

If the user is found, we take the user to the second page shown in the following figure.


Request New Password

Question: Mother's Maiden Name

Note: Answer is case sensitive!

Answer:

The security question and answer ensure that the user is really the person who is associated with the account or e-mail address.

Extranet address book

Consider using a separate address book, if you do not want the users of your Web application to be in the same address book as your company's address book.

- Use Directory Assistance to set up the extranet address book.
- Put the groups in the company's address book and the person documents in the extranet address book to ensure that the users can log in if you use single sign-on (SSO), LDAP or both. **Note:** This may not be necessary in the newer version of Domino (7 or later).

Using interactive data and Web services

This page last changed on Apr 02, 2008 by jservais.

Introduction

In this section, we discuss the various *interactive data* models and how we can better use Web services to provide our user audience with rich client experiences for our Domino Web applications.

Web services and Domino development

The Domino Web service engine provides a built-in capacity within the Design Element catalog to create interactive data models for our Domino applications. By using Web services in our Domino development practices, we can deliver and maintain Domino data to more advanced user interfaces, client types, and devices as well as create a merge-point for integration with third-party technologies and solutions.

After we begin to leverage Web services-based architectures in our Domino development, our applications evolve from conventional Domino form design element data maintenance engines to more global community and industry recognized methods. See Web service design elements for information about creating Domino Web service design elements.

Defining APIs for Domino applications

We have the ability to extend our current application development techniques through the utilization of interactive data models and Web services. In the following sections, we discuss the creation of integration APIs for Lotus Notes, Hybrid, and Domino Web applications.

The definition and utilization of a Domino Application API furthers a global community initiative to separate UI and visual representation of data from the data itself. This is often not a concept that is used by Domino developers. However we can immediately see the benefit from the separation of data and UI first through the integration with other technologies and solutions as well as the flexibility that such a separation affords in our solution development.

For example, the maintenance of multiple selected documents from a rendered view is something that can easily be accomplished after we seperate the maintenance of data from the typical visual rendering of the data. For this example, we need the following Domino design elements:

- View Design Element: *markup*

- Form Design Element: *$$ViewTemplate for markup*
- Agent Design Element: *docmod*

We combine these Domino design elements to create a functional document maintenance engine which hopefully illustrates an initiation to separate the data from the user interface.

```
{code:xhtml|title=Form Design Element
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
        <title>Examples - $$ViewTemplate for markup</title>
</head>

<body>
        <form name="viewbody" action="docmod?openagent" method="post">
                <table>
                        <thead>
                                <tr>
                                        <td class="col0"></td>
                                        <td class="col1">Last Modified</td>
                                </tr>
                        </thead>
                        <tbody>
                                $$ViewBody
                        </tbody>
                </table>
        <input type="submit" value="Update" />
        </form>
</body>
</html>
{null}
In the previous code sample, _$$ViewBody_ represents the location of the _$$ViewBody_ Field
Element.

The following combination of&nbsp; _the markup_ view design element and _$$ViewTemplate for
markup_ form design element (each whose contents are set to text/html), render the
results  shown in the following figure when the view design =eElement is displayed in a
Web browser client.

!markup_renderedexample.gif!
{code:xhtml|title=Rendered _markup_ Source Code}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
        <title>Examples - $$ViewTemplate for markup</title>
</head>
<body>
        <form name="viewbody" action="docmod?openagent" method="post">
                <table>
                        <thead>
                                <tr>
                                        <td class="col0"></td>
                                        <td class="col1">Last Modified</td>
                                </tr>
                        </thead>
                        <tbody>

<tr>
        <td class="col0">
                <input type="checkbox" name="docid" value="9A39152EC88316018525741D001F6B1B" />
        </td>
        <td class="col1">
                03/31/2008 01:43:10 AM
        </td>
</tr>

<tr>
        <td class="col0">
                <input type="checkbox" name="docid" value="E071C457345791238525741D001F6B1C" />
        </td>
```

```
		<td class="col1">
			03/31/2008 01:43:10 AM
		</td>
	</tr>

	<tr>
		<td class="col0">
			<input type="checkbox" name="docid" value="F828447B4B47D9E18525741D001F6B1D" />
		</td>
		<td class="col1">
			03/31/2008 01:43:10 AM
		</td>
	</tr>

	<tr>
		<td class="col0">
			<input type="checkbox" name="docid" value="FFCC7DBBA636343A8525741D001F6B1E" />
		</td>
		<td class="col1">
			03/31/2008 01:43:10 AM
		</td>
	</tr>

	<tr>
		<td class="col0">
			<input type="checkbox" name="docid" value="72A9C554B76CD6208525741D001F6B1F" />
		</td>
		<td class="col1">
			03/31/2008 01:43:10 AM
		</td>
	</tr>

	<tr>
		<td class="col0">
			<input type="checkbox" name="docid" value="892A015FD3439C478525741D001F6B20" />
		</td>
		<td class="col1">
			03/31/2008 01:43:10 AM
		</td>
	</tr>

				</tbody>
			</table>
		<input type="submit" value="Update" />
		</form>
	</body>
	</html>
```

We can now design our agent design element to update the submitted collection of Notes documents via their document unique ID, which we can gather from iterating through the Document Context of the Agent Session.

Domino Web application APIs

Since our Domino Web applications have been (presumably) architected with best practice security considerations, the creation of Domino Web application APIs are simply an extension of current architectures. Through the proper utilization of form design elements, view design elements, agent design elements, and Domino URL commands, we can create custom APIs that both render and maintain data in a Domino Web application.

Hybrid client application APIs

Our hybrid client applications have been (again, presumably) architected with best practice security considerations. Therefore, the creation of Domino Web application APIs for solutions are simply an evolution to a more industry recognized best practices of the separation of data from the rendering and representational elements of the data.

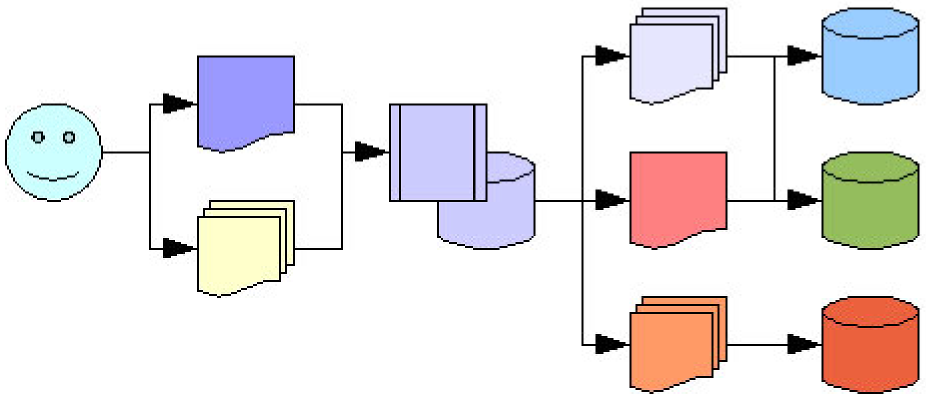
While more advanced architectures and considerations may be required, a hybrid client application, by definition, is design to accommodate multiple client types and therefore different client type environments.

Lotus Notes client application APIs

Creating an API for a Lotus Notes client application delivered via the Domino HTTP Task when the application is currently only Lotus Notes client-accessible allows for an immediate extension of the application to Web browser and mobile client types. For best practices considerations, there are two methods for the creation of a Lotus Notes client application API:

- The evolution of the application to a *hybrid* client application
- The creation of an application middleware/conduit engine architecture to facilitate such an API

When the *evolution* of a Lotus Notes client application to a hybrid client application is either not possible or not necessary, we can implement an application middleware/conduit engine architecture. This type of architecture can act as a communication bridge between the various browser clients and the source data in our Lotus Notes client application. Additionally, this architecture can negate the need to modify the security of your Lotus Notes Client Applications, because they will not be accessible through the browser clients.



In the above figure, we can see the submission of requests for the query and maintenance of Domino data through a middleware/conduit engine architecture. An additional benefit to this approach is the ability to maintain and query data on several different Domino Databases and resources while maintaining a single point of entry (via the middleware/conduit solution).

Working with data

This page last changed on Apr 02, 2008 by jservais.

When you design with the Notes client, you tend to think in a form-based structure. You build views where you have a document per row, and when you click it, that form opens. With Web development, you no longer are restricted to this format. You can build views that take the user to different pages depending upon what they click. You now can work with the Notes data without it being tied to the structure.

In this section, we explore different ways to access information in the Notes databases. Hopefully you can expand your thinking about what you can do with Notes and it will expand more with the future releases of Notes. Explore JSON, play with Dojo, and see what you can do with federated data in query views.

- JSON
- RSS
- Using query views

JSON

This page last changed on Apr 03, 2008 by jservais.

Introduction

Domino has direct support for JSON in views with the &outputformat=JSON parameter. If you want to send information or need additional controls on the information, then you must resort to development. You can use an agent to return or accept JSON and you will have the full control that LotusScript or Java agents provide. If you need the JSON in a different format, then you can use a page to render your JSON.

The JSON object is usually the result of an XMLHTTPRequest sent to a server. This is referred to as Asynchronous JavaScript and XML (AJAX) even though you use JSON instead of XML. All of the popular JavaScript libraries include support for creating the request and processing the resultant string. You pass to the request function the URL that is used to access the server. For example, when getting all the entries under "Riverbend" in a view you can use the following URL:

/database.nsf/viewname?readviewentries&outputformat=JSON&RestrictToCategory=riverbend

This returns a string that contains all the view entries with the category of "Riverbend" in the JSON format.

Getting JSON from Domino views

By using the following Domino URL, the view entries in the JSON format will be returned. You can then access any column or row in the view by accessing the JSON object.

/database.nsf/viewname?readviewentries&outputformat=JSON

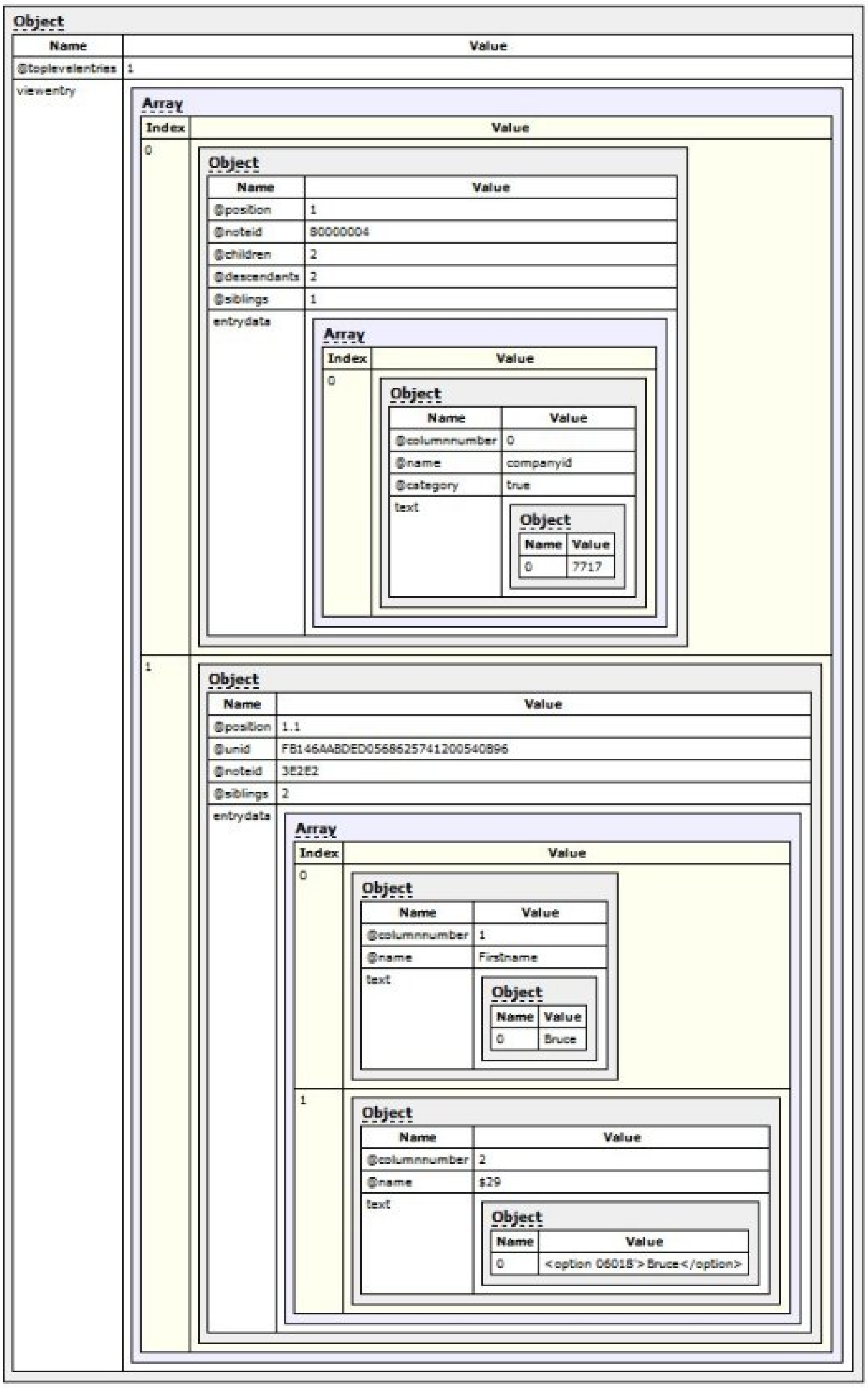
Important

Remember to add &count=xx. Otherwise the number of entries defined in the view properties will be returned. If you first call with &count=0, you can use the value that is returned by @toplevelentries, which is the number of entries or categories in the view.

The following example shows a sample Domino view with categories in JSON.

```
{ "@toplevelentries": "1",
        "viewentry": [
        {       "@position": "1",
                "@noteid": "80000004",
                "@children": "2",
                "@descendants": "2",
                "@siblings": "1",
                "entrydata": [
                {       "@columnnumber": "0",
                        "@name": "companyid",
                        "@category": "true",
                        "text": {"0": "003BE9"}
                }]
        },
        {       "@position": "1.1",
                "@unid": "6AFB146AABDED0568625741200540B96",
                "@noteid": "3E2E2",
                "@siblings": "2",
                "entrydata": [
                {       "@columnnumber": "1",
                        "@name": "Firstname",
                        "text": {"0": "Bruce"}
                },
                {       "@columnnumber": "2",
                        "@name": "$29",
                        "text": {"0": "<option value='006018'>Bruce</option>"}
                }]
        },
        {       "@position": "1.2",
                "@unid": "78053C46688490AD8625741200540B97",
                "@noteid": "3E2E6",
                "@siblings": "2",
                "entrydata": [
                {       "@columnnumber": "1",
                        "@name": "Firstname",
                        "text": {"0": "Melanie"}
                },
                {       "@columnnumber": "2",
                        "@name": "$29",
                        "text": {"0": "<option value='007673'>Melanie</option>"}
                }]
        }]
}
```

To visualize and to help validate your JSON object, you can use the JSON 2 HTML http://json.bloople.net/ Web page. You paste in your JSON object and it displays it graphically or tells you to fix it.



```
Object
Name               Value
@toplevelentries   1
viewentry          Array
                   Index  Value
                   0      Object
                          Name            Value
                          @position       1
                          @noteid         80000004
                          @children       2
                          @descendants    2
                          @siblings       1
                          entrydata       Array
                                          Index  Value
                                          0      Object
                                                 Name            Value
                                                 @columnnumber   0
                                                 @name           companyid
                                                 @category       true
                                                 text            Object
                                                                 Name  Value
                                                                 0     7717
                   1      Object
                          Name            Value
                          @position       1.1
                          @unid           FB146AABDED0568625741200540B96
                          @noteid         3E2E2
                          @siblings       2
                          entrydata       Array
                                          Index  Value
                                          0      Object
                                                 Name            Value
                                                 @columnnumber   1
                                                 @name           Firstname
                                                 text            Object
                                                                 Name  Value
                                                                 0     Bruce
                                          1      Object
                                                 Name            Value
                                                 @columnnumber   2
                                                 @name           $29
                                                 text            Object
                                                                 Name  Value
                                                                 0     <option 06018'>Bruce</option>
```

View of the JSON structure of a Domino view

The JSON from the Domino view has objects and arrays that are defined. The following table lists the objects and their descriptions.

Object name	Description
toplevelentries	The number of top level entries in the view, this would be the number of categories in a categoriezed view.
viewentry	An array of the entries returned
@position	The current entry's position in the view, similar to @docnumber. Examples: 1,2,3.1,3.1.1
@noteid	NoteID of the current entry
@unid	UNID of the current entry
@children	Number of children for the current entry
@descendants	Number of descendantsfor the current entry
@siblings	Number of siblingsfor the current entry
entrydata	An array of the current entry's columns
@columnnumber	Column number starting with 0
@name	The name of the column from the properties box under Programmatic Use.
@category	Is the current entry a category
text	The text returned in the column. If the column had multiple values, each would be an additional object in the text object.

Object names

The format of Object names returned by Domino views requires that you use the array notation to access the data. Domino uses the at sign (@)to indicate object names in the view such as @toplevelentries.

To get the number of entries or categories returned, you can use JavaScript to access the information.

The following example shows sample JavaScript:

```
var text = xhr.responseText;
var data=eval("(" + text + ")");
var top = parseInt(data["@toplevelentries"]);
```

Getting JSON from Domino pages

You can use a Domino Page to provide data in JSON (or any format. Build Domino pages with embedded views (set to html), or computed text (with dblookup or dbcolumn), give it a name (data.json), and finally set the content type to application/json. With the computed-text objects, you can filter the data needed with URL parameters and easily change the data.

This sample is used to pull values for a type-a-head field, where a list of entries is displayed that starts with the characters that the user has typed in. The following sample is based on a category type-a-head field where the user has entered "ad". The @dblookup uses the parameter passed in the URL to filter the result set. You can also use an embedded query view, where the URL parameters are passed to the view and used as part of the SQL query. See the Using query views section.

Page name: AutoSug.json
Content type: application/json
The URL : /db.nsf/autosug.json?open&input=ad

```
{ results: [
<Computed Value>
] }
```

The computed text field has the following formula:

```
ClassCache := "Notes" : "NoCache";
LookupDb := "";
View := "(entry)";
Key := @UrlQueryString("input");
Column := 1;
Temp := @DbLookup(ClassCache; LookupDb; View; Key; Column;[PartialMatch]);

@If(@IsError(Temp); "{id : \"1\", value:\"No Match\", info:\"\"}"; @Implode("{ id:
\""+@Unique(Temp);", "))
```

The following example shows the results:

```
{ results: [ { id: "Administration", value: "Administration", info: "all
Administrationentries"},
        { id: "AdminP", value: "AdminP", info: "all AdminPentries"},
        { id: "Admin", value: "Admin", info: "all Admin entries"}
        ]
}
```

Getting JSON from Domino agents

Using agents allows you to do more complex data manipulation such as combining information from more than one database. You can use Java or LotusScript to build the agent. It does not have to reside in the same database as the data. You can have a central database that provides all JSON data. On the OpenNTF.org website, there are some libraries to read and write JSON data to help accelerate your development. Remember that Domino development is all about collaboration.

The following example shows a sample agent that returns JSON (format like the view) of the first column in a view. In this case, it's a category.

```
Sub Initialize
        Dim session As New notessession
        Dim db As notesdatabase
        Dim doc As notesdocument                ' agent context
        Dim View As NotesView                   ' search view
        Dim vEntries As NotesViewEntryCollection
        Dim vEntry As NotesViewEntry
        Dim strCol As String
        Dim json As String

        Set db = session.currentdatabase
        Set doc = session.DocumentContext

        On Error Goto ErrorHandler

        Print |Content-type: application/json|
        Set view=db.GetView("vCat")
        Set vEntries = view.allentries
        If vEntries.Count = 0 Then ' NO DOCS FOUND   ERROR
                json =  |{toplevel:[{"Status":"No Documents Found"}]}|
                Print json
                Goto TheEnd
        End If
        json =  |{"@toplevelentries":"| & vEntries.Count & |","viewentry":[|
        Set vEntry = vEntries.GetFirstEntry
        While Not(vEntry Is Nothing)
                strCol = vEntry.Columnvalues(0)
                Set vEntry = vEntries.Getnextentry(vEntry)
                If vEntry Is Nothing Then
                        json = json + |{"entrydata":[{"text":{"0":"|+ strcol + |"}}]}|
                Else 'need the , at the end if not the last entry
                        json = json + |{"entrydata":[{"text":{"0":"|+ strcol + |"}}]},|
                End If
        Wend
        json = json +  |]}|

        Print json
        Goto TheEnd
ErrorHandler:
        json =  |{toplevel:[{"ERROR":Error$ & "    Line#: "   &  Erl & |   Object: | &
Lsi_info(2)}]}|
                Print json
        Resume TheEnd
TheEnd:
End Sub
```

RSS

This page last changed on Apr 02, 2008 by jservais.

Domino and RSS

Domino has always had the capacity to provide RSS feeds. There are to basic processes for getting RSS from Notes documents:

- RSS using views and view templates
- Domino RSS Feed Generator database

The Domino RSS Feed Generator database is a quick way to get an RSS feed set up on an existing database. Without customization, the application is simple and not very flexible after you get beyond the basics. It has good support for <enclosure> tags and supports iCalendar or vCard objects. Immediately it doesn't seen to feed HTML in descriptions, which is a popular way of getting adds and images into RSS feeds. If you find a way to do this, updat this wiki with your solution. Otherwise you have to build your own RSS view.

Domino RSS syndication

The following excerpt is from the Help About Using in the Domino RSS Feed Generator database. We included it to give you some idea of the scope of this application.

The RSS Feed Generator database contains a collection of agents and script libraries designed to produce RSS feeds for views in Domino databases, including:

- E-mail, Calendar, and Contact entries from a user's database
- Corporate contacts
- Discussions

RSS feed generator databases have three primary functions:

- Map fields in Domino databases to RSS XML elements
- Generate the RSS feeds
- Syndicate (advertise) feeds

The rss_generator.ntf is a server-based template, and it can only be used on a Domino server. Databases created from the template must reside on the server, and they can only access and generate feeds for databases co-located on the same server.

Databases that are created from the template must include Anonymous in the access control list (ACL), with Reader-level privileges, if the RSS feed database is to be used by anyone. Session authentication must be disabled.

User-based RSS feed databases can be created with server security set to 'Basic' authentication mode and Anonymous is set to 'No access.' When users open the database, the home page redirects users to an "Available Feeds" page, which advertises all feeds listed in the RSS Feed Definitions view.

Views

Create and edit RSS feeds in the RSS Feed Definitions view. The order of the feeds in the view is the order in which they appear in the feed list. You can change the order of the feeds in this view, by moving them up or down the list.

Use the External Web documents view to collect and advertise Web-enabled documents as RSS items. For example, you might use this view to aggregate important company documents sucha s Human Resources PDF forms, corporate policies, and other documents that may not be part of a Domino view.

Setting up an RSS feed generator database

Create the database from the template, as described in Domino Administrator Help. In the RSS Field Definitions view, click **New Feed**. The New RSS Feed Definition form is displayed. Choose the database to serve up as an RSS feed.

Field	**Action**
Database type	Choose the type of database for which to create the RSS feed: • Derived mail database (calculated per user) • Other, common database: The database type determines some aspects of the behavior of the feed. For example, if you create a feed from a mail file, the user's name is displayed as the feed name in the database.
Database	Choose the database for which to create the feed.
View	Choose the database view that will be used to create the feed.

Next, configure the following mandatory fields for the RSS feed description. These map to required XML tags in the RSS file.

Field	**Action**
<title>	Specify the title of the feed, which will appear on the feed list in the database.

<description>	Provide a brief description of the feed. This also appears on the feed list.
<language>	The default is English.
<encoding>	The default is UTF-8.

Finally, configure these optional fields to further refine the RSS feed description.

Field	**Action**
Number of entries to be emitted	Specify the number of items that appear in the RSS feed.
Convert Domino names to RFC822 email addresses	Have Domino-formatted user names (for example, John Doe/West/Acme) be converted to standard Internet e-mail addresses (for example, johndoe@acme.com).
Add additional tag libraries	Specify the location of other tag libraries you want to use
Treat this view as an ordered list?	If you specify this for an RSS feed, this means that RSS readers will remove items that do not fit the criteria of an ordered list. A common example of an "ordered list" is a top-10 feed of best selling books. **Note**: Contacts and Calendar feeds should be configured as ordered lists.

You can also add additional XML information at the bottom of the form to further define your RSS feed. For example, for podcasts, you would use the <enclosure> tag to advertise an audio or video file.

The database also has an internal function to create iCalendar or vCard objects when the <enclosure> tag is associated with an RSS item:

- iCalendar and vCard objects can be retrieved by examining the "URL" property of the <enclosure> tag.
- Since most RSS readers do not process enclosure tags, sending a link to the iCalendar or vCard object reduces the amount of unnecessary data delivered to the client.

You can configure global options for the RSS feed. Select the feed in the view and click **Global Options**. From here, you can choose either of the following options:

- Change the look of the main page of the RSS generator database, including the header.
- Change the look of the redirect page, as well as the default redirect time.

Viewing and using the feed database

Then when Internet users first access the database, they are redirected to the feed list page. They click a feed to add it to a feed reader or aggregator.

RSS using views and view templates

By using a Notes view and a $$Viewtemplate, you can provide an RSS feed. RSS XML can be stored in the document for performance, or it could be calculated in column formula if your RSS is evolving quickly.

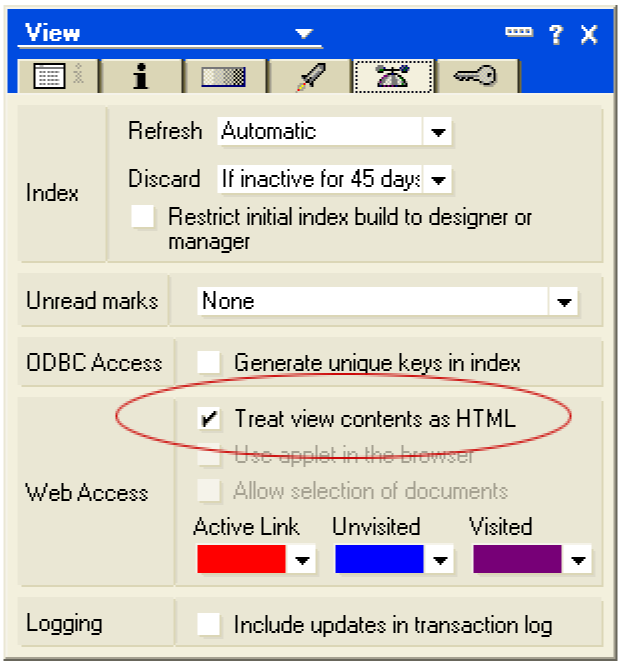
Computed field "RSS" in the form or formula on view column (for documents in the feed)

```
"<item>"+@NewLine
+"<title>"+Subject+"</title>"+@NewLine
+"<description>"+Description+"</description>"+@NewLine
+"<link>"+'link to document'+"</link>"+@NewLine
+"<guid>"+'link to document'+"</guid>"+@NewLine
+"</item>"+@NewLine
```

The RSS view

In the RSS view, set your view selection for the documents you want to publish.

- First column: Date Posted, @created, or @modified, sorted descending, hidden
- Second column: RSS (field) or column formula (above)



Index: Refresh Automatic; Discard If inactive for 45 days; Restrict initial index build to designer or manager ☐

Unread marks: None

ODBC Access: ☐ Generate unique keys in index

Web Access: ✓ Treat view contents as HTML; ☐ Use applet in the browser; ☐ Allow selection of documents

Active Link 🟥 Unvisited 🟦 Visited 🟪

Logging: ☐ Include updates in transaction log

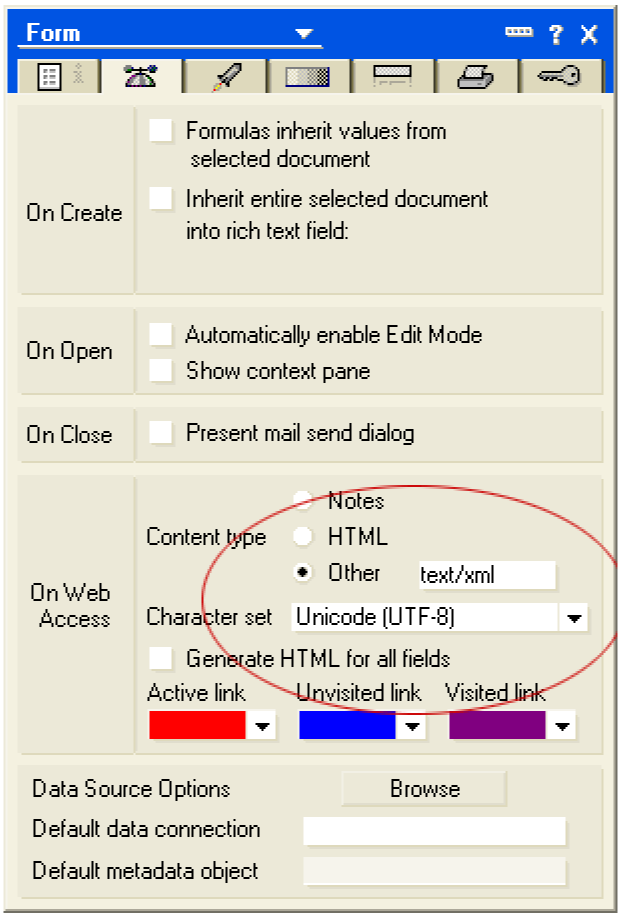
The Treat view content as HTML option

The RSS $$ViewTemplate

Create a form "$$ViewTemplate for RSS". In the form properties, complete the following actions:

1. Change the Character set to UTF-8

2. Set the Content type to other: "application/rss+xml".
3. Test with "text/xml" because with "application/rss+xml" most browsers prompt to add the feed rather than display the page.



On Create	☐ Formulas inherit values from selected document ☐ Inherit entire selected document into rich text field:
On Open	☐ Automatically enable Edit Mode ☐ Show context pane
On Close	☐ Present mail send dialog
On Web Access	Content type: ○ Notes ○ HTML [✓ ● Other] text/xml Character set: Unicode (UTF-8) ☐ Generate HTML for all fields Active link 🟥 Unvisited link 🟦 Visited link 🟪

Data Source Options: Browse
Default data connection:
Default metadata object:

The feed description

This information pertains to the feed not the entries. On "$$ViewTemplate for RSS", complete the following actions:

1. Provide the RSS XML to identify the feed.
2. Use computed text or computed field to retrieve a description and so on.
3. Add a $$ViewBody field to get the RSS view entries.

```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
[<computed text>]
[$$ViewBody]
</channel>
</rss>
```

In the computed text or computed field, enter the following information:

```
"<title>"+'lookup subject'+"</title>"+@NewLine
```

```
+"<link>"+'link to database'+"</link>"+@NewLine
+"<description>"+'lookup description'+"</description>"+@NewLine
```

The result

The following example shows the result in valid XML:

```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
<title>Document One</title>
<link>http://localhost/rss/documentone</link>
<description>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam fermentum
vestibulum est. Cras rhoncus.
Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.
Sed quis tortor. Donec non
ipsum. Mauris condimentum, odio nec porta tristique, ante neque malesuada massa, in dignissim
eros velit at tellus. Donec et
risus in ligula eleifend consectetuer. Donec volutpat eleifend augue. Integer gravida sodales
leo. Nunc vehicula neque ac
erat. Vivamus non nisl. Fusce ac magna. Suspendisse euismod libero eget mauris.</description>
<item>
<title>Document Two</title>
<link>http://localhost/rss/documenttwo</link>
<description>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam fermentum
vestibulum est. Cras rhoncus.
Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.
Sed quis tortor. Donec non
ipsum. Mauris condimentum, odio nec porta tristique, ante neque malesuada massa, in dignissim
eros velit at tellus. Donec et
risus in ligula eleifend consectetuer. Donec volutpat eleifend augue. Integer gravida sodales
leo. Nunc vehicula neque ac
erat. Vivamus non nisl. Fusce ac magna. Suspendisse euismod libero eget mauris.</description>
</item>
<item>
<title>Document Three</title>
<link>http://localhost/rss/documentthree</link>
<description>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam fermentum
vestibulum est. Cras rhoncus.
Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.
Sed quis tortor. Donec non
ipsum. Mauris condimentum, odio nec porta tristique, ante neque malesuada massa, in dignissim
eros velit at tellus. Donec et
risus in ligula eleifend consectetuer. Donec volutpat eleifend augue. Integer gravida sodales
leo. Nunc vehicula neque ac
erat. Vivamus non nisl. Fusce ac magna. Suspendisse euismod libero eget mauris.</description>
</item>
</channel>
</rss>
```

Remember to switch the content type of the $$ViewTemplate back to to "application/rss+xml" if you have been testing it as "text/xml" and validate it by using a service like http://www.feedvalidator.org/ .

Using query views

This page last changed on Apr 03, 2008 by jservais.

To use query views, the Domino server must be using DB2 as the back end and have the DB2 Access server installed.

By using query views on the Web, you can access any data that is stored in the DB2 tables. By passing information to the view, you can build dynamic search queries. You can use @UrlQueryString("ParmName") to retrieve any parameter that is passed to the view. The queries can return fields that are not limited to the ones that the developer created in the view. Unlike normal Notes views that have the query and column values hard coded at design time, you can manipulate the query and return fields. This means that you can build one view that lets you pull the customer information, department information, or book descriptions. The only requirement is that a particular column can only return one data type. If the column returns a text string, then it cannot return a number with a different query, but it could if the number can be displayed as text.

Note
Query views are designed so that you can only build an SQL statement that produces a result set. This is a security measure against inadvertent record deletion or change.

Gotchas

Currently the query views that are done show the view total for columns with total selected. Domino Date fields that have the default time component may show the date off by one day in the view.

Error messages are not very clear. If you get one, first check the fieldname in the SQL query. Then check the DB2 table access. These are the biggest source of errors.

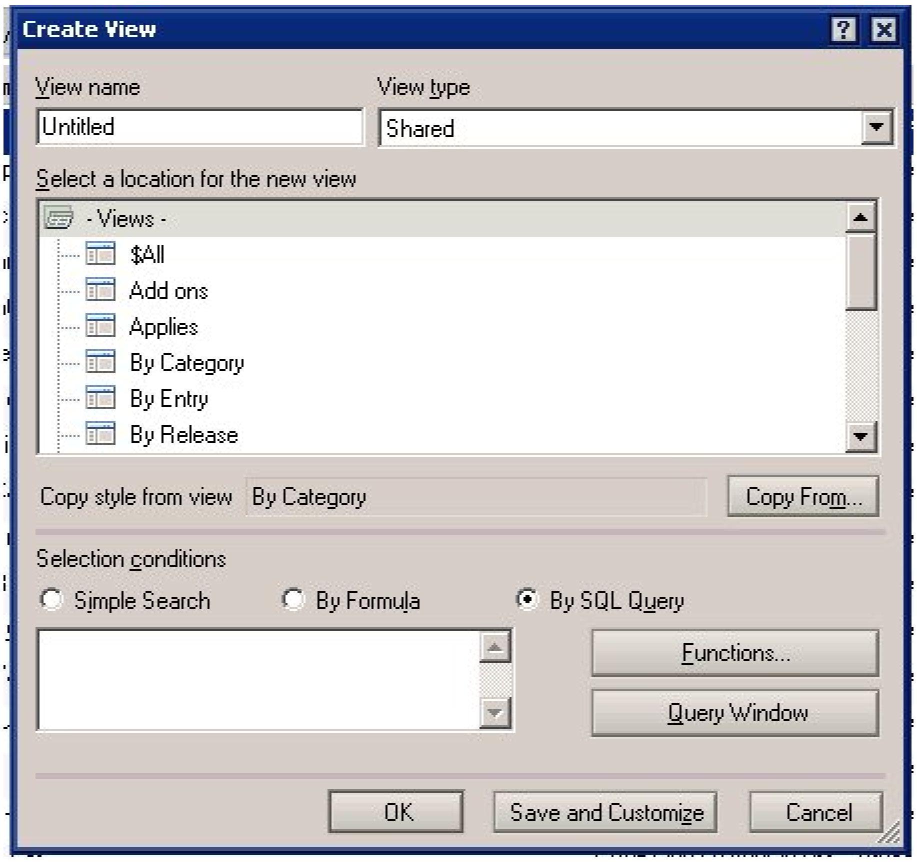
The Query views excel in response time for certain types of queries. If you want to show all 100,000 documents, then a Notes view is faster because the index is already built. However, if you want specific documents, such as all documents created last quarter, then the Query view is significantly faster. Since on the Web, you normally only show a few rows (30 to 50) at a time. In this case, Query views do help.

The following table shows timing results. It shows Notes views versus query views to retrieve documents from a database with 100,000 documents and having a result set of about 1000 documents.

Selection	Notes (indexed)	Query view
Date < 1/1/2007	7 min	3 sec
Field = xxx	3 min	6 sec
range	28 min - first opened	3 sec

The Query view is also smaller since the view index is not stored. Remember that view options such as sortable columns, categories, multiple columns sorted, calculations and extending the column, all cause the view to be slower regardless of the type. See the View design elements section for more information about views.

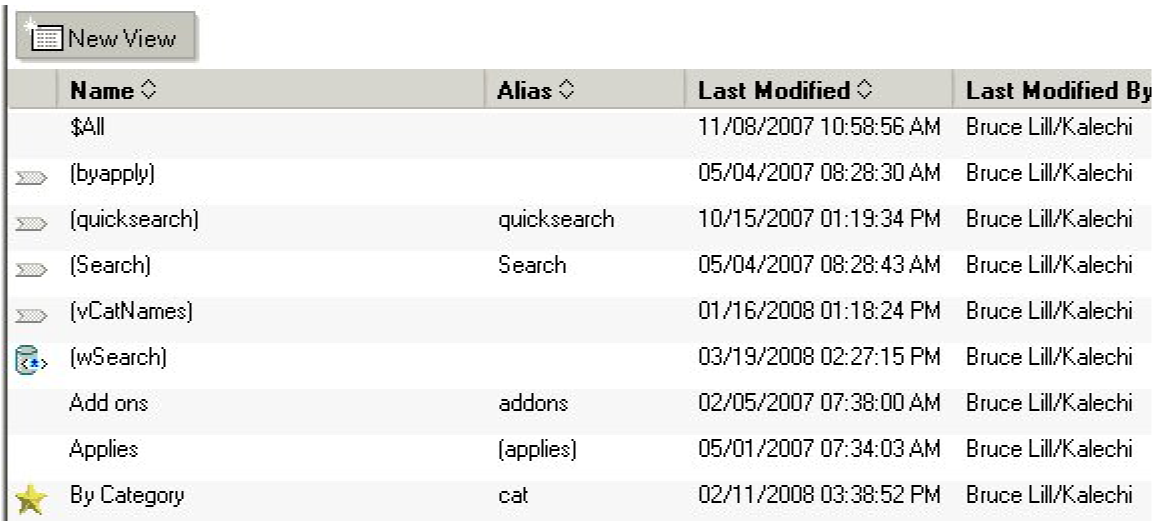
When you create query views, in the Create View window (in the following figure), you must select **By SQL Query** for Selection conditions. This is the only way to have a query view.



Create View window

Tip
When you create a query view, select **By SQL Query** last. When you select a view to copy the design from, it overwrites the Selection conditions.

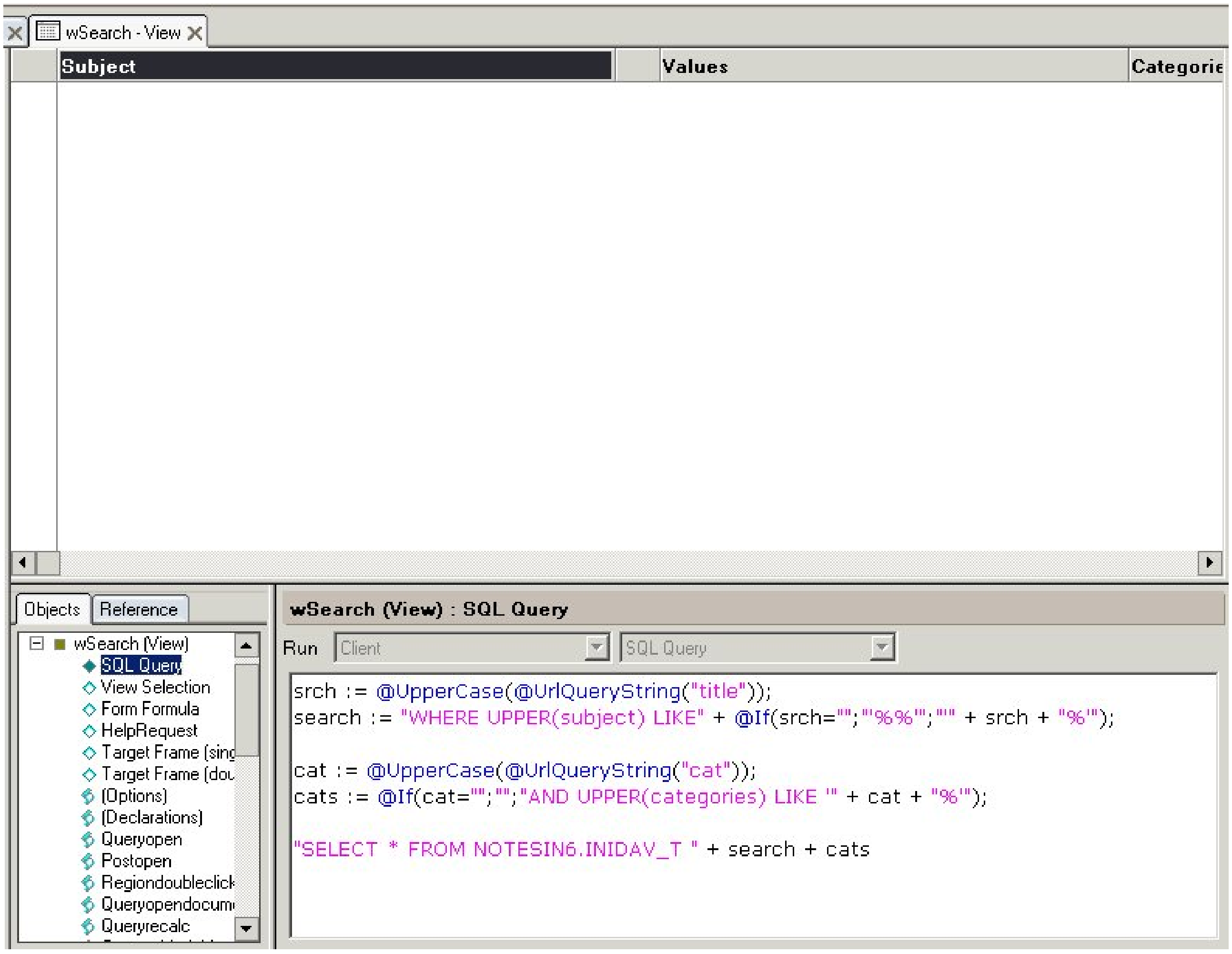
In Designer, the view has a different icon in front of it to indicate that it is a query view.



	Name	Alias	Last Modified	Last Modified By
	$All		11/08/2007 10:58:56 AM	Bruce Lill/Kalechi
	(byapply)		05/04/2007 08:28:30 AM	Bruce Lill/Kalechi
	(quicksearch)	quicksearch	10/15/2007 01:19:34 PM	Bruce Lill/Kalechi
	(Search)	Search	05/04/2007 08:28:43 AM	Bruce Lill/Kalechi
	(vCatNames)		01/16/2008 01:18:24 PM	Bruce Lill/Kalechi
	(wSearch)		03/19/2008 02:27:15 PM	Bruce Lill/Kalechi
	Add ons	addons	02/05/2007 07:38:00 AM	Bruce Lill/Kalechi
	Applies	(applies)	05/01/2007 07:34:03 AM	Bruce Lill/Kalechi
★	By Category	cat	02/11/2008 03:38:52 PM	Bruce Lill/Kalechi

View list in Designer

Now when you open the view in Designer, you have an new object called *SQL Query* in the object list. The following figure shows where you build your query.



```
srch := @UpperCase(@UrlQueryString("title"));
search := "WHERE UPPER(subject) LIKE" + @If(srch="";"%%";"'" + srch + "%'");

cat := @UpperCase(@UrlQueryString("cat"));
cats := @If(cat="";"";"AND UPPER(categories) LIKE '" + cat + "%'");

"SELECT * FROM NOTESIN6.INIDAV_T " + search + cats
```

Query view in Designer

The tables that you normally access are created from DB2 Access Views. The DB2 Access Views table name is created by the server by using the database name that DB2 Access Views is in, along with the DB2 Access Views name. For example, the table CRM_SUP8.TICKETDAV_T is from the crm_support.nsf file where the DB2 Access View named TicketDAV resides. The server takes the first seven characters of the file name, adds a number, and then the DB2 Access Views name. The table with the **_T** at the end is the data, and the table with **_X** is the index.

You can also access multiple tables and display the information in the same view. You are not limited to just data from DB2 Access Views. You can access data in any DB2 table. You can have a view that shows information that is combined from other databases. Using @dblookup is faster with the view in the current database than accessing another database. Therefore, put a query view in the current database to provide the information that is needed. You can also filter it so that only the information that is needed for the database is in the view, making it smaller and faster.

Important
The table name is not guaranteed to be the same if you use a template on a new server. Also you must click the **Create** and **Populate** buttons manually to access the table that is created from the DB2 Access View.

The following code allows a DB2 Query View to be composed dynamically, by passing the Department information as part of the URL.

```
URLParam := @UrlQueryString("Dept");
Clause := @If( URLParam ="" ; "" ; " AND DeptName='"+URLParam+"'");

"SELECT D.DeptID , D.DeptName As DeptName , E.DeptID , E.Lastname AS Lastname FROM " +
  T1 +"  AS D, " + T2 + " AS E WHERE D.DeptID=E.DeptID" + Clause
```

You can get quite complex. The following code example lets a Web user search for support tickets by ticket number, company name, status, or assigned name and have the results displayed in the view categorized by which ever field they want. You can now use one view instead of multiple views. The following example shows the sample SQL query:

```
CatField := @UrlQueryString("cat");
status := @UrlQueryString("stat");
ticket := @UrlQueryString("tkt");
user := @ReplaceSubstring(@UrlQueryString("user");"_";" ");
cmpy := @LowerCase(@ReplaceSubstring(@UrlQueryString("cmp");"_";" "));

company := @If(cmpy="";""; " AND LCASE(COMPANYNAME) LIKE '%" + cmpy + "%' ");
ufilter := @If(user="";""; " AND ASSIGNMENT LIKE '%" + user + "%' ");
filters := "Where STATUS_DESC='" + Status + "' " + company + ufilter;

stat := @If(ticket !=""; "Where TICKETID LIKE '%"+ticket+"%' "; filters);
cat := @If(CatField ="";" STATUS_DESC ";CatField)+ " as db2Cat, ";
"SELECT " + cat + " #NOTEID, #UNID, COMPANYNAME, CONTACTNAME, DESCRIPTION, MODIFIED_DATE,
ASSIGNMENT, STATUS_DESC, TICKETID" +
" FROM CRM_SUP8.TICKETDAV_T " + stat
```

The first column is categorized and uses the following column formula:

```
@If(db2cat ="";"* Not categorized";@Name([CN];db2cat));
```

With this example, the categorized column will be the fieldname that is passed as the &cat parameter in the URL. Therefore, you can have it categorized by contact, status, company, or by any field in the table.

Filtering is similar. You can show only open tickets, tickets for a selected company, or tickets for one engineer, and only one view is needed. If you add some Javascript and JSON, you then have an interactive page that provides a dynamic list of tickets.

5.0 Extending rich client applications for Web clients

This page last changed on Apr 03, 2008 by jservais.

Web-enabling a Domino Database

If you are lucky, you will be given detailed, well architected requirements when designing a new Domino application. If the application is intended to be accessed by both a Lotus Notes client and Web browsers, you will have the luxury of designing for this dual-use up front. However, more often you will be asked to Web-enable an existing Lotus Notes application and make it functional on the Web.

There are some benefits to extending a Lotus Notes application to the Web, as well as some potential pitfalls. However, if you carefully define the requirements for browser access of the application and are aware of the different ways Notes clients and Web browsers handle data, you can avoid the pitfalls and take advantage of the benefits.

Topics in this section

Benefits and pitfalls of extending rich client applications for Web clients

This page last changed on Apr 03, 2008 by dalandon.

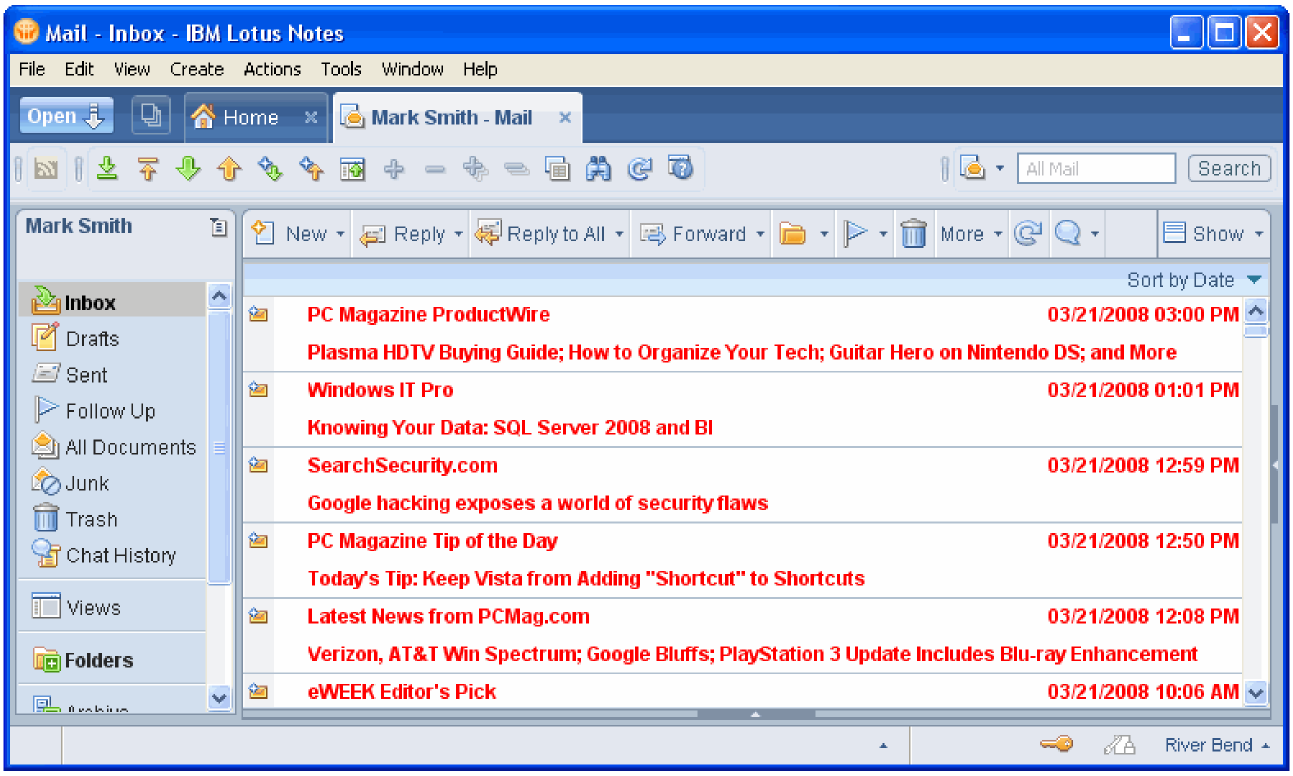
Introduction

Frequently, you are asked to take an existing Lotus Notes application and Web enable it so that users can access the application with a browser in addition to a Lotus Notes client. There are both benefits and pitfalls to having an existing rich client application as a starting point rather than having to create a Web application from scratch.

Benefits

The chief benefit of taking an existing application to the Web is that the application functionality should already be defined. The business logic and workflow are typically already in place and familiar to users. There may even be a requirements document or functional and technical specifications documents for the application. Consequently, you can focus on the Web design and development, and not be distracted by other tasks.

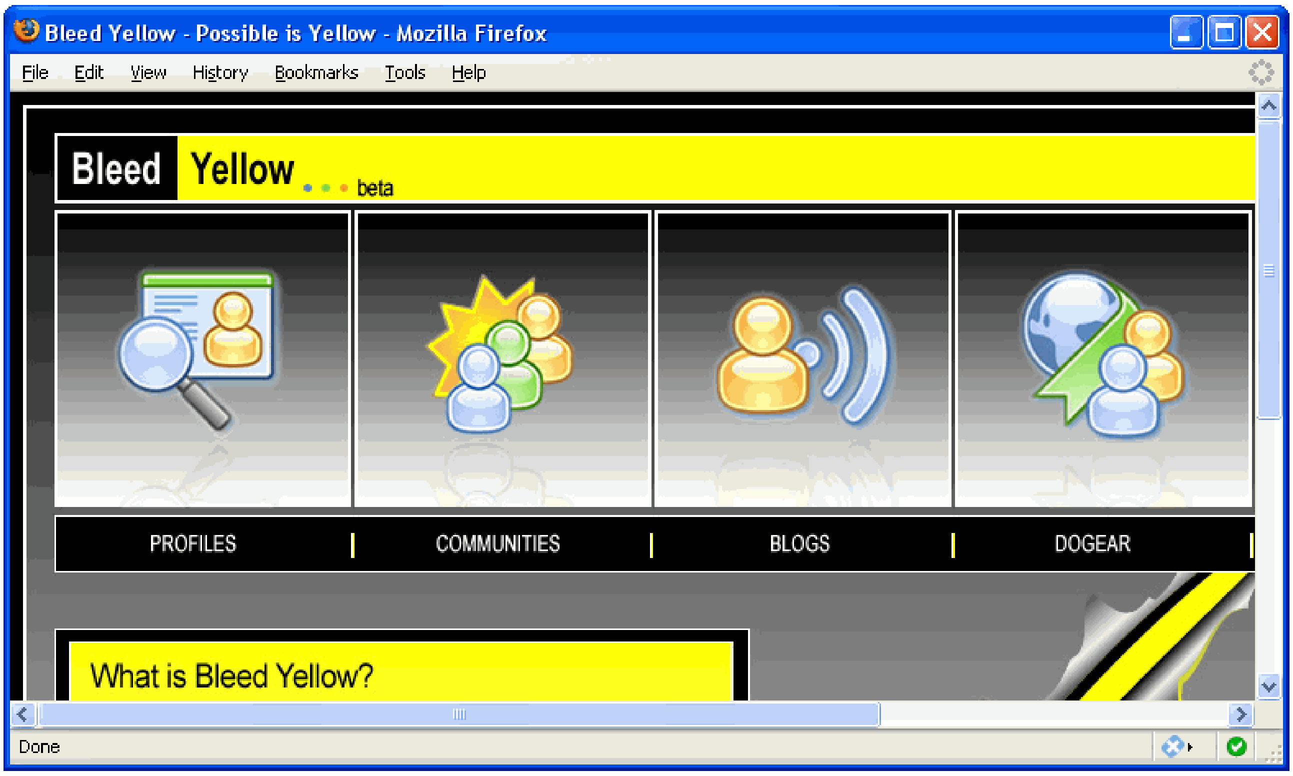
Another benefit of Web enabling a Lotus Notes application is that when creating the browser interface, you are not limited by the rich client UI. Most Lotus Notes applications have a consistent navigation scheme created with a frameset that contains an outline in the frame on the left and the selected content displayed in a frame on the right. Certainly, there is nothing wrong with this consistency among Notes applications, and it can be a benefit to users when they first work with a new database.



PC Magazine ProductWire 03/21/2008 03:00 PM
Plasma HDTV Buying Guide; How to Organize Your Tech; Guitar Hero on Nintendo DS; and More
Windows IT Pro 03/21/2008 01:01 PM
Knowing Your Data: SQL Server 2008 and BI
SearchSecurity.com 03/21/2008 12:59 PM
Google hacking exposes a world of security flaws
PC Magazine Tip of the Day 03/21/2008 12:50 PM
Today's Tip: Keep Vista from Adding "Shortcut" to Shortcuts
Latest News from PCMag.com 03/21/2008 12:08 PM
Verizon, AT&T Win Spectrum; Google Bluffs; PlayStation 3 Update Includes Blu-ray Enhancement
eWEEK Editor's Pick 03/21/2008 10:06 AM

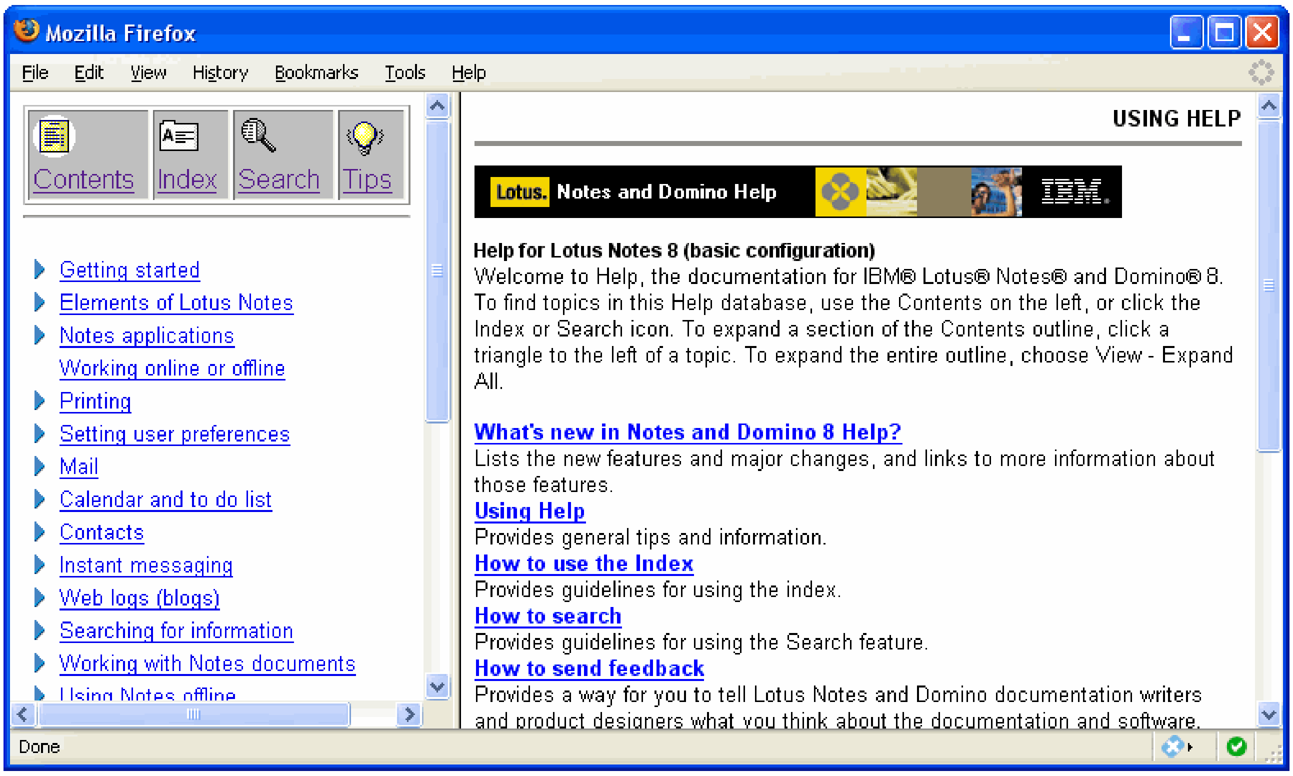
Typical Lotus Notes application UI with navigation on the left

However, when designing for the Web, you have more flexibility in laying out the UI of an application. There may be a compelling reason for the look and feel of the Web interface of a Lotus Notes application to differ from its look and feel in a rich client. For example, the Web interface may become part of the client's intranet where the client already has a defined navigation scheme and uses a color palette that's different from the one that is used in the Lotus Notes application.


Bleed Yellow - Possible is Yellow - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Bleed Yellow beta
PROFILES
COMMUNITIES
BLOGS
DOGEAR
What is Bleed Yellow?
Done

Example Web application UI with navigation on the top

Another benefit of extending a Lotus Notes database to the Web is that Domino automatically attempts to render the existing rich client UI in HTML. You undoubtedly should customize it to enhance the appearance of the application on the Web, but Domino does much of the initial work for you. Having a base markup to work with is often easier than beginning with a blank slate.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
Contents Index Search Tips
USING HELP
Lotus. Notes and Domino Help
IBM.
Done

Getting started
Elements of Lotus Notes
Notes applications
Working online or offline
Printing
Setting user preferences
Mail
Calendar and to do list
Contacts
Instant messaging
Web logs (blogs)
Searching for information
Working with Notes documents
Using Notes offline

Help for Lotus Notes 8 (basic configuration)
Welcome to Help, the documentation for IBM® Lotus® Notes® and Domino® 8. To find topics in this Help database, use the Contents on the left, or click the Index or Search icon. To expand a section of the Contents outline, click a triangle to the left of a topic. To expand the entire outline, choose View - Expand All.

What's new in Notes and Domino 8 Help?
Lists the new features and major changes, and links to more information about those features.
Using Help
Provides general tips and information.
How to use the Index
Provides guidelines for using the index.
How to search
Provides guidelines for using the Search feature.
How to send feedback
Provides a way for you to tell Lotus Notes and Domino documentation writers and product designers what you think about the documentation and software.

Default HTML generated by Domino with minimal customization

Pitfalls

One potential pitfall to avoid when extending a rich client application to the Web is assuming that, because the rich client application already exists, most of the development for Web clients is already done. For example, it is common in Lotus Notes client applications for form validation to be done by using LotusScript in the Querysave event of the form. The following example shows a simple script that ensures a value is entered in the Subject field:

```
Sub Querysave(Source As Notesuidocument, Continue As Variant)
  If Trim(Source.FieldGetText("Subject")) = "" Then
    Messagebox "You must enter a subject.", 16, "Validation Failure"
    Continue = False
  End If
End Sub
```

However, this code does not execute in a browser and must be rewritten by using client-side JavaScript, and for any data where validation is critical, a server-side agent is required. With the exception of agents, you can assume that any LotusScript in the application must be rewritten in JavaScript.

In addition, there are a number of @functions that do not work on the Web such as those that generate modal dialog boxes in the Notes client:

- @DialogBox
- @PickList
- @Prompt

You can determine exactly which rich client functionality must be reprogrammed for the Web by carefully testing the required functionality in a browser.

Another pitfall to avoid is assuming that you need to duplicate exactly all of the rich client functionality on the Web. This is often not the case, and you can avoid unnecessary work by carefully defining the functional requirements for browser clients.

In addition, another pitfall of dual-use applications is that rich clients and Web clients handle data differently. If you do not account for these differences, unintended consequences may result.

Data management in hybrid rich and Web client applications

This page last changed on Apr 03, 2008 by jservais.

Introduction

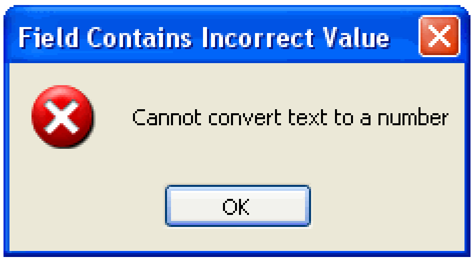
One benefit of a hybrid Lotus Notes and Web application is that both interfaces access the same data store. The documents that you create in Lotus Notes are visible to Web browsers and vice versa. However, Lotus Notes has 17 field types, and data that is submitted from the Web is, for practical purposes, just text. Consequently, there are data management considerations for hybrid applications that help avoid unexpected issues.

Managing the data types

Because data submitted from the Web is text, numeric and date-time data types in a hybrid application require special consideration.

Numbers

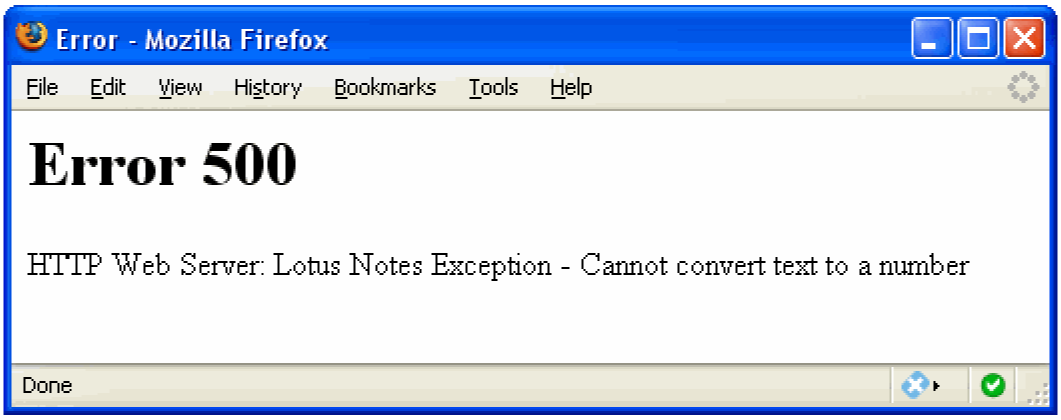
A number field on a Lotus Notes form only accepts numeric data. A Notes user can enter non-numeric data in a number field but when the user attempts to save the document, the Notes client shows the error message shown in the following figure.


Field Contains Incorrect Value

Cannot convert text to a number

Attempting to submit non-numeric data in a number field using Notes

Likewise, if a form submitted from the Web contains non-numeric data in a number field, Domino generate a similar error message as shown in the following figure.



Error 500

HTTP Web Server: Lotus Notes Exception - Cannot convert text to a number

Attempting to submit non-numeric data in a number field using a browser

It appears that there is no problem. Domino prevents users from submitting non-numeric data where a hybrid application requires a number. While this is true, it takes a round trip from the browser to the server for Domino to test the field value and inform the user of the error. We can make the application more user friendly and save bandwidth by testing to see if the field value is numeric using client-side JavaScript.

JavaScript contains a special value, NaN, which is short for "not a number." We can use NaN to write a function to verify a field value is numeric by using the isNaN() and parseInt() functions as shown in the following example:

```
function isNumeric(input) {
  // parseInt() and the parseFloat() attempt to return an integer or a decimal respectively
  var x = parseInt(input, 10);
  if(isNaN(x)) {
    alert('"' + input + '" cannot be converted to a number');
    isNumeric = false;
    return isNumeric;
  }
}
```

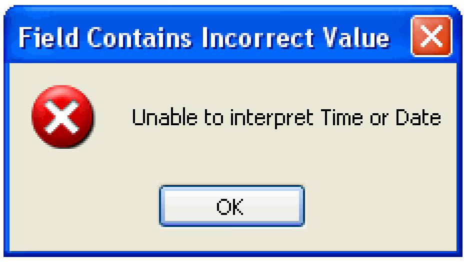
The isNumeric() function can be placed in the jsHeader of a form or an externally referenced JavaScript library and called from the onSubmit event or within a larger form validation function by using syntax similar to the following example:

```
var f = document.forms[0];
if(!(isNumeric(f.NumberField.value))) {
  f.NumberField.focus();
  return false;
}
```

Each call to isNumeric() takes as a parameter the value of the field we want to ensure contains a numeric value. If the field does not contain a number, the user is alerted, focus returns to the field, and execution of the form validation function ceases.

Dates

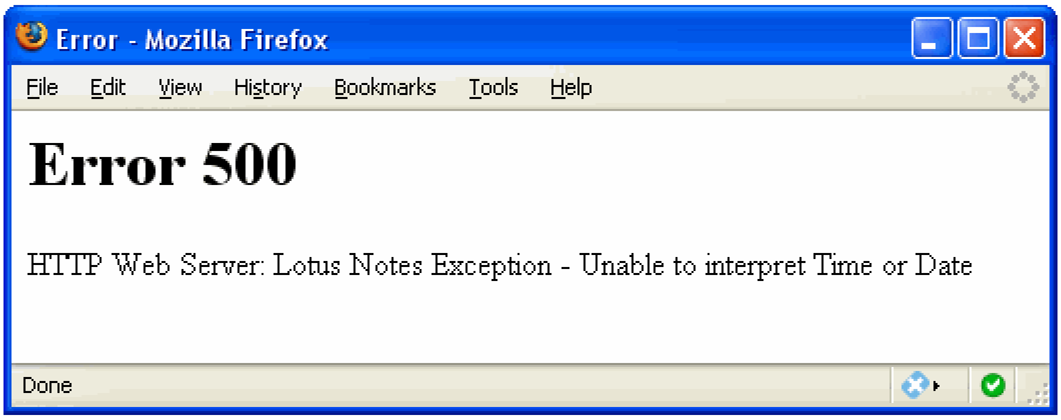
Like number fields, a Date/Time field in Notes requires a valid date/time value or the Notes client shows the error message shown in the following figure.


Field Contains Incorrect Value

Unable to interpret Time or Date

Attempting to submit an invalid date/time value in Lotus Notes

Domino shows a similar error message (see the following figure) if an invalid date/time value is submitted from the Web.



Error 500

HTTP Web Server: Lotus Notes Exception - Unable to interpret Time or Date

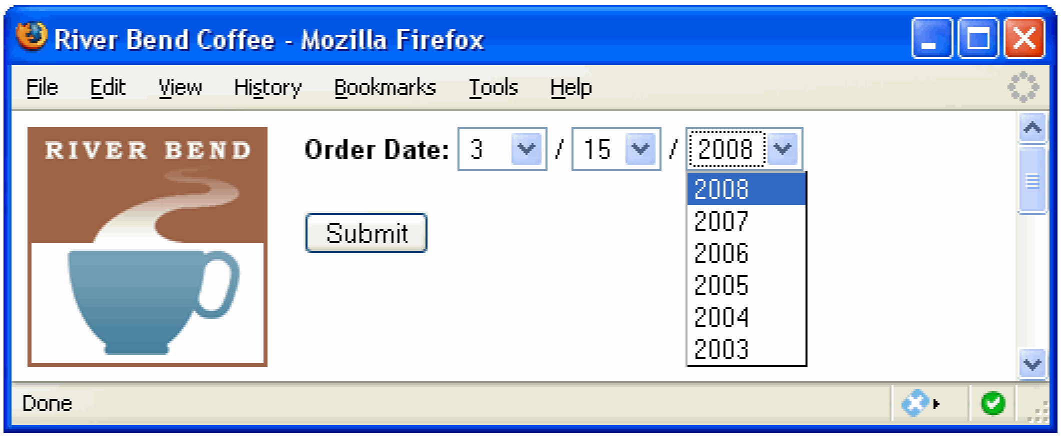
Attempting to submit an invalid date/time value using a browser

However, just as with number fields on the Web, allowing Domino to validate a date/time value requires a round trip to the server, wasting time and bandwidth when the field can first be checked at the client. Unlike a number field, validating a date/time value on the Web is somewhat complex.

When attempting to determine whether a given date value is valid, you must account for several factors such as the following examples:

- Different months have differing numbers of days, and in the case of February, a different number of days depending on the year.
- Users in different countries may enter a date in month/day/year format or day/month/year.
- There are at least three commonly used separators in date values, a forward slash (/), a hyphen (-), or a period (.)

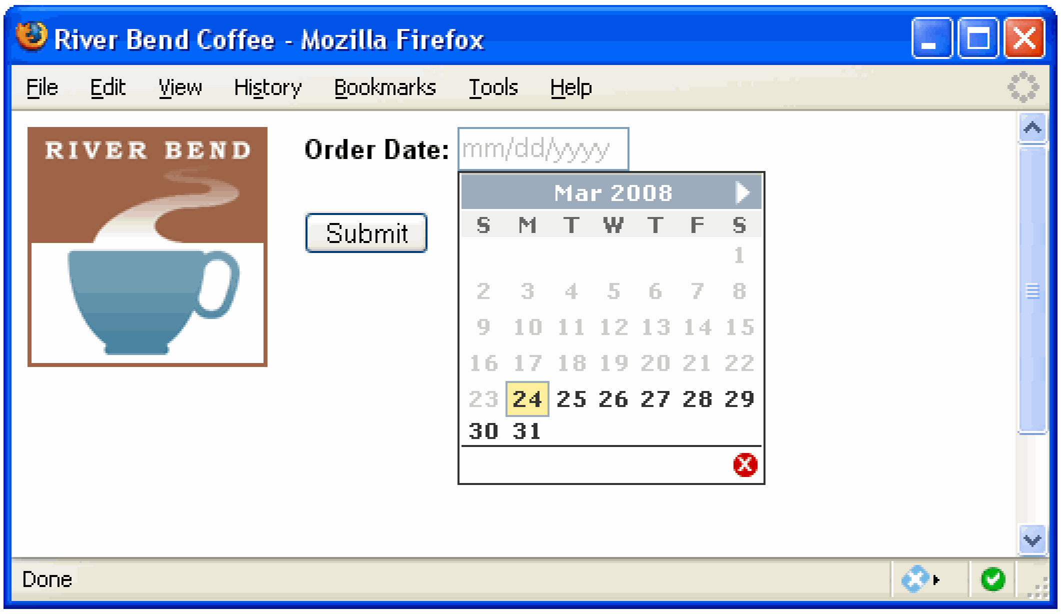
Because of these issues, you sometimes see date values in Web applications that are created by using three separate selection lists rather than a text input field in order to minimize the amount of validation work required as shown in the following figure.


River Bend Coffee - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIVER BEND
Order Date: 3 / 15 / 2008
2008
2007
2006
2005
2004
2003
Submit
Done

Creating a date on the Web by using three separate fields

The selected values are then combined to create a date/time value in a hidden, computed field. However, this approach still requires client-side JavaScript to ensure the selected date is valid (and not something like February 31, 2008) to avoid an unnecessary round trip to the server.

As another option, the ever increasing number of JavaScript frameworks now make it a simple exercise to add a DHTML "date picker" to a Web page, similar to the one that can be displayed in the Notes client as shown in the following figure.



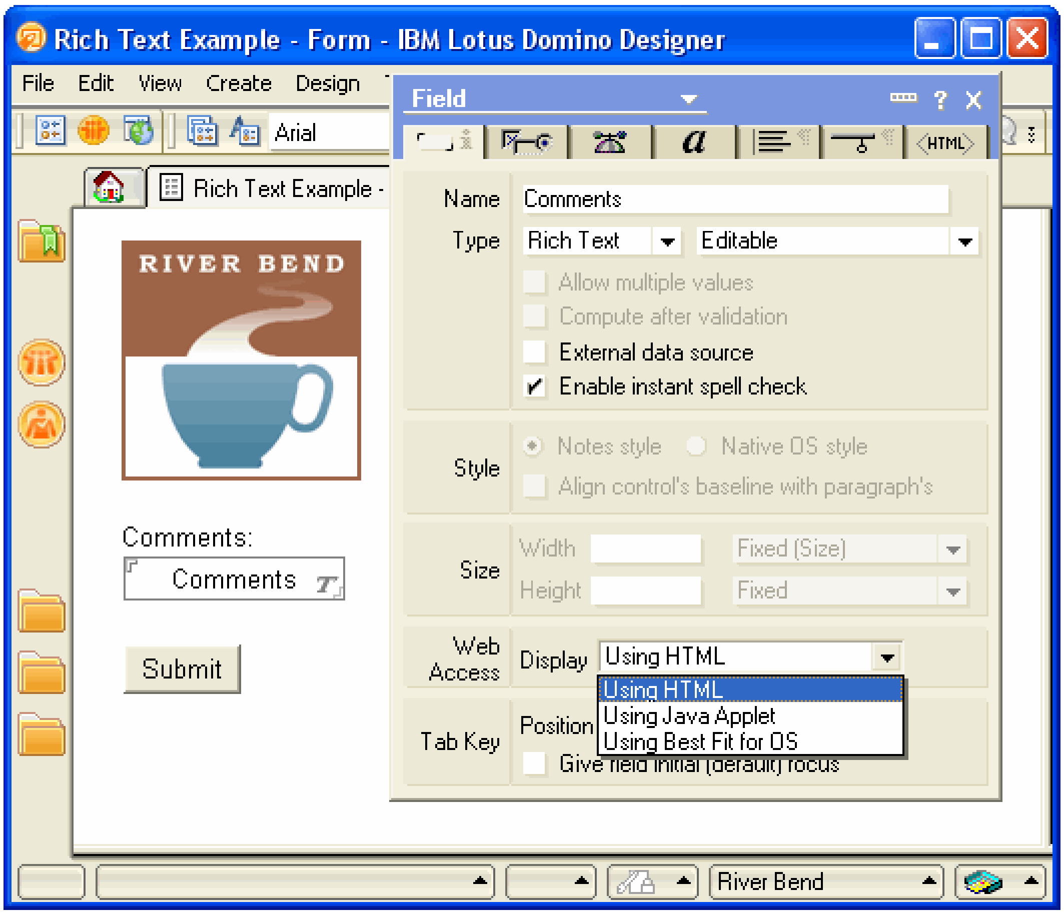
A DHTML date picker

The best approach for implementing time/date fields in your Domino application varies based on your audience and the application requirements. The associated code can be rather lengthy due to the multiple considerations involved in validating time/date fields. However, the following links point to several examples of browser based time/date field implementations and JavaScript validation routines. A search for relevant keywords using the search engine of your choice will undoubtedly yield additional examples:

- http://www.nsftools.com/tips/NotesTips.htm#datepicker
- http://www-10.lotus.com/ldd/sandbox.nsf/Search?SearchView&Query=date%20picker&SearchOrder=0&Start=18
- http://www.expertsrt.com/scripts/Rod/validate_date.php

The issue with rich text

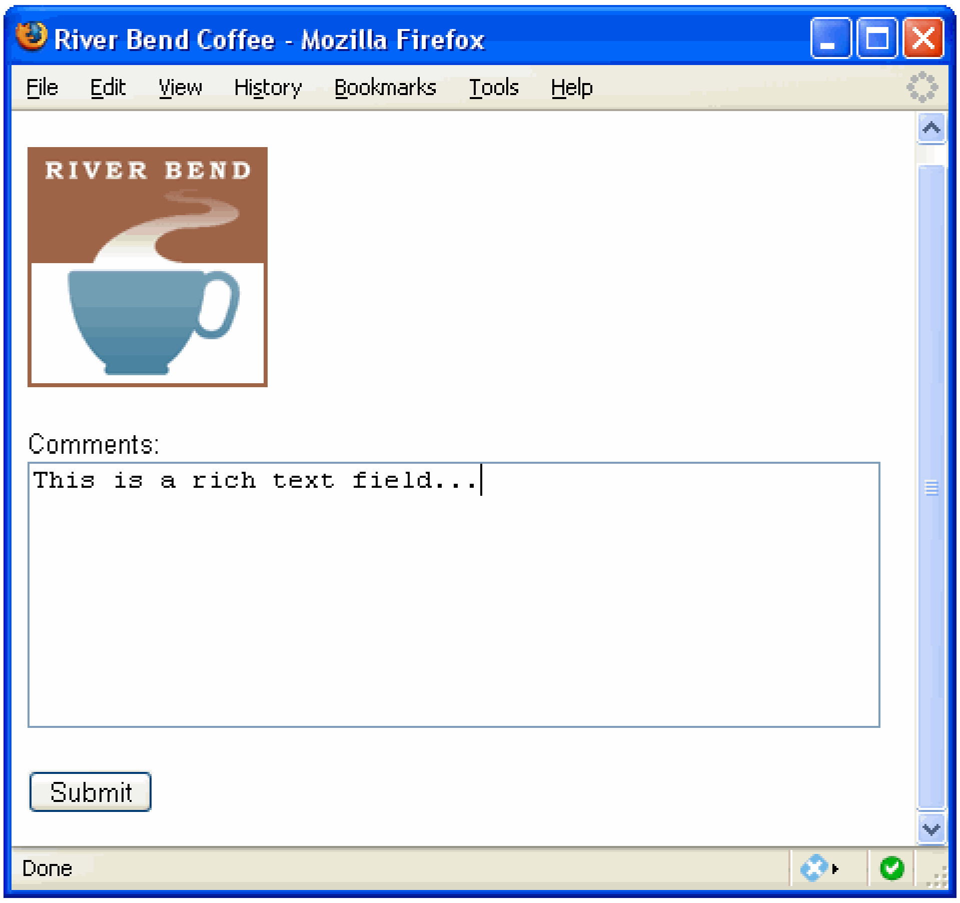
If you have a hybrid application that uses rich text fields, you must be aware of a couple of issues. First, if you create a rich text field on a Notes form, the default option for rendering the field on the Web is "Using HTML" as shown in the following figure.


Rich Text Example - Form - IBM Lotus Domino Designer
File Edit View Create Design
Arial
Rich Text Example -
RIVER BEND
Comments:
Comments
Submit
Field
River Bend

Name: Comments
Type: Rich Text | Editable
Allow multiple values
Compute after validation
External data source
✓ Enable instant spell check
Style: ● Notes style ○ Native OS style
Align control's baseline with paragraph's
Size: Width | Fixed (Size)
Height | Fixed
Web Access: Display: Using HTML
Using HTML
Using Java Applet
Using Best Fit for OS
Tab Key: Position
Give field initial (default) focus

Default option for rendering a rich text field on the Web is "Using HTML"

On the Web, this translates to a text area and users are limited to entering simple text as shown in the following figure.


River Bend Coffee - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIVER BEND
Comments:
Submit
Done

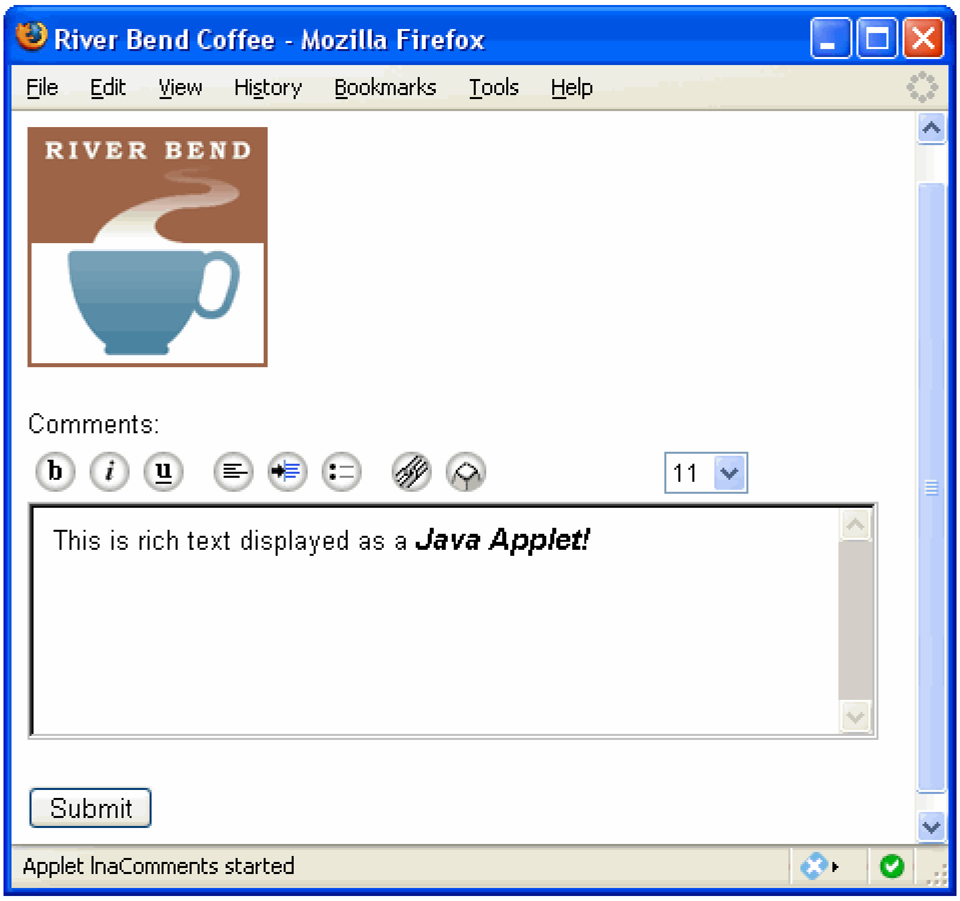
```
This is a rich text field...
```

A rich text field displayed as HTML

Because a text area limits input to simple text, displaying a rich text field on the Web by using HTML is probably the worst choice for hybrid applications where documents that contain rich text fields will be edited by both Notes clients and Web browsers.

Attention
If a user edits a document in Lotus Notes and enters formatted text in a rich text field that is configured to display on the web using HTML, and the document is subsequently edited on the Web, all the rich text formatting is removed.

The second option for displaying a rich text field on the Web is "Using Java Applet." Unlike a text area, with this option, users can enter formatted text from the Web. Better still, if a document has been edited in Lotus Notes and formatting is applied to a rich text field, Domino largely preserves the formatting when the document is edited on the Web. Note that 100% fidelity is not maintained when editing a rich text field in both Notes and on the Web, but it is close.

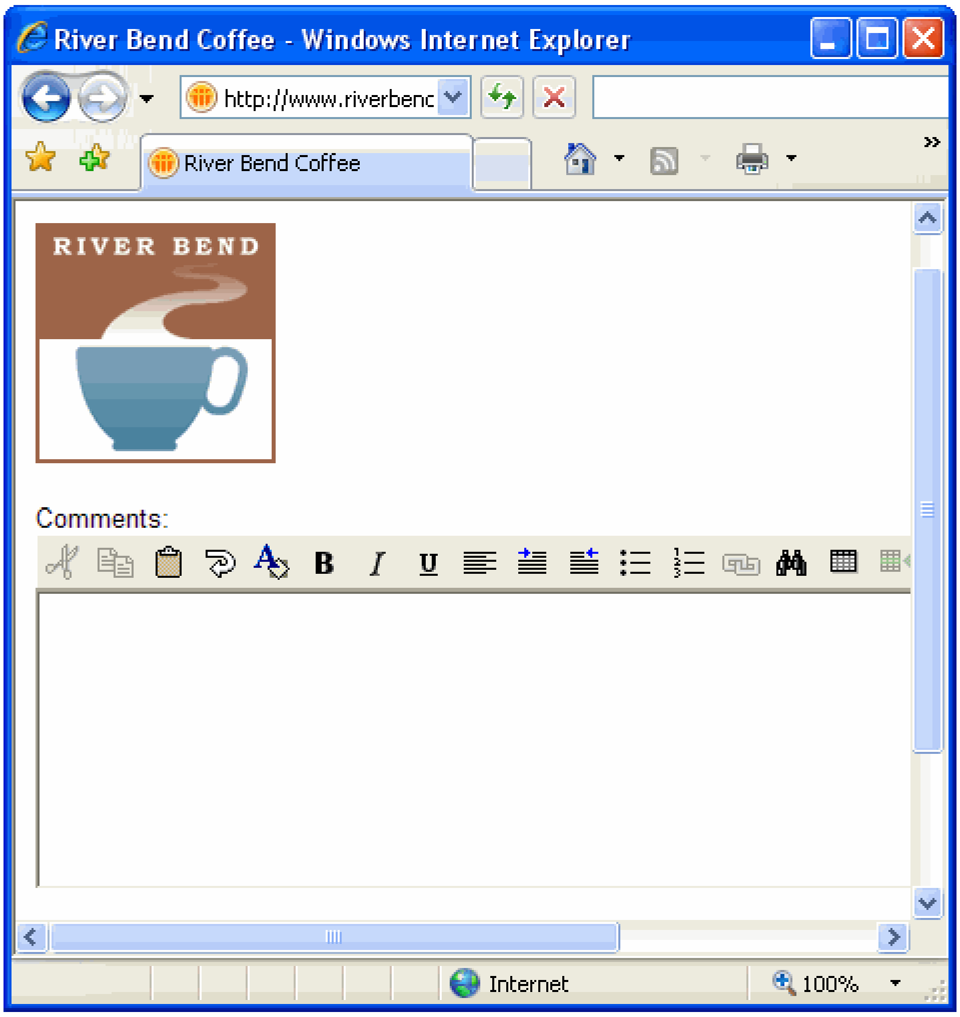

River Bend Coffee - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIVER BEND
Comments:
11
Submit

This is rich text displayed as a ***Java Applet!***

Applet InaComments started

A rich text field displayed using a Java applet

The third option for displaying a rich text field on the Web is "Using Best fit for OS." This option uses an ActiveX® control to display the rich text field, but only in Internet Explorer. Users of other browsers still see the Java applet.



A rich text field displayed in Internet Explorer "Using Best Fit for OS"

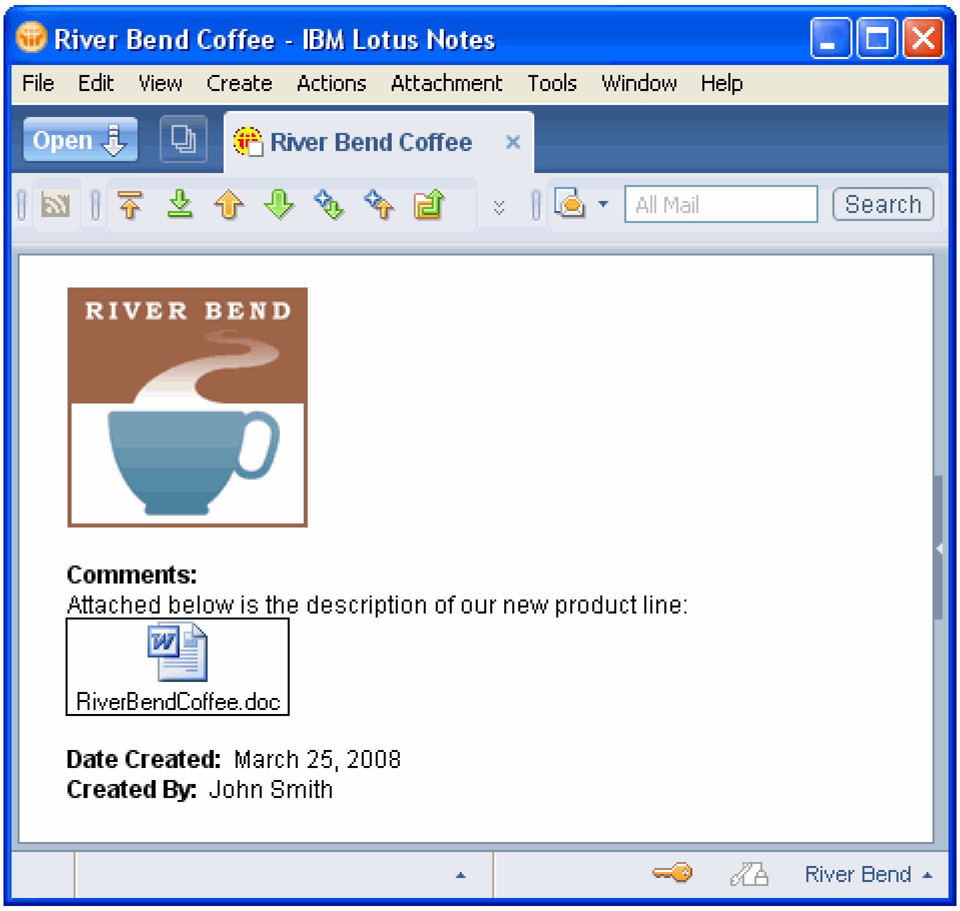
If the browser users of your application use a mixture of Internet Explorer and other browsers, you may want to avoid the "Using Best Fit for OS" option so that users have a consistent interface when working with rich text.

Developers have successfully incorporated other WYSIWYG text editors for the Web, such as FCKeditor and TinyMCE, into Domino Web applications. However these implementations are almost exclusively for browser access only. Consequently, if your hybrid application requires both Notes client and browser users to edit documents that contain rich text fields, the best option in Domino 8.0.1 and earlier is to display rich text fields on the Web by using a Java applet. Ensure that the users of your application understand that there are limitations when editing rich text using a browser and that they should avoid adding certain objects to rich text fields by using a Notes client such as:

- Tables
- Images
- Ordered lists

Handling attachments

The final issue related to data management you want to consider when developing a hybrid application is whether users can add attachments to documents. Using a Notes client, a user can create an attachment in any rich text field. If the document is subsequently viewed by using Lotus Notes or a Web browser, the icon for the attachment is visible in rich text field the attachment was saved in as shown in the following figure.

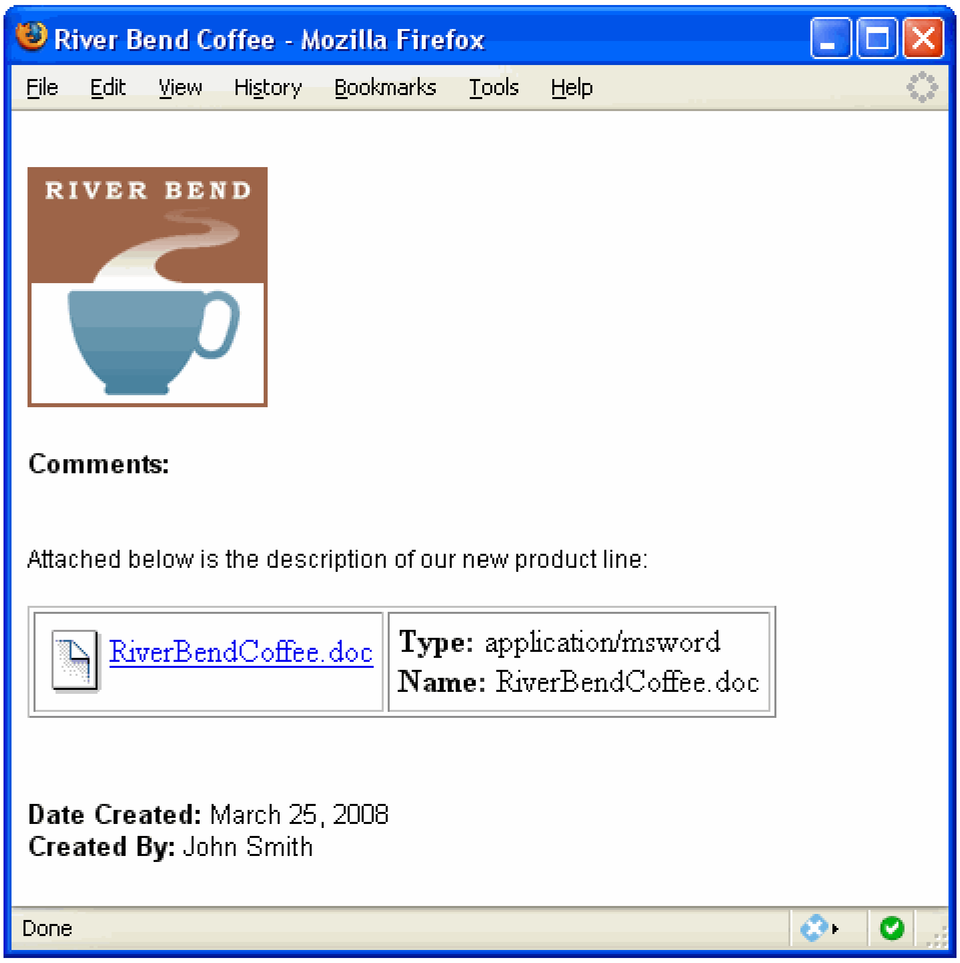

River Bend Coffee - IBM Lotus Notes
File Edit View Create Actions Attachment Tools Window Help
Open
River Bend Coffee
All Mail
Search
RIVER BEND
River Bend

Comments:
Attached below is the description of our new product line:
RiverBendCoffee.doc

Date Created: March 25, 2008
Created By: John Smith

An attachment saved in a rich text field

The Web access display settings for the rich text field determine what happens to attachments when a document is edited on the Web however. If the rich text field is set to display "Using a Java Applet," any attachments that are already part of the document or added from the Web are saved to the rich text field, although the formatting of the attachment icon changes.



Comments:

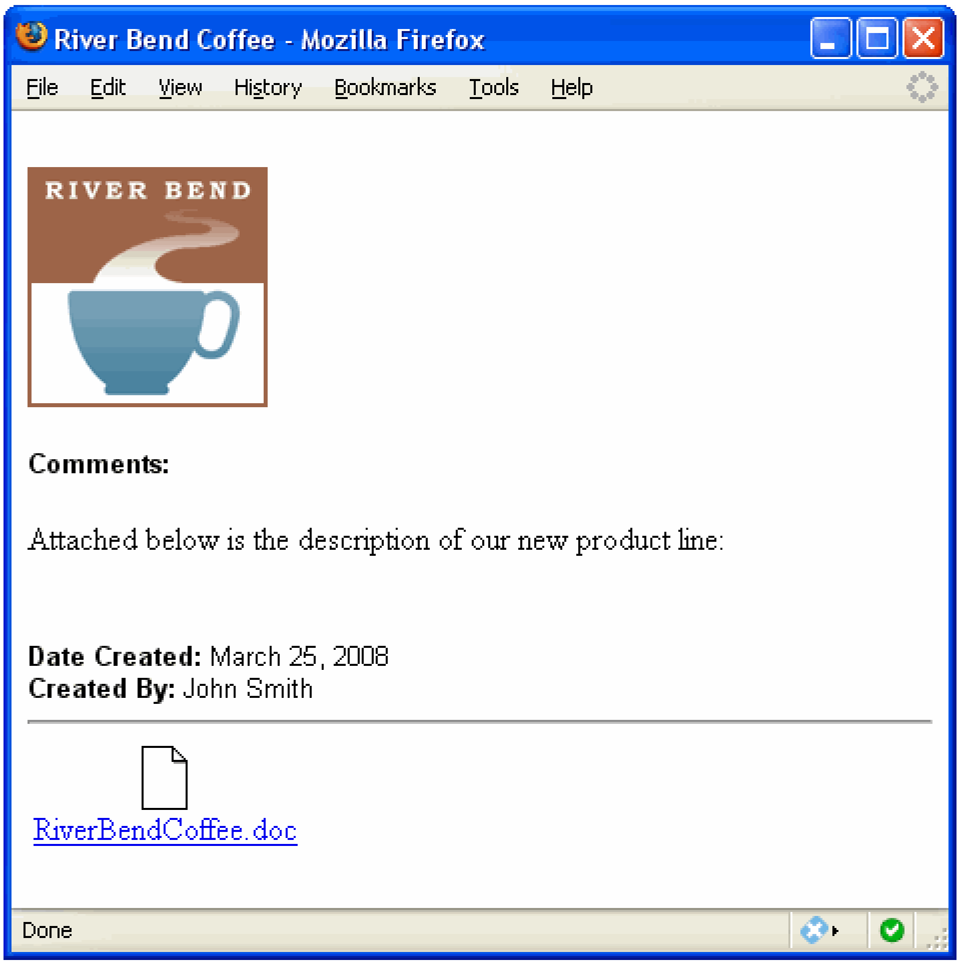
Attached below is the description of our new product line:

RiverBendCoffee.doc	**Type:** application/msword **Name:** RiverBendCoffee.doc

Date Created: March 25, 2008
Created By: John Smith

Saving the document on the Web modifies the display of the attachment

If the rich text field is set to display as HTML, the attachment is saved to the document outside of the rich text field as shown in the following figure.


River Bend Coffee - Mozilla Firefox
File Edit View History Bookmarks Tools Help
RIVER BEND
Done

Comments:

Attached below is the description of our new product line:

Date Created: March 25, 2008
Created By: John Smith

RiverBendCoffee.doc

An attachment saved as part of the document but not in a rich text field

Consequently, when dealing with attachments in hybrid applications, it is best to have at least one rich text field set to display as a Java applet on the Web.

Important
If you have more than one rich text field on a form that is set to display on the Web as a Java applet, be aware that any attachments on the form, whether they were added by using Lotus Notes or a Web browser, and whether they are attached in the first rich text field on the form or subsequent rich text fields, are all saved to the first rich text field when the document is saved from the Web.

Defining functional requirements based on client type

This page last changed on Apr 03, 2008 by dalandon.

Introduction

If you have been asked to Web enable an existing Lotus Notes application, begin by defining the functional requirements for browser access of the application.

Not all clients are created equal

One of the reasons for the term "rich clients" is because they have a rich set of functionality. This is particularly true of Lotus Notes. In contrast, Web browsers have been historically referred to as "thin clients" and have lacked the feature set of client-server applications.

Web technologies have advanced. There are now a number of frameworks that enable the development of rich Web applications. However, it is still not possible to exactly duplicate all the functionality of a Lotus Notes client application in a Web browser. Consequently, when Web enabling a Notes database, it is important to determine what functionality is required on the Web and how it can be implemented.

Functional requirements for Web access

Most Web-enabled Lotus Notes applications do not have the same set of features that are available to both Notes clients and Web browsers. For example, the Lotus Notes interface of a help desk application may allow members of the help desk to create and update trouble tickets. By contrast, the browser interface to the same application may only enable authenticated users to view the status of tickets created for their individual issues.

Before you begin Web enabling an existing Lotus Notes application, you must answer the following questions:

- Will the users who currently access the application in Notes be the same users who access the application on the Web?
- Will the users who access the application with a browser require the same functionality that is available with a Notes client?
- How will users access the application on the Web? That is, will it be part of the corporate intranet or Web site or will it be a stand-alone application?

The answers to these questions will help you define the functional requirements for the browser interface and assist you in creating use cases for design and testing.

Developing hybrid rich client and Web client applications

This page last changed on Apr 03, 2008 by jservais.

When developing a hybrid Lotus Notes and web application, there are several best practices to keep in mind:

- Scope the development effort
- Design for ease of maintenance
- Reconsider security

Important
Before you begin development on the browser interface for a rich client application, ensure that the functional requirements for browser clients have been defined in order to avoid unnecessary work.

Scope the development effort

After you define the functional requirements for the browser interface, begin development by attempting to perform the required functionality in the browser or browsers with which you expect users to access the application. Move through the application step-by-step and document those items that do not work or functionality that is missing. This process helps you to create a realistic level of effort for Web enabling the application.

Having an accurate level of effort is important before undertaking any development project, but is particularly important for Web-enabling an existing Domino application. Clients often feel that creating a Domino Web application is a quick project and Web enabling an existing application is even faster. While Domino provides for rapid Web application development, the complexity of an existing Domino application is generally directly proportional to the amount of time it takes to create a browser interface for it.

Test and consider the following items:

- Application access
- User interface
- Viewing the application data
- Adding or editing the application data

Application access

By necessity, the first item to test for is application access:

- Are you prompted to authenticate when you access the application in a browser?
- If not, should you be?
- Is the data in the application sensitive?

- If so, does it require Secure Sockets Layer (SSL)?

As a developer, you may need to work with your Domino administrator to address authentication with the server and enable SSL for the application. You can find an overview of these topics in 6.0 Server configuration and SSL support.

User interface

After application access, you must examine the user interface and ask the following questions:

- Does the existing application navigation as rendered by Domino work in a browser?
- If the application will be part of an intranet or a component of an existing site, is the navigation consistent with the rest of the site?
- Do the application graphics and color palette match the rest of the intranet or existing Web site?

As part of the user interface, also consider links that were not required in the Notes application but are needed on the Web to help users to navigate such as a link to the corporate home page or intranet splash page. Additionally, most Web sites contain an "About" page, a legal disclaimer, and a contact page. Check if these pages already exist to determine whether you need to create them.

Viewing the application data

Separate from the user interface, test the views that are required to support browser access of the application by asking the following questions:

- Do the views display as expected?
- Can users easily navigate the view, for example, move forward and backward?
- Are there links in the view to open the documents?
- Are the required view actions available and do they work on the Web?

As a result of testing the views, you might find that you want to create $$ViewTemplate forms in the level of effort for Web enabling the application.

Adding or editing the application data

The next step in testing your application is to look at each form in a browser in both read and edit mode and examine is the overall appearance. Ask the following questions:

- Do field labels line up with the corresponding fields?
- Are the fonts on the form an appropriate size and color?
- Do the graphics render correctly?

After you examine the form's appearance, test for required functionality by asking the following questions:

- Are the required form actions available and do they work on the Web?
- Is there a way for the user to submit the form?

- Does the form validation work as you expected?

Remember that any client-side LotusScript functionality must be rewritten in JavaScript, as do @formulas that generate modal dialog boxes in the Notes client such as:

- @DialogBox
- @PickList
- @Prompt

There is a broader list of @formulas that do not work on the Web, or are restricted, in the "@Functions on the Web" document in the "Programming Domino for Web Applications" section of the Domino Designer Help database.

Finally, review form actions and events, such as PostOpen and QuerySave for LotusScript or @formulas that are not supported on the Web.

Design for ease of maintenance

After you begin the work of creating a Web interface for a Lotus Notes application, always design with future application maintenance in mind. Specifically, avoid having two completely different sets of design elements in the application, one to support web clients and one to support Notes clients. There are almost always some forms and views duplicated for each client type. However, every time this occurs, it doubles the amount of work that you have to do when the design of the application changes.

A simple example is the addition of a field to a form. If there is a Web version of the form and a Notes version, the field must be added to two forms instead of just one. If the field data needs to be displayed in a view and there are two versions of the view instead of one, then two additional changes must be made.

Reconsider security

Typically, in a Lotus Notes application, validation of data submitted via a form is done at the client by using @formulas, LotusScript, JavaScript, or all three. If a required field is missing a value or a field fails a validation test, the user is alerted and no data is submitted to the server until the issue is corrected. Many developers duplicate this same functionality on the Web by using client-side JavaScript. However, unlike Lotus Notes, it is trivial for a browser user to intercept form data after client-side validation and modify it before it is sent to the server.

If your Domino Web application is important and its functionality relies on data submitted by your users, validation should be done twice. For the convenience of your users, you should still do initial form validation using client-side JavaScript. This save needless round trips to the server if the user does not complete required fields or enters data in the incorrect format. However, a second validation of user submitted data should be done on the Domino server via a WebQuerySave agent triggered by submission of the form. This ensures that the data validated on the client is the same data that reaches the server.

See Server side user input validation in 4.0 Building Domino Web applications, for a detailed example of using a WebQuerySave agent for form validation.

6.0 Server configuration

This page last changed on Apr 03, 2008 by jservais.

- Tell HTTP commands
- Notes.ini settings for HTTP
- Topics in this section

Domino provides the ability to present information to users through a Notes client and Web browser. To enable the server to present Web pages, the HTTP task must be enabled. You can enable the HTTP task through the console, or to have it run whenever the server is started, add it to the ServerTasks line in the Notes.ini.

The Web server part of the Domino server provides for additional functionality and additional configuration settings and security concerns.

Tell HTTP commands

Command	Result
Tell HTTP Dump Config	Dumps the HTTP configuration to a text file so that you can see how the server is configured.
Tell HTTP Refresh	Refreshes the Web server before the normal refresh. You can specify the refresh cycle interval in the server document. During a Web server refresh cycle, all of the Web site documents, including File protection, authentication realms, and rules, are reloaded by the server.
Tell HTTP Restart	Refreshes the Web server with changes made to settings in the: • Server document for the Web server • File protection, virtual server, and URL mapping documents in the Domino Directory • NOTES.INI file that affects the HTTP server task • HTTPD.CNF and BROWSER.CNF files • Servlets or the servlets.properties fileThis command produces *mostly* the same results as stopping and restarting the Web server but is faster since the HTTP server task remains in memory. All outstanding HTTP requests are processed before the HTTP task restarts. However no HTTP requests are processed during the restart. This command deletes the in-memory page and

	user-authentication caches.
Tell HTTP Show File Access	Displays information about file system protection on the machine and on virtual servers, if you set up virtual servers on the machine.
Tell HTTP Show Security	Displays information about SSL and the server key ring file, including information about whether the server started SSL on the machine. Displays information about SSL for virtual servers if you set up virtual servers on the machine.
Tell HTTP Show Users	Displays the names of users, IP addresses, and the session expiration time for users authenticated with session-based authentication. On servers that are using multiple servers (SSO), authentication may not report sessions accurately by using this command. If the authentication cookie is from the current server, it displays the user name, IP address, and session expiration time for that server. If the authentication cookie is not from the current server, then is does not display session information for users. After a user logs out, this command continues to display the cookie as valid on the server. The session is still valid even though the user has ended the session.
Tell HTTP Show	Virtual servers display a list of virtual servers running on the machine.
Tell HTTP Quit	Stops the Web server task.
Restart Task HTTP	This is a general command and is shorthand for issuing a "tell http quit" followed by a "load http."

Notes.ini settings for HTTP

This following table provides a brief list of the notes.ini settings that effect the HTTP task on the Domino server. We recommend that you research the setting prior to implementing it and make sure it will not cause performance issues. The Domino resources section contains a list of sites hosting information about the Notes.ini setting for you to reference. These settings can be made to the Notes.ini directly or through the server's configuration document. The configuration document lets you easily see what is set, without having to access the Notes.ini directly.

Subject	Release	Description
DominoCompleteDocType	7	Use to have the Domino Web server generate a specific doctype string in the generated HTML. It does not cause the server to generate the correct html for each type.
DominoNoWebAdmin		Specifies whether the HTTP

		server task automatically creates and manages the Domino Web administrator (webadmin.nsf) application.
DominoStrictHTML	7	DominoStrictHTML=2 No additional information is known about this variable.
DominoDisableFileUploadChecks		Allows the file upload control to be disabled on pages served by the Domino Web server.
DominoXUrlProcess	7.0	To enable a Domino Web server's URL command parser to accept **!** or **?** in the URL. Put into the workstation notes.ini to test pages using the exclamation mark (!) with the Domino Designer. For servers, on the Web site document or server document, if "Using Internet Site" is not enabled. It is listed as "Make this site accessible to web search site crawlers."
HTTPDisableGlobalWebsiteRules	8.0.1	To have the default Web site *not* be applied to all Web sites when using Internet sites, set the variable "HTTPDisableGlobalWebsiteRules=1".
HTTPDisableSSIWarnings	7.0.2	Use to allow server-side include (SSI) files to include comments containing "<!--#" without warning to the console and browser.
HTTPDisableUrlCache		Set to "1" to disable the server side cache which was introduced to minimize redundant gzip compressions.
HTTPEnableConnectorHeaders	6.0	Enables the Domino HTTP task to process special headers that are added to requests by a WebSphere 4.0.3 plug-in installed on a foreign Web server. Valid Values are: 0
HTTPEnablePostDataLogging	6	HTTP request logging should be used only for troubleshooting
HTTPEnableResponseContentLogging	6	HTTP request logging should be used only for troubleshooting
HTTPEnableThreadDebug	6.	HTTP request logging should be used only for troubleshooting.

HTTPFormulaCache		Enables (1) or Disables (0) the HTTP server formula cache
HTTPLogFormatAscii		HTTP log files are written in EBCDIC format by default on the i5/OS® platform. You can change the format of these files to ASCII by setting the following in the NOTES.INI file.
HTTPLogUnauthorized	6.0	When set to 1, the Web server logs Error 401 instances to the server console. These instances are generated in two cases: • A user attempts to access a resource but is not authorized for it. • A user has failed to authenticate.
HTTPMultiErrorPage	6	The original R5.x model or R6.x queue of requests model can be selected. This variable is for Notes 6.5.4 FP1 and later.
HTTPSkipTranslationOfNsfRequests	7.0.2	Remove NSF files from Web rules to allow backward compatibility with R5. The Notes.ini variable is "HTTPSkipTranslationOfNsfRequests=1".
HTTPUseNotesMemory		Setting this variable to 1 can prevent a excessive use of the general Memory pool. This excessive usually causes a "Panic: Insufficient Memory" error message.
HTTP_Port		This variable is only used during the collaboration server install. Example : HTTP_Port=8088
HTTP_Pwd_Change_Cache_Hours	6.0	Allows both the old and new passwords to be valid on a server after user request an HTTP password change for this number of hours.
SSLCipherSpec		Determines which SSL-compliant cipher to use to encrypt files on the server.
SSL_Load_Client_Cert	5	Enables LDAP lookups to be done properly when SSL is being used.
SSL_Resumable_Sessions		Specifies the number of resumable SSL sessions that will be cached on the server. Setting

		this variable to 1 disables SSL session resumption on the server.
ReportSSLHandshakeErrors	5	Reports HTTP SSL handshake errors to the console and LOG.NSF. You may want to set this by default, since it only reports errors.
WebAuth_Verbose_Trace		Use this setting to troubleshoot problems with Web server user authentication and Web server group searches for database access verification. With the setting enabled, a Domino Web server records detailed information about specific Web user authentication sessions at the server console.
WebSess_Verbose_Trace		When enabled, the setting allows a Domino Web server to record, at the server console, detailed information about specific Web session-based authentication sessions, such as unauthorized, unauthenticated, or session expiration information.

Topics in this section

- Logging
- Performance considerations
- Security considerations
- Server error handling
- Topology
- Working with Web site rules

Logging

This page last changed on Apr 03, 2008 by dalandon.

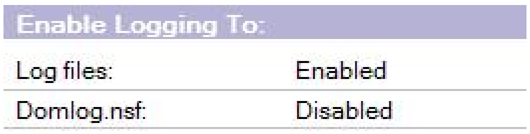
Introduction

The Domino server has the ability to log information in several ways. One method uses a Domino database to store information about requests serviced by the server, while another method uses standard text files to store similar information. Alternatively, applications can log access information on their own with various methods.

We recommend that you use one of the server's methods for logging, because it simplifies your application's duties and leverages the built-in functionality of the Domino server.

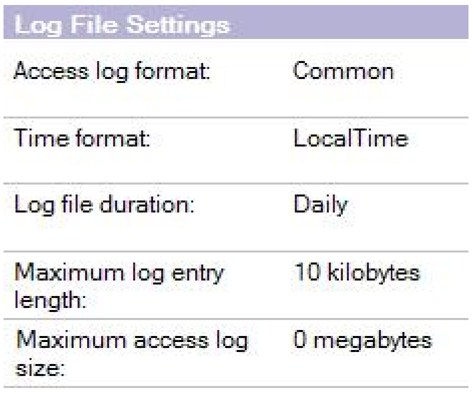
Using the Domino server's access logs

The Domino HTTP task can log request information to text logs in a standard format. To enable the Domino access logs, click the **Internet Protocols -> HTTP** tab of the server document. Then enable the logs as shown in the following figure. You use this setting in the same place regardless of whether you are using Internet Site documents on the server.



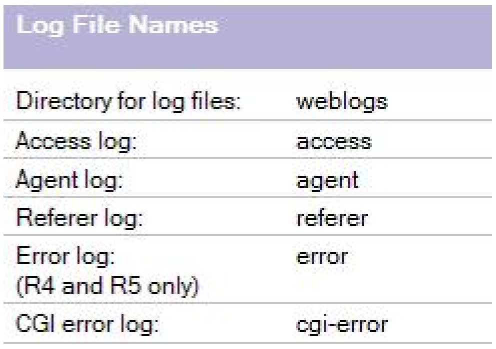
Enable Logging To:	
Log files:	Enabled
Domlog.nsf:	Disabled

There are other settings in the server document that control where these logs exist and the format of each entry as shown in the following figure.



Log File Settings	
Access log format:	Common
Time format:	LocalTime
Log file duration:	Daily
Maximum log entry length:	10 kilobytes
Maximum access log size:	0 megabytes

You can set the Access log format field to either Common or Extended Common. The difference between the two is that the Common format splits up log information among different text files: access, agent, and referer. The Extended Common format logs all of the access, agent, and referer information in one line in the access log files, so that the agent and referer logs are not created. If the Common format is used, then each request to the server corresponds with one line in each of the access, agent, and referer logs, which can be quite difficult to correlate correctly.



Log File Names	
Directory for log files:	weblogs
Access log:	access
Agent log:	agent
Referer log:	referer
Error log: (R4 and R5 only)	error
CGI error log:	cgi-error

The Directory for log files field is located under the data directory unless a full explicit path is specified. The Access log field contain entries for each request made to the Domino HTTP server. The Agent log field contains entries that detail the HTTP client for each request, be it Java, Firefox, Internet Explorer, Opera, and so on. The Referer log field contains information about links that generated other requests that reached the Domino HTTP server.

The following table defines what is placed in the log files.

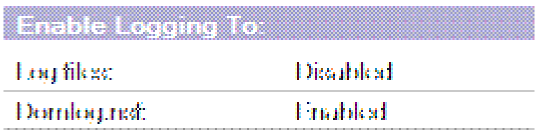
Text file	Records
Access	Depending on the file format you choose, the Access log file records the following Web server request information in the order shown: • **Common** 1. Client DNS name or IP address if DNS name is not available 2. Host header from request, or server IP address if the Host header is not available 3. Remote user if available 4. Request time stamp

	5. HTTP request line 6. HTTP response status code • **Extended Common** 1. Client DNS name or IP address if DNS name is not available 2. Host header from request, or server IP address if Host header is not available 3. Remote user if available 4. Request time stamp 5. HTTP request line 6. HTTP response status code 7. Request content length if available, otherwise "-" 8. Referring URL if available, otherwise "-" 9. User agent if available, otherwise "-" 10. Amount of time, in milliseconds, to process the request 11. Value of the cookie header 12. Translated URL (the full path of the actual server resource, if available)
Agent	User agent if available, otherwise "-"
Referer	URL the user visited to gain access to a page on this site

Using the Domlog database

Another option provided by the Domino server for HTTP logging is the Domlog.nsf database. This database logs information similar to the text access logs listed previously, but stores the data in a Domino database. This database has some built-in views that categorize the log entries by date, user, response code, and others. The database can also be customized in case other groupings are more helpful.

The Domlog can be enabled in the server document as shown in the following figure.



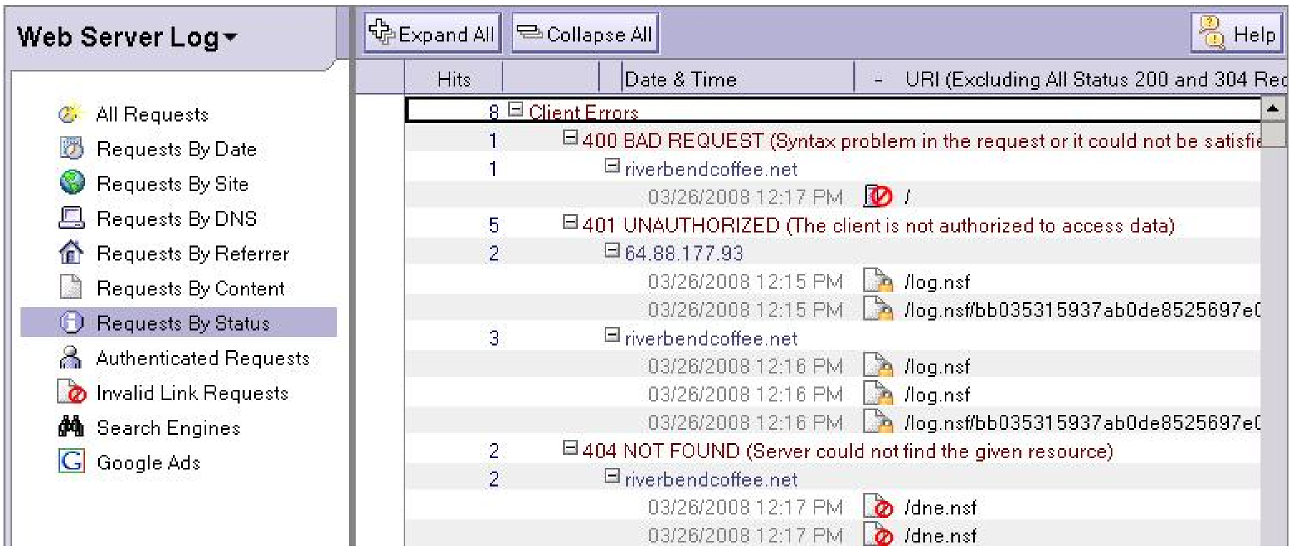
Enable Logging To:

Log files:	Disabled
Domlog.nsf:	Enabled

After the Domlog is enabled, the HTTP task automatically creates the domlog.nsf database the next time it starts. The Domlog continues to grow in size unless it is manually cleaned out. A built-in cleanup agent is contained in the domlog.nsf database, but it is *disabled* by default. We strongly recommend that you enable this agent or create a cleanup agent of your own, because the Domlog can grow to use large amounts of disk space and potentially lead to server crashes.

Important
Do not create a full text index for the Domlog database.

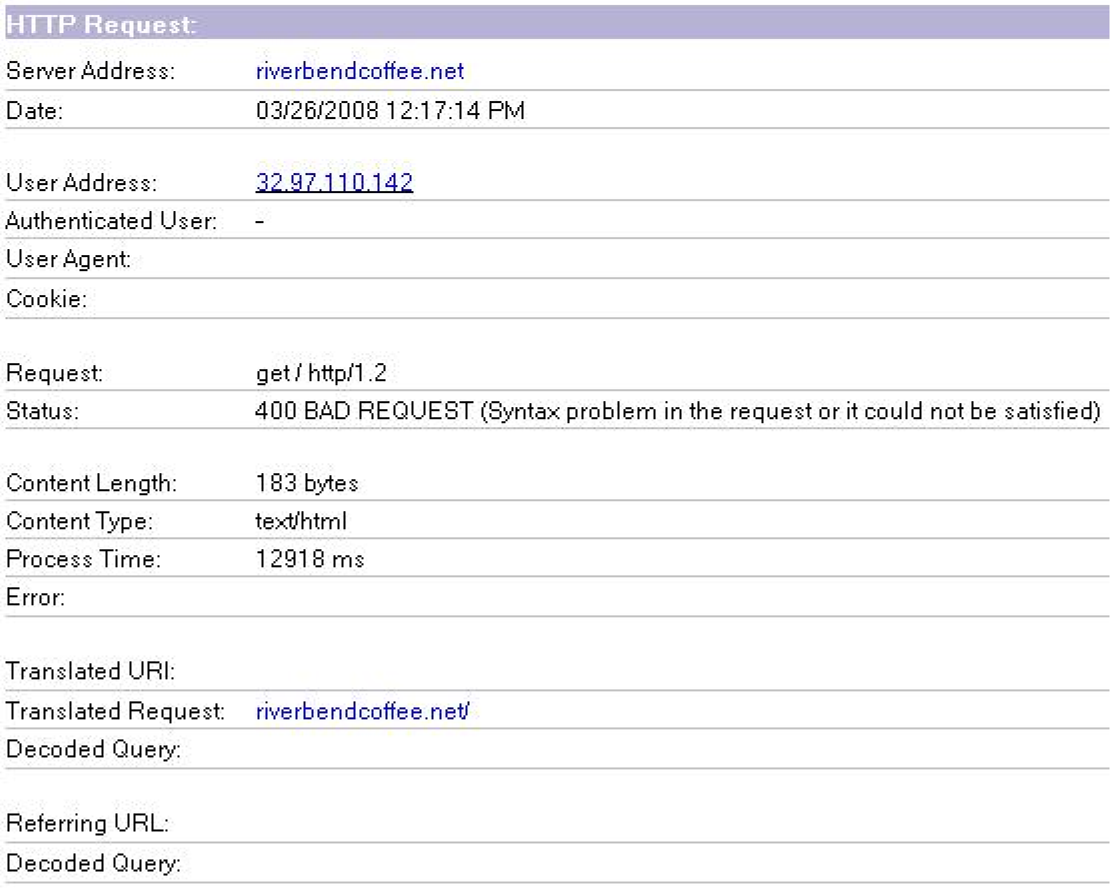
The following figure shows the views that are already included in the Domlog database.



Hits	Date & Time	- URI (Excluding All Status 200 and 304 Rec
8	⊟ Client Errors	
1	⊟ 400 BAD REQUEST (Syntax problem in the request or it could not be satisfie	
1	⊟ riverbendcoffee.net	
	03/26/2008 12:17 PM	/
5	⊟ 401 UNAUTHORIZED (The client is not authorized to access data)	
2	⊟ 64.88.177.93	
	03/26/2008 12:15 PM	/log.nsf
	03/26/2008 12:15 PM	/log.nsf/bb035315937ab0de8525697e0
3	⊟ riverbendcoffee.net	
	03/26/2008 12:16 PM	/log.nsf
	03/26/2008 12:16 PM	/log.nsf
	03/26/2008 12:16 PM	/log.nsf/bb035315937ab0de8525697e0
2	⊟ 404 NOT FOUND (Server could not find the given resource)	
2	⊟ riverbendcoffee.net	
	03/26/2008 12:17 PM	/dne.nsf
	03/26/2008 12:17 PM	/dne.nsf

Views included in the Domlog database

The following figure shows one of the documents that is created in the Domlog database.



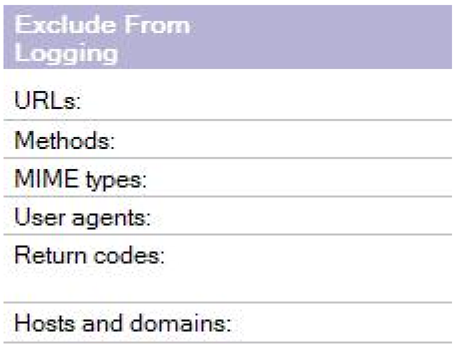
HTTP Request:

Server Address:	riverbendcoffee.net
Date:	03/26/2008 12:17:14 PM
User Address:	32.97.110.142
Authenticated User:	-
User Agent:	
Cookie:	
Request:	get / http/1.2
Status:	400 BAD REQUEST (Syntax problem in the request or it could not be satisfied)
Content Length:	183 bytes
Content Type:	text/html
Process Time:	12918 ms
Error:	
Translated URI:	
Translated Request:	riverbendcoffee.net/
Decoded Query:	
Referring URL:	
Decoded Query:	

Document created in the Domlog database

Common log settings

In the server document, you can modify a few fields if you want to exclude certain URLs, methods, and so on, as shown in the following figure. These settings affect both the Domlog and text logs.


Exclude From Logging

URLs:	
Methods:	
MIME types:	
User agents:	
Return codes:	
Hosts and domains:	

Custom logging

It is possible for your application to log its own information in other formats. For example, it may be helpful for your application to log specific actions within a particular form. The actions can be tracked by creating entries within a separate database or even within the application itself.

Custom logging can also include print statements within agents that are meant to track the execution path of an agent. These can be helpful for debugging purposes, but we recommend that you disable these during normal operations, because the extra lines may not add much individually but can bloat the log.nsf file of the server.

Performance considerations

This page last changed on Apr 03, 2008 by dalandon.

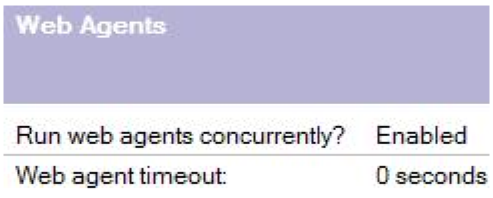
Performance should always be considered when developing applications. This section contains best practices for performance, but there may be many more.

Using Summary Reports instead of views

While View Indexing allows caching of view content, which facilitates faster access to Domino data, the repeated runtime queries of the View Index for certain reusable data subsets can cause undue stress on the server as well as impact the user experience. To achieve the same functionality, we discuss the creation and maintenance of a *Summary Report*. A *Summary Report* is a document that contains metric data and other analytic content. Applications can use this report to facilitate low-cost metadata queries against large data collections.

Concurrent Web agents

By default, agents that run as a result of HTTP requests are run one after the other in a sequential, serial manner. This can be a huge performance bottleneck if you plan to use agents in your application, so you may have to modify the server configuration to allow Web agents to run concurrently. This option is in the Server Document on the **Internet Protocols -> Domino Web Engine** tab. You must reload the HTTP task in order for the change to take effect.



Web Agents

Run web agents concurrently?	Enabled
Web agent timeout:	0 seconds

Security considerations

This page last changed on Apr 03, 2008 by jservais.

Introduction

If you are going to place your application for the world to access it, you cannot expect everyone to only look at the data that you want them to. You have to be proactive and ensure that the user can only access the data that you want them to see. While the Domino administrator tries to ensure that the server is secure, you as the developer to make certain that the database is secure.

Development considerations

- Protect all views by including a $$ViewTemplateDefault with no embedded view or $$ViewBody field and put a redirect meta tag in the HTML header as shown in the following example.

```
URL:=@GetProfileField("Setup" ; "URL" );
"<meta http-equiv=\"refresh\" content=\"0; URL="+url+"\"></head>"
```

- Use a "$$ViewTemplate For ViewName" for each view that is seen from the Web.
- Create a Web redirect form to prevent access to the default design elements. See Working with Web site rules for more information about creating rules.
    - Navigators: Set the Incoming URL Path set to **/**.nsf/$DefaultNav and the redirect to your home page or warning page.
    - Forms: Set the Incoming URL Path set to **/**.nsf/$DefaultForm and the redirect to your home page or warning page.
    - Views: Set the Incoming URL Path set to **/**.nsf/$DefaultView and the redirect to your home page or warning page.
- To prevent misuse of database searching, include a $$SearchTemplateDefault with no $$ViewBody field.
- Protect your agents from being invoked by malicious users by using the following methods:
    - Validating the cgi variables such as referring_server.
    - Check that the form being processed is allowed.
    - If the agent is not set to "run as Web user", then set a hidden computed field on the form to pass user information.
    - Use a cookie, if the site is set to SSO.
- Do not rely solely on CGI variables in your security implementation.
- Be aware that custom authentication using @Password may be compromised.
- Use SSL for any database with sensitive information.

Server considerations

- Review the access control list (ACL) settings for every database and template on your server, including .NSF, .NTF, .BOX, .NS2, .NS5, .NS4, .NSG, and .NSH.
- Set Anonymous and Default access privileges to "No Access" for all databases except for the home page and Domino Web Configuration and Custom Login databases, if you use them. This only lets validated users into databases.
- Create a separate organization for all Domino servers that are located on the Internet or outside your firewall system. Then cross-certify the external Domino servers with internal Domino servers. You can use Directory Assistance and cascade your primary Domino Directory, if your Domino users need access to the site.
- Secure the login and databases with sensitive data by using SSL.

Note
Domino 6 and later do not allow database browsing, but you can download an application from the Lotus Sandbox that allows customers to retain this functionality if they really want it. Therefore it is no longer a consideration for you to turn off database browsing for Web clients, so users cannot look to find databases, unless they know the database name to access it.

Additional topics in this section

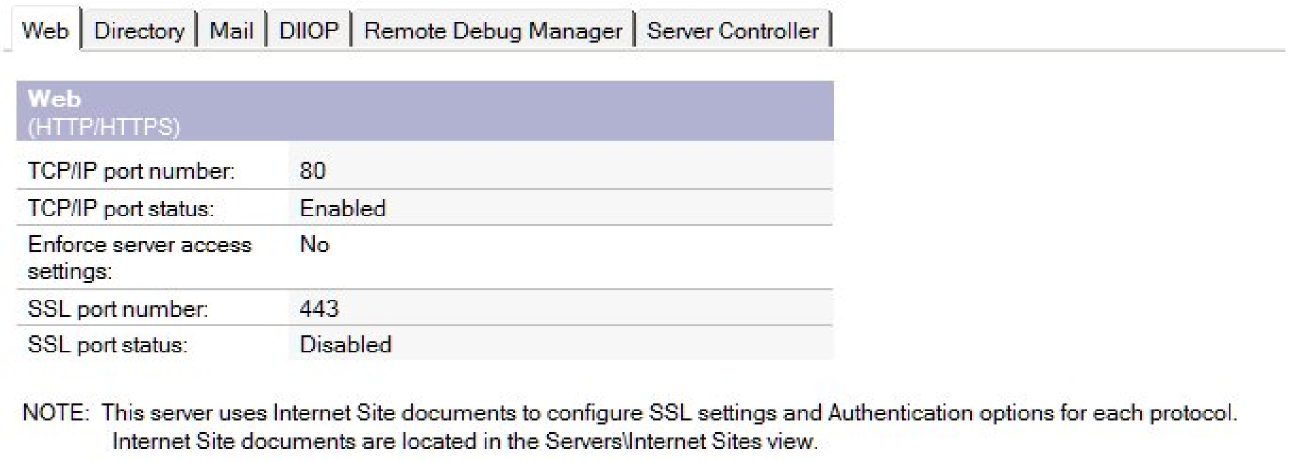
- SSL support

SSL support

This page last changed on Apr 03, 2008 by dalandon.

Introduction

Secure Sockets Layer (SSL) is cryptographic protocol that encrypts the communications between the server and the browser for Web browsing. It can be also used for secure communications with LDAP, POP3, IMAP, DIIOP, and SMTP protocols. You can set up a Domino server so that clients and servers that connect to the server use SSL to ensure privacy on the network. You set up SSL on a protocol-by-protocol basis by enabling SSL for each protocol on the server document in the Domino Directory.



Web
(HTTP/HTTPS)

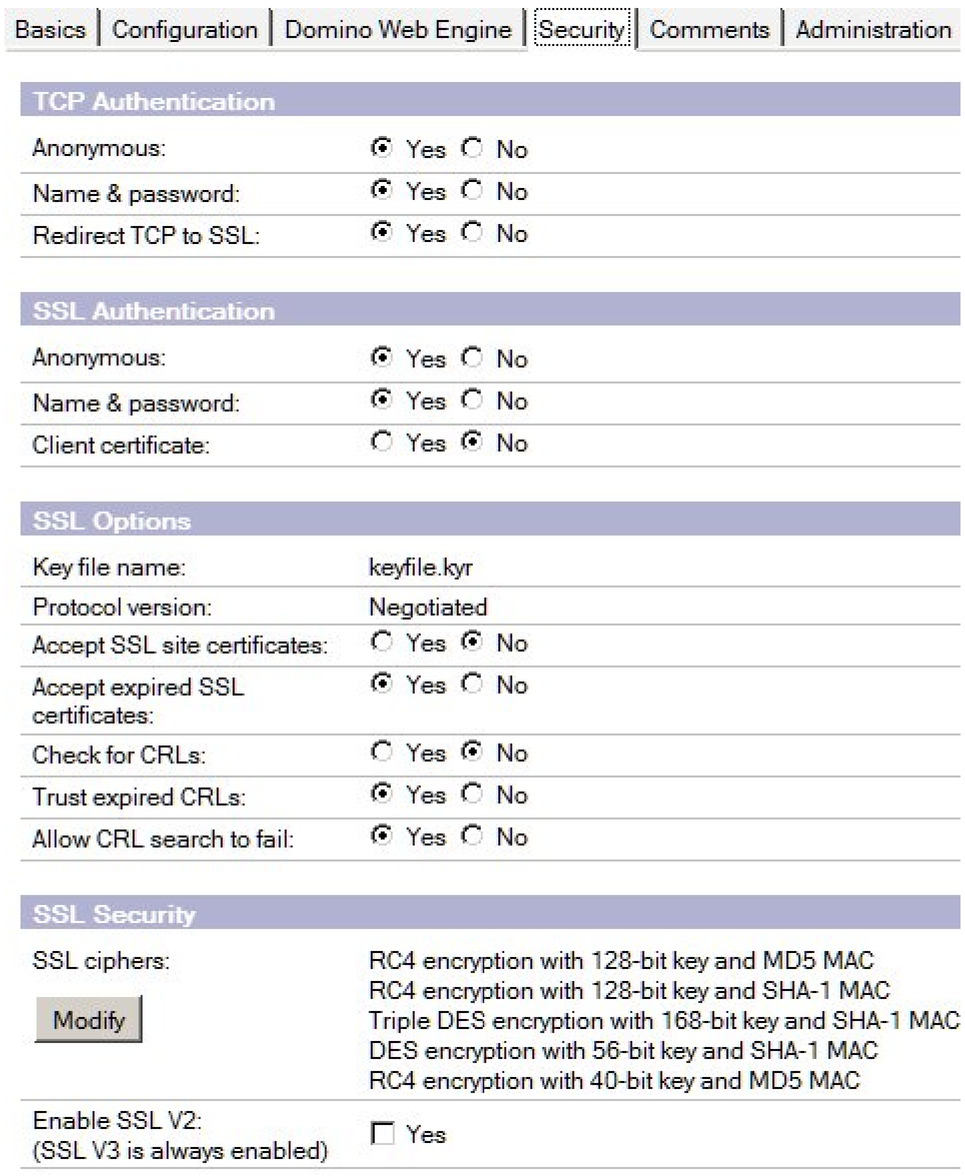
TCP/IP port number:	80
TCP/IP port status:	Enabled
Enforce server access settings:	No
SSL port number:	443
SSL port status:	Disabled

NOTE: This server uses Internet Site documents to configure SSL settings and Authentication options for each protocol. Internet Site documents are located in the Servers\Internet Sites view.

Setting up SSL

To set up SSL on your server, you need a server certificate (saved in a key ring) from an Internet certificate authority (CA). You can get a server certificate from either a Domino with a self-certificate or third-party CA and then install it in a key ring. A *key ring* is a binary file that uniquely identifies the server. The key ring file is stored on the server's hard drive and contains a public key, name, an expiration date, and a digital signature. The key ring also contains root certificates that are used by the server to make trust decisions.

The key ring's file name is entered either on the server document, if you are not using Internet Documents or on the Web site document.



TCP Authentication

Anonymous:	[✓ Yes] No
Name & password:	[✓ Yes] No
Redirect TCP to SSL:	[✓ Yes] No

SSL Authentication

Anonymous:	[✓ Yes] No
Name & password:	[✓ Yes] No
Client certificate:	Yes [✓ No]

SSL Options

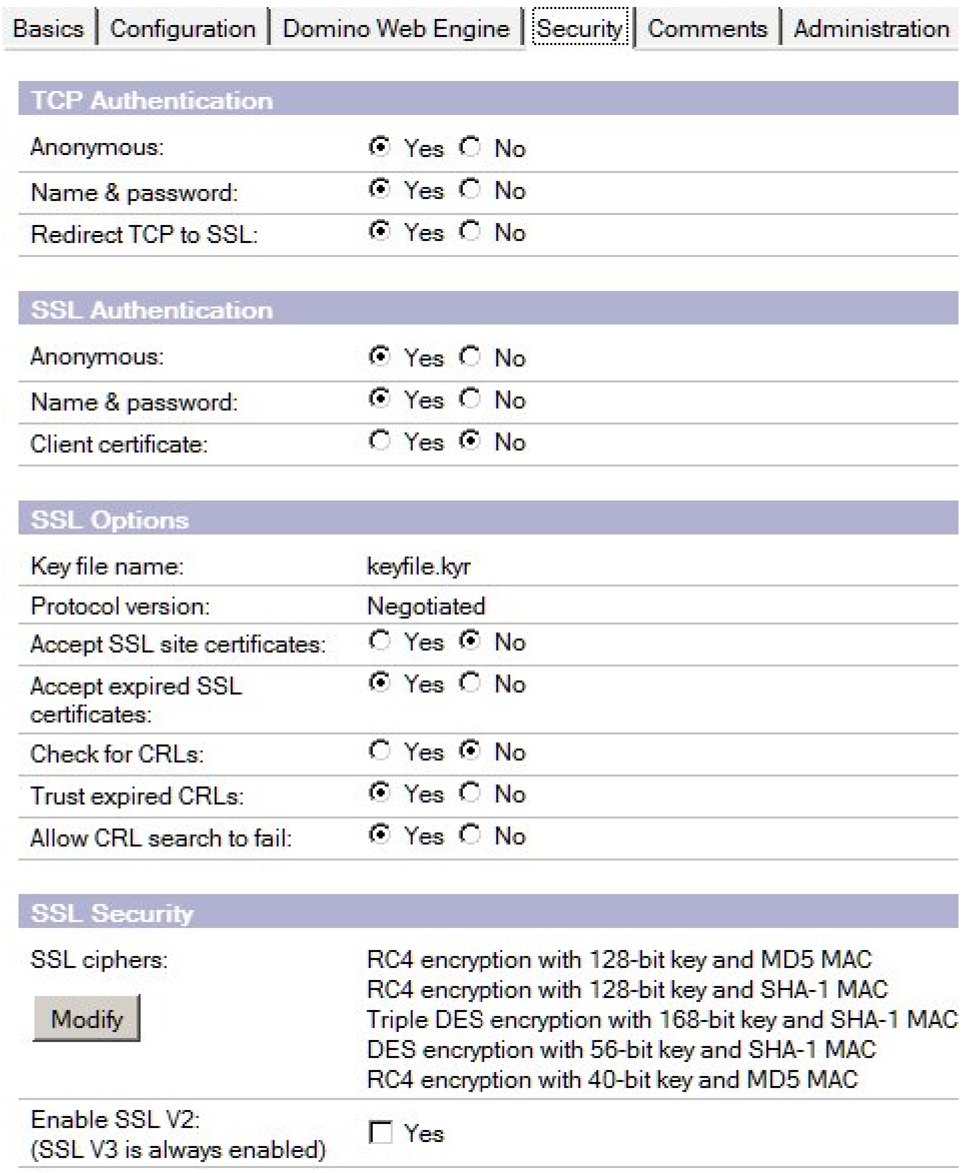
Key file name:	keyfile.kyr
Protocol version:	Negotiated
Accept SSL site certificates:	Yes [✓ No]
Accept expired SSL certificates:	[✓ Yes] No
Check for CRLs:	Yes [✓ No]
Trust expired CRLs:	[✓ Yes] No
Allow CRL search to fail:	[✓ Yes] No

SSL Security

SSL ciphers:	RC4 encryption with 128-bit key and MD5 MAC RC4 encryption with 128-bit key and SHA-1 MAC Triple DES encryption with 168-bit key and SHA-1 MAC DES encryption with 56-bit key and SHA-1 MAC RC4 encryption with 40-bit key and MD5 MAC
Enable SSL V2: (SSL V3 is always enabled)	☐ Yes

Server document

For each Web site document, enter the key ring file name, if it is to use SSL.


Basics | Configuration | Domino Web Engine | Security | Comments | Administration

TCP Authentication

Anonymous:	[✓ Yes] No
Name & password:	[✓ Yes] No
Redirect TCP to SSL:	[✓ Yes] No

SSL Authentication

Anonymous:	[✓ Yes] No
Name & password:	[✓ Yes] No
Client certificate:	Yes [✓ No]

SSL Options

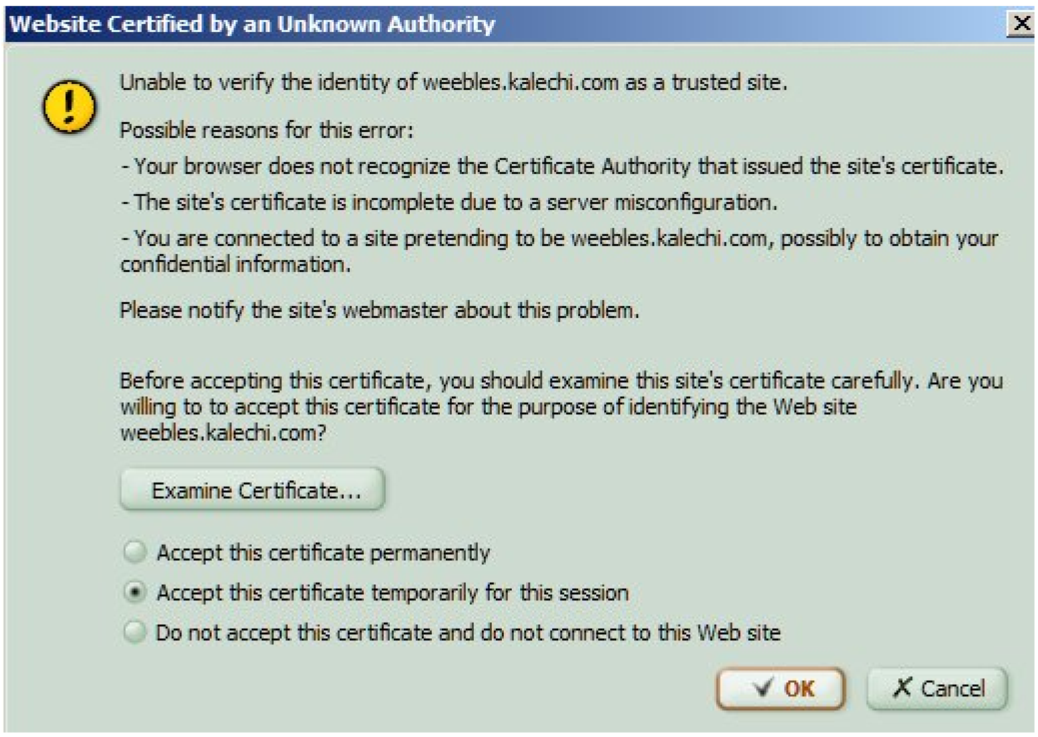
Key file name:	keyfile.kyr
Protocol version:	Negotiated
Accept SSL site certificates:	Yes [✓ No]
Accept expired SSL certificates:	[✓ Yes] No
Check for CRLs:	Yes [✓ No]
Trust expired CRLs:	[✓ Yes] No
Allow CRL search to fail:	[✓ Yes] No

SSL Security

SSL ciphers: Modify	RC4 encryption with 128-bit key and MD5 MAC RC4 encryption with 128-bit key and SHA-1 MAC Triple DES encryption with 168-bit key and SHA-1 MAC DES encryption with 56-bit key and SHA-1 MAC RC4 encryption with 40-bit key and MD5 MAC
Enable SSL V2: (SSL V3 is always enabled)	☐ Yes

Web site document

For internal use or testing, you can use a server certificate created by the Domino server called a *self-certificate*. Since it is not one of the trusted CAs, your browser gives you a warning about trust. You can accept the certificate and you will not be prompted again. Each browser and version have a different sequence needed to accept the certificate. This may be easy for testing or a controlled group of users, but for general public use, it is best to buy a certificate from a trusted source.



Unable to verify the identity of weebles.kalechi.com as a trusted site.

Possible reasons for this error:

- Your browser does not recognize the Certificate Authority that issued the site's certificate.

- The site's certificate is incomplete due to a server misconfiguration.

- You are connected to a site pretending to be weebles.kalechi.com, possibly to obtain your confidential information.

Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to to accept this certificate for the purpose of identifying the Web site weebles.kalechi.com?

Accept this certificate permanently
[✓ Accept this certificate temporarily for this session]
Do not accept this certificate and do not connect to this Web site

Firefox warning message

Additional topics in this section

- Setting up SSL with a self-certified certificate

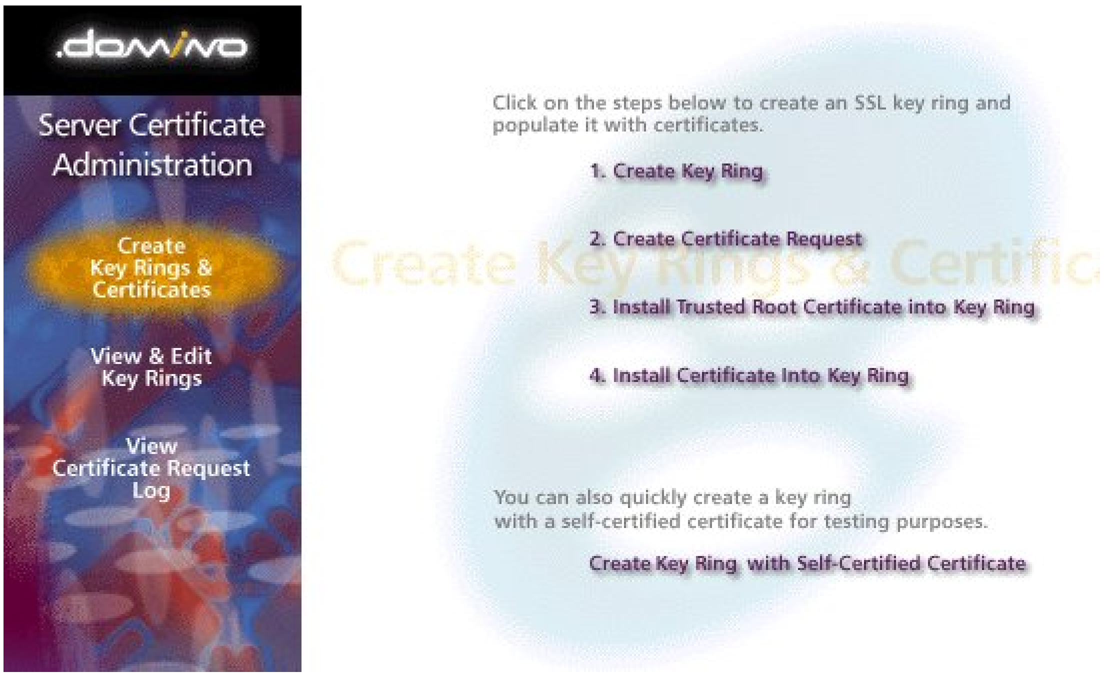
Setting up SSL with a self-certified certificate

This page last changed on Apr 03, 2008 by dalandon.

For development, test with SSl if your production site is going to run Secure Sockets Layer (SSL). You can either spend the money for a CA certificate or create your own. If your testing is going to be done with a controlled group, then a self-certified certificate will work. Make sure to tell the users to accept the certificate.

Creating a self-certified certificate

From the Notes client, open the Server Certificate Admin application, and then click **Create Key Rings & Certificates**.

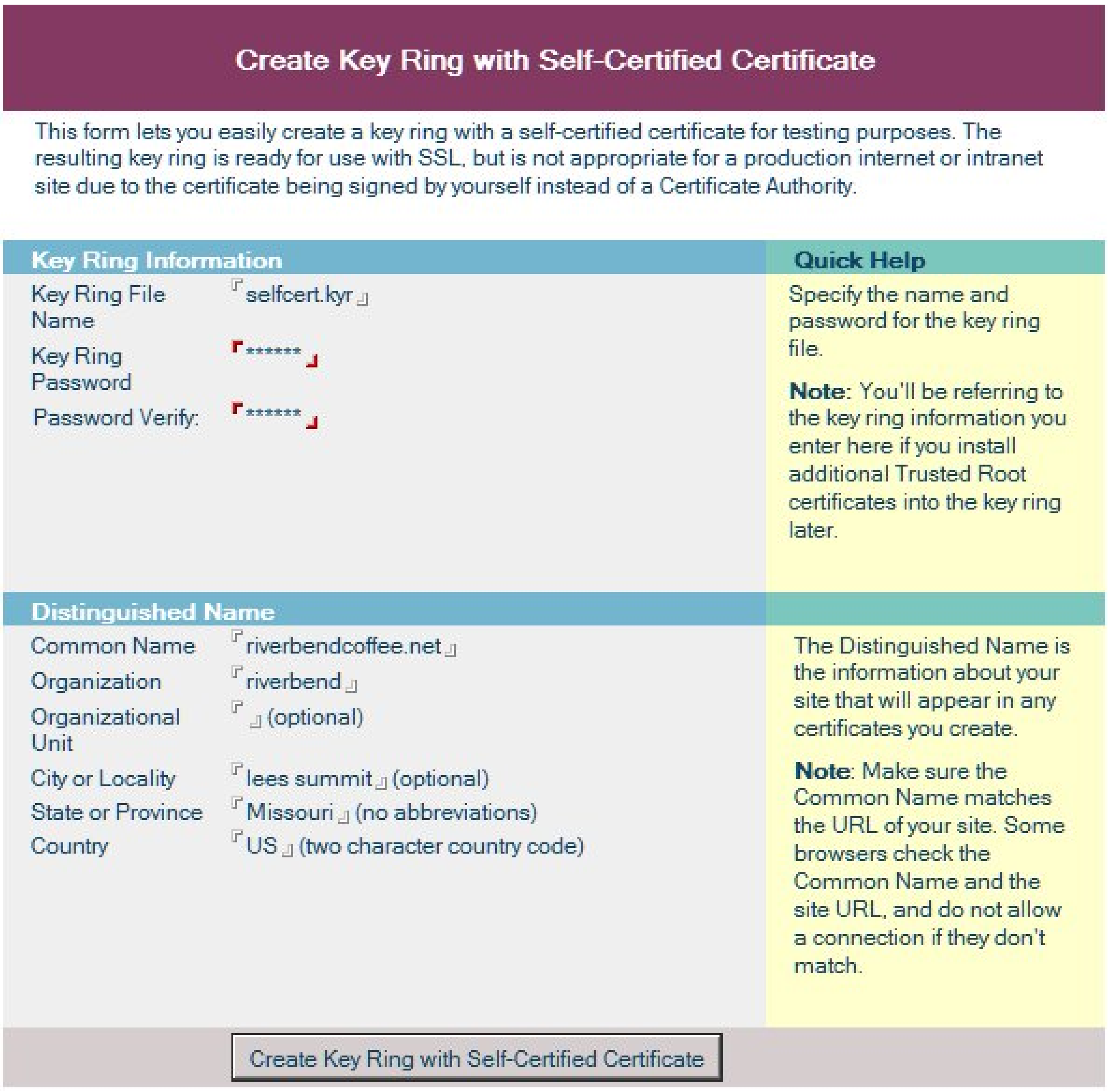

.domino
Server Certificate Administration
Create Key Rings & Certificates
View & Edit Key Rings
View Certificate Request Log
Create Key Rings & Certific
1. Create Key Ring
2. Create Certificate Request
3. Install Trusted Root Certificate into Key Ring
4. Install Certificate Into Key Ring
Create Key Ring with Self-Certified Certificate

Click on the steps below to create an SSL key ring and populate it with certificates.

You can also quickly create a key ring with a self-certified certificate for testing purposes.

Home page

Click **Create Key Ring with Self-Certified Certificate** and complete the fields as shown in the following figure.


Create Key Ring with Self-Certified Certificate

This form lets you easily create a key ring with a self-certified certificate for testing purposes. The resulting key ring is ready for use with SSL, but is not appropriate for a production internet or intranet site due to the certificate being signed by yourself instead of a Certificate Authority.

Key Ring Information

Key Ring File Name	selfcert.kyr
Key Ring Password	******
Password Verify:	******

Quick Help

Specify the name and password for the key ring file.

Note: You'll be referring to the key ring information you enter here if you install additional Trusted Root certificates into the key ring later.

Distinguished Name

Common Name	riverbendcoffee.net
Organization	riverbend
Organizational Unit	(optional)
City or Locality	lees summit (optional)
State or Province	Missouri (no abbreviations)
Country	US (two character country code)

The Distinguished Name is the information about your site that will appear in any certificates you create.

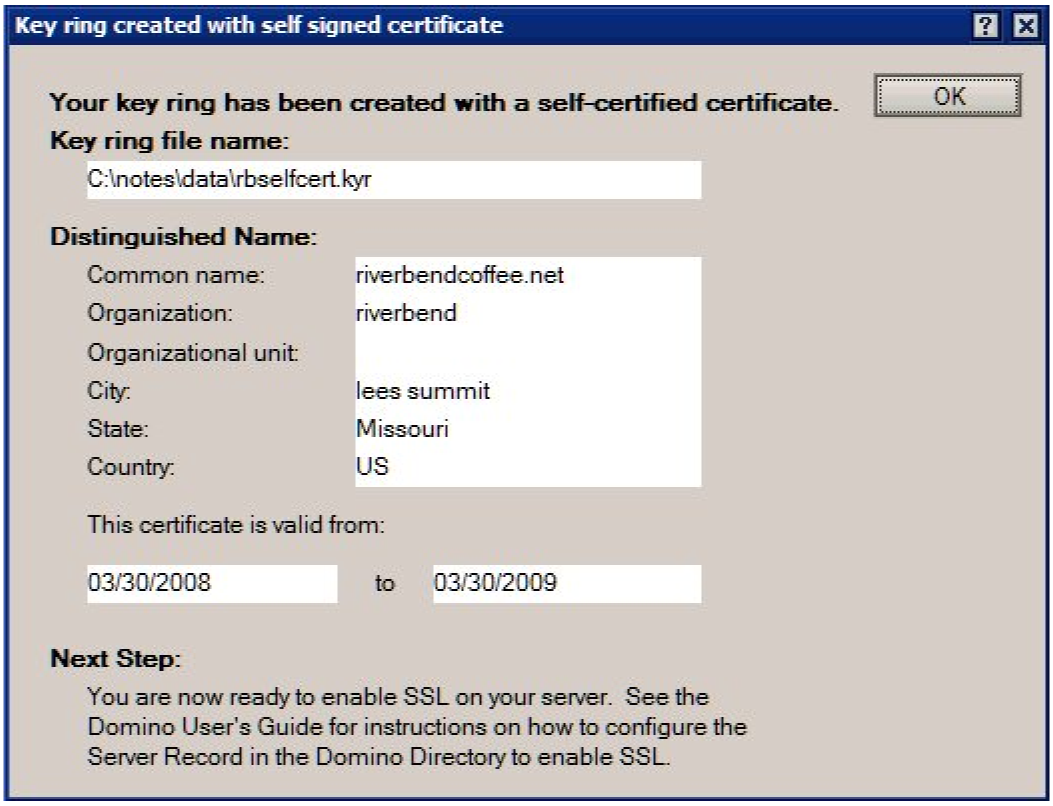
Note: Make sure the Common Name matches the URL of your site. Some browsers check the Common Name and the site URL, and do not allow a connection if they don't match.

Entry form

Then click **Create Key Ring with Self-Certified Certificate**. Use the values from the following table to complete the fields in the Key ring created with self-signed certificate window shown below the table.

Field	Enter
Key ring file name	A file name with the extension .KYR.
Key ring password	At least 12 case-sensitive, alphanumeric characters.
Common name	A descriptive name that identifies the server certificate, such as, RiverBend CA.
Organization	The name of the organization, for example, a company name such as Acme.
Organizational Unit	(Optional) Name of certifier division or department.
City or Locality	(Optional) The organization city or locality.

State or Province	Three or more characters that represent the state or province in which the organization resides, for example, Massachusetts. (For U.S. states, enter the complete state name, not the abbreviation.)
Country	A two-character representation of your country, for example, US for United States or CA for Canada.


Key ring created with self signed certificate

Your key ring has been created with a self-certified certificate.

Key ring file name:
C:\notes\data\rbselfcert.kyr

Distinguished Name:

Common name:	riverbendcoffee.net
Organization:	riverbend
Organizational unit:	
City:	lees summit
State:	Missouri
Country:	US

This certificate is valid from:
03/30/2008 to 03/30/2009

Next Step:
You are now ready to enable SSL on your server. See the Domino User's Guide for instructions on how to configure the Server Record in the Domino Directory to enable SSL.

Key ring created with self-signed certificate window

Now copy the key ring file and stash (.STH) file from you local hard drive to the data directory of the Domino server. You either have to rename the files to keyfile.kyr and keyfile.sth or change the file name on the Web site documents.

Now configure the server to use SSL for the ports that you want encrypted.

Server error handling

This page last changed on Apr 03, 2008 by jservais.

The authentication, authorization, and general errors can be handled on a Web site or individual database basis. If there is error handling in the database, it is used. Otherwise, what is configured in the Domino Web Configuration database is used for error handling. The Domino Web configuration database can be set up by a Web site or server. For a global error form that handles missing files on the OS file system, then use the Global error form. See the Error handling section for information about using database specific error forms.

Topics in this section

Global 404 error form

This page last changed on Apr 03, 2008 by jservais.

To have an error page that handles all 404 errors, including those from missing files in the html folder, set the following entry in the server's notes.ini file or in the server's configuration document:

```
HTTPMultiErrorPage=/error.html
```

The value (/error.html) must point to an HTML page that has anonymous access. The error page URL is relative to the server's base domain. For example, for the www.riverbend.com Web site, the error page is www.riverbend.com/error.html. Since pages or forms in databases have not worked, it has to be an HTML file that is stored in data/domino/html.

After putting the file on the server and updating the notes.ini file, you must restart the server for the changes to be accepted.

Web server configuration database

This page last changed on Apr 03, 2008 by jservais.

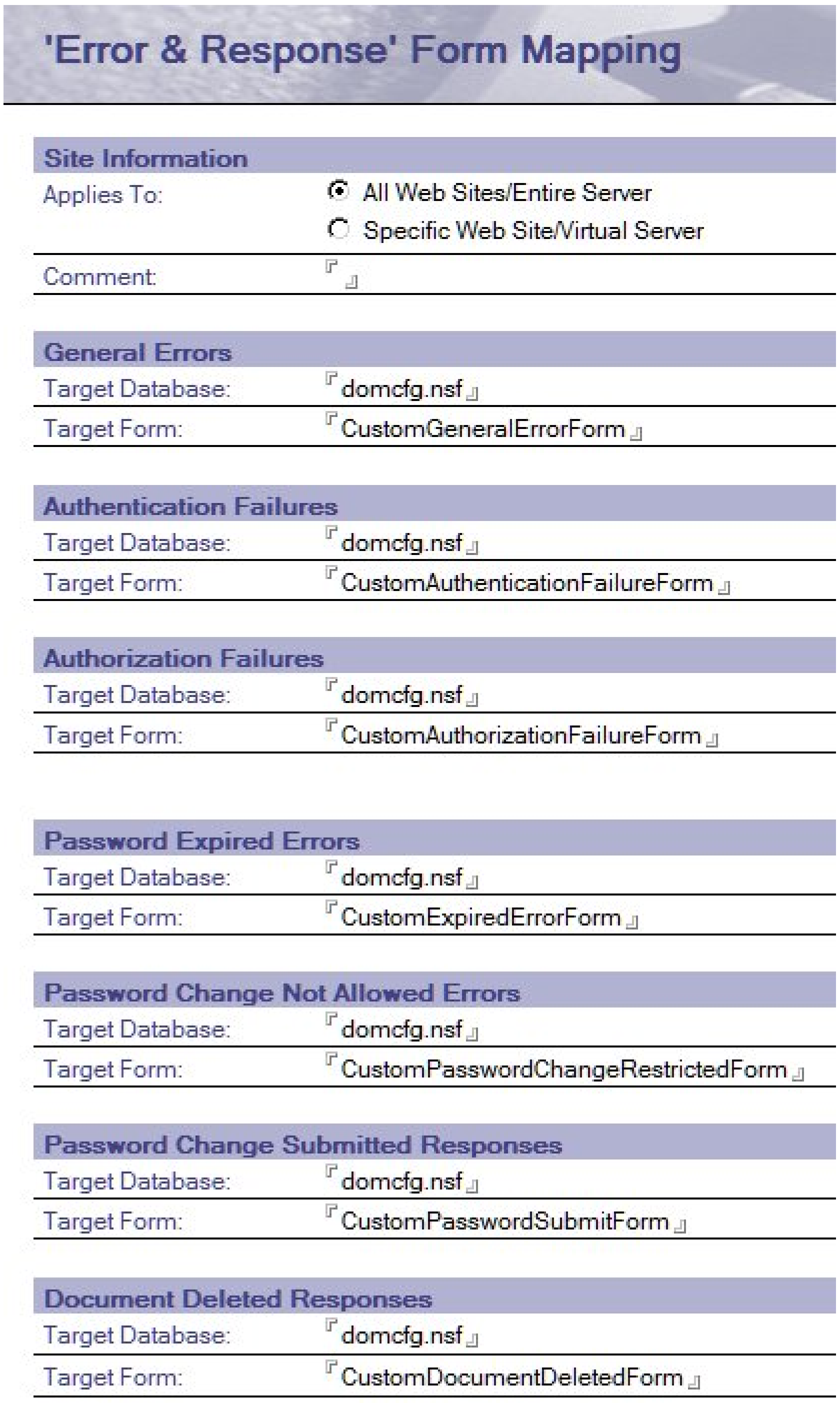
General information

Also known as the Domcfg since it is named domcfg.nsf on the server, the Web server configuration database allows us to specify custom forms for logging in, changing a password, and reporting errors. The database is not created on a server by default, so administrators must create this database and should provide the file name domcfg.nsf, since the server looks for that file name explicitly. The server detects this database automatically. Therefore, no restart is required.

Error and response form mapping

You can specify the default error forms that will be used by an individual Web site or by server. The error forms are used by most specific to least specific, so error forms in a database are used first, then the Web site specific ones are use, then the server specific forms are use, and finally the default Domino error pages are used.

You can create forms inside the Domino Web configuration database or in another database. The database must have at least reader access for users to see the error forms. If you select Specific Web Site/Virtual Server, then you see a field to enter the Web site, virtual server name, or IP address for the server.



'Error & Response' Form Mapping

Site Information

Applies To:	[✓ All Web Sites/Entire Server] Specific Web Site/Virtual Server
Comment:	

General Errors

Target Database:	domcfg.nsf
Target Form:	CustomGeneralErrorForm

Authentication Failures

Target Database:	domcfg.nsf
Target Form:	CustomAuthenticationFailureForm

Authorization Failures

Target Database:	domcfg.nsf
Target Form:	CustomAuthorizationFailureForm

Password Expired Errors

Target Database:	domcfg.nsf
Target Form:	CustomExpiredErrorForm

Password Change Not Allowed Errors

Target Database:	domcfg.nsf
Target Form:	CustomPasswordChangeRestrictedForm

Password Change Submitted Responses

Target Database:	domcfg.nsf
Target Form:	CustomPasswordSubmitForm

Document Deleted Responses

Target Database:	domcfg.nsf
Target Form:	CustomDocumentDeletedForm

Topology

This page last changed on Apr 03, 2008 by jservais.

The location of Domino and other servers within the Riverbend Coffee and Tea Company environment can affect the way applications operate. If the application will reside on a server located in the demilitarized zone (DMZ), then it may have different and restricted access to some of the internal servers. If the application resides on a server located inside the intranet, then that can mean it has greater access to other internal servers for purposes of external database lookups and data retrieval.

In addition, there may be proxy servers between the application and users of that application, which can impact caching and other factors that can affect how we build a particular application.

When designing and creating applications, always keep in mind where the application will reside within the Riverbend Coffee and Tea Company environment.

Working with Web site rules

This page last changed on Apr 03, 2008 by jservais.

Web site rules let you adjust the way Domino will use URLs and authentication. The rules are created by the Domino administrator in the Domino Directory under Configuration/Web/Internet Sites. The rules are tied to a specific Web site document, except for the rules that are tied to the default Web site. With Web site documents, you can specify the way different sites work on your server. You can have www.riverbendcoffee.com point to the home page, while mail.riverbendcoffee.com takes the user to their mail file.

The rules let you do such things as have easier navigation. Instead of the user typing in www.riverbendcoffee.com/web/docs/supporthelp.nsf/help.html, you can use a substitution rule to change it to www.riverbendcoffee.com/help.

Topics in this section

- Directory rules
- HTTP Response Header rules
- Overriding Session Authentication rules
- Redirection and Substitution rules

Directory rules

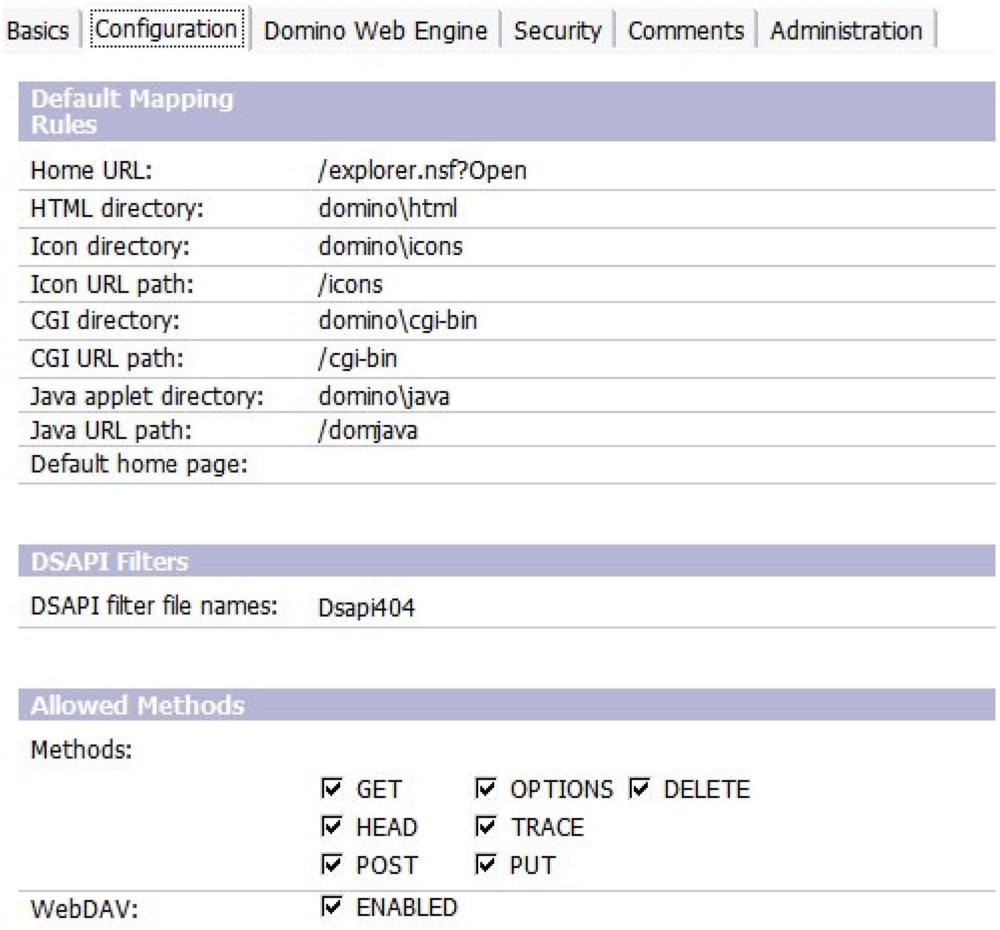
This page last changed on Apr 03, 2008 by dalandon.

A directory rule maps a folder to a URL pattern. When the Domino server receives a URL that matches the pattern, the server uses the folder to access the resource. Directory rules can only be used to map the location of files that are to be read directly, such as HTML files and graphic files, and executable programs, such as CGI programs, to be run by the operating system. Directory rules cannot be used to map the location of resources, such as Domino databases or servlets.

When you install a Domino Web server, some folders are created automatically. These folders are mapped by directory rules that are defined on the Configuration tab of the Web site document. When the Web server starts, it automatically creates internal rules to map these folders to URL patterns.

The following folders are created by default:

- HTML folder for non-graphic files
- Icons folder for graphic images such as the view gifs
- CGI-bin folder for CGI programs
- Java folder for Java applets



Default Mapping Rules

Home URL:	/explorer.nsf?Open
HTML directory:	domino\html
Icon directory:	domino\icons
Icon URL path:	/icons
CGI directory:	domino\cgi-bin
CGI URL path:	/cgi-bin
Java applet directory:	domino\java
Java URL path:	/domjava
Default home page:	

DSAPI Filters

DSAPI filter file names:	Dsapi404

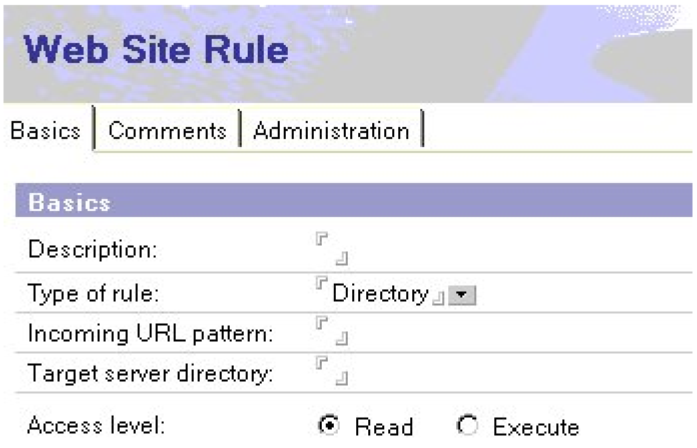
Allowed Methods

Methods:

✓ GET ✓ OPTIONS ✓ DELETE
✓ HEAD ✓ TRACE
✓ POST ✓ PUT

WebDAV: ✓ ENABLED

Web site document



Web Site Rule

Basics | Comments | Administration

Basics

Description:	
Type of rule:	Directory
Incoming URL pattern:	
Target server directory:	
Access level:	[✓ Read] Execute

Directory rule

HTTP Response Header rules

This page last changed on Apr 03, 2008 by jservais.

- Reasons for using Response Header rules
- Creating Response Header rules

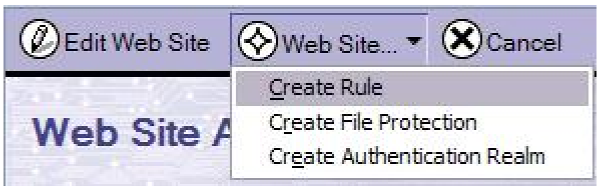
Reasons for using Response Header rules

You might want to disguise the fact that your application is being served by a Lotus Domino HTTP server. For this reason, you can create an HTTP Response Header rule document to overwrite the Server response header with a custom value. Response Header rules can also be useful if you want custom headers to be sent from the server. These headers can then be read and possibly set to new values by your applications.

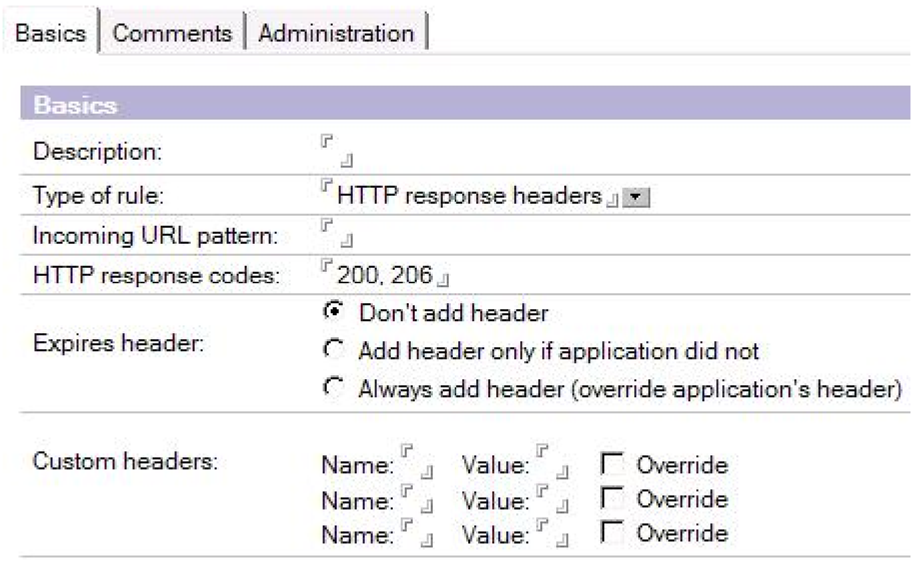
Creating Response Header rules

Note: In order to use these documents, the Domino server must be using Internet sites.

The Domino server can overwrite HTTP response headers if your application needs to do so. To create one of these rules, open a Web site document from the Internet site view of the server's address book. Select Web Site-> Create Rule as shown in the following figure.



On the Basics tab (see the following figure), change the Type of Rule field to be **HTTP response headers**. Then configure the fields as shown in the figure.

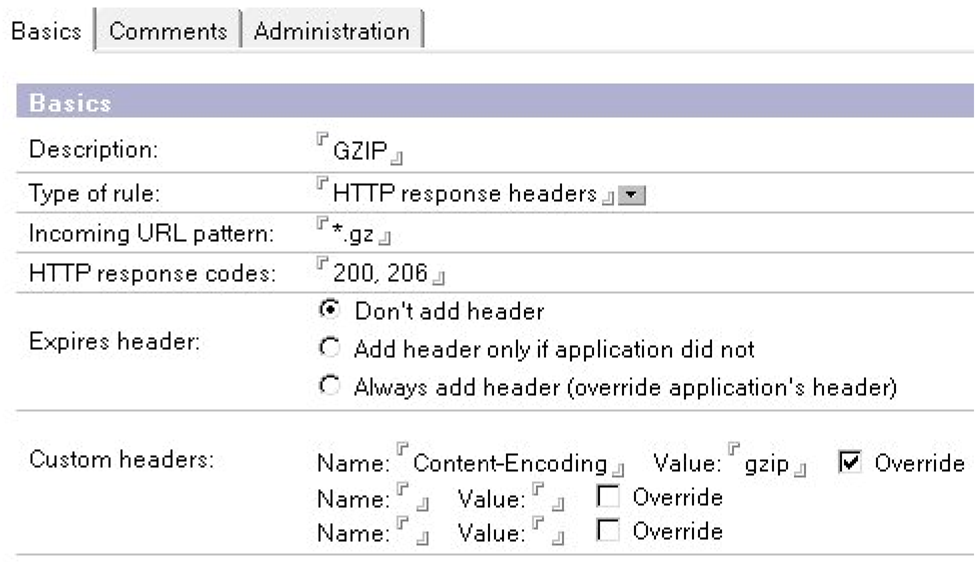


Basics	Comments	Administration

Basics

Description:	
Type of rule:	HTTP response headers
Incoming URL pattern:	
HTTP response codes:	200, 206
Expires header:	[✓ Don't add header] Add header only if application did not Always add header (override application's header)
Custom headers:	Name: Value: ☐ Override Name: Value: ☐ Override Name: Value: ☐ Override

One use is to support gzip files. Gzip is used by Domino for Domino Web Access, but it is not available to developers yet. You can create a Response Header rule to allow the use of gzip files that are stored on the hard drive. With the gzip rule, any file that has an extension of .gz that is requested from the site receives a Content-Encoding header with a value of "gzip". This causes the Web browser to decompress the file prior to using it to render the page.



Basics	Comments	Administration

Basics

Description:	GZIP
Type of rule:	HTTP response headers
Incoming URL pattern:	*.gz
HTTP response codes:	200, 206
Expires header:	[✓ Don't add header] Add header only if application did not Always add header (override application's header)
Custom headers:	Name: Content-Encoding Value: gzip ☑ Override Name: Value: ☐ Override Name: Value: ☐ Override

Response header for GZIP files

Further documentation is available in the Administrator Help file and the KnowledgeBase.

Overriding Session Authentication rules

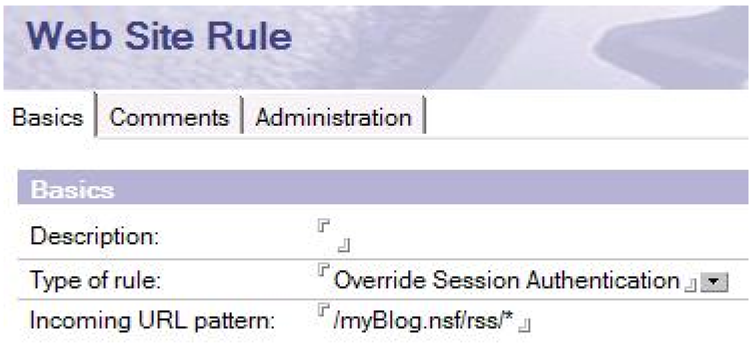
This page last changed on Apr 03, 2008 by jservais.

> **Note**
> This type of Web site rule is only usable when Internet sites are being used on the server.

If Session Authentication is enabled on the Domino server, then the browser client is expected to send a cookie to the server to identify the user who is attempting to access the server resource. If a request reaches the server without a cookie, then the user is presumed to not be authenticated and is seen as an anonymous user.

If Session Authentication is disabled on the server, then the client sends a special Authentication header that contains the user name and password for the user. This is known as *HTTP Basic Authentication*. Some technologies, such as RSS feed readers and WebDAV clients, are only able to use Basic Authentication. If there are any RSS feed readers, WebDAV clients, or other Basic Authentcation-only clients accessing the server, they require Basic Authentication be enabled on the server. Unfortunately this may conflict with other requirements that necessitate having Session Authentication enabled on the server.

For this reason, the Override Session Authentication Rule was introduced in Domino 7.0.2. This rule allows the server to use Basic Authentication for specific requests, while Session Authentication is used for all other requests. The rule must be configured for the URLs that need to use Basic Authentication. This is done by populating the Incoming URL Pattern field after you create the rule.



Description:	
Type of rule:	Override Session Authentication
Incoming URL pattern:	/myBlog.nsf/rss/*

Wildcard characters can be used to match multiple URLs, and multiple rule documents can be created if necessary.

Redirection and Substitution rules

This page last changed on Apr 03, 2008 by jservais.

Introduction

Both Redirection and Substitution rules change the incoming URL to point somewhere else. The main difference is that substitution rules replace the matching section of the URL using wildcards, while the Redirection rule is used recursively to replace a different section of the URL. The Substitution rule is great for when you need to change a folder name and do not want URLs to change. Redirection can be used to hide the Domino file extensions.

Using Redirection rules

Redirection rules can be used on the server for various reasons:

- **Provide aliases for databases**:
  Sometimes it is helpful to have a shorter name or alias for a database that will be accessed via HTTP. For this reason, we can create a Redirection document on the Domino server to make it easier for users to reach our application, or specific pages or resources within the application.
- **Disguise the Domino HTTP server**:
  For security reasons, it may be helpful to disguise the specific type of server that is hosting our Web applications. If a user knows that a Domino server is serving an application, that user can try to craft attacks to exploit vulnerabilities specific to that type of server. Redirection rules can be used so that users do not see that an application has an .nsf extension, and HTTP response header rules can be used to override the Server response header.

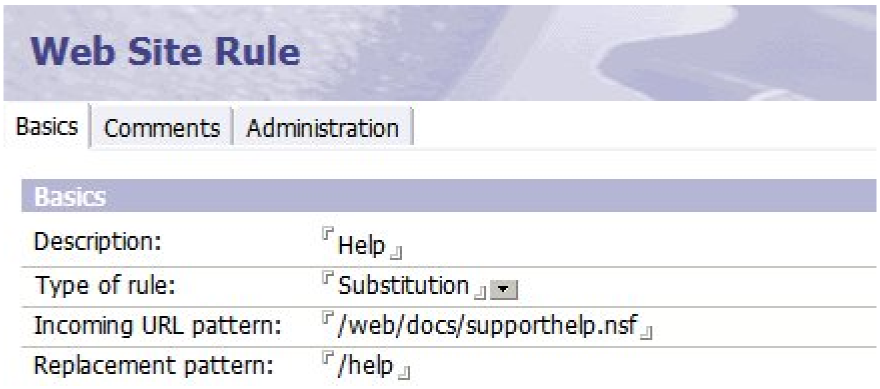
Using Substitution rules

Substitution rules use wildcards. If you do not use one, then the server appends /* to the end of the rule. You can use the rule on the server for the following reasons:

- **Move folders:**
  You need to move databases to a new folder but don't want to have old URLs break. A substitution rule with an incoming pattern of /help/* and a replacement pattern of /product/docs, would let the user access the databases in product/docs with either /help/ or /product/docs in the URL.

- **Make the URL easier to remember:**
  So that the user does not have to remember

www.riverbendcoffee.com/web/docs/supporthelp.nsf/help.html, you make a substitution rule to change it to www.riverbendcoffee.com/help.



Description:	Help
Type of rule:	Substitution
Incoming URL pattern:	/web/docs/supporthelp.nsf
Replacement pattern:	/help

Substitution rule

7.0 Developer tools and resources

This page last changed on Apr 03, 2008 by jservais.

This section provides links to additional information.

Topics in this section

- Domino resources
- Web development resources
- Web development tools

Domino resources

This page last changed on Apr 03, 2008 by jservais.

Useful sites for Domino Web information

Site	Description
Lotus Developer Domain http://www.ibm.com/developerworks/lotus	The main site for Domino information
PlanetLotus.org	A feed from 242 Lotus blogs updated hourly.
OpenNTF.org	The home of open source Notes applications.
Lotus Greenhouse https://greenhouse.lotus.com/home/login.jsp	Lotus Greenhouse is a premier showcase Web site to experience Lotus products.
Notes.ini settings http://www.ibm.com/developerworks/lotus/documentation/notes-ini	developerWorks - viewable by starting letter.
Notes.ini settings http://www.admincamp.de/notesini- original http://www.kalechi.com/notesini- backup	User maintained - viewable by name, categories, and versions and fully searchable

Other topics

- Web development resources
- Web development tools

Web development resources

This page last changed on Apr 03, 2008 by jservais.

Useful Web development Web sites

Site	Description
aListApart.com	CSS and Web best practices
Bookmarklets https://www.squarefree.com/bookmarklets/webdevel.html	Web development bookmarklets for all browsers
CSSZenGarden.com	For CSS examples and information
kuler http://kuler.adobe.com	kuler allows you to create, rate and download color themes. This site provides inspiration when trying to create a color scheme for a Web site or Notes application.
slayeroffice http://slayeroffice.com/tools/color_palette	slayeroffice has a number of good articles about Web development, but its most useful feature is a color palette creator. The site allows you to enter a base color and two colors to mix it with and presents you with 10 shades of the mixture.
URLencoder/URLdecoder Utility http://www.albionresearch.com/misc/urlencode.php	
w3c.org	The home of the Internet standards
w3cSchools.com	The basic how-to for HTML and JavaScript

JavaScript libraries and reference sites

Site	Description
Ajaxian.com	A site dedicated to improving Web development
devguru.com	A another site for Web development information
Dojo http://www.dojotoolkit.org/	Home of the Dojo JavaScript Library
jQuery http://jquery.com	A fast, concise, JavaScript Library that simplifies working with HTML documents

JSON.org http://json.org	The home page for JSON information
MooTools http://mootools.net	A compact, modular, OO JavaScript framework that is designed for the intermediate to advanced JavaScript developer
Prototype http://www.prototypejs.org/	JavaScript Library
script.aculo.us script.aculo.us	Provides an easy-to-use, cross-browser user interface
Yahoo UI http://developer.yahoo.com/yui	Provides a set of utilities and controls, written in JavaScript
EXT http://extjs.com	Provides for a cross-browser UI libraries

Other topics

- Domino resources
- Web development tools

Web development tools

This page last changed on Apr 03, 2008 by jservais.

Useful sites for Web Development tools

Site	Description
Eclipse http://www.eclipse.org	Eclipse project has lots of add-on tools. It is the basis of the Notes 8 client, which makes it a good platform to start learning it now.
CSS Tab Designer http://www.highdots.com/css-tab-designer	A tool to build nice tags without being a graphic artist.
Notepad++ http://notepad-plus.sourceforge.net/uk/site.htm	Editor for text, HTML, JavaScript, C, and so on with extensions.
Scite http://scintilla.sourceforge.net/	Text editor.
Mozilla Firefox http://www.mozilla.com/en-US/	Browser extension.
Web Developer Toolbar https://addons.mozilla.org/en-US/firefox/addon/60	Useful Web development tools, such as live CSS edit.
Firebug https://addons.mozilla.org/en-US/firefox/addon/1843	You can edit, debug, and monitor CSS, HTML, and JavaScript live in any Web page.
View Source Chart https://addons.mozilla.org/en-US/firefox/addon/655	Creates a graphic view of a Web page.
HTML Validator https://addons.mozilla.org/en-US/firefox/addon/249	Helps you validate the current page.
FireShot https://addons.mozilla.org/en-US/firefox/addon/5648	A browser screen capture tool.
LiveHTTPHeaders http://livehttpheaders.mozdev.org/?ver=0.13.1	View HTTP headers of a page while browsing.
Javascript Debugger https://addons.mozilla.org/en-US/firefox/addon/216	The way to debug JavaScript.
Microsoft Internet Explorer	Browser add-ons.
Microsoft Script Debugger and Debug toolbar http://msdn2.microsoft.com/en-us/downloads/default.aspx	Contains all Microsoft add-ons for Internet Explorer.
Fiddler http://www.fiddlertool.com/fiddler	An HTTP debugging proxy that logs all HTTP traffic between your computer and the Internet.
DebugBar	Debugger for IE lets you see the HTML, JavaScript

http://www.debugbar.com	and CSS.

Other topics

- Domino resources
- Web development resources

Looking ahead to version 8.5

This page last changed on Apr 04, 2008 by heinsje.

Looking ahead to Domino and Domino Designer 8.5

With Lotus Domino and Domino Designer 8.5, IBM provides new options for Web application development. Developers can leverage these options for new or existing Domino applications to provide modern interfaces that exploit Web 2.0 capabilities.

Domino Designer 8.5

Domino Domino 8.5 is re-parented into Eclipse to provide new developer capabilities while preserving the developer's and company's investment in applications and skills. Developers have new and refined navigation options, new editors for CSS, HTML, and XML, and additional capabilities and design elements.

New options for rich text fields on Notes forms

Developers can use Dojo control for rich text fields in Notes forms for Web applications. This new option is in addition to the Domino supplied applet, HTML or 'Best Fit for OS' option that is available in Domino Designer 8.0 and earlier. As a developer, you can select the new "Using JavaScript control" option in the rich text property box in Domino Designer 8.5, and the Domino-supplied Dojo control is provided for rich text fields in the Web application.

Well-formed XML

The HTML generated by the Domino 8.5 Web server can be valid XML that can be processed by XML-based processors. This does not include the following code:

- passthru HTML,
- Application-supplied HTML in HTML Head Content
- HTML in the templates in datadirectory/domino/templates

Auto-classing support

With Domino 8.5, class attributes are automatically added to certain HTML that is generated for Notes objects. Developers can apply style sheets to a larger set of Notes objects than in prior versions and use JavaScript on a larger set of Notes objects.

Theming support

Developers can exploit new Domino Designer 8.5 options to apply themes to all design elements in an NSF for use in Web browser applications running on Domino 8.5.

XPages and custom controls

An *XPage* is a new design element based on JavaServer Faces (JSF) technology that lets developers create Web 2.0 enabled pages for use in Web browser applications running on Domino 8.5. XPages remove the barrier of Web programming in Domino by providing advanced page design capabilities and complete control of the generated markup. They also allow access to any kind of data and provide an easy method for localizing applications. XPages includes the following features:

- AJAX enabled (for example, partial page refresh, type ahead capability, and so on)
- Advanced Web control library (tabbed panel, and so on)
- Full support for styling by using CSS
- Fully extensible by using custom controls (composite controls or Java-based controls) or JSF extensions
- Support for multiple clients (Web, rich client, and so on)
- JavaScript scripting language support for client-side and server-side action
- Pre-built simple actions provided for most common cases
- Direct access to Java libraries on the server

A custom control is a collection of controls that are stored as a single object. Similar to subforms in Domino Designer, custom controls are design elements that you can create once and add to multiple XPages. When you update a custom control, every XPage that uses that custom control gets updated with the changes, saving developer time and effort.