

Domino Web Access 7 Customization

Integrate your organization's design
into Domino Web Access

Extend and enhance Domino Web
Access functionality

Leverage new Version 7
design capabilities



Philip Monson
Wolfgang Fey
Shu Sia Lukito
Purvi Trivedi



International Technical Support Organization

Domino Web Access 7 Customization

August 2006

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (August 2006)

This edition applies to Version 6 and Version 7 of IBM Lotus Notes and Domino.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this Redpaper	ix
Become a published author	x
Comments welcome	x
Chapter 1. Introduction to Lotus Domino Web Access customization	3
1.1 Domino Web Access defined	4
1.1.1 Address Book and Journal	4
1.1.2 Domino Offline Services	4
1.1.3 Heterogeneous environment support	4
1.2 The history of Domino Web Access	5
1.3 Why customize Domino Web Access	5
1.4 How customizable Domino Web Access is	6
1.4.1 Skin customization	6
1.4.2 Feature customization	6
1.5 Upgrading a customized Domino Web Access environment	7
1.5.1 Use a custom forms file	7
1.5.2 Isolate your custom code	7
1.5.3 Document your changes	7
1.5.4 Other considerations	8
1.6 Going offline with customized Domino Web Access	8
1.7 Browser considerations	8
1.8 Language pack considerations	9
1.9 IBM support policy for customized Domino Web Access	10
Chapter 2. Customization scenario - ITSO Bank	11
2.1 Corporate standard design customization	12
2.2 Mail management compliance	13
2.2.1 Custom dialog box creation	14
2.2.2 Action menu operations	14
2.2.3 Application integration	15
2.2.4 Further customization examples	15
2.3 Domino Offline Services	15
Chapter 3. Customization considerations	17
3.1 General customization considerations	18
3.1.1 Domino Web Access Customization in Domino 6.5	18
3.1.2 Custom_JS form	20
3.2 Domino Web Access architecture	21
3.2.1 Templates and forms files	22
3.2.2 Special icon note items	24
3.3 Domino Web Access Web page generation	25
3.3.1 Domino Web Access Web page architecture	25
3.3.2 Domino Web Access Web page generation process	26
3.3.3 Proxy documents	27
3.4 Domino Web Access Web pages	28

3.4.1	Main pages	28
3.4.2	Other pages	28
3.5	Domino Web Access scenes and subscenes	29
3.5.1	Standard read and edit scenes and subscenes	29
3.6	Domino Web Access skins	30
3.6.1	Skin group	31
3.6.2	Skin type (skin)	31
3.6.3	Which skin is used	31
3.6.4	Skin components	32
3.7	Forms map table	33
3.8	Domino Web Access special Web page elements	33
3.8.1	Special server-side tags	33
3.8.2	Extensible tags	35
3.8.3	Special URL arguments	35
3.8.4	Special Domino Web Access formulas	37
3.9	Domino Web Access table of contents (TOC)	39
3.10	How to edit Domino Web Access Markup	40
3.11	Taking customized Domino Web Access offline	41
Chapter 4. Skin customization techniques		43
4.1	Skin customization overview	44
4.2	Modifying the stylesheet	45
4.2.1	Change the background of the top toolbar	45
4.2.2	Change background color of the left panel	46
4.2.3	Change the background color of the view label	47
4.2.4	Change the color of the selected item in the outline	48
4.2.5	Change the highlight color in the outline	48
4.2.6	Change the of the Active, Inactive, and highlighted tab	49
4.2.7	Changing the background color for the table of contents toolbar	50
4.2.8	Change the background color for the action bar	51
4.2.9	Change the background color of the column header	51
4.2.10	Change the menu highlight color	52
4.2.11	Change the background color for the message	53
4.3	Modifications to h_ShimmerSkin-h_ListFolder	54
4.3.1	Change the background color around the arrow on the divider	54
4.3.2	Move search next to preferences	55
4.3.3	Remove trash icon	56
4.4	Modifying forms and subforms	57
4.4.1	Change banner to company logo	57
4.4.2	Change the About information box to display company logo	58
4.4.3	Modifications to s_SessionInfo form	58
4.5	Adding a new skin component	60
4.6	Corporate Login screen	61
4.6.1	Corporate logo	61
4.6.2	Adding functionality to display weekday, date, and time	63
Chapter 5. Feature customization techniques		65
5.1	Implementation overview	66
5.2	Domino Web Access functions for Action menu operations	67
5.2.1	DM_getMenuByPos	67
5.2.2	DM_getMenuById	67
5.2.3	DM_getMenuByLabel	67
5.2.4	DM_removeMenu	67

5.2.5	DM_newMenu	67
5.2.6	DM_getParentNode	68
5.2.7	DM_getChildNodes	68
5.2.8	DM_updateActionBar	68
5.2.9	DM_getSubmenuItem	68
5.2.10	DM_createContextMenu	68
5.2.11	DM_getBranch	69
5.3	Overwrite New Message action	69
5.3.1	Create a new dialog page	69
5.3.2	Modify Custom_JS form	71
5.3.3	Modify s_MailMemoDictionary subform	74
5.3.4	Modify s_MailMemoEdit subform	75
5.4	Remove Action Tools → Block Mail from Sender	77
5.4.1	Add Action New → Account	77
5.5	Add a new form, ccount	78
5.5.1	Create a new form in Designer	78
5.5.2	Create document in Notes client	79
5.5.3	Add the FormsMap form to forms7.nsf	79
5.5.4	Create the FormsMap mapping document	80
5.5.5	Special subforms for page creation	80
5.5.6	Add the forms and subforms for the scene	81
5.5.7	Use the action New Account	87
5.5.8	Preferences	89
Chapter 6. Administrative customizations		93
6.1	Different customizations for different audiences	94
6.1.1	How to use different forms databases	94
6.2	notes.ini variables affecting Domino Web Access	95
6.2.1	Configuration settings versus notes.ini parameters	95
6.2.2	Obsolete notes.ini variables in Domino Web Access 7	96
6.2.3	New or modified notes.ini variables in Notes and Domino 7	96
6.3	Domino Web Access redirect personal options	98
6.3.1	Configuration document options	99
6.3.2	DWA redirect options	101
Appendix A. Additional customization tips		105
Creating a custom help document in the Domino Web Access help database		106
Different techniques to include externalized code		106
Tools helpful for Domino Web Access Development		106
NotesPeek		106
Firefox extensions and tools		107
JavaScript debugger		113
Internet Explorer tools		113
Documentation		114
Additional tools		115
Wrapper functions for action menu operations		116
removeActions function		116
addActions function		123
Appendix B. Sneak preview: upcoming features		133
What is in Notes and Domino Version 7.0.2		134
Appendix C. Additional material		137
Locating the Web material		137

Using the Web material	137
How to use the Web material	138
Related publications	139
IBM Redbooks	139
Online resources	139
How to get IBM Redbooks	139
Help from IBM	139

Notices

This information was developed for products and services offered in the U.S.A.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®
@server®
Redbooks (logo) ™
developerWorks®
iNotes™
Domino Designer®

Domino®
IBM®
Lotus Notes®
Lotus®
Notes®
QuickPlace®

Rational®
Redbooks™
Sametime®
WebSphere®

The following terms are trademarks of other companies:

Java, JavaScript, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Internet Explorer, JScript, Microsoft, MSDN, Outlook, Visual Studio, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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

Preface

Domino® Web Access is a client that allows users to access different Domino services using a Web browser. Domino Web Access provides the browser user with access to a number of features that were previously only available for users with non-browser clients, such as Lotus® Notes®.

As you read this IBM® Redpaper, you will discover that Domino Web Access is highly customizable. Domino Web Access is categorized as a markup-based application. As a markup-based application, the markup or script is the factor that defines the look, feel, and function of the application. Therefore, if you know what to change, you can customize Domino Web Access by modifying this markup or script. This Redpaper provides you with the guidance that you need to determine where changes should be made in Domino Web Access to accomplish the customization relevant to your organization's needs. In this Redpaper we use a fictitious financial institution called ITSO Bank to illustrate the customizations necessary for their business needs.

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Cambridge, Massachusetts Center.

Philip Monson is a Project Leader at the ITSO Lotus Center in Cambridge, Massachusetts. Phil has been with Lotus/IBM for 16 years, joining the company when the early versions of Notes were rolled out for internal use. He has served in management, technical, and consulting roles in the IT, Sales, and Development organizations.

Wolfgang Fey works for ebf-EDV Beratung Foellmer GmbH in Muenster, Germany (<http://www.ebf.de>). He is a Lotus Certified Professional Administrator R5 and Domino 6.5 at the Principal level. He has more than 13 years of experience in solution design and architecture, as well as in system consulting and integration and project management. He has in-depth knowledge of the banking and insurance industries and in the operation of data processing centers. He has successfully managed various projects in which he was involved in the design and architecture of Lotus Notes/Domino infrastructure and the development of several Notes/Domino applications for the Notes Client and the Domino Web interface as well as customizations to Domino Web Access since Version 5.0.8. He was one of the IBM Redbook authors of *Domino Web Access 6.5 on Linux*, SG24-7060.

Shu Sia Lukito is a Senior IT Specialist in the Washington, DC, metro area. She has over nine years of industry-level experience in application design and development, specializing in Lotus/Domino and WebSphere® Portal products. She has successfully designed and developed applications for telecommunications, medical, and insurance companies. She has been with IBM for six years. Her areas of expertise include Lotus Notes/Domino applications (for Notes and Web clients), Domino Web Access, and WebSphere Portal theme customization. She has written articles on Domino Web Access customization topics and WebSphere Portal theme customization.

Purvi Trivedi is a Staff Software Engineer with the IBM Lotus Software Support team working with PSP customers. She has been with IBM for three years and her areas of expertise include Domino Web Access, Domino Offline Services, Domino Access For Microsoft® Outlook®, Notes Client installation, and Notes Client Plug-in for WorkPlace Rich Client. She holds a Bachelors degree in Computer Science from the University of

Massachusetts at Amherst and is currently pursuing a Masters degree in Software Engineering at Brandeis University.

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

- ▶ Vinod Seraphin, Senior Technical Staff Member, Domino Web Access Architect, IBM Software Group, Lotus
- ▶ John LeJeune, Domino Web Access Development, IBM Software Group, Lotus
- ▶ John Immerman, Development Manager, IBM Software Group, Lotus
- ▶ Jane L. Wilson, Knowledge System Architect. IBM Software Group, Lotus
- ▶ Jason Dumont, Senior Product Manager, IBM Software Group, Lotus

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll 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 our papers to be as helpful as possible. Send us your comments about this Redpaper or other Redbooks™ in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Introduction to Lotus Domino Web Access customization

This chapter is an overview of IBM Lotus Domino Web Access and the topics covered in this Redpaper. Specifically, we introduce you to:

- ▶ What Domino Web Access is
- ▶ The history of Domino Web Access
- ▶ Why to customize Domino Web Access
- ▶ How to customizable it
- ▶ Upgrading a customized Domino Web Access environment
- ▶ Going offline with customized Domino Web Access
- ▶ Browser considerations
- ▶ Language pack considerations
- ▶ IBM support policy for customizations

1.1 Domino Web Access defined

Domino Web Access (formerly iNotes™ Web Access) is a sophisticated Web client that delivers leading Domino messaging, collaboration, and personal information management (PIM) capabilities to Web browsers. Browser users can take full advantage of Domino services through an ultra-intuitive, easy-to-use interface, both online and offline, seamlessly.

Domino Web Access 7 was architected using the latest Web application development technologies, such as AJAX, and can be centrally administered, and touts a low-touch to the desktop deployment model.

Domino Web Access is a client that allows users to access different Domino services using a Web browser. Domino Web Access provides the browser user with access to a number of features that were previously only available for users with non-browser clients, such as Lotus Notes. These features are in the areas of messaging, calendar and scheduling, personal information management, task management, and personal journal.

1.1.1 Address Book and Journal

One major difference between the Domino Web Access client and the Notes client is that the local address book and local journal do not exist with Domino Web Access. Instead, contact and journal entries are stored in the mail file as opposed to separate databases. For users that maintain both a Notes client and Domino Web Access, client agents are available to sync contact and journal entries. The agent, which is part of the Domino Web Access template, must be run from the Notes client.

1.1.2 Domino Offline Services

Users can also work offline to manage e-mail messages, contacts, calendars, to-do items, and so forth via Domino Offline Services (DOLS).

1.1.3 Heterogeneous environment support

Domino Web Access can be used independently or together with the Lotus Notes client. Users can use the Notes client while they are in their office environment, and use Domino Web Access while they are remote and their only choice is to use a Web browser, such as when they are at another user's PC or on their home PC.

Note: Until Lotus Domino Version 6.5, Domino Web Access was called iNotes Web Access, or iNotes. For this reason, many elements of Domino Web Access (such as mail templates) still carry the iNotes name. For the sake of simplicity in this text, all versions of the product including those released prior to Domino 6.5 are referred to as Domino Web Access.

The Domino Web Access environment is comprised of five components:

- ▶ Supported browser
- ▶ Domino Server running the HTTP task
- ▶ The mail template iNotesX.ntf and dwaX.ntf
- ▶ Shared forms database (Forms5.nsf, Forms6.nsf, or Forms7.nsf)
- ▶ Domino Offline Services (DOLS) for offline use

Important: All of these components need to be considered in any case of customizing Domino Web Access. Sometimes there are several ways to implement a specific customization in different places. In all cases the customization should be tested in a real-world environment.

For an overview of Domino Web Access 7, refer to the product page at:

<http://www-142.ibm.com/software/sw-lotus/products/product1.nsf/wdocs/dwawhatsnew>

1.2 The history of Domino Web Access

The first browser-based mail client that could access Domino mail files was WebMail. WebMail was introduced in Lotus Notes and Domino 4.6. It required the Web Mail template (mailw46.ntf) or the Combo Mail template (mailc46.ntf). In Notes and Domino 5.x to 7.x, WebMail template design elements were combined into the Standard Mail template (mail50.ntf, mail60.ntf, mail7.ntf).

Domino Web Access (then called iNotes) was introduced in Domino 5.0.8 as a new standard browser-based mail client for Lotus Domino. Key points are:

- ▶ Enhanced Web access to Domino messaging/PIM capabilities
- ▶ Targeted at *richer* browsers such as Internet Explorer® and Mozilla that have DHTML support
- ▶ Uses powerful Web technologies (XML, JavaScript™, DHTML, AJAX) to provide enhanced usability, performance, and interface functionality over WebMail
- ▶ Works with Domino Offline Services (DOLS) to allow users to access Web applications (primarily mail files) offline.

1.3 Why customize Domino Web Access

Domino Web Access has a very rich set of features. With that in mind, why do customers still want to customize it? As flexible and feature-rich as any application can be, there are always unique customer needs or creative desires to be met. Some common customizations requested by customers are:

- ▶ Look and feel customization
- ▶ Preferences customization
- ▶ Add and remove features
- ▶ Implementation of a mail retention policy
- ▶ Integration with other systems (such as document management systems, J2EE™ applications, Portal applications, relational databases, and so on)

Before deciding to customize Domino Web Access, note that every release adds more options and flexibility to customize Domino Web Access through administrative settings, for example, configuration documents, notes.ini settings, and so on (see Appendix A, “Additional customization tips” on page 105, for more information). The Domino Web Access design team has devoted increasing attention to customization, and with Lotus Domino 7 the Domino Web Access template is even more customizable.

1.4 How customizable Domino Web Access is

As you read this Redpaper, you will discover that Domino Web Access is highly customizable. Domino Web Access is categorized as a markup-based application. As a markup-based application, the markup or script is the factor that defines the look, feel, and function of the application. Therefore, if you know what to change, you can customize Domino Web Access by modifying this markup or script. This Redpaper provides you with the guidance you need to determine where changes should be made in Domino Web Access to accomplish the customization relevant to your business needs. In this Redpaper we use a fictitious financial institution called ITSO Bank to illustrate the customizations necessary for their business needs. You will read about it in Chapter 2, “Customization scenario - ITSO Bank” on page 11.

There are two main areas of customizations discussed in this Redpaper:

- ▶ Skin customization
- ▶ Feature customization

These areas are covered in Chapter 4, “Skin customization techniques” on page 43, and Chapter 5, “Feature customization techniques” on page 65.

In Domino Web Access 7.0.1, the only tool required for customization is the Domino Designer® Client. However, “Tools helpful for Domino Web Access Development” on page 106 discusses additional tools that are helpful for the customization process.

1.4.1 Skin customization

Chapter 4, “Skin customization techniques” on page 43, discusses how ITSO Bank changes the look and feel of Domino Web Access in order to comply to their standard corporate GUI design. This is followed by slightly more complicated examples involving moving skin components and also explains how to move the search bar next to the Preferences button and to remove the additional trash icon. The final example demonstrates how to create a new skin component. Also illustrated is creating a skin component that displays the remaining database size based on mail file quota.

Based on the scenario examples provided in this paper, you will be able to design sophisticated real-world customizations. You will also find how much easier skin customization with Domino Web Access Version 7.0.1 is compared to Version 6.5.

Skins are now easier to edit as the skins editor tool is not necessary for Forms7.nsf. All skins are stored as file resource design elements. The names of relevant design elements will be the skin group name followed by a hyphen followed by the skin type name.

This Redpaper equips you with techniques on:

- ▶ Domino Web Access look and feel customization
- ▶ Rearranging the layout of a skin
- ▶ Removing skin components
- ▶ Creating new skin components

1.4.2 Feature customization

Chapter 5, “Feature customization techniques” on page 65, demonstrates how extensible Domino Web Access truly is. We start by covering various operations that can be performed on action menus, such as overwrite, remove, and add Domino Web Access action menus. Then we get into more complex customizations involving creating a custom dialog box and

implementing a QuerySave agent. In addition, we cover an example of the common customizations for the Preferences dialog. Finally, we show you how to create a custom main page in Domino Web Access by providing a step-by-step example.

Based on the examples provided in the feature customization section, you will be well equipped to extend the customizations to meet your organization's needs. You will learn techniques to:

- ▶ Manipulate action menus (overwrite, create, remove).
- ▶ Create custom dialog box.
- ▶ Implement the QuerySave agent.
- ▶ Create a new main page.
- ▶ Change the default settings in preferences.

1.5 Upgrading a customized Domino Web Access environment

Chapter 3, "Customization considerations" on page 17, discusses how Domino Web Access has improved from previous releases. Before deciding to add a custom feature to Domino Web Access, review the latest features. You may find that the feature your organization needs is now available out of the box. For example, prior to Version 7.0.1, there is no capability to import country-specific corporate holidays.

Unlike some of the mail template customizations for the Lotus Notes client, custom code in Domino Web Access will not automatically work after a version upgrade. The reason is that there are common external script files that may change from release to release. Additionally, various releases emit the script tags in different ways. Therefore, it is important to consider the following best practices for minimizing problems during the upgrade:

- ▶ Use a custom forms file.
- ▶ Isolate your custom code.
- ▶ Document your changes.
- ▶ Other considerations.

1.5.1 Use a custom forms file

Prior to Domino Web Access 7 it is difficult to have Domino Web Access use a custom forms file other than the one that comes with the product (formsX.nsf, where *X* represents the version number). "Different techniques to include externalized code" on page 106 shows in detail how Domino Web Access can be forced to use a different default forms database.

1.5.2 Isolate your custom code

It is a good practice in Domino Web Access to separate your custom code from the out-of-the-box Domino Web Access code. For example, instead of embedding custom Notes Dictionary fields or vars, you can have them in a separate subform and then insert the subform into the s_MailMemoDictionary subform. Refer to 5.3.3, "Modify s_MailMemoDictionary subform" on page 74 for more information. There are also different techniques to externalize code and caching operations to consider. These topics are covered in detail in "Different techniques to include externalized code" on page 106.

1.5.3 Document your changes

Even though documentation is an important practice for any application development effort, it is that much more important in Domino Web Access because you have to track the changes.

Then you will be able to port the changes to the next version by implementing the logic to the new version of the forms file (formsX.nsf).

In some cases you might be able to simply copy and paste the custom code. However, if Domino Web Access has changed its implementation, you might have to re-write the code so that it will work in the new version. For example, the action menu structure has changed from Version 6.5 to 7.0.1. Refer to 5.1, “Implementation overview” on page 66, for more information.

Important: It is important to know that if you have created custom main pages and dialog box pages, you should not simply copy and paste the entire forms or subforms from the older forms file to the newer one. The markup and scripts that are on the head portion of these forms or subforms are different from release to release. You should start with one of the main pages or dialog boxes that came out of the box and follow a similar procedure as performed when you created the custom pages.

1.5.4 Other considerations

In some cases Domino Web Access customization is only required for a subset of the users in an organization. If so, you can consider setting up a separate environment for these users. This can be done with two different forms files on the same server as outlined in “Different techniques to include externalized code” on page 106. With such separation, if there are resource constraints you have the option to upgrade those users who use out-of-box Domino Web Access first and schedule the upgrade for the other group later.

1.6 Going offline with customized Domino Web Access

Domino Offline Services provides a way for users to take Domino Applications and mail files offline, work in them, and synchronize the changes with an online replica on the Domino server. Changes made to the Forms7.nsf file on the domino server are not automatically included in the DOLS install. For performance reasons, the DOLS install is a pre-packaged fileset that includes the standard Forms7.nsf. Customers have the ability to replace the out-of-box Forms7.nsf with the custom Forms7.nsf from the fileset. In 3.11, “Taking customized Domino Web Access offline” on page 41, we detail the steps for bringing Domino Web Access customizations offline.

1.7 Browser considerations

In the real world there will almost never be two identical implementations of Domino Web Access. So if the customization of Domino Web Access is considered for a particular environment, the development team has to know everything about the browser software used. Also, it is important to know which local desktop restrictions and security software are in use and which security settings in the browser are set.

Note: For the customization examples within this Redpaper, the Redpaper team used Microsoft Internet Explorer 6.0 and Mozilla Firefox 1.0.x.

It is possible to do specific customizations for each browser type differently. For example, in our ITSO Bank scenario the skin customization is done only for Microsoft Internet Explorer, but the corporate logo is done for both, as shown in Figure 1-1.

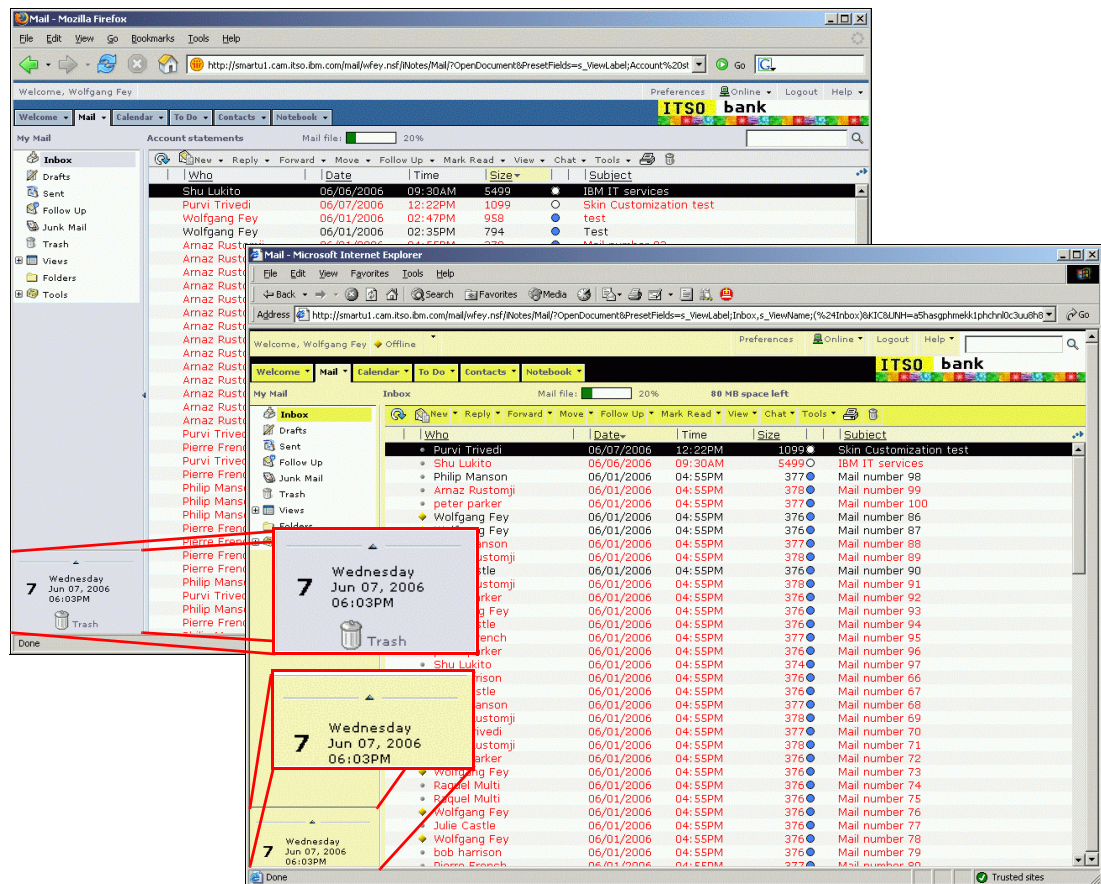


Figure 1-1 Different customization sample for Mozilla Firefox and Microsoft Internet Explorer

1.8 Language pack considerations

Doing customization to a multilingual Domino environment is also possible using the same techniques as we discussed for a single language environment. The design elements used are all the same and have no language-specific names. The only difference in the design is the \$Language item on the respective design notes. The value will reflect the relevant language the design note is to be used for.

Attention: Implementing Domino Web Access customizations in a multilingual Domino environment requires applying the customizations to the design elements for each language.

Also, skins are not language-specific, so any implementation should avoid placing user-visible text directly within a skin. Rather, this is placed within skin components or subforms.

1.9 IBM support policy for customized Domino Web Access

Customizing Domino database templates, including the Standard Mail template, is a common practice with many customers. This has led to numerous questions regarding what type of customization can be made to the Domino Web Access mail template. While it is possible to customize Domino Web Access, and several customizations are discussed in this Redpaper, please understand this disclaimer:

IBM Lotus Support neither certifies nor supports Domino Web Access installations using altered templates or other customizations. IBM Lotus Support will instruct customers who open incidents resulting from a customized template to revert to the stock template to see whether the problem still occurs. If it does, IBM Lotus Support will troubleshoot the problem as it exists in the stock template. If the problem does not exist in the stock template, IBM Lotus Support will recommend that the customer remove the modifications and submit an enhancement request of the desired functionality for the next release. This policy includes the Forms6.nsf database as well.

The current, detailed support policy can be found online in technote 1100952 at:

<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21100952>



Customization scenario - ITSO Bank

In this chapter we introduce a financial institution, ITSO Bank, that has decided to customize their Domino Web Access user interface (UI) to adopt their corporate standard Web design. Additionally, ITSO Bank addresses a Sarbanes-Oxley (SOX) mail management compliance issue by storing copies of outgoing e-mail messages in a separate Domino Database repository.

The following customizations are addressed in detail:

- ▶ Incorporate ITSO Bank corporate design:
 - Color scheme
 - Corporate logo
 - Display available space
 - Corporate login page
 - Updating DOLS to bring customizations offline
- ▶ Mail management compliance:
 - Store a copy of each mail sent out.
 - Remove functionality to block mail from the sender.
- ▶ Integrate an account statement form:
 - Create new documents.
 - Read and edit existing documents.
 - Enable the documents to be forwarded by mail.

This chapter outlines these customizations in general and points to the places in this book where the specific implementation is documented.

Important: Note that this Redpaper does not address the business and legal perspectives of SOX compliance. The scenario is solely used to outline the various techniques used to accomplish Domino Web Access customizations.

2.1 Corporate standard design customization

By implementing a corporate standard design for ITSO Bank we demonstrate how skin customization is accomplished in Domino Web Access. We examine stylesheet modifications to illustrate implementing the corporate UI look and feel. We then discuss modifying HTML layout files to modify and create skin components.

Figure 2-1 displays the out-of-the-box Domino Web Access user interface in comparison with the customized look and feel of ITSO Bank UI shown in Figure 2-2 on page 13.

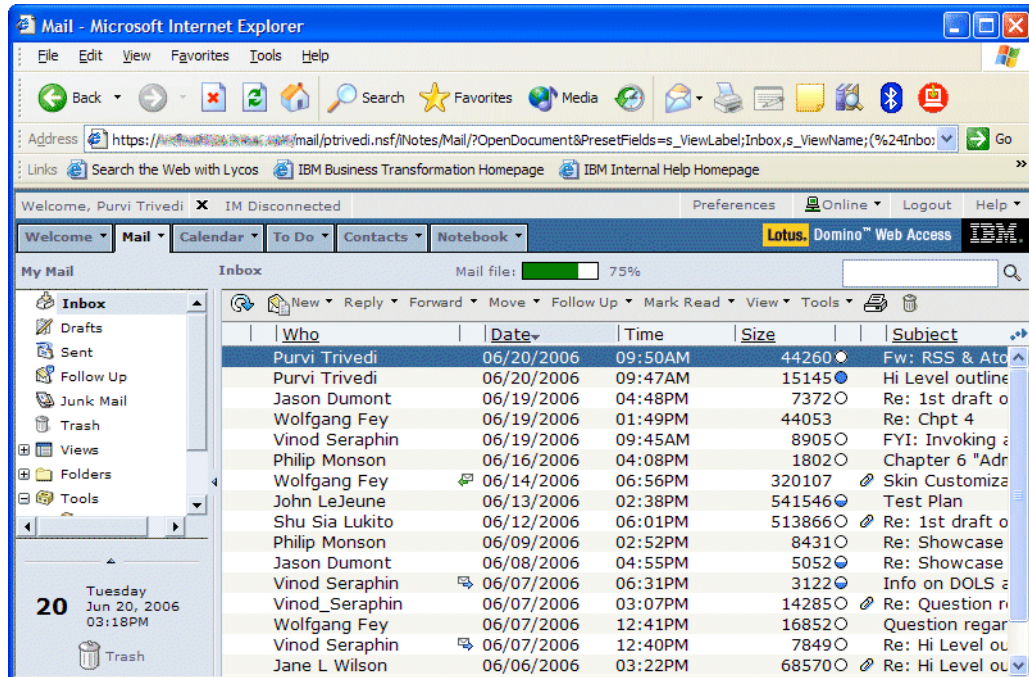


Figure 2-1 Before skin customization

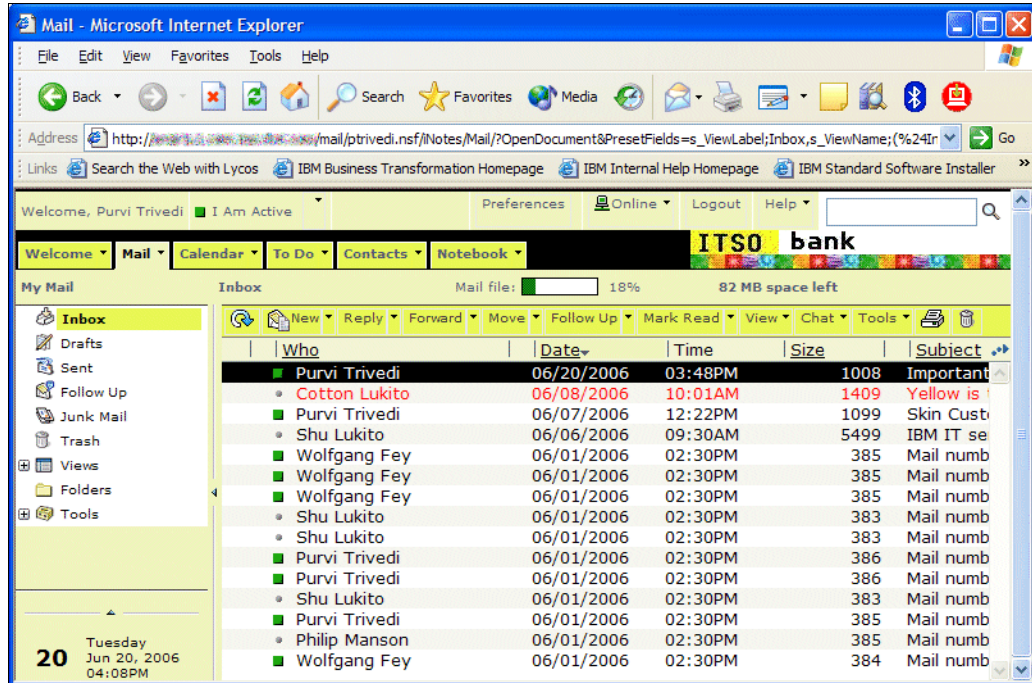


Figure 2-2 After skin customization

Chapter 4, “Skin customization techniques” on page 43, provides you with step-by-step instructions on how to accomplish the standard corporate design for ITSO Bank:

- ▶ Modifying stylesheet
- ▶ Modifying UI elements
 - Background color for divider arrow
 - Removing trash icon
 - Relocating search bar
- ▶ Modifying forms and subforms
 - Introducing the corporate logo
- ▶ Added a new skin component
 - Available mail file space indicator
- ▶ Customizing the login screen

2.2 Mail management compliance

ITSO Bank has a business requirement to implement and integrate an e-mail management compliance system. The main reason is to be SOX compliant as well as having a secure copy of every e-mail in a central repository. Other possible customizations utilizing the same techniques outlined here are integration with other Notes and Domino applications as well as non Notes and Domino applications and the integration of new user interfaces such as the new account form illustrated later. The scenario is used to demonstrate the following techniques:

- ▶ Usage of a custom dialog box
- ▶ Action menu operations
- ▶ Integration of a new main page
- ▶ Further customization techniques

Chapter 5, “Feature customization techniques” on page 65, discusses ITSO Bank mail management compliance customization in detail and provides you with the step-by-step instructions on how to accomplish the different implementations and integrations.

2.2.1 Custom dialog box creation

ITSO Bank requires their employees to select the document category and type for outgoing mail messages from a list of given values prior to sending them. These meta data items are stored in the document and are used for categorization in the mail compliance storage. This is accomplished by overwriting the Domino Web Access *New Message* action so that it prompts the employees to populate the header information via a dialog box (see Figure 2-3) before displaying the Domino Web Access new message page.

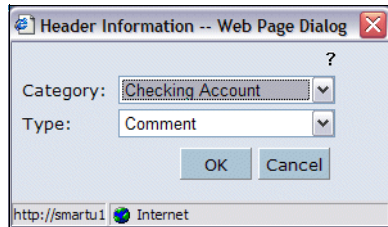


Figure 2-3 Header Information dialog box

The following customization operations are covered in detail by the corresponding sections of Chapter 5, “Feature customization techniques” on page 65:

- ▶ Creating modal dialog box
- ▶ Providing additional JavaScript functions for integration of the dialog box
- ▶ Storing data from dialog box in memo form
- ▶ JavaScript call to existing Domino Web Access functions (such as openNewShimmerDoc)

2.2.2 Action menu operations

ITSO Bank does not want its employees to filter any incoming messages to ensure that every customer e-mail gets delivered and nothing important risks being blocked by a rule. Therefore, ITSO Bank decided to remove the *Block Mail from Sender* item from the Tools action menu, as shown in its default state in Figure 2-4.

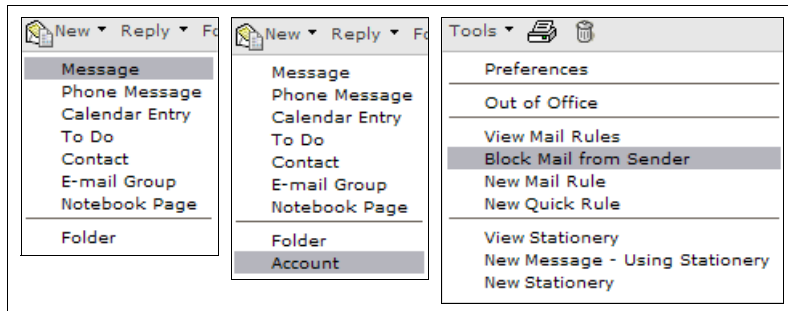


Figure 2-4 Action menu operations

The following customization operations are covered in detail by the corresponding sections of Chapter 5, “Feature customization techniques” on page 65:

- ▶ Remove Tools → Block Mail from Sender submenu
- ▶ Overwrite the New menu and New → Message submenu

2.2.3 Application integration

In order to provide a new form integrated in the Domino Web Access interface to create new account statements, ITSO Bank wants to incorporate an existing Notes form. The form has five fields in which to enter the appropriate data, as shown in Figure 2-5. For the first approach the documents created are stored in the mail database of the user. The technique to store a document in a different database is outlined in 2.2.4, “Further customization examples” on page 15.

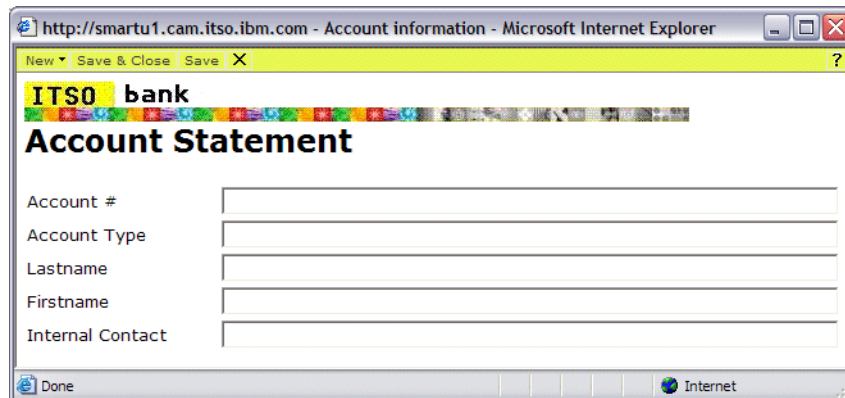
The image shows a screenshot of a Microsoft Internet Explorer browser window. The address bar displays the URL "http://smartu1.cam.itso.ibm.com - Account information - Microsoft Internet Explorer". The browser's menu bar includes "New", "Save & Close", "Save", and "X". The page content features the ITSO Bank logo at the top, followed by the heading "Account Statement". Below the heading, there are five input fields with labels: "Account #", "Account Type", "Lastname", "Firstname", and "Internal Contact". The browser's status bar at the bottom shows "Done" and "Internet".

Figure 2-5 New account form

The following customization operations are covered in detail in the corresponding sections of Chapter 5, “Feature customization techniques” on page 65:

- ▶ Adding the New → Account submenu to create new documents based on the account form.
- ▶ Creating the backend form.
- ▶ Creating the Domino Web Access form.
- ▶ Integrating the form into Domino Web Access using the formsmap technique.
- ▶ Reading existing documents through Domino Web Access.
- ▶ Implementing an edit function for existing documents.
- ▶ What is needed to forward a document by mail.

2.2.4 Further customization examples

The following are additional helpful techniques to create the code for the customizations operating in the background of Domino Web Access. They are covered in detail in the corresponding sections of Chapter 5, “Feature customization techniques” on page 65:

- ▶ How to work with Custom_JS form
- ▶ Use of QuerySave agents
- ▶ Ability to interact with other applications (for example, copies of outgoing e-mail messages stored in a Domino database repository)

2.3 Domino Offline Services

For the mobile ITSO Bank employee the above customizations can be delivered via Domino Offline Services through a Web browser. This will boost productivity by providing a smooth

transition from online to offline. This is discussed in more detail in 3.11, “Taking customized Domino Web Access offline” on page 41.



Customization considerations

This chapter outlines general customization considerations and improvements in Release 7 for Domino Web Access. In addition, we provide an in-depth discussion of the architecture and page generation logic. This chapter includes:

- ▶ General customization considerations
- ▶ What is new regarding customization since Version 6.5
- ▶ Domino Web Access architecture and components
- ▶ Domino Web Access Web page generation components, logic, and process
- ▶ Domino Web Access Web pages
- ▶ Domino Web Access scenes, subscenes, skins, and skin components
- ▶ Forms map
- ▶ Domino Web Access special Web page elements
- ▶ How to edit Domino Web Access markup

3.1 General customization considerations

Corporate layout and identity, as well as more functionality tailored to the specific needs in a browser-based application environment, have made customization of Domino Web Access very popular in companies like ITSO Bank. Tight integration with other Notes applications (as well as non- Notes applications) is also very much desired for customization.

The next few sections provide an overview of the customization options in Domino Web Access Version 6.5 and the subsequent improvements in Domino Web Access 7.

3.1.1 Domino Web Access Customization in Domino 6.5

Domino Web Access Version 6.5 was the first release that focused on making Domino Web Access much more customizable. A Web developer or anyone with development skills in JavaScript, CSS, and HTML could perform customizations. Version 6.5 also added a Custom_Welcome form and a Custom_Banner subform. These and other features are discussed in detail in Chapter 11 of *Domino Web Access 6.5 on Linux*, SG24-7060, all are an improvement over the older releases of Domino Web Access where they were stored as attachments to documents. How to modify images in Release 6.5 of Domino Web Access is described in Chapter 11 of the IBM Redbook *Domino Web Access 6.5 on Linux*, SG24-7060.

Skin customization

Prior to Domino Web Access 7.0, you needed an additional tool for skin customization, called *DWA Skin Editor*, downloadable from the Lotus Sandbox on developerWorks® at:

<http://www-10.lotus.com/1dd/sandbox.nsf/ecc552f1ab6e46e4852568a90055c4cd/66aed8a94144eafa85256e2e006d38ba>

Now in Release 7, the design elements for skins are stored as file resources. You can easily export the file resources and make modifications with your editor of choice (for example, Notepad, or your favorite professional text editor tool like Rational® Application Developer or Visual Studio®). Once you finish with your changes you can import the modifications back into Domino Designer. Also, you can use the Open with action to have Designer export this to its temp directory and then invoke your favorite editor to edit this file. The Refresh action may then be used to import in any saved changes to the temp file.

Action menu operations

The action menu implementation in Release 7 has changed drastically. The newer object structure makes action menu manipulation easier. For instance, the function call that creates a new menu entry does this automatically for the top menu as well as for the right-click menu. Additionally, there are new functions that make working with the action menu far easier than in previous releases. Refer to 5.1, “Implementation overview” on page 66, for more details.

Enable and disable functional areas

This item refers to a new INI setting that allows a way to easily turn off functional areas. Previously there was a way to customize the template and remove elements in the Table of Contents view, but it was too complicated for most people. This notes.ini setting makes it much easier to do something like solely expose the mail and contacts functionality. With this feature if mail is disabled in addition to not having the Mail tab, the Mail panels in Preferences also disappear, and so do various actions that do not make sense if Mail is not available.

One of the significant improvements in the notes.ini settings is the capability to easily disable certain functional areas. This is done by the notes.ini Variable iNotes_WA_Areas. The specific values are documented in “iNotes_WA_Areas” on page 96.

New dwa: tags

The new dwa: tags are used in cases where only a custom Domino Web Access formula is being called. It removes the need to invoke the formula evaluator and improves performance. This new tag is designed to be extensible. The tags are defined in dwa_tags form. Example 3-1 shows an example of a tag definition.

Example 3-1 Tag definition and usage example

The following shows a sample of a tag definition:

```
<tagdefinition name="getvars" outputformat="jscript">
<description>Provide access to the h_Vars functionality. </description>
<argument type="text" optional="yes" name="unidstring" />
<argument type="text" optional="yes" name="prefixstring" />
<argument type="text" optional="yes" name="includeall" />
<bodyfunction namespace="haiku" builtin="h_Vars" />
</tagdefinition>
```

and here's an example of the usage:
<dwa:getvars includeall="0" />

Base64 encoding replaced by @formula

Formulas on forms and subforms are no longer compiled and Base64 encoded and hence are easily modifiable.

Base64 encoding was necessary for performance reasons for optimizations. In Release 7, server-side forms processing code has been enhanced to compile these formulas the first time the form is processed and then to store the compiled formula within a server cache to avoid having to recompile it every time the form is utilized. This is a huge improvement for customization.

Reduced line length to meet Designer limitations

Reducing the line length is another enhancement. Prior to Version 6.5.5, if a form had been built and saved with edits in a tool other than Domino Designer, the line length sometimes would be truncated when edited in Designer. This would result in broken script files.

Since Version 7, the line length in the build process has been reduced so that all forms can be edited with Designer without breaking the scripts. This enhancement has been implemented in Versions 6.5.5 and 7.

QueryOpen/QuerySave agents

In traditional Domino Web applications developers are familiar with the QueryOpen and QuerySave agents. Now this option is available in Domino Web Access as well. This feature was developed for Domino Web Access 7 and also added to Version 6.5.5.

In Domino Web Access, QueryOpen and QuerySave agents must reside in the mail template and can be written in any of the supported agent languages (Notes formula, LotusScript, or Java™).

Obfuscation list as file resource

One of the customization challenges in Domino Web Access is that JavaScript code used in Domino Web Access is obfuscated (or condensed). This is necessary to improve the runtime performance of the application.

The obfuscation list, a text file that describes the mapping of the meaningful names to the obfuscated names, is now included as a file resource in the forms database, in both Release 6.5.5 and 7. The descriptive names give developers a better idea of what the JavaScript functions are for.

For releases prior to 7 the obfuscation list could be retrieved as additional material for the IBM Redbook *Domino Web Access 6.5 on Linux*, SG24-7060.

Tip: To use the obfuscation list, export the *ObfuscationList.txt* file resource from Domino Designer to the disk drive. You can then open it with an editor of choice and perform lookup on any obfuscated abbreviation. Refer to Example 3-2 on page 20 to see what the list looks like.

Example 3-2 Sample section of obfuscationlist.txt

```
=====
The keys on the left have replaced the following functions and variables in DWA
code:
=====
AA IWAOfflineCtrl_DoInstall
AB getViewList
AC bEditable
AD refreshtree
AE sFormsFile
AF tableMenu
...
```

Lite skin fully incorporated

Lite skins were originally introduced to speed up performance by reducing the number of images employed within Domino Web Access pages. Starting in Release 7, the lite skins are incorporated into the normal skins, so there is no difference anymore in loading the default UI versus the lite UI of Domino Web Access.

The lite skins are no longer utilized by Domino Web Access Release 7.

3.1.2 Custom_JS form

The *Custom_JS* form in Version 7 still plays an important role in Domino Web Access customization. It serves as a container for a collection of customization entry point functions that are included in almost every Domino Web Access page. The core part of this was the introduction of a common non-obfuscated form named *Custom_JS*, which is an external script file emitted along with just about every Domino Web Access page.

Within this script file there are some key callback routines that get invoked from the normal code and allow specific places where a developer might gain control and alter Domino Web Access functionality or behavior. At present, the key entry points are just before the action menu is displayed, during the JavaScript onload handling, and whenever a submit is about to occur to the server (in other words, an action is about to be executed).

The *Custom_JS* form is also a place where custom static script routines might reside. It is important to remember that only static scripts should be placed here.

Custom_JS is cached and therefore it should not be used to put user-specific scripts or formulas there. The more appropriate form for user-specific code is in form *s_SessionInfo*. For

example, if you have a lookup based on the user logged in, you should have the lookup in s_SessionInfo. If the lookup is performed in Custom_JS, the formula would be evaluated for the first user who logged in, and the output is cached on the server. This evaluated value will then be incorrect for all other users.

Note: HTTP responses to URL requests from the forms file that have the &MaxExpires (or &MX) argument specified will be cached on the server.

In “Different techniques to include externalized code” on page 106 there are two different ways shown to externalize code from forms and subforms.

Tip: Add to the s_SessionInfo form if per-user or per-mail file formula computations are needed. This can be done via a subform or via an external script (for example, if a user-specific profile setting should be retrieved and assigned to a global javascript var).

The functions provided as a standard entry point include:

- ▶ Scene_Actions, called just before the action buttons are added to the action bar. Some possible customizations include adding a new item, removing an item, or changing the position of an action item.
- ▶ Scene_PostLoad, called after the body is loaded.
- ▶ Scene_PreSubmit, called just before a form is submitted, but after Domino Web Access has done its own validation.

Additional useful functions in Custom_JS are documented in Table 3-1.

Table 3-1 Callbacks and functions

Function	Meaning
Scene_Actions	Add, remove, or change actions.
Scene_PostLoad	Invoked after page has executed its onLoad handlers (but still within onload event).
Scene_PreSubmit	Invoked prior to submitting an action to the server.
API_SetWelcomePageInboxColumns	Set which columns get displayed for the Mail panel in the Welcome page.
API_TimeZones	Add additional time zones.
API_GetSelectedDocs	Return array of strings representing UNIDs.
API_IsView	Return true if view page.
API_GetMailfilePath	Return URL path up to and including .nsf.

3.2 Domino Web Access architecture

In order to customize Domino Web Access it is important to understand the architecture and the processes of how the different components are integrated and work together, as well as how they are used to construct Domino Web Access pages. Figure 3-1 shows an overview graphic of the general Domino Web Access architecture. As shown, the user’s mail files inherit their design from the dwa7 template. The page is finally generated with the HTML and

JavaScript from the forms7 database and the document and view contents read and stored to the user's mail file.

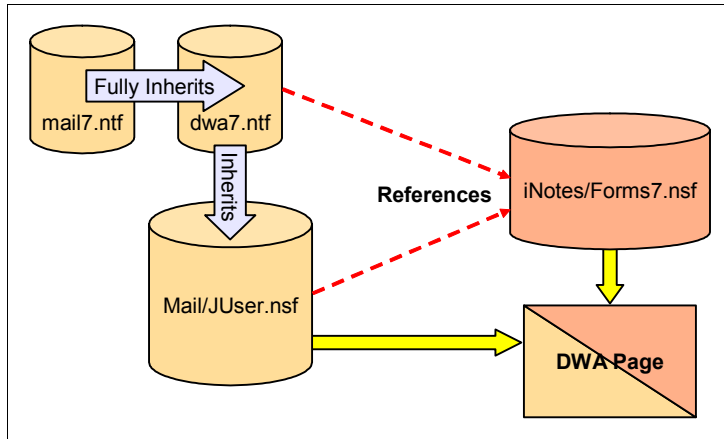


Figure 3-1 Domino Web Access architecture overview

3.2.1 Templates and forms files

The following subsections describe the technical architecture and design elements of Domino Web Access and the components needed for the page generation.

The Domino Web Access template inherits fully from the standard mail template. This allows the same user experience offered by the standard mail template when using the Notes client. For typical Notes applications this is about it and all the design elements are located within the template.

However, for Domino Web Access the template also has a reference to another Notes database known as the forms file. This is where the majority of the design elements that impact the Domino Web Access UI reside and where you will need to make most of your customizations.

When the Domino Web Access template is associated with a user's mail file (when either creating a new mail file or doing a design/replace), the reference to the forms file also moves over to the mail file. This reference to the forms file is stored in a notes item in the icon design note. This is explained in more detailed in 3.2.2, "Special icon note items" on page 24.

So the key thing to understand is that design elements within the forms file and the user's mail file work together to help build a typical Domino Web Access page.

Domino Web Access template

A Domino Web Access user's mail file is based on the *iNotesX.ntf* or *dwaX.ntf* template, where *X* refers to a version number, for example, *iNotes6.ntf* or *dwa7.ntf*.

The relationship between mail template file name and version, the Lotus Domino version, and the forms file name is documented in a table in the public technote 1158614, which is available online at:

http://www-1.ibm.com/support/docview.wss?rs=0&q1=1158614&uid=swg21158614&loc=en_US&cs=utf-8&cc=us&lang=en

The differences between the Domino Web Access templates and the standard mail templates (*mail7.ntf*, for example) are transparent to users when accessing a Domino Web Access mail file via the Notes client. The Domino Web Access template offers full compatibility with the

Notes client. In other words, users who access mail files from the Notes client will not notice anything different.

Although the Domino Web Access template has all of the standard forms and views associated with the standard mail template (for compatibility with the Notes client), most of the design elements used to generate and display Domino Web Access in the browser are retrieved from the shared forms database. The only design elements used from the Domino Web Access template are:

- ▶ Standard views/folders
- ▶ Proxy documents

The Domino Web Access template maintains some native design elements that are distinct from the standard mail template, such as:

- ▶ One agent to synchronize the journal entries called *Synchronize Journal*. Another agent to synchronize the address book called *Synchronize Address Book* is contained within the standard mail template and supports the synchronization for both WebMail and Domino Web Access.
- ▶ Views explicitly for Domino Web Access:
 - Tasks view named Haiku_TasksAll
 - TOC views named Haiku_TOC) and iNotes_ArchiveTOC
 - Proxy document view named iNotes
 - Meeting Notices view named iNotesNotices
 - Contacts view named iNotes_Contacts
 - Notebook view named \$Journal

Note: The new Domino Web Access templates for Version 7.0.1 delivered through IBM Lotus Notes access for SAP solutions are also supported by Domino Web Access though the SAP functionality is only available in the Notes client. For more information refer to:

<http://www-128.ibm.com/developerworks/lotus/library/notesforsap/>

The Shared Forms database

The *FormsX.nsf* (*X* refers to a version number) database is one of the databases included as part of Domino Web Access. It contains most of the JavaScript, pass-through HTML, and images used to implement the User Interface (UI) of Domino Web Access.

All the forms, subforms, and graphics used by Domino Web Access (except for certain images in mail views that are either in the Domino icons directory or within the mail template) reside in the FormsX.nsf database located within the <domino data>\iNotes\ subdirectory on the server.

The reason for keeping design elements in a different database, instead of in individual mail databases, is that they can be cached on the server. All the Web browsers accessing mail files on a server will use the same design elements, which can be loaded from the server's cache. Caching the elements on the server allows better performance on the server.

The forms database is used by the Web server to assemble the appropriate forms based on user action and then serve them to the client as the Domino Web Access UI. It is located in the <domino data>\iNotes\ directory on the Domino Web Access server.

The Forms database includes the following design elements:

- ▶ Forms
- ▶ Subforms

- ▶ Graphics - stored as file resources
- ▶ JavaScript
- ▶ Skins (HTML and CSS files used to define the general layout of various pages) - stored as file resources
- ▶ Forms map table (maps Notes forms to corresponding Domino Web Access forms) - stored as documents in a special view

3.2.2 Special icon note items

In 3.2.1, “Templates and forms files” on page 22, a reference has been mentioned between the Domino Web Access mail template and the forms database. This reference is stored in special items of the icon design note. Figure 3-2 shows the icon design note property box with the special items.

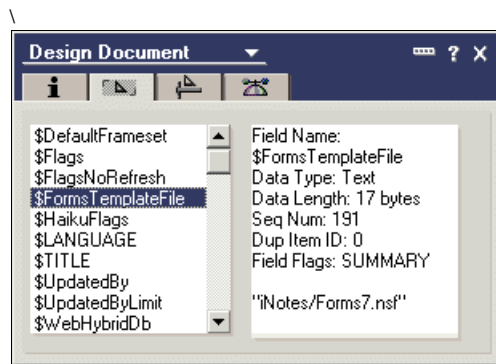


Figure 3-2 The icon design note properties

Within the icon design note of databases created or updated with a Domino Web Access mail template is a field named \$WebHybridDB. The value of this field is set to 1, indicating that this is a Domino Web Access mail database. This is how the Domino Web server identifies a Domino Web Access mail file. The icon note also includes a field called \$FormsTemplateFile.

For the dwa7.ntf template file the default values for special Domino Web Access items of the icon design note are shown in Table 3-2.

Table 3-2 Default values for special Domino Web Access properties of icon design notes

Property	Value
\$HaikuFlags	1
\$WebHybridDb	1
\$FormsTemplateFile	iNotes/Forms7.nsf

These three items govern the association and move over when a Domino Web Access template is assigned to a mail file. The first two are used to trigger some different code paths on the Domino Web server — whether to use the traditional Domino Web page algorithms or the Quickplace and Domino Web Access introduced ones. This is what allows an ?OpenDocument on any document in the mail file to either display the normal WebMail rendering or the Domino Web Access rendering based on which template is being employed.

Attention: Be aware that at present there are some security checks built into the Domino server to only honor a fixed list of forms files. You might be tempted to experiment with adding these items to other application templates, but be warned that this is supported only for Domino Web Access templates.

Even if it is possible to change these item values, they should remain untouched. There is a better way to change the forms database, which is described in “Different techniques to include externalized code” on page 106. It shows in detail how Domino Web Access can be forced to use a different default forms database.

3.3 Domino Web Access Web page generation

It is important to understand how the Web pages for Domino Web Access are generated by the http server. The following sections give an overview of how page generation works and which components are involved in the process.

3.3.1 Domino Web Access Web page architecture

The main pages of Domino Web Access use the form *h_PageUI* for their construction. Figure 3-3 shows the main Notes form *h_PageUI* elements used for the construction of the main Domino Web Access Web pages. For the differentiation of main pages and other pages see 3.4, “Domino Web Access Web pages” on page 28.

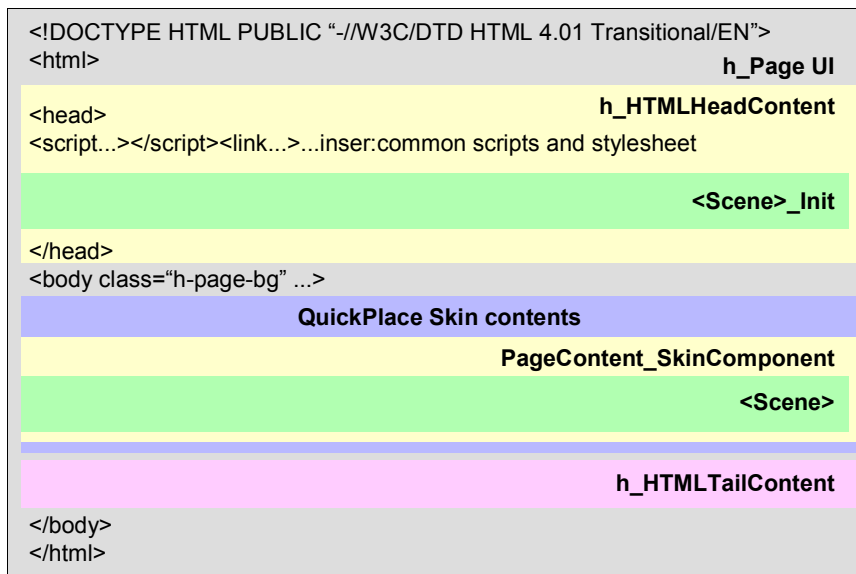


Figure 3-3 Domino Web Access Web page construction

h_PageUI

The form *h_PageUI* contains the HTML opening and closing tags and includes various other subforms and components. The inclusion of subforms is not done in the way Notes normally uses subforms. Subforms are included with the special DWA tag *<InsertNotesSubForm name=...>*. These special tags can be cascaded so that a subform can include a subform that again can include a subform, and so on. More detailed information is covered in 3.8, “Domino Web Access special Web page elements” on page 33.

h_HTMLHeadContent

The first subform included is the `h_HTMLHeadContent`. This subform also includes a subform named `s_CommonVars`. Both contain a lot of JavaScript code and also special Domino Web Access @formulas to set up the Domino Web Access page environment. See 3.8.4, “Special Domino Web Access formulas” on page 37, for more information about how to use @formulas in Domino Web Access. The `h_HTMLHeadContent` also includes the scene initialization for the active *scene*. For details about scenes refer to 3.5, “Domino Web Access scenes and subscenes” on page 29. For example, the included subform could be the custom subform `s_AccountRead_Init` designed for ITSO Bank. To create a new subform for initialization of a new main page see “Subform `s_AccountRead_Init`” on page 83.

Body

After the HTML header the body starts next. The body contains a special DWA tag called `<QuickPlaceSkin>`. This tag is a placeholder that is replaced with the full contents of the appropriate skin (within the appropriate skin group) for this page. The skin consists of the `PageContent` skin component and possibly other additional skin components or subforms. The page content skin component is again a subform named `PageContent_SkinComponent`. This subform includes more subforms to construct the page. One of them is the scene subform for the document type or view being displayed. In the ITSO Bank scenario, this is `s_AccountRead`, which contains the unique HTML and JavaScript code for the page. The subform is described in “Subform `s_AccountRead`” on page 84.

h_HTMLTailContent

The next step is to close the body tag and to load the `h_HTMLTailContent` subform that contains additional JavaScript code and @formulas to load some helper functions into the browser.

At last the html page is closed.

The special HTML tags as well as the special URL parameters for the Domino Web Access page construction are explained in 3.8.3, “Special URL arguments” on page 35, and 3.8.4, “Special Domino Web Access formulas” on page 37.

3.3.2 Domino Web Access Web page generation process

When a database is opened with a browser, the Domino Web server processes the URL and inspects the database icon note for a field called `$WebHybridDB`. If this field is found and its value is set to 1 (default value), the Domino Web server recognizes that this is an Domino Web Access mail database.

Next, the Domino Web Access logic in the Domino Web server is invoked, overriding the server’s normal operation. The Domino Web Access logic locates the shared forms database by retrieving the value from the field `$FormsTemplateFile` (also found in the database icon note). The Domino URL `?OpenDatabase` results in a server-side redirect to an `?OpenDocument` URL to display the `s_Start` scene. This scene then redirects to the desired first functional area view page, based on server configuration settings or personal user preference settings. If no specific page is specified, then a URL is generated to the first document the user has access to within the view called `s_TOC` (Table of Contents).

Special tags (as shown in 3.8, “Domino Web Access special Web page elements” on page 33) are interpreted by the server to:

- ▶ Insert skin.
- ▶ Insert skin components.
- ▶ Insert subforms.

- ▶ Evaluate computed blocks (@formulas).
- ▶ Determine persisted and computed-only items.

All URLs to top-level pages have two levels (view level and document level) after the .nsf extension. The parts of a Domino Web Access URL are annotated in Figure 3-4.

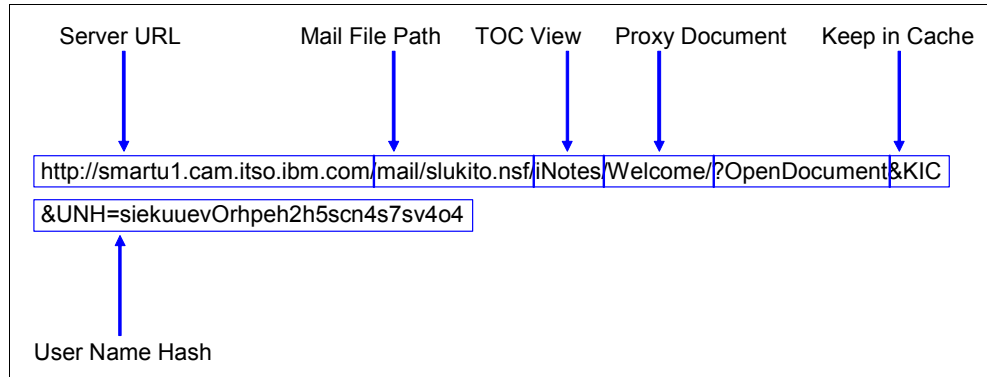


Figure 3-4 Annotated Domino Web Access URL

The Domino Web Access mail template contains a special view named iNotes, which houses special proxy design elements (as shown in 3.3.3, “Proxy documents” on page 27) being used by the HTTP task to process the URL and insert the appropriate forms, subforms, and other elements into the HTML stream to the browser.

3.3.3 Proxy documents

Domino Web Access uses the QuickPlace® architecture for generating Web pages. Every page generated is generated as a result of a ?OpenDocument or ?EditDocument URL. There is no concept of opening a view, but pages may be generated that contain data from one or more Domino views.

Displaying a view is done via Proxy documents. Proxy documents can be opened via the ?OpenDocument URL command and can include data from a view. Examples of proxy document design elements are iwaMail, iwaCalendar, and iwaContacts. They are located in the Domino Web Access mail template.

The special Domino Web Access page generation logic takes care of a bunch of special markup tags and formula code in the page source. The most commonly used are explained in 3.8, “Domino Web Access special Web page elements” on page 33.

The Proxy documents are viewable from the Notes client (open db to view hidden views). The meaning of these documents is to wire specific scenes for specific pages. Figure 3-5 shows an example of a proxy document.

Tip: Proxy documents can be viewed with the NotesPeek utility. This utility is shown in “NotesPeek” on page 106.

Figure 3-5 shows a proxy document in the Notes client. This is possible but a little bit more work to see the fields and document structure through the document properties dialog.

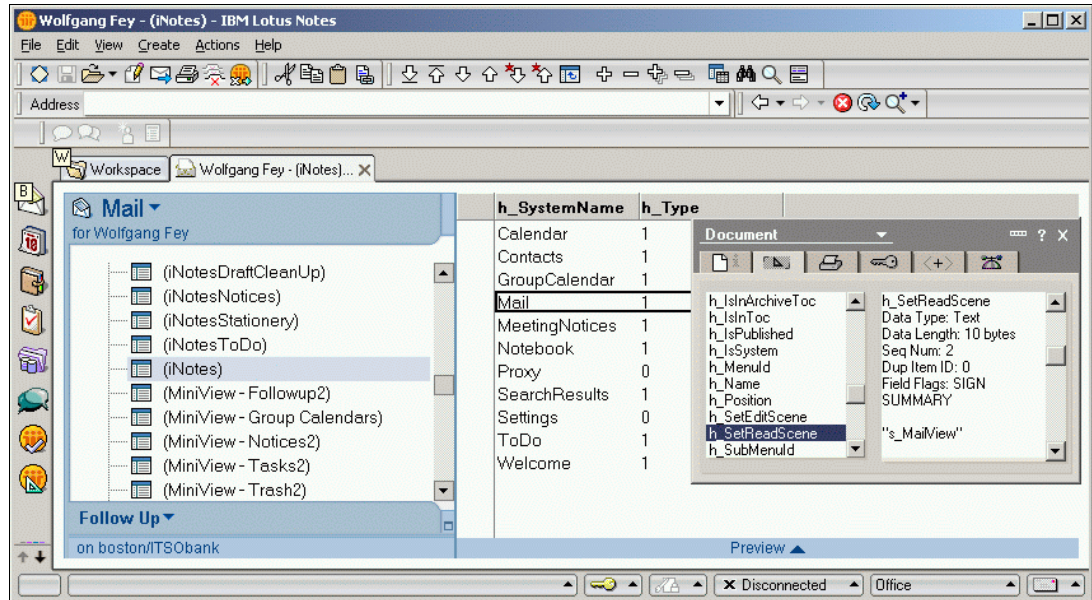


Figure 3-5 Proxy document

3.4 Domino Web Access Web pages

There are two types of Web pages in Domino Web Access: main pages and other types of pages. The following subsections outline samples of these page types and also some differences.

3.4.1 Main pages

All main pages in Domino Web Access use the `h_PageUI` form for page construction by default. The following paragraph lists examples of main pages:

- ▶ Mail view page
- ▶ New Message page
- ▶ Read Contact Object page
- ▶ Edit Calendar Entry page
- ▶ Preferences page
- ▶ Out of Office page

Examples of main pages in Domino Web Access are:

- ▶ Memo
- ▶ To Do
- ▶ Appointment
- ▶ Contact

3.4.2 Other pages

Unlike main pages, the *other* type of page does not use `h_PageUI` form. It uses the other form by utilizing the “&Form=” argument to override the form specified for the document being manipulated. Dialog pages are examples of other pages. The following lists some examples of dialog box in Domino Web Access:

- ▶ Delivery options

► About Domino Web Access

In order to clarify the differences between the main and other pages, let us examine the URL of the New Memo main page:

```
http://smartul.cam.itso.ibm.com/mail/slukito.nsf/($Drafts)/$new/?EditDocument&Form=h_PageUI&PresetFields=h_EditAction;h_New,s_NotesForm;Memo
```

And the URL for the delivery options dialog box:

```
http://smartul.cam.itso.ibm.com/mail/slukito.nsf/iNotes/Proxy/?OpenDocument&Form=s_DeliveryOptions
```

As you can see from the example above, the URL for the main page uses the `&Form=h_PageUI` argument versus the URL for the dialog box, which uses the `&Form=s_DeliveryOptions` argument to override the form.

Only the `h_PageUI` form utilizes the `<QuickplaceSkin>` tag. In other words, the Domino Web Access dialog pages do not use this tag, and hence do not leverage skins or scenes. Rather, they reference the appropriate skin group stylesheet and define the dialog page layout entirely within the form.

3.5 Domino Web Access scenes and subscenes

In 3.4, “Domino Web Access Web pages” on page 28, we already mentioned that all pages use the `h_PageUI` form, which uses the `<QuickplaceSkin>` tag to bring in a skin. The skin has a very important `PageContent` skin component into which plays a *scene*. Therefore scenes are really what give each Domino Web Access page its unique visible content.

A scene consists of two subforms — one with the scene name followed by “_Init” to represent content to be emitted in the HEAD area of a page and the other with the actual scene name, which is emitted in the body of the page. Refer to Figure 3-6 on page 30 for a visualization of the subforms belonging to a scene.

Example 3-3 Scene subforms

```
<SceneName>_Init - Content within the <head> section of the page  
<SceneName> - Content within the <body> section of the page
```

Scenes are relevant only for skins that employ the `PageContent_SkinComponent` and forms that employ the `<QuickplaceSkin>` tag.

3.5.1 Standard read and edit scenes and subscenes

Some key computed variables determine which scene is played, as shown in Example 3-4.

Example 3-4 Scene selection based on variable

```
h_SetReadScene: used when ?OpenDocument URL is in effect  
h_SetEditScene: used when ?EditDocument URL is in effect
```

Domino Web Access utilizes a common read scene and edit scene for almost all read object pages and edit object pages, respectively. This allows a mechanism to share code and markup common to several objects. Hence, most Domino Web Access object pages will have `h_SetReadScene` set to `s_StdPageRead` and `h_SetEditScene` set to `s_StdPageEdit`. Nothing prevents the same scene to be specified for both the `h_SetReadScene` and

h_SetEditScene. For example, the Domino Web Access preferences page is set up in this manner.

If all the Domino Web Access object pages used the same standard read and edit scene, how does Domino Web Access get the New Calendar Entry page to look different from the New Message page? Domino Web Access's standard read and edit scenes implement logic to load additional subforms that contain the actual object-specific layout details. This collection of subforms is known as *subscenes*.

Like a scene, a subscene also consists of a subform to be used within the head section of the page and a subform to be used within the body section of the page. In addition, there is a subform that defines the *schema* for the object. Each object typically has both a read subscene and an edit subscene, and the subscene dictionary is used for both. For a standard read object page, here is a summary of the relevant subforms that comprise the subscene:

- ▶ <subscene prefix>+Read_Init: <head> portion
- ▶ <subscene prefix>+Dictionary: defines "schema" for object
- ▶ <subscene prefix>+Read: <body> portion

For a standard edit object page, the relevant subscene subforms are:

- ▶ <subscene prefix>+Edit_Init: <head> portion
- ▶ <subscene prefix>+Dictionary: defines the "schema" for object
- ▶ <subscene prefix>+Edit: <body> portion

For the ITSO Bank scenario, implementing a whole new main page is demonstrated in 5.5.5, "Special subforms for page creation" on page 80. We show how the different subforms are created and contribute to the page content.

Figure 3-6 shows the structure of the html document regarding the insertion of the scene subforms.

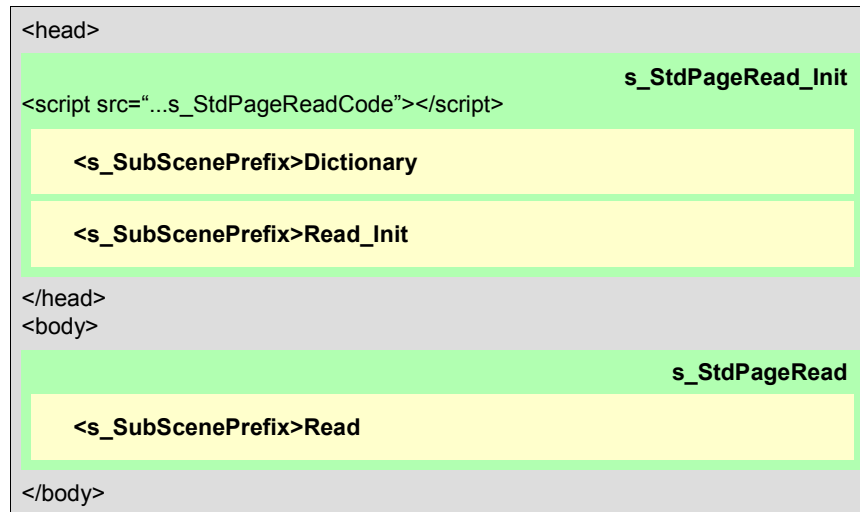


Figure 3-6 Visual display of scenes and subscenes

3.6 Domino Web Access skins

A skin, also referred to as a *skin type*, defines the layout, HTML, and CSS for a Domino Web Access Web page. This is done through a set of files stored as file resources in the forms database. These files are grouped together by the browser they are made for.

3.6.1 Skin group

A skin group defines a set of skins sharing the same stylesheet. There are two primary skin groups in the shared forms database for Domino Web Access:

- ▶ h_ShimmerSkin - used for Internet Explorer
- ▶ h_ShimmerSkin_Gecko - used for Mozilla and Firefox

Contents of both skin groups are identical except for stylesheet.

There are two less significant skin groups:

- ▶ h_ShimmerSkin_UB - for unsupported browsers, displays a special form to direct users to WebMail instead of using the Domino Web Access frontend
- ▶ h_ShimmerSkin_ACC - for accessibility, currently not used

3.6.2 Skin type (skin)

Skin types are members of a specific skin group. Any skin type can only be a member of one skin group. If it is needed in two skin groups there have to be two skin types. Every skin type defines a specific layout in HTML markup to emit within the page body to govern the overall page layout. For example, the h_ListFolder skin type defines the layout for any view or folder that is displayed through Domino Web Access.

The primary Domino Web Access skin types are:

- ▶ h_ListFolder
- ▶ h_MailPage
- ▶ h_Edit
- ▶ h_Page
- ▶ h_Portal

For Domino 7.0.0 and 7.0.1, the Forms7.nsf file contained two skin types that are not used:

- ▶ h_DwaLite
- ▶ h_PortalLite

They were introduced in Version 6.5.5 for the display of the lite skin, which minimizes the use of images to gain better performance. For Domino 7, the spirit of these changes was incorporated into the main h_ListFolder and h_Portal skins, and hence these were no longer utilized, and have been removed in Version 7.0.2.

There is also one special skin type: h_Stylesheet (CSS file). It contains the cascading stylesheet called CSS definitions for the skin group. This one file is the most important to modify if corporate design in terms of colors and fonts have to be brought into Domino Web Access.

3.6.3 Which skin is used

Special items in the JavaScript influence the skin group and skin type to use on a specific page:

- ▶ h_SetSkinGroup NOTESVAR sets the skin group to use, defined in the s_CommonVars subform.
- ▶ h_SkinTypeOverride NOTESVAR provides a means of specifying the specific skin type.

For example, if a mail page is displayed for reading mail, the variable `h_SkinTypeOverride` is set to `'h_MailPage'`. To get the specific value of a Domino Web Access page just look into the page source and search for the variables mentioned.

If the `h_SkinTypeOverride` variable is not set, `h_Type` `NOTESVAR` determines the skin by the simplified logic shown in Example 3-5.

Example 3-5 Simplified synopsis of skin type selection logic (if `h_SkinTypeOverride` is not explicitly specified)

```
If page is opened with ?EditDocument in the URL use h_Edit skin  
Else If h_Type=="1" (QP Type of Folder) use the h_ListFolder skin  
Otherwise use the h_Page skin (last fallback strategy)
```

Special items identify skin group and skin type being used for further usage in JavaScript on the page if needed. For example, if it is needed to know whether a document is in edit mode, or if a mailpage or notebookpage is currently displayed, these variables can be used:

- ▶ `h_CurrentSkinName` var identifies skin group
- ▶ `h_CurrentSkinType` var identifies skin type

To get these variable contents from an actual page, the page's html source can be displayed by the browser. This is shown in 4.1, "Skin customization overview" on page 44.

3.6.4 Skin components

As discussed previously, skin components are subforms that implement a specific visible area of a page outside the main essential data content of a view or form. Skin components provide a mechanism for centralizing the html or javascript to realize this visible area and reuse the component within different skins.

Unlike a plain subform, the skin component has a mechanism to specify arguments when the component is defined within a skin, and the skin component can then render itself differently based on the argument. For instance, the `PanelToggle` skin component is set up to be initially expanded when used within the `h_ListFolder` skin, but collapsed when used within the `h_Portal` skin.

Their purpose is to externalize code or page fragments to specific subforms and use them as an included block in the html page design. Skin components are used in the skin type (as shown in 3.6.2, "Skin type (skin)" on page 31) to put specific functional areas on the page. Examples are:

- ▶ TOC
- ▶ Outline
- ▶ Actionbar
- ▶ Tools
- ▶ Online awareness

Tip: It is a best practice to introduce new skin components when implementing new visual areas for the overall page. The ITSO Bank example that shows the space left in the current mail file in megabytes until the quota is reached is implemented as a skin component. Refer to 4.5, "Adding a new skin component" on page 60, for the steps to implement a skin component.

3.7 Forms map table

To map the fields of a backend Notes form to a Domino Web Access form definition, a mapping document is needed in forms7.nsf. This forms mapping document needs to be defined with a form called FormsMap in the forms7.nsf, which is not included in the product. Figure 5-6 on page 79 shows the definition for the fields needed for this form.

Formsmap is actually a Notes view in the forms database. This view, as shown in Figure 3-7, houses all the formsmap documents that define the read and edit scenes utilized for a particular Notes backend form type.

s_NotesForm	s_SubFormPrefix	h_SetReadScene	h_SetEditScene
Account	s_Account	s_StdPageRead	s_StdPageEdit
Appointment	s_Appointment	s_StdPageRead	s_StdPageEdit
CleanSheet	s_NotebookPage	s_StdPageRead	s_StdPageEdit
Delivery Report	s_MailDeliveryReport	s_StdPageRead	s_StdPageEdit
DOLSOOfflineConfiguration		s_DOLSOOfflineConf	s_DOLSOOfflineConf
fa_RcvdVoiceMail	s_MailMemo	s_StdPageRead	s_StdPageEdit
fa_VoiceMail	s_MailMemo	s_StdPageRead	s_StdPageEdit
Group	s_Group	s_StdPageRead	s_StdPageEdit
h_PageUI			
h_PrintUI			
JournalEntry	s_NotebookPage	s_StdPageRead	s_StdPageEdit
Mailrule	s_Rule	s_StdPageEdit	s_StdPageEdit
Memo	s_MailMemo	s_StdPageRead	s_StdPageEdit
NonDelivery Report	s_NonDelivery	s_StdPageRead	s_StdPageEdit
Notice	s_MeetingNotice	s_StdPageRead	s_StdPageEdit
OutOfOfficeProfile	s_OutOfOffice	s_StdPageEdit	s_StdPageEdit
Person	s_Contact	s_StdPageRead	s_StdPageEdit
Personal Stationery	s_MailMemo	s_StdPageRead	s_StdPageEdit
Phone Message	s_MailPhoneMessage	s_StdPageRead	s_StdPageEdit
QR_BlockSender	s_BlockSender	s_StdPageEdit	s_StdPageEdit
QR_ListEdit	s_QRList	s_StdPageEdit	s_StdPageEdit
QuickMailrule	s_QuickRule	s_StdPageEdit	s_StdPageEdit
Reply	s_MailMemo	s_StdPageRead	s_StdPageEdit

Figure 3-7 Forms map view in Notes client

If the scene in the formsmap document is set to s_StdPageRead or s_StdPageEdit, then the value of the s_SubFormPrefix column defines the subscene prefix for the backend form.

For more details and how to create or modify formsmap documents refer to 5.5.3, “Add the FormsMap form to forms7.nsf” on page 79.

3.8 Domino Web Access special Web page elements

This section outlines several additional special design elements Domino Web Access uses for Web page construction.

3.8.1 Special server-side tags

Domino Web Access introduced new special server-side html tags to implement several special functions within the application. The following sections outline the different kinds of special tags and functions and how they are used.

Insert content from other design elements

There are three new special tags that are recognized by the Domino http task to insert additional content in the actual page:

- ▶ <InsertNotesSubForm Name=SubFormName>
- ▶ <QuickPlaceSkin>
- ▶ <QuickPlaceSkinComponent Name=xxx>

First <InsertNotesSubForm Name=SubFormName> is used to insert the specified subform content as passthrough html (or JavaScript) in the place where the special tag resides. If the inserted code contains another insertion tag, this is also parsed and expanded by the server.

The name of the subform may be a computed value. To get the name computed use a formula like @{...}. The formulas are explained in 3.8.4, “Special Domino Web Access formulas” on page 37. Conditional insertions are also possible by having the Name attribute value be evaluated by a formula. @If may be used to consider other criteria and conditionally include one subform or another. If the desire is in some cases to not insert anything, an empty name value will insert nothing.

<QuickPlaceSkin> inserts the contents of the skin type for the current skin group. See 3.5, “Domino Web Access scenes and subscenes” on page 29, for an additional explanation.

<QuickPlaceSkinComponent Name=xxx> inserts the contents of a subform named xxx_SkinComponent. Because skin components are also subforms, one could think that these are redundant to <InsertNotesSubForm>, but <QuickplaceSkinComponent> has optional Format and Argument attributes used by some skin components.

Argument translates to javascript arg, which is accessible by the skin component. The argument can be used to display different content from the same skin component in different skins. For instance, providing “0” as an argument for the Logout skin component, as shown in Example 3-6, results in nothing being displayed if the user is logged in. However, if the user is not logged in, the Login link is displayed. If you wish to remove the Logout option from Domino Web Access, we encourage you to use this argument rather than removing the skin component altogether (because the skin component doubles as the login option for a mail file opened with anonymous access).

Format supports the special <Item> tag to allow means of excluding some markup if the tag is determined at runtime to be not displayed. Real tag contents are emitted there. Example 3-6 shows how this special tag is used in skins with all optional parameters.

Example 3-6 Format and argument parameters used by <QuickPlaceSkinComponent>

```
<QuickPlaceSkinComponent Name="Logout" Argument="0"
Format={<td class="s-toptoolbar-text s-toptoolbar-bg-middle" nowrap><Item></td>}>
```

Persisted and computed for display items

As outlined in 3.5.1, “Standard read and edit scenes and subscenes” on page 29, and shown in Figure 3-6 on page 30, there are special tags to define fields and items for the mapping between the JavaScript and HTML frontend in Domino Web Access and the Notes backend document. This mapping is maintained through the actual scene’s <Subformprefix>_Dictionary subform. Other forms and subforms may also include the special tags for individual field mappings.

NotesDictionary Tag

<NotesDictionary>...</NotesDictionary> is the pair of tags enclosing all fieldmappings in the dictionary. The enclosed block may contain NotesVar and NotesField tags.

NotesField Tag

<NotesField Name=ItemName> defines a persisted field. The field may have up to four optional parameters:

- ▶ Type=Text|TextList|RichText|Mime|Number|NumberList| Time|TimeList defines the fieldtype.
- ▶ InitialValue=... evaluated if item does not exist within document.
- ▶ Value=... evaluated as page is generated and as page is saved.
- ▶ Includeif="@formula" is an optional attribute for conditional inclusion (new in Forms7.nsf).

For example, <NotesField Name=Subject>.

NotesVar Tag

<NotesVar Name=ItemName> defines a computed-for-display-only item. This tag has the same element attributes as the Notes field. See Example 5-4 on page 74 for a sample usage of NotesVar.

3.8.2 Extensible tags

As part of the Domino Web Access performance and scalability improvements in Domino 7, an additional page generation enhancement supports the specification of namespace qualified tags. This allows for replacing stand-alone invocations of special Haiku @DbCommands such as the h_RunAgent formula. This results in the compiled version of:

```
@{@DbCommand("Haiku"; "h_RunAgent"; AgentName; "1")}
```

Being replaced with the following:

```
<dwa:runagent agentname="AgentName" commonagent="1" />
```

The first is what is found in Forms6.nsf, and the second is what is found in Forms7. The first occurrence of a dwa: tag results in the Web server looking for and processing a special dwa_tags form. This form contains a special XML syntax that defines the various dwa: tags. If you search the dwa_tags form for runagent, you will find the XML stream (some line breaks are added for legibility) shown in Example 3-7.

Example 3-7 XML stream for runagent tag definition

```
<tagdefinition name="runagent">
  <description> Haiku::RunAgent
  @DbCommand("Haiku", "h_RunAgent", "<Agent name>" [, "<CommonAgent? 1:0"]])
  -----The last parameter was changed to a boolean flag since the
  code threw away the arguments anyway. Run a named database agent.
  </description>
  <argument type="fieldname" name="agentname" />
  <argument type="text" optional="yes" name="commonagent" />
  <bodyfunction namespace="haiku" builtin="h_RunAgent" />
</tagdefinition>
```

This allows for the replacement of various compute blocks that were calling the special Haiku @DbCommands with special dwa namespace tags that are easier to read, are self-documenting, and avoid having to initialize and utilize the compute engine, which is a relatively expensive operation.

3.8.3 Special URL arguments

Domino Web Access uses special URL arguments to control environment settings and the mode and context of how a document is displayed.

Default form override

`&Form=FormName` triggers the http server to use the specified form rather than any form value associated with the current document. For example, a developer can use the `&Form=h_Dumpltms` URL argument. This will display a page listing all NotesDictionary variables instead of the normal form. So this parameter can be used to override the form specified for the document being displayed.

Setting default field values

`&PresetFields=` provides a way to override Notes Dictionary items by specifying multiple name/value pairs. Each name and value is separated by a semicolon (;) and each pair is separated by a comma (.). For example, the mail page uses this argument to identify the view or folder to display and what label to give it:

```
filename.nsf/iNotes/Mail/?OpenDocument&PresetFields=
s_ViewLabel;Inbox,s_ViewName;(%24Inbox)&...
```

(The %24 is the URL encoding of the \$ character.) URL arguments can also be used to control the caching of the pages to maximize performance, while insuring that the user is working with the latest data.

This URL argument is useful for passing the values from one page to another. See 5.3.2, “Modify Custom_JS form” on page 71, for a sample usage and more detailed explanation of `&PresetFields`.

Setting Last-Modified

`&CacheResults` (or `&CR` in Version 7 and later) returns the HTTP Last-Modified response header set to the last modified design of forms file.

Set maximum expiration time

`&MaxExpires` (or `&MX` in Version 7 and later) returns the HTTP Expires response header set to nearly one year from today to keep the page contents in the browser cache. Setting the Expires header to a value in the future results in the browser not even asking the server for a more recent version if this response is already in the browser cache. Reusing static content from the cache greatly improves client performance, reducing both bandwidth consumed and server CPU cycles. However, you need to be careful to avoid using stale information. For this reason, Domino Web Access typically adds an `&TimeStamp` to avoid this problem. If the resource changes (for example, a newer forms file is installed on the server), this argument value would be different and the application would no longer use the stale information in the cache.

Avoiding stale code or data in the cache

Domino Web Access adds a `&TimeStamp` (or `&TS` in Version 7 and later) to avoid using stale data. The value for the argument is intelligently calculated based on the resource (for example, static code in the forms file uses the design last modified time of the forms file), so when the resource changes the value changes, and hence the stale code or data in the cache is ignored.

Tip: Making sure that the page returns a proper Last-Modified response header allows for more efficient processing if the user invokes the browser's Refresh action or traverses to the same page again. The browser sends the server an If-Modified-Since GET request in such scenarios, and the server can very efficiently return Not-Modified if nothing more recent is available. A user can hold down the Ctrl key and click the browser's Refresh action to prevent the browser from sending a conditional GET. This is a good debugging technique to make sure a problem is not resulting from stale data in the browser cache.

Skin selection

&ui (equals inotes | inotes_lite | portal | portal_lite | webmail) causes the specified skin or even WebMail to be used for page generation.

For example, if you add &ui=portal to the URL that opens Domino Web Access, it will open Domino Web Access using portal skin instead of the default inotes skin. If you want this change to be permanent, you can set this via an INI value to the desired skin, as described in 6.2.3, "New or modified notes.ini variables in Notes and Domino 7" on page 96.

The default also can be set via the personal options of the Domino Web Access redirect database, as described in 6.3.2, "DWA redirect options" on page 101.

User Name Hash

The parameter &unh=... contains a calculated hash value from the user's full name. This hash value is used to avoid collision of user-specific data pages or script that lands in the cache. For example, Domino Web Access caches the main view pages (as there is no real data in this page, just the layout html; the view data is retrieved separately). However, there are variables that specify what level of access the current user has to this page, and that is used to govern which actions are offered to the current user. If the argument is missing, it is assumed that there is no user-specific data within this page.

This is an option in terms of customization to be used to control self-developed pages or externalized scripts to be loaded from the browsercache or not, if the page contains user specific data.

3.8.4 Special Domino Web Access formulas

In this section we discuss special Domino Web Access formulas.

Formula blocks in HTML and JavaScript

Domino Web Access has a special notation syntax to display and include computed text blocks. These are noted in the @formula language. Not all @ functions work and some are available only to Domino Web Access. All formula blocks have to be included in a block like @{...}.

These formulas appeared Base64 encoded in earlier forms files (Forms5, Forms6) like:

```
#B64#<base64 results of NSFFormulaCompile>
```

The output of the formula can contribute to the page, and therefore it may also run output through special filters:

- ▶ jsdata: JavaScript variable data (apostrophes are escaped)
- ▶ html: valid html data
- ▶ xml: valid xml data (new in Release 7.0.2 and later)

The notation of that would look like @{{...};jsdata;html} and is shown in Example 3-8.

Example 3-8 Sample @formula in Domino Web Access

```
haiku.CalendarProfileOwnerName = '@{@Name([Canonicalize];  
@GetProfileField("calendarprofile";"Owner"))};jsData}';
```

Table 3-3 shows the characters quoted by the xml filter.

Table 3-3 Characters quoted by xml filter

Special character	Quoted filter output
&	&
<	<
>	>
'	'
"	"

The xml filter would be used to properly quote xml element values. Example 3-9 shows how to use the xml filter in a form.

Example 3-9 Form containing XML

```
<?xml version="1.0" encoding="UTF-8" ?>  
  <root>  
    @{"abc&<>'\"";xml}  
  </root>  
<NotesDictionary>  
  <NOTESVAR NAME={$ContentType} VALUE={"text/xml"}>  
  <NOTESVAR NAME={$CacheControl} VALUE={"no-store"}>  
</NotesDictionary>
```

Tip: Consider that server-side generated pages as a result of Urls to the forms file which have &MaxExpires specified are normally cached and therefore the output of the @formulas are cached as well. “Different techniques to include externalized code” on page 106 shows a way to create user-specific output like lookups.

Special @DBCommand formula

Domino Web Access implements a new @DBCommand formula to deliver a great set of special functions. These functions include, for example, calls to get current mail database properties, properties about the current user and mail file, environment settings, profile document access, and so on.

Tip: Sometimes it is very useful to investigate existing forms and subforms to learn about the formulas, the @DBCommand, and the data structures delivered by the calls. Additional information about a found @DbCommand value might be found within the dwa_tags form.

The general notation for @DBCommand is:

```
@DbCommand("Haiku"; "<command_name>" [; "Arg1"; "Arg2"; ...])
```


@DbCommand("haiku";"h_Vars") emits all non-system items in a document as JavaScript vars with values. This command has up to three optional Args:

- ▶ Arg1: UNID of document in current DB being processed (current document if not specified)
- ▶ Arg2: prefix string (string to prepend to each item; useful if you desire to put items as properties of another javascript object) (default "var ")
- ▶ Arg 3:
 - "1": Emit all items on current note (default).
 - "0": emit only if specified in a NotesDictionary block.

@DbCommand("haiku";"h_RunAgent"; "Arg1" [; "Arg2"]) runs the specified agent (which can contribute to page output). This call has one required and one optional parameter:

- ▶ Arg 1: Required - Name of agent
- ▶ Arg 2: Optional
 - "1": Agent is in forms file.
 - "0": Agent is in mail file (default).

@DbCommand("haiku";"h_GetProfileField";"calendarprofile";"Owner") retrieves the value of the field given as the last parameter from the profile with the form name given as the third parameter. The example retrieves the value of the field Owner from the profiledocument calendarprofile.

@DbCommand("Haiku";"h_GetNABName") retrieves the name of the public names and the address book.

Example 3-10 shows two samples using the @DBCommand formula.

Example 3-10 Usage samples for @DBCommand

```
<NotesVar>NAME=s_FF INITIALVALUE={@DbCommand("Haiku";"s_GetFormsFile")}
ExcludeFromJSVars=yes>
<NotesVar>Name=s_UNH
INITIALVALUE={@DbCommand("Haiku";"s_Hash";@Name([Abbreviate];@UserName))}>
```

3.9 Domino Web Access table of contents (TOC)

By default, Lotus Domino Web Access displays six tabs, also called functional areas: Welcome, Mail, Calendar, ToDo, Contacts, and Notebook. However, you may notice that one or more tabs are missing in a mail file or the order of these items is different from the default of Domino Web Access. Even replacing the design of the mail file, this could also not solve the problem.

This has been observed if the mail files had once inherited from a customized Domino Web Access template. If you open the hidden *Haiku_TOC* view, you will see that the design element for the missing tab is not in the list. There might even be design elements for customized tabs. The design elements cannot be deleted from the Notes Client user interface (UI). Attempting to do so will give the error #02:41. Neither can design elements be added.

We suggest that you use an agent to delete those design elements. Example 3-11 shows a sample LotusScript code that will delete all the design elements. Once done, you can re-add them by replacing the design of the mail file with the standard Domino Web Access template.

Example 3-11 Lotus script to remove the TOC documents

```
Dim session As New NotesSession
Dim db As NotesDatabase
Set db = session.CurrentDatabase
Dim view As NotesView
Dim note As NotesDocument
Dim note1 As NotesDocument

Set view = db.GetView( "(Haiku_TOC)" )
Set note = view.GetFirstDocument
Set note1 = view.GetNextDocument(note)

While Not (note Is Nothing)
    Call note.Remove(True)
    If Not (note1 Is Nothing) Then
        Set note = note1
        Set note1 = view.GetNextDocument(note)
    End If
Wend
```

The customization of the TOC in terms of hiding functional areas from it should not be necessary with Domino Web Access 7, because there is a new notes.ini variable introduced to control the display of the areas displayed in the Domino Web Access browser windows. For details refer to “Enable and disable functional areas” on page 18 and “iNotes_WA_Areas” on page 96.

The order of the tabs in the TOC is taken from the order of the proxy documents in the TOC view in the user’s mail file, normally derived from the dwa7 template. So if the order has been modified by changing the proxy documents this has to be redone in a Domino Web Access 6 to 7 migration. The description of how to change the order of the TOC tabs in DWA has been described in Chapter 6 of the IBM Redbook *iNotes Web Access Deployment and Administration*, SG24-6518.

3.10 How to edit Domino Web Access Markup

With Version 7 of Domino Web Access the only tool really needed for modification and customization is Domino Designer:

- ▶ Forms and subforms: Domino Designer
- ▶ Images: Domino Designer within shared resources\files
- ▶ Skin elements: Domino Designer within shared resources\files

To start with skin modifications dive into file resources prefixed with h_ShimmerSkin, but use the one skin group assigned to the targeted browser. Use the javascript variables h_CurrentSkinName and h_CurrentSkinType discussed in 4.2, “Modifying the stylesheet” on page 45, to determine which skin group to edit.

3.11 Taking customized Domino Web Access offline

The ability to work while disconnected from the network increases the productivity of ITSO Bank's mobile employees. Domino Web Access uses Domino Offline Services (DOLS) to take the mail file offline and gives the disconnected user full fidelity access to Web applications, like their mail files, via a Web browser.

The Domino Web Access mail template is already DOLS-enabled, so no configuration needs to be done in order to take the mail file offline. To decrease the performance overhead, the out-of-box forms7.nsf database is included in the filesets that DOLS downloads to the local client. So if the forms7.nsf database is being customized there are considerations about putting these modifications into the fileset. The following steps can be used to replace the standard Form7.nsf with the customized Forms7.nsf from the pre-packaged fileset n_SHIMMER7_en.exe. This example is only for the Windows® English DOLS fileset but can be applied to the LINUX English fileset (l_SHIMMER7_en, l_SHIMMER7_en.inf) and all supported server languages (n_SHIMMER7_fr.exe, n_SHIMMER7_fr.inf).

Important: Backup n_SHIMMER7_en.exe and n_SHIMMER7_en.inf before making any changes.

The steps are:

1. n_SHIMMER7_en.exe is located under Domino\data\domino\html\download\filesets.
2. n_SHIMMER7_en.exe is a self-extracting executable, so we can unzip it using any zip application.
3. Replace the standard Forms7.nsf with the customized Form7.nsf underneath the \data\iNotes folder of the unzipped contents.
4. Maintaining the folder structure, create a new zip file. On Windows XP, one can do this by selecting **efnl27w.dll** and the data folder, followed by the right-clicking **Send to** → **compressed folder**.
5. Convert the above zip file into a self exacting executable and name it n_SHIMMER7_en.exe.
6. Replace the old n_SHIMMER7_en.exe with the new executable.
7. n_SHIMMER7_en.inf is the versioning file. For existing installs, to push out the new customized fileset, increment the Version string (for example, the out-of-box Version = 7.0.1.0 so set Version = 7.0.1.100).

Important: If there is a Domino Web Access hotfix applied to the server, the Version string might be higher. So be sure to increment the current string by one. That is, 7.0.1.100 becomes 7.0.1.200, and so on.



Skin customization techniques

ITSO Bank desires their mail user interface to conform to their corporate standard design. This allows a smooth visual transition as users navigate between Domino Web Access and other internal applications. Customizations implemented to achieve this look and feel are:

- ▶ Modifying stylesheet
- ▶ Modifying h_ShimmerSkin-h_ListFolder
- ▶ Modifying forms and subforms
- ▶ Adding a new skin component
- ▶ Customizing the login screen

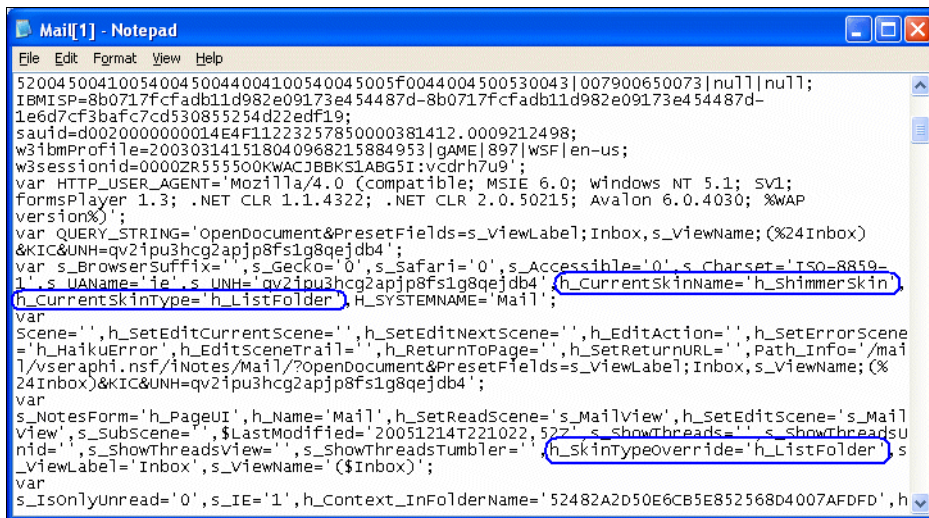
4.1 Skin customization overview

As discussed in Chapter 3, “Customization considerations” on page 17, most of the design (HTML, JavaScript, images, and CSS) that contributes to the look and feel of Domino Web Access resides within the Domino Web Access forms file.

Domino Web Access uses Lotus Quickplace themes technology. This entails having *skin groups* that represent different *themes*. Each theme is comprised of a set of skins that represents specific layout templates along with a shared stylesheet. Therefore, to change the overall look and feel of Domino Web Access, these skins would need to be modified. Customizations explored in this chapter deal specifically with `h_ShimmerSkin`, for IE, but can easily be extended for Mozilla/Firefox (`h_ShimmerSkin_Gecko`). Within each skin group the following key skin types are used in the following pages:

- ▶ `h_ListFolder`, `h_Portal`, `h_DwaLite`, `h_PortalLite`: Used for most view pages. `h_ListFolder` is the default. `h_Portal` is used if `&ui=portal` is in effect. `h_DwaLite` and `h_PortalLite` are only used by Forms6 (Version 6.5.5 and later). They are not used in Versions 7.0.0 and 7.01 and are eliminated in Version 7.0.2.
- ▶ `h_MailPage`: Used for most object screens.
- ▶ `h_ApptPage`: Used by CalendarEntry, ToDo, and Out of Office and Preferences screen (has a different class specified for the BodyDiv outer div within the main PageContent area displayed by the PageContent skin component).

To determine which skin to modify examine the `h_CurrentSkinName` and `h_CurrentSkinType` variables to figure out which skin group and skin type is being used. To get to these variables click the **View** → **Source** menu (in Internet Explorer) and see how these variables are highlighted (Figure 4-1).



```
File Edit Format View Help
5200450041005400450044004100540045005f0044004500530043|007900650073|nu11|nu11;
IBMISP=8b0717fcfadbl1d982e09173e454487d-8b0717fcfadbl1d982e09173e454487d-
1e6d7cf3baf7cd530855254d22edf19;
suid=d002000000014E4F11223257850000381412.0009212498;
w3lmprof1le=200303141518040968215884953|GAME|897|WSF|en-us;
w3sessionid=00002R55500KwACJBKSLABG5I:vcdh7u9';
var HTTP_USER_AGENT='Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1; sv1;
FormsPlayer 1.3; .NET CLR 1.1.4322; .NET CLR 2.0.50215; Avalon 6.0.4030; %WAP
version%)';
var QUERY_STRING='openDocument&PresetFields=s_ViewLabel;Inbox,s_ViewName;(%24Inbox)
&KIC&UNH=qv21pu3hcg2apjp8fs1g8qe1db4';
var s_BrowserSuffix='',s_Gecko='0',s_Safari='0',s_Accessible='0',s_Charset='ISO-8859-
1',s_UAName='ie',s_UNH='qv21pu3hcg2apjp8fs1g8qe1db4',h_CurrentSkinName='h_ShimmerSkin',
h_CurrentSkinType='h_ListFolder',H_SYSTEMNAME='Mail';
var
Scene='',h_SetEditCurrentScene='',h_SetEditNextScene='',h_EditAction='',h_SetErrorScene
='h_HaikUError',h_EditSceneTrail='',h_ReturnToPage='',h_SetReturnURL='',Path_Info='/mai
l/vseraphi.nsf/iNotes/Mail/?OpenDocument&PresetFields=s_ViewLabel;Inbox,s_ViewName;(%
24Inbox)&KIC&UNH=qv21pu3hcg2apjp8fs1g8qe1db4';
var
s_NotesForm='h_PageUI',h_Name='Mail',h_SetReadScene='s_MailView',h_SetEditScene='s_Mail
View',s_SubScene='',LastModified='20051214T221022.52Z',s_ShowThreads='s_ShowThreadsU
nid=',s_ShowThreadsView='',s_ShowThreadsTumbler='',h_SkinTypeOverride='h_ListFolder',s
_ViewLabel='Inbox',s_ViewName='($Inbox)';
var
s_IsOnlyUnread='0',s_IE='1',h_Context_InFolderName='52482A2D50E6CB5E852568D4007AFDFD',h
```

Figure 4-1 View → Source

As displayed in Figure 4-1, we can see that the skin group we are dealing with is `h_ShimmerSkin` and the skin type is `h_ListFolder`.

Tip: Most of the color changes can be done by customizing `h_ShimmerSkin-h_Stylesheet` with the exception of some minor color changes that can be done in `h_ShimmerSkin-h_ListFolder` and `s_SessionInfo` form.

4.2 Modifying the stylesheet

How do you know which CSS definitions to change in order to change the appearance of an HTML element? If you examine the HTML that gets generated when you click View → Source from Internet Explorer, you will find that it does not offer enough details to help you figure out what to change.

The best tool to use here is either Web Developer or DOM Inspector for Mozilla Firefox or Developer toolbar for IE. These tools make it much easier to discover the style that applies to the HTML element. These tools are discussed in detail in Appendix A, “Additional customization tips” on page 105.

Important: It is always good development practice to make backups of any database or design elements you intend to modify in a way that will allow you to reverse any changes.

To make modifications to h_ShimmerSkin-h_StyleSheet:

1. From the Domino Designer client, under shared resources → Files, select the file **h_ShimmerSkin-h_StyleSheet** and click the **Open** or **Open with** button to open the specified file in an application of your choice. The open options are available as buttons on top and in the right-click menu.
2. Open the file h_ShimmerSkin-h_StyleSheet using a text editor or a CSS editor.
3. If the application saves the file and closes it, the file resource becomes updated by Designer.
4. Restart the HTTP server.
 - tell http quit
 - dbcache flush
 - load http

4.2.1 Change the background of the top toolbar

The following steps explain how to change the background of the top toolbar in Domino Web Access, as displayed in Figure 4-2.

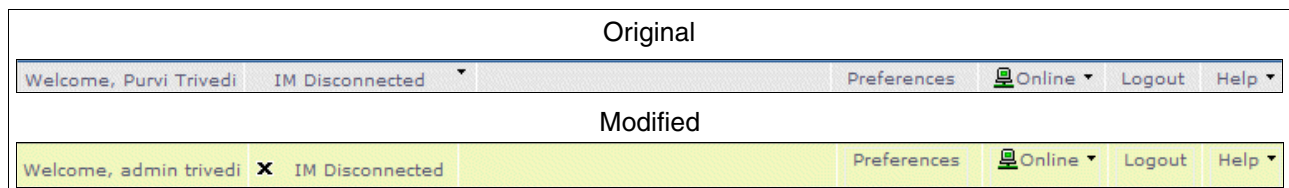


Figure 4-2 Top Toolbar

Modify the style background-color under the .s-toptoolbar-bg, .s-toptoolbar-bg-middle, and .s-toptoolbar-bg-last classes.

- ▶ The .s-toptoolbar-bg-middle class controls the middle of the toolbar, except for X (IM Presence status image area) and help.

The border-right style sets the color of the borders separating each button.

- ▶ The .s-toptoolbar-bg-last class controls the X and Help part of the top toolbar.

Table 4-1 shows the original and modified `.s-toptoolbar-bg`, `.s-toptoolbar-bg-middle`, and `.s-toptoolbar-bg-last` classes.

Table 4-1 .Original and modified functions for top toolbar

Original	Modified
<pre>... .s-toptoolbar-bg { background-color:#e8e8e8; } ...</pre>	<pre>... .s-toptoolbar-bg { background-color:#F0F5BE; } ...</pre>
<pre>... .s-toptoolbar-bg-middle { background-color:#e8e8e8; border-right: 1px solid #ffffff; } ...</pre>	<pre>... .s-toptoolbar-bg-middle { background-color:#F0F5BE; border-right: 1px solid #ffffff; } ...</pre>
<pre>... .s-toptoolbar-bg-last { background-color:#e8e8e8; } ...</pre>	<pre>... .s-toptoolbar-bg-last { background-color:#F0F5BE; } ...</pre>

4.2.2 Change background color of the left panel

To change the background color of the left panel in Domino Web Access, as displayed in Table 4-3 on page 47, modify the style background-color under the `.s-leftpanel` class.

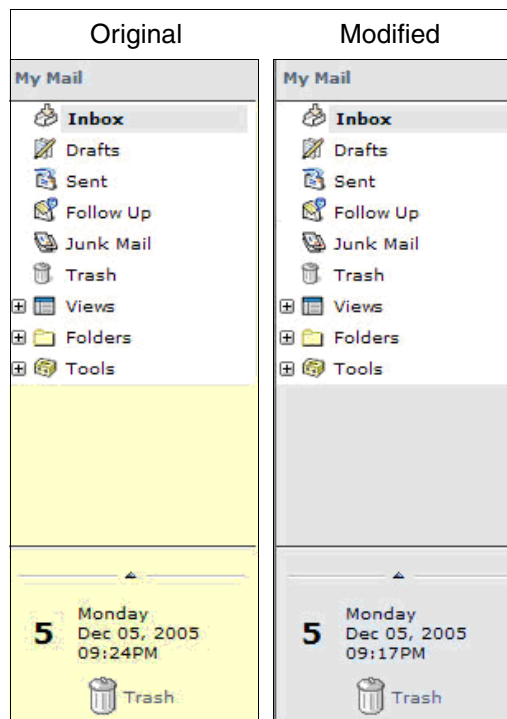


Figure 4-3 Left panel

Table 4-2 shows the original and modified `.s-leftpanel` class.

Table 4-2 `.s-leftpanel`

Original	Modified
<pre>s-leftpanel { background-color: #dee3ef; } ... </pre>	<pre>s-leftpanel { background-color: #F0F5BE; } ... </pre>

4.2.3 Change the background color of the view label

To change the view label in Domino Web Access as displayed in Figure 4-4, modify the style `background-color` under the `.s-view-active` and `.s-margin` classes.

- ▶ The `.s-view-active` class controls the background of the view label.
- ▶ The `.s-margin` class controls the margin color of the view label.

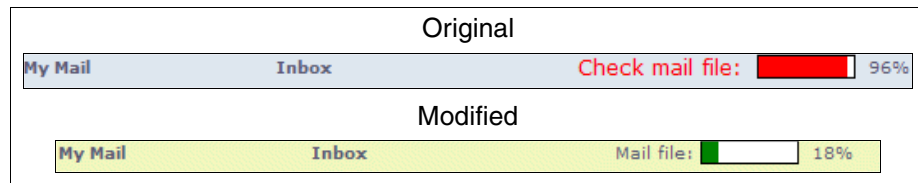


Figure 4-4 View label

Table 4-3 shows the original and modified `.s-view-active` and `.s-margin` classes.

Table 4-3 `.s-view-active` class

Original	Modified
<pre>s-view-active { background-color: #dee3ef; } ... </pre>	<pre>s-view-active { background-color: #F0F5BE; } ... </pre>
<pre>s-margin{ border-top:4px solid #dee3ef; border-bottom:4px solid #dee3ef; height:100%; } ... </pre>	<pre>s-margin{ border-top:4px solid #F0F5BE; border-bottom:4px solid #F0F5BE; height:100%; } ... </pre>

4.2.4 Change the color of the selected item in the outline

To change the color of the selected item in the outline in Domino Web Access as displayed in Figure 4-5, modify the style background-color under the `.s-outlineCellSelected` class.

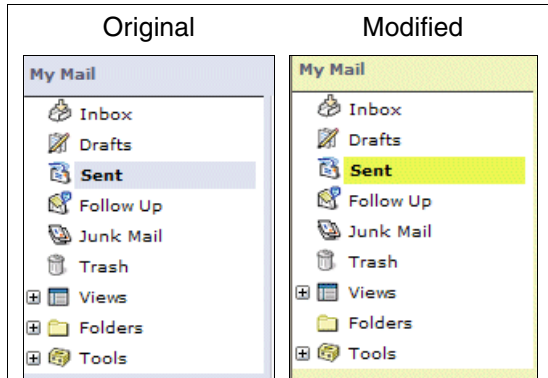


Figure 4-5 Selected item in outline

Table 4-4 shows the original and modified `.s-outlineCellSelected` class.

Table 4-4 `.s-outlineCellSelected`

Original	Modified
<pre>s-outlineCellSelected{ background-color:#dee3ef; border-left:1px solid #ffffff; border-right:5px solid #ffffff; border-top:1px solid #ffffff; border-bottom:1px solid #ffffff; } ... </pre>	<pre>s-outlineCellSelected{ background-color:#F0F5BE; border-left:1px solid #ffffff; border-right:5px solid #ffffff; border-top:1px solid #ffffff; border-bottom:1px solid #ffffff; } ... </pre>

4.2.5 Change the highlight color in the outline

To change the highlight color in the outline in Domino Web Access as displayed in Figure 4-6, modify the style background-color for a element of the `s-outlineEntry: hover` class.

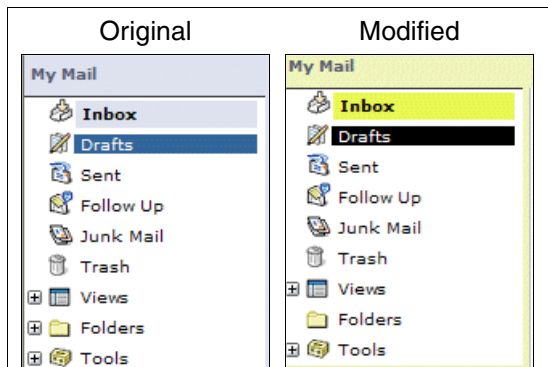


Figure 4-6 Highlight color in the outline

Table 4-5 shows the original and modified `.a.s-outlineEntry: hover` class.

Table 4-5 `.a.s-outlineEntry: hover`

Original	Modified
<pre> ... a.s-outlineEntry: hover{ color:#ffffff; background-color:#336699; text-decoration:none; } ... </pre>	<pre> ... a.s-outlineEntry: hover{ color:#ffffff; background-color:black; text-decoration:none; } ... </pre>

4.2.6 Change the of the Active, Inactive, and highlighted tab

The following steps explain how to change the Active, Inactive, and highlighted tab in Domino Web Access, as displayed in Figure 4-7.

Note: This change will be applied to all tabs: Welcome, Calendar, To Do, Contacts, and Notebook.

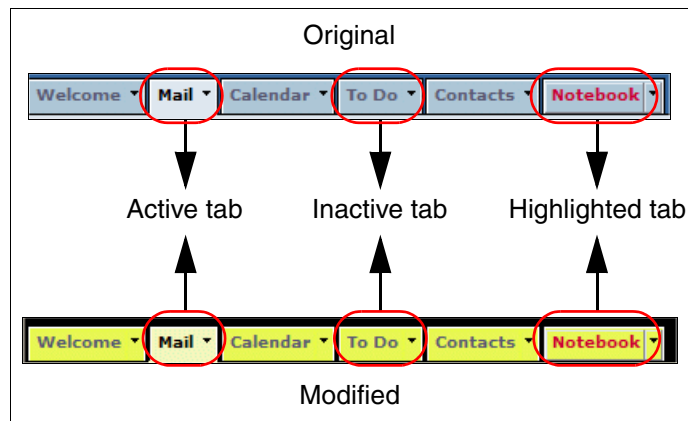


Figure 4-7 Active, Inactive, and highlighted tabs

Modify the following classes and styles:

1. Under the `.s-toc-tab-active` class modify style `background-color` and `border-bottom` for the active tab color and border. There is a highlighted border around the active tab in the same color as the tab. This border color can be modified under the `.dm-toca-hyper` class.
2. Under the `.s-toc-tab-inactive` class modify style `background-color`.
3. Under the `.dm-toci-hyper` class modifies the border style. This changes the border of the tab when the mouse hovers over the tab.

Table 4-6 shows the original and modified `.s-toc-tab-active`, `.s-toc-tab-inactive`, `.dm-toci-hyper`, and `.dm-toca-hyper` classes.

Table 4-6 `.s-toc-tab-active`, `.s-toc-tab-inactive`, and `.dm-toci-hyper`

Original	Modified
<pre>... .s-toc-tab-active{ background-color:#dee3ef; border-bottom:1px solid #dee3ef; } ...</pre>	<pre>... .s-toc-tab-active{ background-color:#F0F5BE; border-bottom:1px solid #F0F5BE; } ...</pre>
<pre>... .s-toc-tab-inactive{ background-color:#adc3d6; border-bottom:1px solid #000000; } ...</pre>	<pre>... .s-toc-tab-inactive{ background-color:#E8F575; border-bottom:1px solid #000000; } ...</pre>
<pre>... .dm-toci-hyper{ border:1px solid #acc3d6; color: #595973; } ...</pre>	<pre>... .dm-toci-hyper{ border:1px solid #E7F845; color: #595973; } ...</pre>
<pre>... .dm-toca-hyper{ border:1px solid #dfe3ef; color: #000000; } ...</pre>	<pre>... .dm-toca-hyper{ border:1px solid #F0F5BE; color: #000000; } ...</pre>

4.2.7 Changing the background color for the table of contents toolbar

The following steps explain how to change background color for the table of contents toolbar in Domino Web Access, as displayed in Figure 4-8.

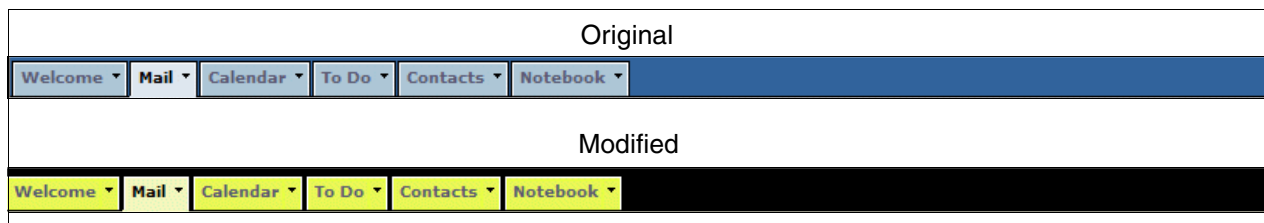


Figure 4-8 Table of contents

Modify the style background-color under the `.s-toc-bar-bg` class.

Table 4-7 shows the original and modified `.s-toc-bar-bg` class.

Table 4-7 `.s-toc-bar-bg`

Original	Modified
<pre>s-toc-bar-bg { background-color:#336699; } ... </pre>	<pre>s-toc-bar-bg { background-color:black; } ... </pre>

4.2.8 Change the background color for the action bar

To change background color for the action bar in Domino Web Access as displayed in Figure 4-9, modify the style background-color under the `.dm-actionbar` class.

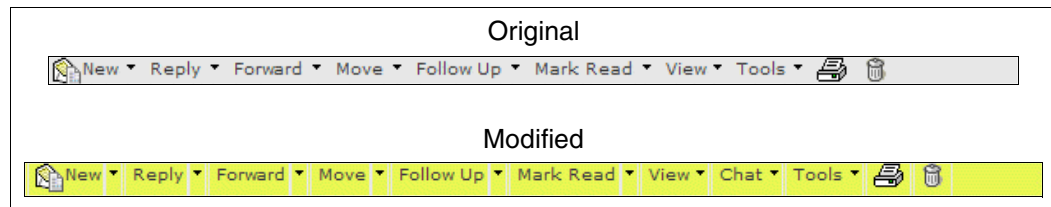


Figure 4-9 Action bar

Table 4-8 shows the original and modified `.dm-actionbar` class.

Table 4-8 `.dm-actionbar`

Original	Modified
<pre>dm-actionbar{ border-top: 1px solid #4A494A; border-bottom: 1px solid #808080; background-color: #E8E8E8; width: 100%; padding-top:1px; padding-left:1px; } ... </pre>	<pre>dm-actionbar{ border-top: 1px solid #4A494A; border-bottom: 1px solid #808080; background-color: #e8f757; width: 100%; padding-top:1px; padding-left:1px; } ... </pre>

4.2.9 Change the background color of the column header

The following steps explain how to change background color of the column header in Domino Web Access, as displayed in Figure 4-10.

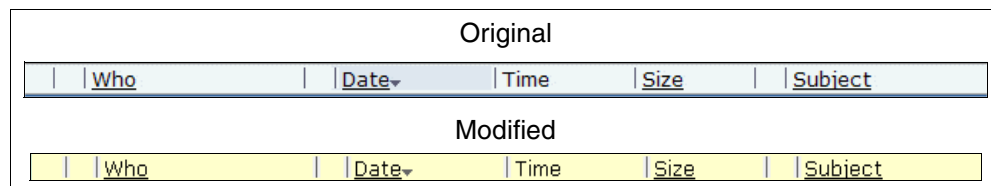


Figure 4-10 Column header

Modify the following classes and styles:

- ▶ Under `.vl-column-header`, `.ll-column-header` class, modify style `background-color`.
- ▶ Under `.vl-coumn-border`, `.ll-column-bar` class, modify style `border-left` and `border-right`.

Table 4-9 shows the original and modified `.vl-column-header`, `.ll-column-header`, `.vl-column-bar`, and `.ll-column-bar` class.

Table 4-9 Column header and border

Original	Modified
<pre>vl-column-header, .ll-column-header{ background-color:#eff3f7; border-bottom:1px solid #999999; font-family:verdana,helvetica,arial,sans-serif; font-size:x-small; height:1.4em; white-space:nowrap; overflow-x:hidden; position:relative; left:0px; top:0px; z-index:0; } ... </pre>	<pre>vl-column-header, .ll-column-header{ background-color:#F3F6D6; border-bottom:1px solid #999999; font-family:verdana,helvetica,arial,sans-serif; font-size:x-small; height:1.4em; white-space:nowrap; overflow-x:hidden; position:relative; left:0px; top:0px; z-index:0; }g ... </pre>
<pre>vl-column-bar, .ll-column-bar{ z-index:1; background-color:#5a5973; border-left:2px solid #eff3f7; border-right:2px solid #eff3f7; height:1em; width:5px; position:absolute; overflow:hidden; cursor:default; } ... </pre>	<pre>vl-column-bar, .ll-column-bar{ z-index:1; background-color:#5a5973; border-left:2px solid #F3F6D6; border-right:2px solid #F3F6D6; height:1em; width:5px; position:absolute; overflow:hidden; cursor:default; } ... </pre>

4.2.10 Change the menu highlight color

To change the menu highlight color in Domino Web Access, as displayed in Figure 4-11, modify the style `background-color` for the highlight color and style `color` for the foreground color under the `.dm-pop-x-hyper-hover` class.

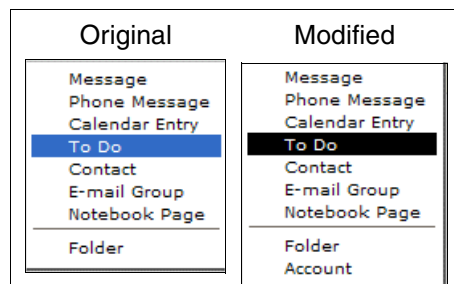


Figure 4-11 Menu highlight color

Table 4-10 shows the original and modified `.dm-pop-x-hyper-hover` class.

Note: The default background-color is set to *highlight* so DWA will use the default operating system color setting.

Table 4-10 `.dm-actionbar`

Original	Modified
<pre>... .dm-pop-x-hyper-hover{ color: highlighttext; background-color: highlight; } ...</pre>	<pre>... .dm-pop-x-hyper-hover{ color: highlighttext; background-color: black; } ...</pre>

4.2.11 Change the background color for the message

To change the background color for the message in Domino Web Access, as displayed in Figure 4-12, modify the style background-color under the `.s-form-mail` class.

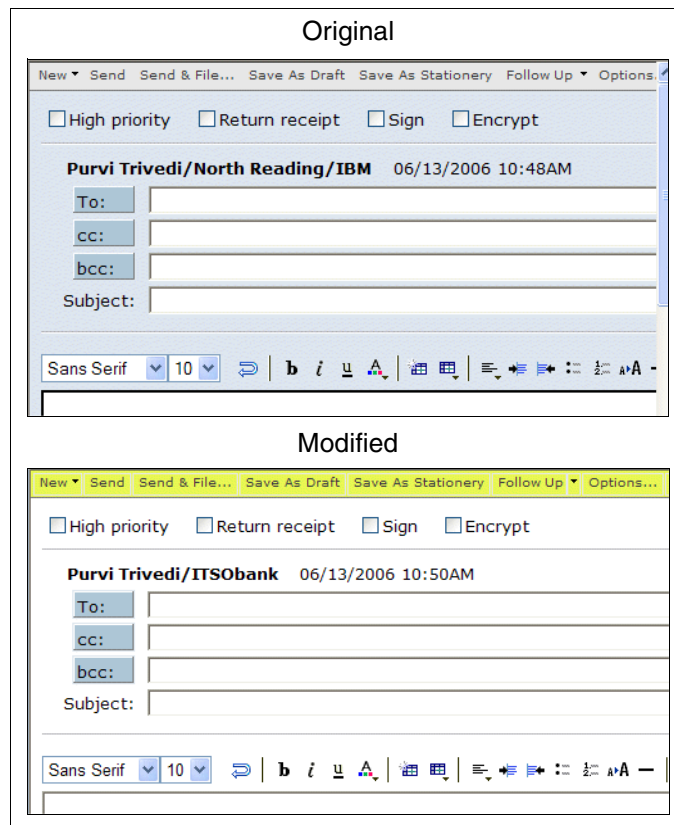


Figure 4-12 New message

Note: This modification will apply to opening, creating, replying, and forwarding the message window.

Table 4-11 shows the original and modified .s-form-mail class.

Tip: The style background-color can be set to a color like #FOF5BE or it can be set to a window that will pick up the color from the user's workstation settings under Control Panel → Display Properties → Appearance → Advanced Appearance → window.

Table 4-11 ..s-form-mail class

Original	Modified
<pre>s-form-mail { padding:6px; background-color: #dee3ef; width: 100%; height:100%; } ... </pre>	<pre>s-form-mail { padding:6px; background-color: window; width: 100%; height:100%; } ... </pre>

4.3 Modifications to h_ShimmerSkin-h_ListFolder

The layout of the DWA Web pages is stored in the skin html files. As we discussed in 4.1, “Skin customization overview” on page 44, the skintype value in the page source points to which layout file needs to be modified. All layout files are stored in the Forms7.nsf as file resources.

To make modifications to h_ShimmerSkin-h_ListFolder:

1. From the Domino Designer client, under shared resources → Files, export h_ShimmerSkin-h_ListFolder. The export option is available as a button on top and in the right-click menu.
2. Open h_ShimmerSkin-h_ListFolder using a text editor.
3. Make the required modifications and save the file at your chosen location.
4. Delete the original h_ShimmerSkin-h_ListFolder from the Forms7.nsf database.
5. Create a new resource and select the modified h_ShimmerSkin-h_ListFolder.
6. Restart the HTTP server.
 - a. tell http quit
 - b. dbcacheflush
 - c. load http

4.3.1 Change the background color around the arrow on the divider

To change the background color around the arrow on the divider in Domino Web Access as displayed in Figure 4-13, modify the bgcolor attribute of the s-leftpanel-divider class.

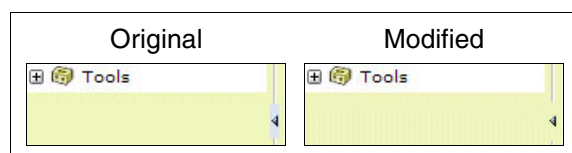


Figure 4-13 Background color around the arrow

Table 4-12 shows the original and modified s-leftpanel-divider class.

Table 4-12 Original and modified s-leftpanel-divider class

Original	Modified
<pre> ... <td class="s-leftpanel-divider" width="9" height="100%" valign="middle" align="center" bgcolor="#dee3ef"> <table height="100%" valign="middle" align="center"> <tr><td height="50%">&nbsp;</td></tr> <tr><td height="10px" valign="middle" align="center" bgcolor="#dee3ef"> ... </pre>	<pre> ... <td class="s-leftpanel-divider" width="9" height="100%" valign="middle" align="center" bgcolor="#F0F5BE"> <table height="100%" valign="middle" align="center"> <tr><td height="50%">&nbsp;</td></tr> <tr><td height="10px" valign="middle" align="center" bgcolor="#F0F5BE"> ... </pre>

4.3.2 Move search next to preferences

The following steps explain how to move the search next to preferences in Domino Web Access, as displayed in Figure 4-14.

1. In order to move a skin component, the search bar, from the original place below the ITSO bank logo to above the tools menu, search the code for the string *quicksearch*. The code of the QuickSearch SkinComponent should be found in the file highlighted in Example 4-1 directly after the usageindicator SkinComponent.

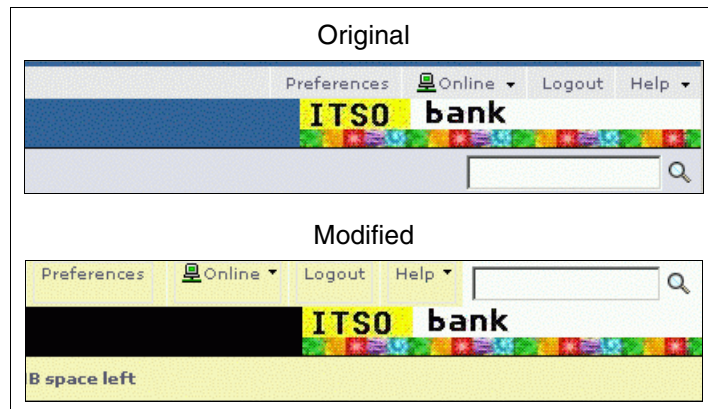


Figure 4-14 Search skin component customization

Example 4-1 h_ShimmerSkin-h_ListFolder HTML original code fragment

```

...
<td class="s-view-title s-view-subtitle" valign="middle" align="right" nowrap>
<InsertNotesSubForm name="UsageIndicator_SkinComponent">
</td>
</tr></table></span></td>
<td id="SearchBin" valign="middle" width=150 nowrap style="padding-top:4px;"
align="left">
<InsertNotesSubForm name="QuickSearch_SkinComponent">
</td>
</tr>
</table>
...

```

2. Cut the whole HTML table cell including the enclosing `<td></td>` tags and insert it directly after the HTML table cell of the Help SkinComponent, as shown in Example 4-2.

Example 4-2 Insertion point after the last table cell of the tools menu

```

...
<td class="s-toptoolbar-text s-toptoolbar-bg-last" nowrap>
<InsertNotesSubForm name="Help_SkinComponent">
</td>

```

Insert QuickSearch SkinComponent here

...

Important: Sometimes it is very important to keep the empty table cell to prevent breaking the table structure. In this example it is not needed to keep the empty cell. Eventually it is required to put a nonbreaking space placeholder character (HTML code ` `;) into the table cell.

3. After that delete the content of the old QuickSearch SkinComponent table cell, but keep the table cell to prevent breaking the table layout.

4.3.3 Remove trash icon

The following steps explain how to remove the trash icon in Domino Web Access, as displayed in Figure 4-15. The motivation of ITSO Bank was to have only one trash icon displayed. But think of the different areas that do not have a trash icon in the outline areas and also that this one removed here is an animated trash bin, showing whether there is something in the trash.

1. Removing the trash icon requires the same steps as moving a SkinComponent. First, find the right SkinComponent by looking for the HTML structure and removing the appropriate code.

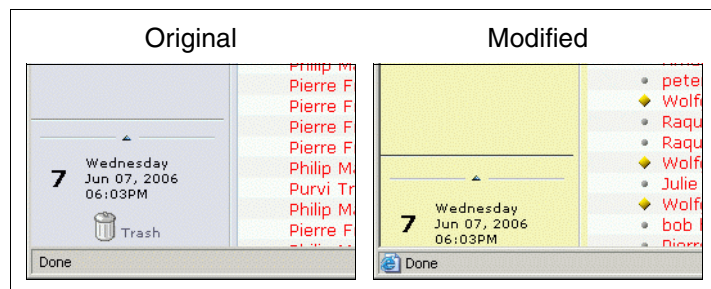


Figure 4-15 DWA page without the Trash bin icon

2. In `h_ShimmerSkin-h_ListFolder` searching for `Trash` should lead to the code shown in Example 4-3.

Example 4-3 Locate trashicon skin component

```

...
<tr>
<td id="TrashBin" valign="top" width=150 nowrap style="padding-top:4px;"
align="center">
<InsertNotesSubForm name="Trash_SkinComponent">

```

```

</td>
</tr>
...

```

3. In this case, leaving an empty HTML cell instead of removing the whole cell is the best decision. The resulting code is shown in Example 4-4.

Example 4-4 *h_ShimmerSkin-h_ListFolder code without Trash SkinComponent*

```

...
<tr>
<td>
</td>
</tr>
...

```

4.4 Modifying forms and subforms

ITSO Bank corporate logo changes are controlled by the Custom_banner subform and AboutBox forms. Also, some look and feel customizations are made in the s_SessionInfo form, as shown in the following sections.

4.4.1 Change banner to company logo

The following steps explain how to change the default banner to the ITSO Bank company logo in Domino Web Access, as displayed in Figure 4-16.

1. Under Shared Resource → Files, create a new file resource for itsso.gif.

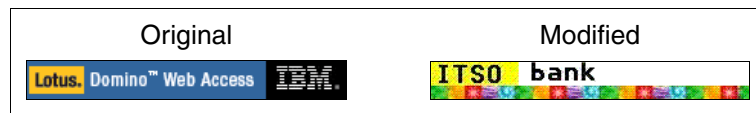


Figure 4-16 *Banner*

2. Under Shared Code → Subforms, modify the Custom_Banner subform to point at the itsso.gif. Change the width and height dimension so the graphic is not stretched or compressed out of proportion. The tooltip can also be adjusted by modifying the title value.

Table 4-13 shows the original and modified Custom_Banner subform.

Table 4-13 *Custom_Banner subform*

Original	Modified
<pre> <script> SV('<div id="ProductLogo"></div> '); </script> </pre>	<pre> <script> SV('<div id="ProductLogo"></div>'); </script> </pre>

4.4.2 Change the About information box to display company logo

The following steps explain how to change the About information box to display the ITS0 Bank logo in Domino Web Access, as displayed in Figure 4-17.

1. From Designer Client, open Forms → AboutBox.

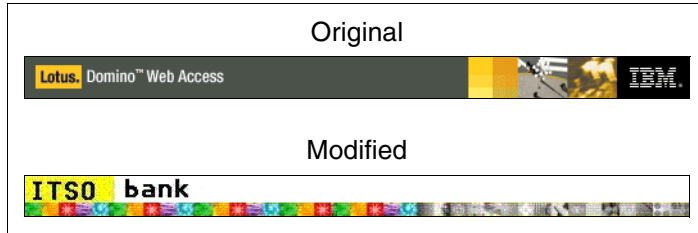


Figure 4-17 About box

2. Search for AboutBanner.gif and replace it with the graphic. In this case ITS0Banner.gif is used.

Note: The size of ITS0Banner.gif is the same as AboutBanner.gif — 640 x 512.

Table 4-14 shows the original and modified AboutBox form.

Table 4-14 Before and after AboutBox form

Original	Modified
<pre>... <tr><td colspan=3 style="padding:0;border-bottom: black 1px solid"></td></tr> ...</pre>	<pre>... <tr><td colspan=3 style="padding:0;border-bottom: black 1px solid"></td></tr> ...</pre>

4.4.3 Modifications to s_SessionInfo form

In the s_SessionInfo form, there is an array of special colors that controls the colors for various design elements.

1. To access this array, open the s_SessionInfo form in Designer client. Once the form is open, search for sTmp2=, as shown in Example 4-5.

Example 4-5 sTmp2 array

```
...
var sTmp2="#dee3ef, #adc3d6, #336699, #ffffff, #e0e0e0, #000000, #ffffff, #e8e8e8,
#eff3f7, #cc3300, tan, darktan, #999999, #f6f3e2, #dee3ef";
...
```

2. Table 4-15 shows the mapping of elements to index.

Table 4-15 Mapping elements to array index

Element	Array index
Active Tab Background	0

Element	Array index
Inactive Tab	1
Highlight Background	2
Highlight Text	3
Drop down manager -- Selected Background	4
Drop down manager -- Selected Text	5
Even row background for list	6
Odd row background for list	7
Column header background	8
Hypertext hover (used by outline)	9
Active GifName	10
Inactive GifName	11
Actionbar border color	12
Sorted contacts column color	13
Sorted column list color	14

3. For ITSO bank requirements, we are modifying the background color highlight for a selected message (index 2) and sorted column color (index 14). Figure 4-18 shows the original and modified page design.

Original					
Who	Date	Time	Size	Subject	
admin trivedi	06/07/2006	09:02AM	1005	test2	
admin trivedi	06/05/2006	01:59PM	1004	test	

Modified					
Who	Date	Time	Size	Subject	
■ Purvi Trivedi	06/07/2006	12:22PM	1099	Skin Customization test	
● Shu Lukito	06/06/2006	09:30AM	5499	IBM IT services	
● Wolfgang Fey	06/01/2006	02:30PM	385	Mail number 929	

Figure 4-18 Background color for highlighted messages and sorted column color

4. Table 4-16 shows the original and modified sTmp2 array.

Table 4-16 sTmp2 array in s_SessionInfo form

Original	Modified
<pre> ... var sTmp2="#dee3ef, #adc3d6, #336699, #ffffff, #e0e0e0, #000000, #ffffff, #e8e8e8, #eff3f7, #cc3300, tan, darktan, #999999, #f6f3e2, #dee3ef"; ... </pre>	<pre> ... var sTmp2="#dee3ef, #adc3d6, #000000, #ffffff, #e0e0e0, #000000, #ffffff, #e8e8e8, #eff3f7, #cc3300, tan, darktan, #999999, #f6f3e2, #f0f5be"; ... </pre>

4.5 Adding a new skin component

The most challenging part of modifying the skin layout is to add new skin components. This requires adding new code. In this section we add a new skin component to display the remaining space in the mail file. This skin component will only display when the mail file quota is implemented.

Restriction: Since we cache the top-level view page, adding the quota calculation to the main page is not ideal. This will be cached and become stale. The DWA quota implementation suffered from same issue and has been overhauled in Version 7.0.2.

For an example this might be OK, but we wanted to make sure that you were aware of this type of problem. A way to make the implementation more robust is to introduce a XML subform that returns the quota information and have the skincomponent retrieve it via a XMLHttpRequest request.

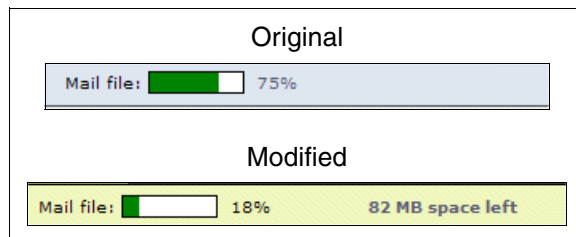


Figure 4-19 New skin component shown in DWA

The steps are:

1. The first step is to implement the JavaScript code to calculate and display the value in an appropriate format. To implement this code as a new SkinComponent, we need to create a new subform named *AvailSize_SkinComponent*. The code needed for the calculation is shown in Example 4-6.

Example 4-6 JavaScript code for *AvailSize_SkinComponent*

```
<NotesDictionary>
<NOTESVAR name=s_DBQuotaSize type=TextList
initialValue={@DbCommand("Haiku";"h_GetDBQuotaSize")}>
</NotesDictionary>
<Script>
var ib_dbqs=s_DBQuotaSize;
var ib_array = ib_dbqs.split(",");
var ib_as = parseInt(ib_array[0]);
var ib_qs = parseInt(ib_array[1]);
var ib_avail = Math.round((ib_qs - ib_as)/1024);
if (ib_avail > -1) {document.write(ib_avail + " MB space left");};
</script>
```

2. Since the subform has been saved to the forms7.nsf database, it can be used in the h_ShimmerSkin-h_ListFolder skin. Example 4-7 shows the h_ShimmerSkin-h_ListFolder HTML code with the inserted table cell for displaying the new skin component. Since the output is being written into the document at the place where the SkinComponent is inserted, there has to be enclosing table cell tags <td></td> around the skin component.

Example 4-7 h_ListFolder skin with new AvailSize_SkinComponent

```
...  
<td class="s-view-title s-view-subtitle" valign="middle" align="right" nowrap>  
<InsertNotesSubForm name="UsageIndicator_SkinComponent">  
</td>  
<td class="s-view-title s-view-subtitle" valign="middle" align="right" nowrap>  
<InsertNotesSubForm name="AvailSize_SkinComponent">  
</td>  
</tr>  
...
```

Attention: Skins are not language-specific, so any implementation should avoid placing user-visible text directly within a skin, as done in this example. Rather, this is placed within skin components or subforms. If customization is for only one language, user-visible text may be placed within the skin.

4.6 Corporate Login screen

The IBM Redbook *Domino Web Access 6.5 on Linux, SG24-7060*, shows how to customize the main logo on the redirection page in the Domino Web Access redirection template. The following sections show how the whole login screen could be customized for representing the corporate design and adding other functionality.

The following examples show how to customize the login page to display:

- ▶ Corporate logo
- ▶ Adding functionality to display weekday, date, and time

4.6.1 Corporate logo

To display the corporate logo it should be imported as an image resource to the iwaredir.ntf template in an appropriate size. In this example we use a custom logo previously used for the About Information box. The size of ITSOBanner.gif is 519x32 pixel.

Figure 4-20 shows the ITSO Bank custom logo and its insertion as an image resource.



Figure 4-20 ITSObank banner

This has to be imported as an image resource, as shown in Figure 4-21.

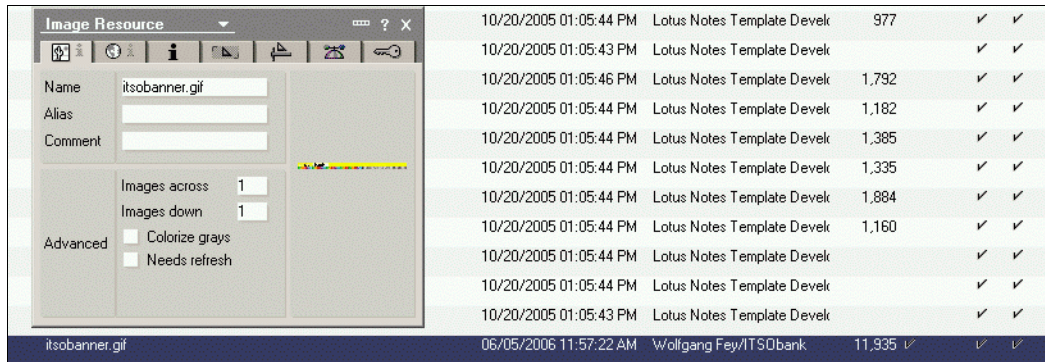


Figure 4-21 Imported itsobanner.gif in Domino Designer

Now the image resource can be used and referenced in the DWALoginPage form, which is used to display the login dialog in the browser, as shown in Figure 4-22.

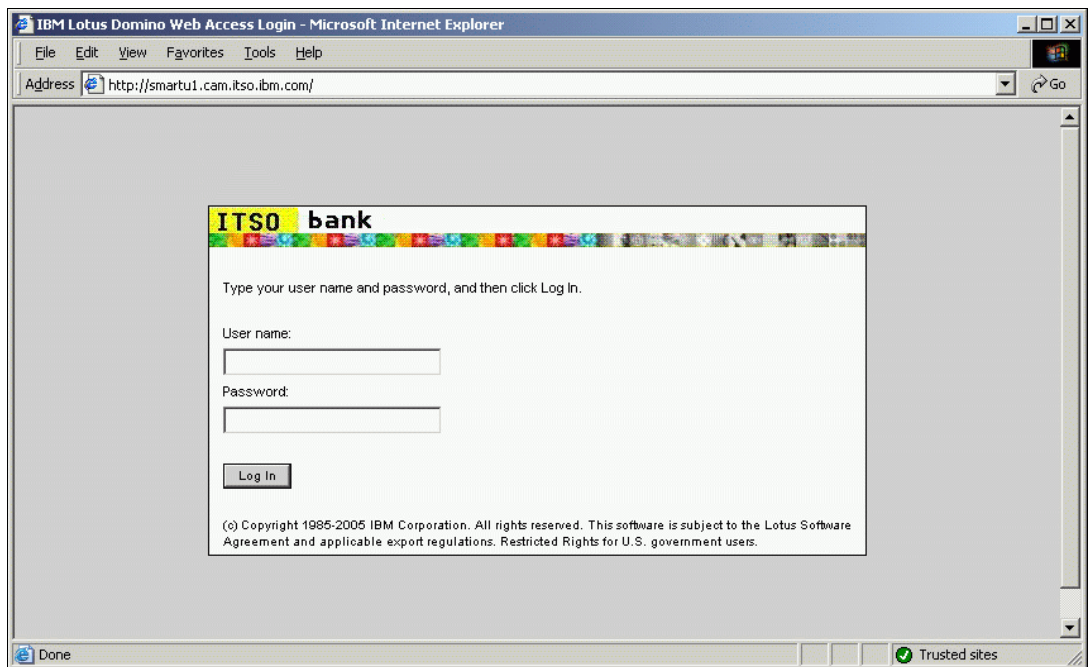


Figure 4-22 Corporate Login screen

Figure 4-23 shows the code extract of the DWALoginForm form in Domino Designer referring to the new image resource.

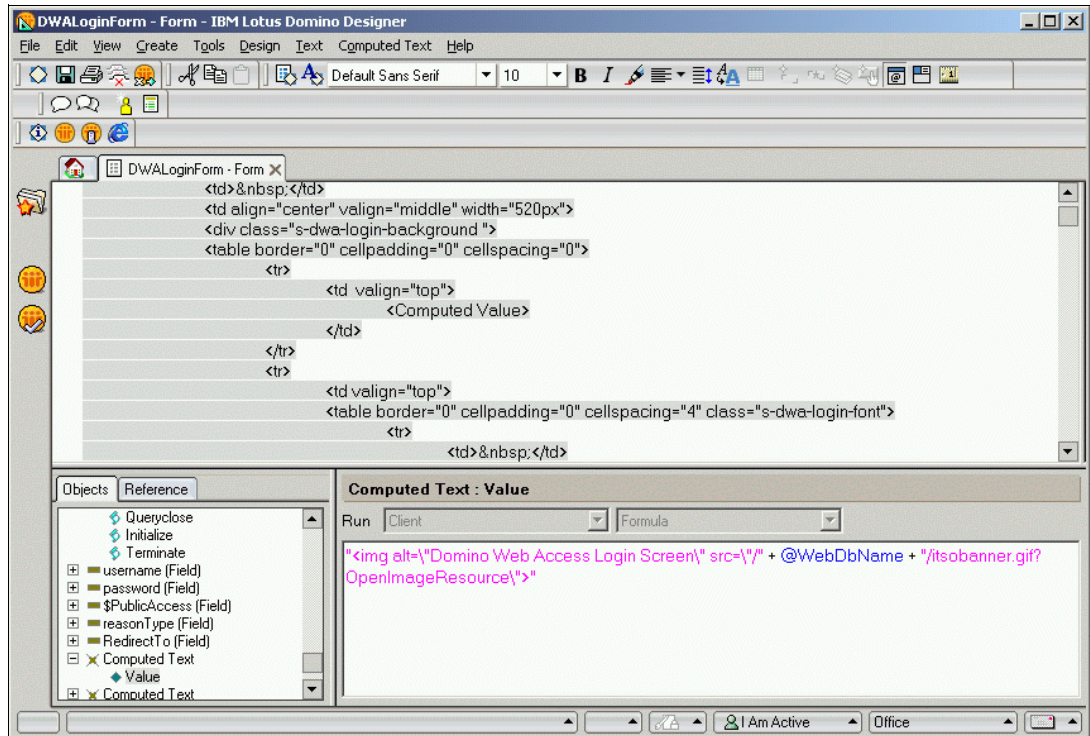


Figure 4-23 Domino Designer showing computed text for the modified image resource

4.6.2 Adding functionality to display weekday, date, and time

In order to add some convenient functionality, like including the actual weekday, date, and time on the login form, a modification to the DWALoginForm form has to be done. First find the paragraph in DWALoginPage that holds the table cell with the corporate logo, as shown in Figure 4-23. Then insert the JavaScript code provided in Example 4-8 as a whole new table row.

Example 4-8 JavaScript code for this example

```

...
<tr>
  <td style="font-weight:bold;"><script language="javascript">
    <!--
      var Dateval = new (Date);
      document.write(Dateval.toLocaleString());
    //-->
  </script>
</td>
</TR>
...

```

After refreshing the http task the login page should look like Figure 4-24.

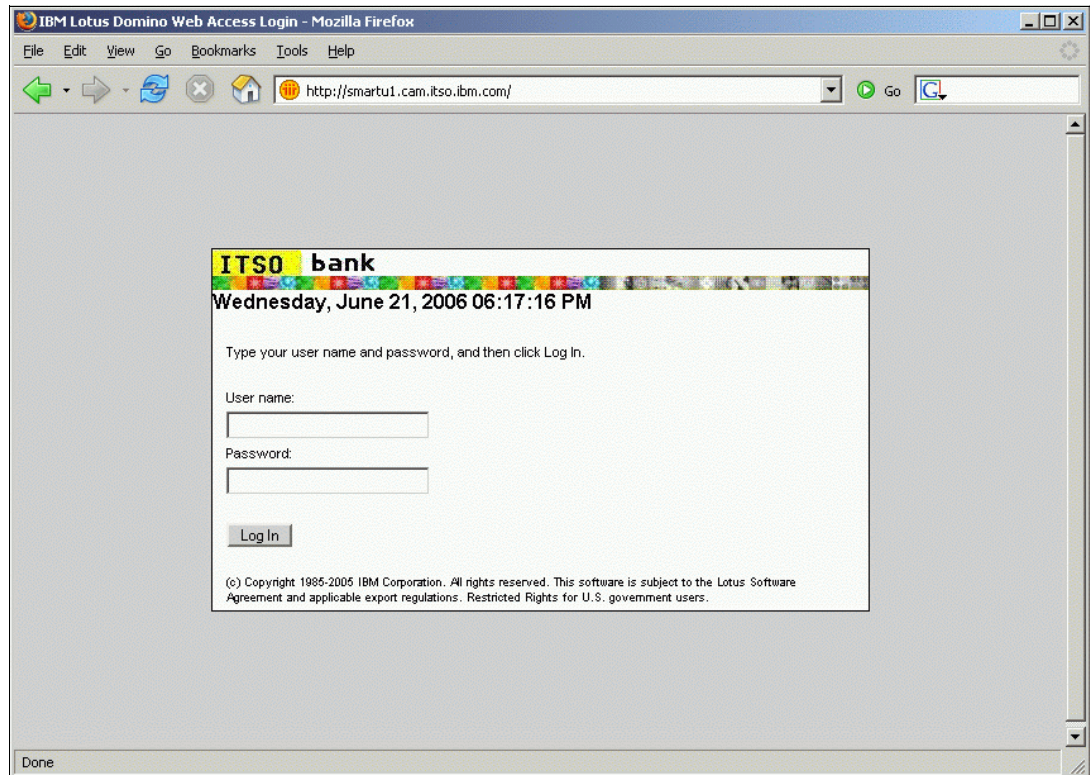


Figure 4-24 DWA login page with date and time shown



Feature customization techniques

This chapter shows how to implement the various changes needed for the ITSO Bank scenario. ITSO Bank is implementing a simplified e-mail management compliance process by storing a copy of all outgoing mail messages in a repository database. In this example the Redpaper team utilizes a Domino database as the repository. However, in the real world the repository choices are virtually limitless — it can be a document management system, relational database, content management system, and so on.

For our example customization, we perform the following:

- ▶ Overwrite an existing action menu.
- ▶ Remove an action menu.
- ▶ Add a new action menu.
- ▶ Add a custom dialog page.
- ▶ Add a custom main page, with the ability to view, edit, and create.
- ▶ Modify preferences.

5.1 Implementation overview

Before getting into the details on how to perform these actions, let us examine an overview of the ITSO Bank customization, as displayed in Figure 5-1.

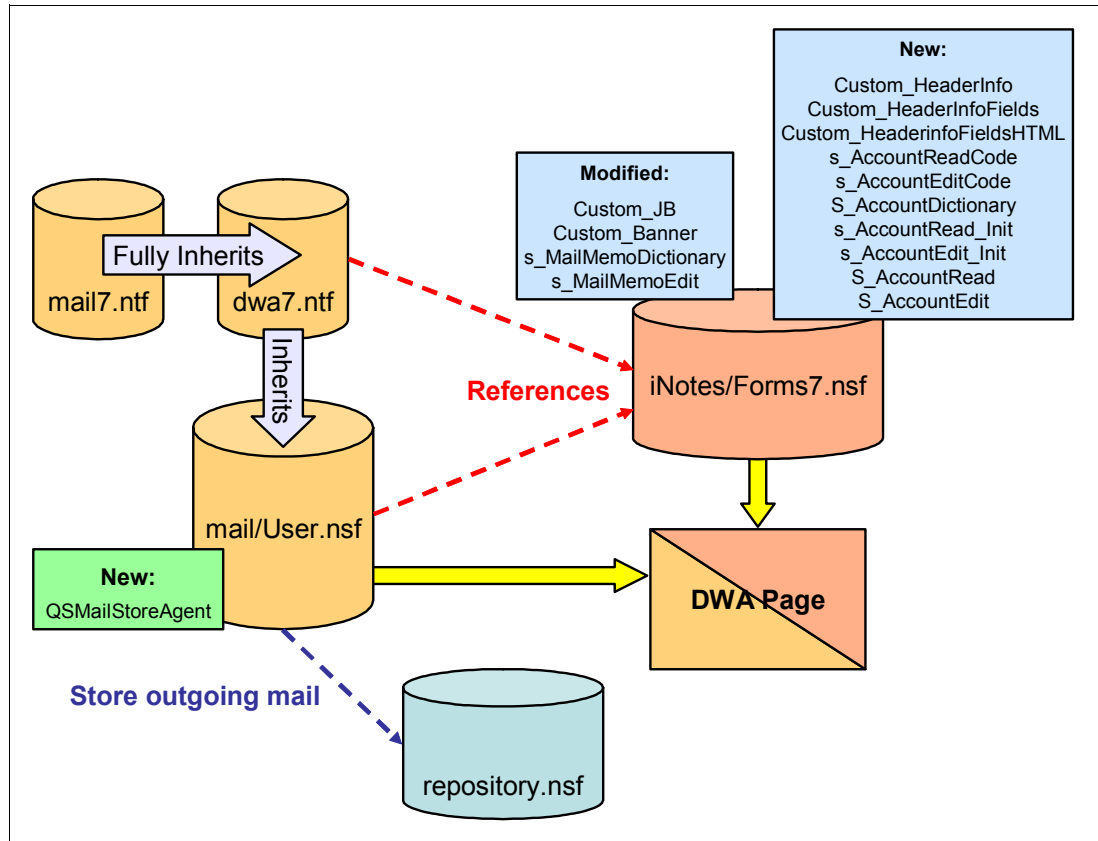


Figure 5-1 ITSO Bank customization overview

From Figure 5-1, you can see that we have added the following new design elements to the forms file (forms7.nsf):

- ▶ Form: Custom_HeaderInfo
- ▶ Subform: Custom_HeaderInfoFields
- ▶ Subform: Custom_HeaderInfoFieldsHTML
- ▶ Subform: s_AccountReadCode
- ▶ Subform: s_AccountEditCode
- ▶ Subform: s_AccountDictionary
- ▶ Subform: s_AccountRead_Init
- ▶ Subform: s_AccountEdit_Init
- ▶ Form: s_AccountRead
- ▶ Form: s_AccountEdit

And we have modified the following design elements from the forms file:

- ▶ Form: Custom_JS
- ▶ Form: Custom_Banner
- ▶ Subform: s_MailMemoDictionary
- ▶ Subform: s_MailMemoEdit

Additionally, we have added a new agent in the mail template file Agent: QSMailStoreAgent.

Finally, we have created a Domino database (repository.nsf) that will act as the repository in this example implementation. Copies of outgoing messages will be stored in this database.

5.2 Domino Web Access functions for Action menu operations

Before getting into hands-on examples for manipulating action menus, let us examine some of the helpful JavaScript functions that are available in Domino Web Access 7.0.1.

5.2.1 DM_getMenuByPos

This action gives you the handle to the menu node object using the menu position as the parameter.

- ▶ Obfuscated name: Dap
- ▶ Parameters:
 - p1 = branch ID
 - p2 = position

5.2.2 DM_getMenuById

This action gives you the handle to the menu node object using the menu node ID as the parameter.

- ▶ Obfuscated name: DPN
- ▶ Parameters:
 - p1 = menu node or node ID

5.2.3 DM_getMenuByLabel

This function gives you the handle to the menu node object using the label as the parameter.

- ▶ Obfuscated name: CyE
- ▶ Parameters: DM_getMenuByLabel(p1,p2)
 - p1 = tree ID
 - p2 = menu label

5.2.4 DM_removeMenu

Given a menu ID, this function removes the menu item.

- ▶ Obfuscated name: DGa
- ▶ Parameter: DM_removeMenu(p1)
 - p1 = menu node or node ID

5.2.5 DM_newMenu

This action creates a new menu item. It adds the item to the action bar as well as to the right-click menu.

- ▶ Obfuscated name: Dbc
- ▶ Parameters: DM_newMenu(p1,p2,p3)
 - p1 = container ID, can be node or branch ID
 - p2 = unique ID (optional)

- p3 = boolean value indicating whether this is the default action on ENTER press (optional)

5.2.6 DM_getParentNode

This action returns the parent node.

- ▶ Obfuscated name: DRo
- ▶ Parameters:
 - p1 = menu node object or node ID

5.2.7 DM_getChildNodes

This action returns an array of child nodes.

- ▶ Obfuscated name: DIh
- ▶ Parameters:
 - p1 = menu node object or node ID

5.2.8 DM_updateActionBar

This action updates the action bar after a change to the label, icon, or disabled status.

- ▶ Obfuscated name: CzB
- ▶ Parameters:
 - None

5.2.9 DM_getSubmenuItem

This action returns a menu object (not the html, but the data) by its ID.

- ▶ Obfuscated name: Cwo
- ▶ Parameters:
 - p1 = branch ID
 - p2 = submenu level
 - p3 = context (optional, the value is either “actionState” or “contextState”)

5.2.10 DM_createContextMenu

This action assigns an oncontextmenu event handler to the specified HTML element. The specified menu tree will be shown.

- ▶ Obfuscated name: DWh
- ▶ Parameters:
 - p1 = HTML element ID
 - p2 = branch ID to show
 - p3 = callback function when context menu is about to show

5.2.11 DM_getBranch

This action returns an ordered array of the specified menu node object's siblings.

- ▶ Obfuscated name: DZo
- ▶ Parameters:
 - p1 = menu node or node ID
 - p2 = return sorted array
 - p3 = state, can be null

5.3 Overwrite New Message action

ITSO Bank requires their employees to add header information to outgoing mail messages prior to sending them. This is accomplished by overwriting the Domino Web Access New Message action so that it prompts the employees to populate the header information via a dialog box (see Figure 5-2) before displaying the Domino Web Access new message page.

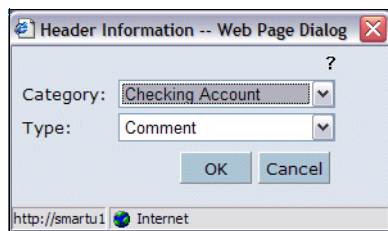


Figure 5-2 Header Information dialog box

The following steps are necessary for accomplishing this task:

- ▶ Create Custom_HeaderInfo form, a new dialog page.
- ▶ Modify Custom_JS form.
- ▶ Modify the s_MailMemoDictionary subform.
- ▶ Modify the s_MailMemoEdit subform.

5.3.1 Create a new dialog page

In order to create a new dialog page, you must create a custom form in the forms file. We recommend that you start with a form in forms7.nsf that implements an existing modal dialog box, such as s_DeliveryOptions. You should keep the majority of the head portion and modify the body portion to fit your purposes.

It is important to keep the head portion of this form because these dialogs bring in certain common external script files that may change from release to release of Domino Web Access, and are typically brought in via some server-side evaluated @formulas. Additionally, various releases emit the script tags in different ways. Earlier releases emitted them via specific `<script ...></script>` tags on the page. Newer releases have started emitting these via script as well, further reducing the size of the various pages. But they rely on various other JavaScript globals as well, located within the `<head>...</head>` section of the page.

Starting with a copy and retaining the majority of the head portion and modifying the body portion allows you to leverage the same modal dialog box foundation used within other modal dialog boxes in Domino Web Access.

We have created a new form called Custom_HeaderInfo based on the s_DeliveryOptions form and modified the body portion, as in Example 5-1.

Example 5-1 Body portion of Custom_HeaderInfo dialog

```

<FORM NAME="s_HeaderInfo">
<script>
  var theForm=document.forms['s_HeaderInfo'];
  var IF="H_REQUESTING_A_DELIVERY_REPORT_STEPS_01.html";</script>
</script>
<table border="0" width="250" style="margin-left:5px;margin-right:5px;">
  <tr>
    <td width="30%">&nbsp;</td>
    <td align="right"><a onclick="javascript:OE(IF)"></a></td>
  </tr>
  <tr>
    <td class="s-form-label" width="30%">Category:</td>
    <td>
      <select class="s-form-input" name="DOC_CATEGORY" style="width:100%">
        <option>Checking Account</option>
        <option>Saving Account</option>
        <option>Loan</option>
      </select>
    </td>
  </tr>
  <tr>
    <td class="s-form-label">Type:</td>
    <td>
      <select class="s-form-input" name="DOC_TYPE" style="width:100%">
        <option>Comment</option>
        <option>Complaint</option>
        <option>Information</option>
        <option>Other</option>
      </select>
    </td>
  </tr>
  <tr><td width="100%" colspan="2">
    <table border="0" cellspacing="3" cellpadding="1" width="100%">
      <tr>
        <td width="50%">&nbsp;</td>
        <td width="25%" align="right">
          <input class="s-form-button" type="button" id="HeaderDataBtnOK"
onclick="BDA('OK')" value="OK" style="width:100%">
        </td>
        <td width="25%" align="left">
          <input class="s-form-button" type="button" id="HeaderDataBtnCancel"
onclick="BDA('Cancel')" value="Cancel" style="width:100%">
        </td>
      </tr>
    </table>
  </td></tr>
</table>
<script language="JavaScript">
function dj(){
  self.returnValue=false;

```



```

}
function BDA(yX){
  if (yX == "OK") {
    var ID = window.dialogArguments;
    var field = theForm.DOC_CATEGORY;
    ID.DCat = field.options[field.selectedIndex].text;
    var field = theForm.DOC_TYPE;
    ID.DType = field.options[field.selectedIndex].text;
    self.returnValue=true;
  }
  else
    self.returnValue=false;
  self.close();
}
</script>
</FORM>
</body></html>

```

ITSO Bank implements a simple dialog box with no lookup for its selection lists. For a more complex dialog box sample that includes lookup and demonstrates how the lookup is cached, refer to the IBM DeveloperWorks article titled "Manipulating Data in Domino Web Access" at: <http://www-128.ibm.com/developerworks/lotus/library/dwa-data/>

This article also shows you how to add a new online help content. Since we used the Delivery Options form as the starting point, the question mark icon(?) in the ITSO Bank Header Information dialog box (Figure 5-2 on page 69) brings up the online help for delivery options. To change the target for the help function replace the IF Variable value (IF="H_REQUESTING_A_DELIVERY_REPORT_STEPS_01.html";) with the value of the html document target in the Domino Web Access help database or just empty the value to point to the default help entry point. "Creating a custom help document in the Domino Web Access help database" on page 106 describes how to create a custom help document.

5.3.2 Modify Custom_JS form

Now, let us see how we overwrite the New Message action menu. Action menu operations (add, remove, re-order, overwrite) are done by modifying the Scene_Actions function in the Custom_JS form. Scene_Actions is called just before the action buttons are added to the action bar.

Example 5-2 Overwrite New Message action in Version 7.0.1

```

function Scene_Actions( s_SceneName, o_Window, s_TopBranchId ){
...

  switch (s_SceneName) {
    case 'Mail':
      var menuNode;

      //Overwrite the "New" action
      menuNode = CyE(s_TopBranchId, "New");
      menuNode.onclick = "javascript:showHeaderDialog()";
      var a_NewBranchId = menuNode.$branchId;

      //Overwrite the "New\Message" action

```

```

        menuNode = CyE(a_NewBranchId, "Message");
        menuNode.onclick = "javascript:showHeaderDialog()";
        break;
    default:
        break;
}
...
}

```

Action menu operations is one of the areas where customization is vastly improved in Domino Web Access 7 as compared to the earlier version. Notice how much more intuitive the menu structure in Version 7.0.1 (see Example 5-2 on page 71) is as compared to how it was in Version 6.5 (see “Wrapper functions for action menu operations” on page 116). In Version 7.0.1, you can use the new functions for action menu operations (see 5.2, “Domino Web Access functions for Action menu operations” on page 67) to get the handle to the menu object. Once you get the handle to the menu object, you can easily overwrite its methods and properties. In Version 6.5, you have to understand the structure of the menu in order to overwrite it.

ITSO Bank is only concerned with action menu operations in the Mail tab (s_SceneName = ‘Mail’). Table 5-1 provides a listing of other scenes in Domino Web Access that you might consider working with.

Table 5-1 Scene names mapping

Scene name	Description
Welcome	Welcome tab
Calendar	Calendar tab
ToDo	To Do tab
Contacts	Contacts tab
Notebook	Notebook tab
s_MailMemo	Memo, reply, forward, stationery
s_MailPhoneMessage	Phone message entry
s_Appointment	Calendar entry
s_ToDo	To do entry
s_Contact	Contact entry
s_Group	Group document
s_NotebookPage	Notebook entry
s_OutOfOffice	Out of office setup

Tip: You can discover which scene you are working with by using the JavaScript alert statement to the Scene_Actions function (in Custom_JS form) to display the value of s_SceneName with a JavaScript statement like alert(s_SceneName).

As described in Example 5-2 on page 71, the New Message action menu onclick event is replaced with the javascript:showHeaderDialog() statement. This function (see Example 5-3) uses Internet Explorer's window.showModalDialog DOM method to display the custom dialog box. Domino Web Access implements something comparable for Gecko, so this will work on Firefox and Mozilla as well. Using this function, the dialog box automatically behaves as a modal dialog.

ITSO Bank employees must select a document category and type (from the dialog box) and click the **OK** button in order to proceed with composing a new message. If they click the Cancel button, they will not be able to compose the memo.

Example 5-3 showHeaderDialog function

```
function showHeaderDialog() {
    //
    // this function displays a dialogbox prior to composing a new message
    //
    var ID = new Object;

    // The Header Information dialogbox is not a main page,
    // you can see that we override the form using &Form=Custom_HeaderInfo
    // so that it uses the form specified instead of h_PageUI
    var shref = lT(self) +
'/iNotes/Proxy/?OpenDocument&Form=Custom_HeaderInfo&l=en@{@If(s_Charset="";"";"&ch
arset=" + s_Charset)}';

    var lI = window.showModalDialog(shref, ID,
"dialogWidth:300px;dialogHeight:175px;dialogTop:200px;dialogLeft:200px;help:no;sta
tus:no;center:no;");

    // We are passing the values of document category and type
    // through the &PresetFields, which is the third argument
    // of the openNewShimmerDoc function
    if (lI) { //User clicks OK
        var str_preset = 'DOC_CATEGORY;' + encodeURIComponent(ID.DCat) +
',DOC_TYPE;' + encodeURIComponent(ID.DType);
        openNewShimmerDoc('($Drafts)', 'Memo', str_preset);
    }
}
```

The showHeaderDialog function utilizes an ID object, which returns the values selected from the dialog box. After the ITSO Bank employee clicks the OK button, it makes a call to the Domino Web Access openNewShimmerDoc function. The values returned by the ID object (document category and type) are passed to the newly composed memo document using &PresetFields (see Figure 5-3). Notice how the URL of the new message has the information about the custom fields in Figure 5-3.

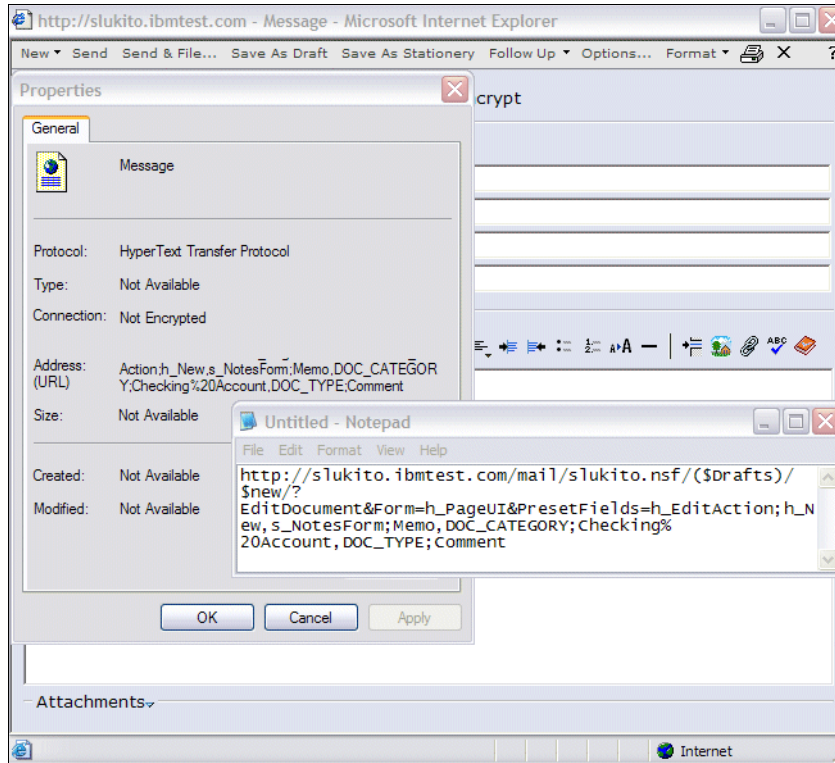


Figure 5-3 New message

5.3.3 Modify s_MailMemoDictionary subform

In order for the header information fields (document category and document type) to persist in the memo document, we need to add <NOTESFIELD> definitions for those two fields (see Example 5-4) in the s_MailMemoDictionary subform.

As discussed in 1.5.2, “Isolate your custom code” on page 7, we recommend that anytime you modify existing Domino Web Access forms or subforms, you should separate your changes from Domino Web Access code. One way to do so is by putting your custom modifications into a subform (see Example 5-4) and insert your custom subform in the Domino Web Access form or subform to modify.

Example 5-4 Custom_HeaderInfoFields

```
<NotesDictionary>
<NOTESFIELD NAME={DOC_CATEGORY}>
<NOTESFIELD NAME={DOC_TYPE}>
<NOTESVAR NAME={$$QuerySaveAgent} TYPE={Text} INITIALVALUE={"QSMailStoreAgent"}>
</NotesDictionary>
```

In order to insert the subform, you will add the following statement at the beginning of the s_MailMemoDictionary subform:

```
<InsertNotesSubform NAME={Custom_HeaderInfoFields}>
```

5.3.4 Modify s_MailMemoEdit subform

In addition to the <NOTESFIELD> tags, Domino Web Access also needs the HTML representation of the fields as a way to obtain the values selected by ITSO Bank employees via the dialog box.

Here is another implementation of the best practices by separating the custom HTML fields in the Custom_HeaderInfoFieldsHTML subform (see Example 5-5).

This subform is inserted into the beginning of the s_MailMemoEdit subform using the following statement:

```
<InsertNotesSubForm NAME={Custom_HeaderInfoFieldsHTML}>
```

Example 5-5 Custom_HeaderInfoFieldsHTML

```
<input type="hidden" name="DOC_CATEGORY" value="@{DOC_CATEGORY}" />  
<input type="hidden" name="DOC_TYPE" value ="@{DOC_TYPE}" />
```

QuerySaveAgent

As part of its e-mail management compliance process, ITSO Bank will store copies of all outgoing e-mail messages in a custom Domino database (repository.nsf). We implement the QuerySave agent (see Example 5-6) to accomplish this. This agent will be invoked before the message is sent out. Unlike other design changes that take place in the forms file, QueryOpen and QuerySave agents must reside in the mail template.

In order to enable QueryOpen and QuerySave agents in Domino Web Access, add the following line to the server's notes.ini file:

```
iNotes_WA_QueryAgents=1
```

Additionally, you will have to create a <NOTESVAR> for \$\$QuerySaveAgent in the s_MailMemoDictionary subform (see Example 5-4 on page 74).

Example 5-6 QSMailStoreAgent agent

```
Sub Initialize
```

```
Dim session As New NotesSession  
Dim db As NotesDatabase  
Dim doc As NotesDocument  
Dim repository_db As NotesDatabase  
Dim repository_doc As NotesDocument
```

```
Set db = session.CurrentDatabase  
Set doc = session.DocumentContext
```

```
'Get the handle to the repository database  
Set repository_db = New NotesDatabase(db.Server, "repository.nsf")  
If Not (repository_db.IsOpen) Then  
    Messagebox "Cannot open the repository database!", 48, "Error"  
    Exit Sub  
End If
```

```
'Store a copy of the message in the repository
Set repository_doc = repository_db.CreateDocument
Call doc.CopyAllItems(repository_doc)
Call repository_doc.Save(True, True)
```

End Sub

Figure 5-4 shows a sample repository entry, which displays a copy of the message and the additional header fields.

If desired, you can also add a callback function to handle the error more gracefully. For example, instead of simply printing the error out at the server console (or log) when the repository database is not found, you could prompt the user with the error message box instead.

Refer to the IBM DeveloperWorks article titled “Manipulating Data in Domino Web Access” at the following Web site for more details:

<http://www-128.ibm.com/developerworks/lotus/library/dwa-data/>

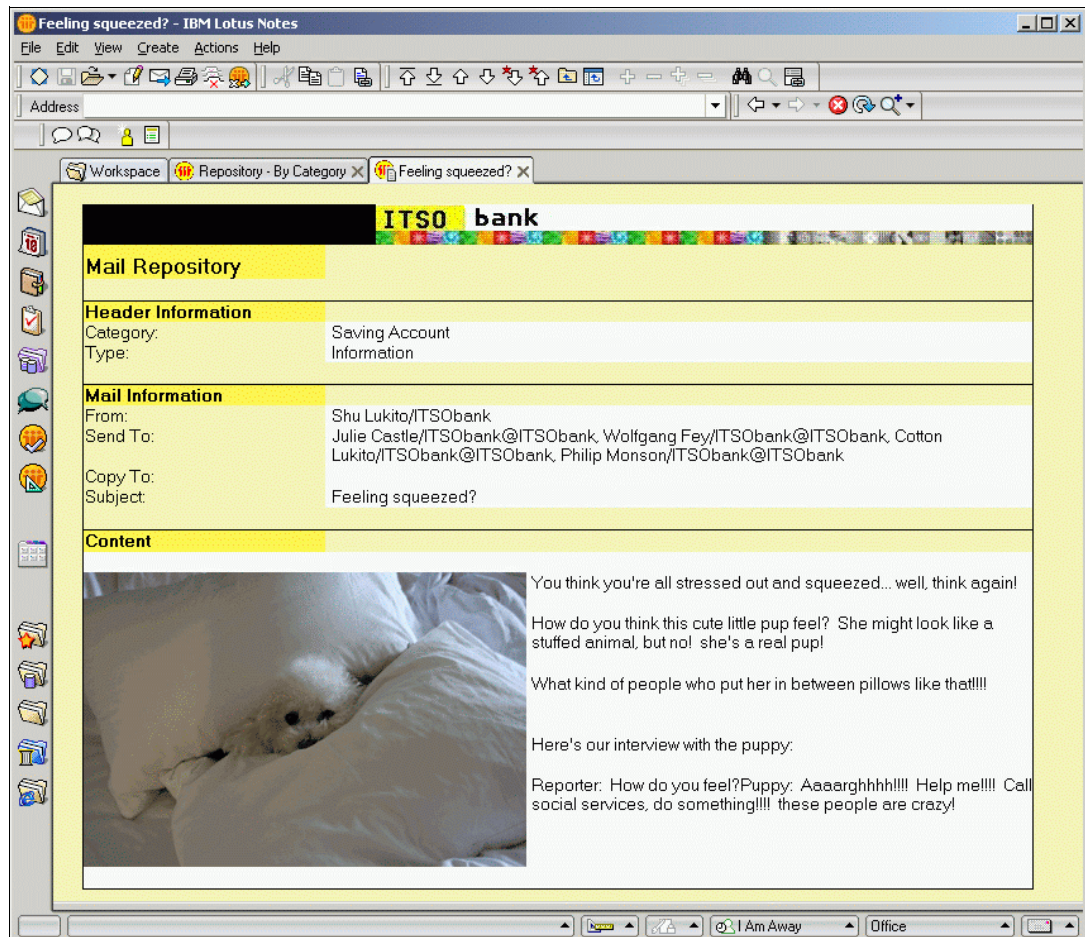


Figure 5-4 Mail Repository entry

5.4 Remove Action Tools → Block Mail from Sender

ITSO Bank does not want its employees to filter any incoming messages and has decided to remove the Block Mail from Sender menu from the Tools action. We accomplish this by adding the javascript code described in Example 5-7 to the Custom_JS form.

Again, we can see here how easy it is to remove action menus in Version 7.0.1 as compared to how it was done in Version 6.5 (see “Removing action menu in Version 6.5” on page 116). You can also see examples of removing action menus in Version 7.0.1 utilizing wrapper functions there.

Example 5-7 Remove Tools → Block Mail from Sender action menu

```
function Scene_Actions( s_SceneName, o_Window, s_TopBranchId ){  
  
...  
  
    switch(s_SceneName) {  
        case 'Mail':  
            // Remove the "Tools\Block Mail from Sender" action  
            var toolsNode = CyE(s_TopBranchId, "Tools");  
            var bmsNode = CyE(toolsNode.$branchId,"Block Mail from Sender");  
            DGa(bmsNode.$id);  
            break;  
        }  
  
...  
    }  
}
```

5.4.1 Add Action New → Account

ITSO Bank wants to streamline some of its daily processes by allowing their employees to create a new account directly from the mailboxes. Copies of the newly created account information is stored in the employees' mailboxes and a back-end process will add the newly created accounts to the main account database. In our sample, we will not include the back-end implementation.

Example 5-8 demonstrates how to modify Custom_JS in order to create a new action menu in Domino Web Access. The equivalent code for the previous Version 6.5 as well as the wrapper functions in Version 7.0.1 are displayed in “Adding new action menu in Version 6.5” on page 124.

Example 5-8 illustrates the usage of the addAction function call.

Example 5-8 Add New → Account action menu

```
function Scene_Actions( s_SceneName, o_Window, s_TopBranchId ){  
  
...  
  
    switch(s_SceneName) {  
        case 'Mail':  
            menuNode = CyE(s_TopBranchId, "New");  
            var a_NewBranchId = menuNode.$branchId;  
            addAction(menuNode.$id, menuNode.$label, menuNode.$url, menuNode.$icon, menuNode.$text, menuNode.$type, menuNode.$parent, menuNode.$branchId, menuNode.$children);  
        }  
  
...  
    }  
}
```

```

...

//Add a menu under the "New" action
menuNode = Dbc(a_NewBranchId);
with (menuNode) {
    pos = 999;
    label = "Account";
    onclick = "javascript:openNewShimmerDoc('$ToDo','Account')";
    DFB = "Creates a new account form";
}
}

...
}

```

5.5 Add a new form, ccount

The following actions show how to add a new main page to Domino Web Access. To perform that several steps need to be done:

1. Create a new form in Domino Designer.
2. Test the form in the Notes client.
3. Add the Formsmap form to forms7.nsf.
4. Create the forms mapping document.
5. Add the forms and subforms for the dictionary, edit, and read scenes.

5.5.1 Create a new form in Designer

Creating a form in the Notes client is not really necessary if the new form is used for Domino Web Access only. However, there are some cases where a new form is useful:

- ▶ If the form has to be accessed from the Notes client as well as Domino Web Access.
- ▶ A form is an easy way to create test data.
- ▶ A form allows you to inspect documents via the Notes client.

Domino Web Access does not really care if there is such a form in the mail file. It only cares about the forms map table to know which field in the Domino Web Access form has to be stored and read from which field in the backend form.

Note: The only place that Domino Web Access relies on a Notes form is in the Forward any document object functionality, where the WebEngine's render to form capability is used to construct HTML to forward the document by mail.

For the example shown in this chapter a new account form in the mail file is used here. The Notes form has five fields:

- ▶ Account#
- ▶ Account Type
- ▶ Lastname
- ▶ Firstname
- ▶ Internal Contact

Figure 5-5 shows this simple form in Domino Designer.

ITSO bank	
Account Statement	
Account#	acctno T
Type	acctype T
Lastname	lastname T
Firstname	firstname T
Internal contact	internalcontact T
Hidden:	
	Form T

Figure 5-5 New account form in Designer

5.5.2 Create document in Notes client

To test the newly created form, create a new document with the form in the Notes client. This form should be displayed correctly and also the document should be readable after it is saved. If a view is needed to select the documents created with the new form it should also be built now. If all forms and views are built into the dwa7.ntf template they can be used with Domino Web Access easily.

5.5.3 Add the FormsMap form to forms7.nsf

To map the Notes form to a Domino Web Access form definition later, a mapping document is needed in forms7.nsf. This forms mapping document needs to be defined with a form called FormsMap in the forms7.nsf, which is not included in the product. Figure 5-6 shows the definition for the fields needed for this form.

FormsMap	
h_SetEditScene	h_SetEditScene T
h_SetReadScene	h_SetReadScene T
s_NotesForm	s_NotesForm T
s_SubFormPrefix	s_SubFormPrefix T
Form	Form T

Figure 5-6 FormsMap form definition in Designer

5.5.4 Create the FormsMap mapping document

After adding the formsmap form the mapping document can be created. Creating the mapping defines several parameters for the mapping. Figure 5-7 shows the mapping document.

- ▶ The edit scene
- ▶ The read scene
- ▶ The Notes form name
- ▶ The subform prefix

FormsMap	
h_SetEditScene	⌘ s_StdPageEdit_⌘
h_SetReadScene	⌘ s_StdPageRead_⌘
s_NotesForm	⌘ Account_⌘
s_SubFormPrefix	⌘ s_Account_⌘
Form	FormsMap

Figure 5-7 FormsMap definition for new Account form

The edit scene and read scene are set to the default scenes that come with the product. These are subforms in the forms7.nsf database and define mostly javascript code to set up the environment on the browser to edit or read documents through the Domino Web Access proxy mechanism.

The Notes form name is the corresponding backend form name and should be stored in a field of the backend form called Form.

Note: After adding the specific new formsmap document, the form can be removed again with Domino Designer. But in case some changes have to be made, the form is needed again, so it might be better to keep the formsmap form in the forms7 database.

5.5.5 Special subforms for page creation

The subform prefix is the first part of the name for the five subforms needed to define the Notes fields and items, the JavaScript, and the HTML to display the Domino Web Access form in the browser. The names for the subforms are:

- ▶ s_AccountDictionary
- ▶ s_AccountEdit_Init
- ▶ s_AccountEdit
- ▶ s_AccountRead_Init
- ▶ s_AccountRead

These subforms are shown in detail later.

After the forms mapping has been defined and saved it is shown in the Notes client in the System\Forms view, as shown in Figure 5-8.

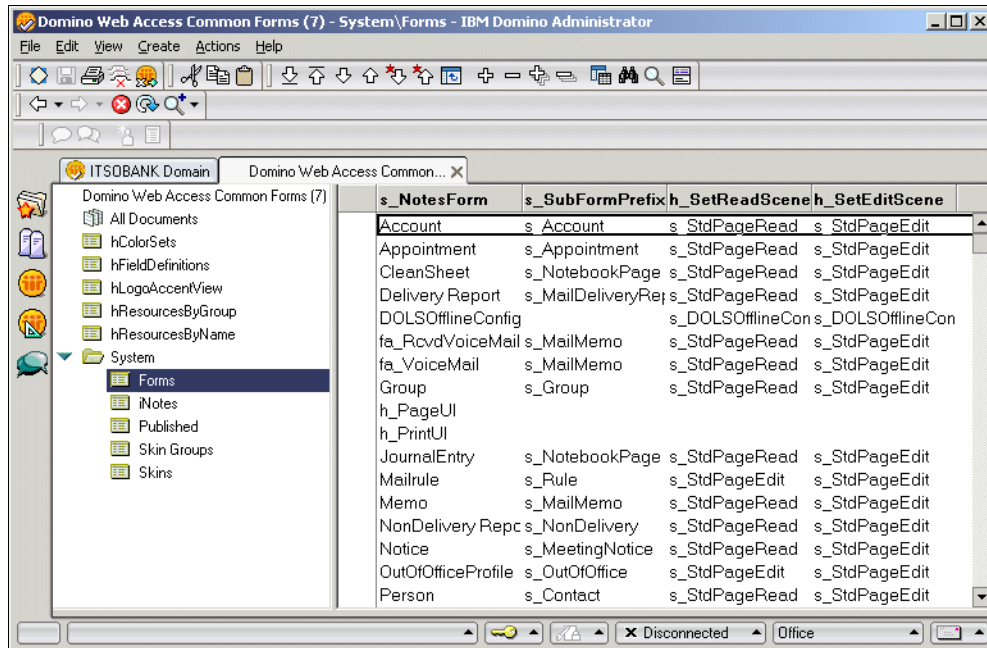


Figure 5-8 Account FormsMap in forms7.nsf in System\Forms view

5.5.6 Add the forms and subforms for the scene

For the creation of the whole page content of the ITSO Bank account form main Domino Web Access page there are five subforms and two forms needed. The subforms are already mentioned in 5.5.5, “Special subforms for page creation” on page 80. The two forms contain the externalized JavaScript code for the creation of the action bars in the context of reading or editing a document.

A good starting point for the creation of the five subforms is to take a look at an existing set of subforms, for example, the set for the journal called notebook page. These forms are not that complicated and only have a minimum of code to deliver for the view fields of the notebook page but also contain the whole code for embedding the Domino Web Access DHTML rich text editor. In the account form sample only text fields are used to keep it simple.

Subform s_AccountDictionary

In order to define the Notes fields and some additional variables needed for the page creation the subform contains a section called NotesDictionary. See Example 5-9.

Example 5-9 Code for the s_AccountDictionary subform

```
<NotesDictionary>
<notesvar name=InitializeFields type=text>
<notesvar name="CalendarOwner"
initialvalue={@Name([Canonicalize];@DbCommand("haiku";"h_GetProfileField";"calenda
rprofile";"Owner"))}>
<notesfield name="Principal"
initialvalue={@If(CalendarOwner!="";CalendarOwner;@UserName(0))}>
<notesfield name="From" authorfield="Yes" initialvalue={@UserName(0)}>
<NOTESFIELD NAME=$NoPurge TYPE=Text INITIALVALUE={"1"}>
```

```

<NOTESVAR NAME=h_Name TYPE=Text>
<NOTESFIELD NAME=Form TYPE=Text INITIALVALUE="Account">
<NOTESFIELD NAME="acctno" TYPE=Text>
<NOTESFIELD NAME="accttype" TYPE=Text>
<NOTESFIELD NAME="lastname" TYPE=Text>
<NOTESFIELD NAME="firstname" TYPE=Text>
<NOTESFIELD NAME="internalcontact" TYPE=Text>
</NotesDictionary>

```

The <NOTESFIELD> tags define the Notes items that will be saved with the document when the document is saved by the user. Such entries along with NOTESVAR entries define which Notes items will be represented as JavaScript vars for the h_Vars custom Domino Web Access formula.

Subform s_AccountEdit_Init

This subform contains code to include the s_AccountEditCode form in the page and additional variables for the page creation. Also, the pagetitle is set to something useful.

Note: There are different techniques available to include externalized code into a form or subform. For more information see “Different techniques to include externalized code”.

The IF variable is set to an empty string in this example. This variable is used to open the context-specific help from the help database, if a user clicks the question mark (?) in the form. The empty string leads to the default Domino Web Access help page.

The value for SceneTitleSkinComponentTitleText sets the title for the window in a document opened in edit mode.

The formula @{{s_FF}} returns the server root relative path to the current forms database for accessing resources in the current forms database and is described in 3.8.4, “Special Domino Web Access formulas” on page 37.

Example 5-10 Code for s_AccountEdit_Init subform

```

<NotesDictionary>
<NOTESVAR name=h_SkinTypeOverride TYPE=TEXT initialValue={"h_MailPage"}>
</NotesDictionary>
<script>
  SceneTitleSkinComponentTitleText="Account information";
  IF="";
</script>
<script>
src="@{{s_FF}}/iNotes/Proxy/?OpenDocument&Form=s_AccountEditCode@{{s_StaticJSArgs}}">
</script>

```

Subform s_AccountEdit

In this subform the HTML layout of the page has to be done. This is also where the HTML form is designed and the JavaScript calls are created to get the data from the frontend form to the backend document.

The main part is the HTML code construction for the table used to display the form. There are some special JavaScript functions used: SV (unobfuscated name *pageWrite*) is a function to write the output of a JavaScript script into the current document. So here is a simplified if

condition in the (condition?true:false) notation used to display either code for Firefox/Gecko or Internet Explorer. The function “DLI” (unobfuscated name *gbIsGecko*) is used to decide whether the current browser is Gecko based. The function DJg (unobfuscated name *unlockTextAreaHeight*) is used to implement for Gecko something comparable to the overflow-y:auto CSS style.

The last call used haiku.LB.add("qP()") (unobfuscated name for LB is *oOnloadChain*), which is a function to put the function that is given as a parameter in the onload chain of the current document. Here there is the function qP() (unobfuscated name *initInputFields*) added, which is used for the initialization of the fields in the browser. This particular function is defined in the s_AccountEditCode form, which is documented in “Form s_AccountEditcode” on page 85.

Example 5-11 Code for the s_AccountEdit subform

```
<table border="0" width="100%" cellpadding="0" cellspacing="0">
  <tr><td></td></tr>
  <tr><td><h2>Account Statement</h2></td></tr>
  <tr><td>
    <table border="0" cellpadding="2" cellspacing="2" width="100%">
      <tr><td class="s-form-label" width=150px>Account #</td><td><script>
        SV('<textarea class="s-form-input" name="acctno" id="acctno" rows=1"
        style="width:100%;' + (DLI ? 'height:3.1ex' onkeyup="DJg(event)" :
        'overflow-y:auto') + '></textarea>');</script></td></tr>
      <tr><td class="s-form-label">Account Type</td><td><script> SV('<textarea
        class="s-form-input" name="accttype" id="accttype" rows=1"
        style="width:100%;' + (DLI ? 'height:3.1ex' onkeyup="DJg(event)" :
        'overflow-y:auto') + '></textarea>');</script></td></tr>
      <tr><td class="s-form-label">Lastname</td><td><script> SV('<textarea
        class="s-form-input" name="lastname" id="lastname" rows=1"
        style="width:100%;' + (DLI ? 'height:3.1ex' onkeyup="DJg(event)" :
        'overflow-y:auto') + '></textarea>');</script></td></tr>
      <tr><td class="s-form-label">Firstname</td><td><script> SV('<textarea
        class="s-form-input" name="firstname" id="firstname" rows=1"
        style="width:100%;' + (DLI ? 'height:3.1ex' onkeyup="DJg(event)" :
        'overflow-y:auto') + '></textarea>');</script></td></tr>
      <tr><td class="s-form-label">Internal Contact</td><td><script> SV('<textarea
        class="s-form-input" name="internalcontact" id="internalcontact" rows=1"
        style="width:100%;' + (DLI ? 'height:3.1ex' onkeyup="DJg(event)" :
        'overflow-y:auto') + '></textarea>');</script></td></tr>
    </table>
  </td></tr>
</table>
<!-- HIDDEN FIELDS -->
<div style="display:none">
  <input name="From" id="From">
  <input name="Principal" id="Principal">
  <input name="Type" id="Type" value="Account">
  <input name="Form" id="Form" value="Account">
</div>
<script>haiku.LB.add("qP()");</script>
```

Subform s_AccountRead_Init

All variables used for the construction of the form in read mode are defined in this subform. It also contains a reference for the s_AccountReadCode form, which defines additional

externalized JavaScript functions and is documented in “Form s_AccountReadcode” on page 87. The use of the script tag with an external form reference will allow for caching of the external JavaScript and possibly save future bandwidth associated with the page.

The @s_StaticJSArgs delivers the desired URL arguments for URLs to external static script pages. See Example 5-12.

Example 5-12 Code for the s_AccountRead_Init subform

```
<NotesDictionary>
<NOTESVAR NAME=h_SkinTypeOverride TYPE=TEXT INITIALVALUE={"h_MailPage"}>
<NOTESVAR NAME=h_Details TYPE="Text">
</NotesDictionary>
  <script
    src="@{s_FF}/iNotes/Proxy/?OpenDocument&Form=s_AccountReadCode@{s_StaticJSArgs}
  ">
</script>
```

Subform s_AccountRead

This subform contains the HTML code needed for reading an account document. The HTML creates a table to display the content in a proper format. Again there are some JavaScript functions used: “SV” and “haiku.LB.add” are already explained in “Subform s_AccountEdit” on page 82.

The function “AdjustWindowSize” is used to properly size and position the window in the onload JavaScript event.

Example 5-13 Code for the s_AccountRead subform

```
<table border="0" width="100%" cellspacing="0" cellpadding="0">
  <tr><td></td></tr>
  <tr><td><h2>Account Statement</h2></td></tr>
  <tr><td>
    <table border="0" width="100%" cellspacing="0" cellpadding="0">
      <tr><td class="s-form-label" style="height:.5ex" colspan="2"></td></tr>
      <tr><td class="s-form-label-bold" width="20%" valign="BASELINE"
        align="left">Account #</td><td class="s-form-label">&nbsp;</td><td
        class="s-form-label"
        width="90%"><script>SV(window.acctno);</script></td></tr>
      <tr><td class="s-form-label-bold" width="20%" valign="BASELINE"
        align="left">Account Type</td><td class="s-form-label">&nbsp;</td><td
        class="s-form-label"
        width="90%"><script>SV(window.acctype);</script></td></tr>
      <tr><td class="s-form-label-bold" width="20%" valign="BASELINE"
        align="left">Lastname</td><td class="s-form-label">&nbsp;</td><td
        class="s-form-label"
        width="90%"><script>SV(window.lastname);</script></td></tr>
      <tr><td class="s-form-label-bold" width="20%" valign="BASELINE"
        align="left">Firstname</td><td class="s-form-label">&nbsp;</td><td
        class="s-form-label"
        width="90%"><script>SV(window.firstname);</script></td></tr>
      <tr><td class="s-form-label-bold" width="20%" valign="BASELINE"
        align="left">Internal Contact</td><td class="s-form-label">&nbsp;</td><td
        class="s-form-label"
        width="90%"><script>SV(window.internalcontact);</script></td></tr>
    </table>
  </td></tr>
```

```

    </td></tr>
</table>
<script> haiku.LB.add("AdjustWindowSize()");</script>

```

The form now displays only a minimum of default actions in the action bar. To add an edit action to the action bar at the appropriate place there has to be added some additional JavaScript code to the `s_AccountRead` form.

The code consists of two helper functions, `CTf()` and `DSy()`. `DSy` is a parser for the location referer and returns back the actual document's URL. This `DSy` function is used by the `CTf` function, which replaces the read command in the URL with an edit command and replaces the location URL of the active document browser window. This function is invoked by the edit button, which is finally inserted into the action bar by the `kk()` function call.

Example 5-14 Additional JavaScript code for edit action

```

function CTd()
{
var shref=DSy();
var Bpd=Ns(shref,"PresetFields");
sP=Bpd.replace(' ,h_SetCommand;h_ShimmerNotesIDPW', '');
sP+=((sP != '')? ',': '') + "h_EditAction;h_ShimmerEdit,s_ReadToEdit;1";
var wL= shref.substring(0,shref.indexOf("?OpenDocument"))
+ "?EditDocument&Form=h_PageUI&PresetFields=" + sP;
window.location.replace(wL);
}

function DSy(){
var sHref=gLoc.href;
var Cix=sHref.indexOf(' ,h_SetCommand;h_ShimmerNotesIDPW');
if (-1 != Cix)
    sHref=sHref.substring(0,Cix);
aUrLComp=sHref.split("/");
var n;
for (n=aUrLComp.length-1;n>=0;n--)
    if (aUrLComp[n].indexOf('?OpenDocument')!=-1){
        aUrLComp[n-1]=h_PageUnid;
        break;
    }
return aUrLComp.join("/");
}

function kk () {
    Nv("Edit","javascript:CTd()", 'Edit this document', false, 333);
}

```

Form `s_AccountEditcode`

This form is used to hold all the JavaScript functions needed to display a valid and proper edit form in the browser. The functions have important meanings:

- ▶ Function `kk` (unobfuscated name `SubScene_Init`)

This function is called to initialize the actual subscene to display. This function is called during the page creation in the browser. From here another function, `Nv`, is called.

► Function Nv (unobfuscated name *AddSceneAction*)

This function is used to add an action to the actual scenes action bar menu. It takes two parameters: the first is the actiontitle to display, and the second is the URL to call.

► Function kl (unobfuscated name *SubScene_SubmitLogic*)

This function is called from the submit event of the form. In our case we make sure the principal field has a proper value, and if not we set one. The parameter Bx, which is not used in our example, delivers a handle to the actual pageframe to the function.

► Function mk (unobfuscated name *getObjTitle*)

This function has two tasks:

- Set the document to open in edit mode next time, if it is saved only (not send).
- The second task is to try to update the view window by calling the appropriate helper function to reflect the new document just created. The function name need only be a similar function to the one implemented by the main view.

For example, the Notebook view page implements BHV() onubfuscated *nbViewRefresh* (nb for Notebook) function, and the Notebook object code tries to call such a function (if it exists). Placing logic in a try block allows for graceful coping with cases where it does not exist (perhaps because the Notebook view page is no longer displayed within the other window).

Because we do not implement a specific view displaying the account documents, there is no place to put a specific view-related function. So we try to call the function Bh1() (unobfuscated name *mailViewRefresh*), which is the view refresh for the general mail view.

Example 5-15 Code for the s_AccountEditcode form

```
<NotesDictionary>
<NOTESVAR NAME={$ContentType} VALUE={"application/x-javascript"}>
</NotesDictionary>
function kk(){
  Nv("Save & Close", "javascript:abPreSubmit('h_Jump')", 'Save and close this
  page', false, 20);
  Nv("Save", "javascript:abPreSubmit('h_TempSave')", 'Protect your current edits
  from accidental loss without leaving this page', false, 90);
  Nv("Cancel", "javascript:abPreSubmit(\"h_Exit\")", 'Discard all the changes you
  have made', false, 336);
  if(h_IsNewDoc == "0") Nv("Print", "javascript:0G()", 'Print this
  document', false, 334);
}
function kl(Bx, submitAction){
  theForm.Principal.value=(window.Principal ? window.Principal : haiku.Kq);
  return submitAction;
}
function mk(window, submitAction, wN){
  if(wN == "h_TempSave") theForm.h_SetEditNextScene.value="s_StdPageEdit";
  if(theForm.h_SetReturnURL.value == '<script>self.close();</script>')
    theForm.h_SetReturnURL.value='<script>try{
      var oWin=window.opener?window.opener:window.parent;
      oWin.Bh1();
    }catch(e){}
    self.close();
  </script>';
  return submitAction;
}
```



```

}
function qP(){
  if (window.acctno) theForm.acctno.value=acctno;
  if (window.accttype) theForm.accttype.value=accttype;
  if (window.lastname) theForm.lastname.value=lastname;
  if (window.firstname) theForm.firstname.value=firstname;
  if (window.internalcontact) theForm.internalcontact.value=internalcontact;
}

```

Form s_AccountReadcode

This form contributes the externalized JavaScript code for the creation of the page to read an account form-based document.

The functions kl() and mk() are already explained in “Form s_AccountEditcode” on page 85. Disregard function fh(). It was used at one time to display an error message for unimplemented actions.

Example 5-16 Code for the s_AccountReadcode form

```

<NotesDictionary>
<NOTESVAR NAME={$ContentType} VALUE={"application/x-javascript"}>
</NotesDictionary>
function fh(){alert("Sorry, not yet implemented.");}
function kl(Bx, submitAction){return submitAction;}
function mk(Bx, submitAction, originalAction){
  if('h_Delete'==originalAction){
    theForm.h_SetReturnURL.value='<script>try{
      var oWin=window.opener?window.opener:window.parent;
      oWin.Bhl();
    }catch(e){}
    self.close();
    <\script>';
  }
  return submitAction;
}

```

5.5.7 Use the action New Account

If the new design has been applied to the forms database, all Domino Web Access users are able to use the new action *New Account* in the menu to create a new account document within the Domino Web Access interface.

Create new document in Domino Web Access

If everything went OK and users click the Account link in the New menu in the inbox, users will see a browser window like the one in Figure 5-9.



The screenshot shows a Microsoft Internet Explorer window titled "Account information - Microsoft Internet Explorer". The browser's address bar is empty. The page content includes the "ITSO bank" logo at the top, followed by the heading "Account Statement". Below the heading is a form with five input fields, each with a label on the left and a value in the input box:

Account #	39845987
Account Type	Lease
Lastname	Monson
Firstname	Phil
Internal Contact	Wolfgang Fey

Figure 5-9 Account form

Read the document

Reading an already saved document through Domino Web Access displays the document in a window, as shown in Figure 5-10.



The screenshot shows a Microsoft Internet Explorer window with the address bar containing the URL: <http://smartu1.cam.itsso.ibm.com/mail/wfey.nsf/20965393ba1c567f852571840...>. The page content includes the "ITSO bank" logo at the top, followed by the heading "Account Statement". Below the heading is a list of account details:

Account #	75728236
Account Type	Rent
Lastname	Bergland
Firstname	John
Internal Contact	Wolfgang Fey

Figure 5-10 Reading account document

Forward an account document by mail

If there has been a view created, selecting the account documents in the mail file and the backend form has been applied to the user's mail database, then one or more documents can be selected and forwarded by mail. This functionality of Domino Web Access needs the Notes backend form to render the document into the body of the new message, as displayed in Figure 5-11.

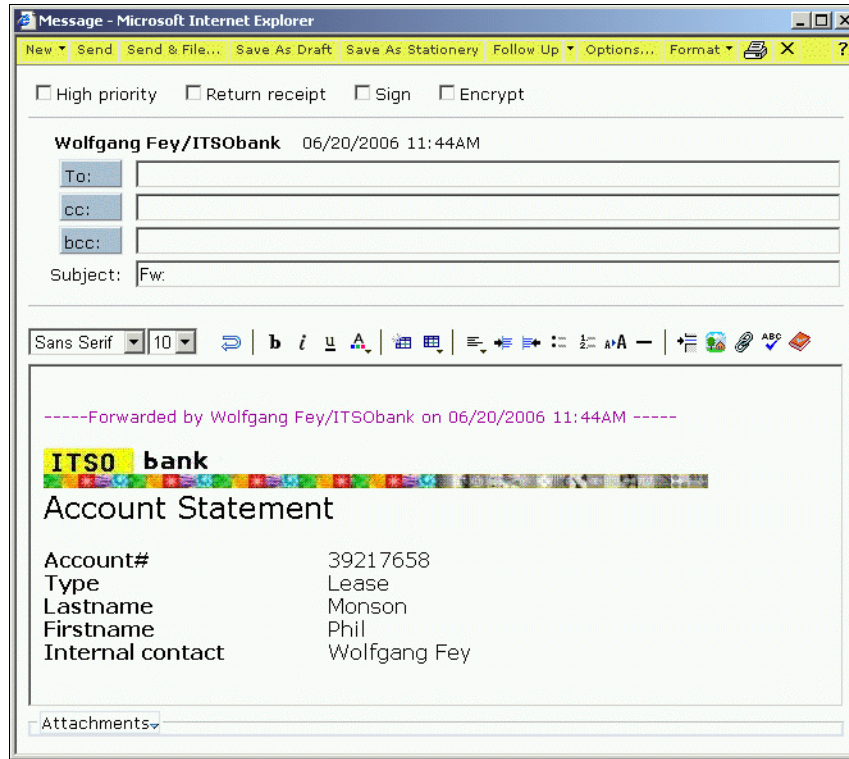


Figure 5-11 Forwarding an account document by mail renders the document into the body field

5.5.8 Preferences

The intention of ITSO Bank is to enable awareness and chat by using Sametime® functionality by default on every user's mail file. This is accomplished by changing the default for the instant messaging setting in the preferences document. This section describes the modifications needed for that customization sample:

- ▶ Modify `s_MailPreferenceEditCode` form.
- ▶ Create the LotusScript agent to modify the preferences document.

Modify `s_MailPreferenceEditCode` form

In order to change the default of the option in the profile document, the form `s_MailPreferenceEditCode` has to be modified. This form contains the code to set the default values in the preferences form.

To modify the default, open the form in Designer and search for the string "SametimeAwareness|0" and change the value to "SametimeAwareness|1". That is all for this particular case. Save and exit the document. Example 5-17 shows the code fragment with all default values of the preferences page in a JavaScript array already with our modified value.

Example 5-17 Code fragment of Form s_MailPreferenceEditCode

```
aFlds=new Array( "MailSaveOpt|1", "MailSort|0", "EnableFTIndex",
"DefaultFolderName" ,"NewMailAlert", "NewMailCheck|1", "NewMailInterval"
,"SoftDeleteExpireTime|48","MailMsgFmt|" + (!haiku.DefaultFormatPlainText ? "1" :
"0"),"MailEditor|1" , "UnreadStyle|red", "ListViewType|0" ,"DateFormat|MM-dd-yyyy"
,"DateSeparator|/" , "TimeFormat|hh:mmt" , "TimeSeparator|:" , "DateFormatLong|dddd,
MMMM dd, yyyy" ,"UseStartTimeZone|" , "UseCurrentTimeZone|1" , "UseAddlTimeZone|0"
,"CurTimeZoneLabel" ,"AddlTimeZoneLabel" ,"AdditionalTimeZone" ,"CurrentTimeZone"
,"CalendarFirstDayFiveDay|" + 1 ,"CalendarFirstDayWeek|" + 1
,"CalendarFirstDayMonth|" + 0 ,"StartupView|" + Bhd[0].systemName
,"DisableDragDropAndInPlace|0" ,"NamePreference|0" ,"SametimeAwareness|1"
,"EnableSignMail|0" , "EnableEncryptMail|0" , "EnableEncryptUnTrustInetCertsMail|0"
,"ReuseChildWindows|0" ,"SCDictionary" );
```

All new Domino Web Access users will now get the new default for that setting in the preferences dialog.

Create a LotusScript agent to modify the preference for existing users

Example 5-18 is the LotusScript agent that can be run to modify the existing inotesprofile documents to enable instant messaging for all users.

Note: This agent can be added to any database on the server, like the names.nsf. It can be set to run from the actions menu or on a schedule. Remember to modify servername and mailpath before using this agent.

Example 5-18 Agent to enable instant messaging

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim doc As NotesDocument
    Dim Count As Integer
    Set db = session.CurrentDatabase
    Set doc = db.GetProfileDocument("inotesprofile")
    count = "1"
    doc.SametimeAwareness = count
    Call doc.save(False,False)
End Sub
```

An administrator can modify this setting for all existing mail files on one server by modifying the example below. This can be done using the methods and properties of the NotesDbDirectory class, which allows you to cycle through all of the databases on a particular server. Example 5-19 is a sample provided to illustrate one way to approach this issue.

Example 5-19 Agent to enable instant messaging on all mail files

```
Sub Initialize
    Dim db As NotesDatabase
    Dim acl As NotesACL
```

```

Dim entry As NotesACLEntry
Dim cprofile As NotesDocument
Dim pos As Integer
Dim found As Integer
Dim servername As String
Dim mailpath As String
Dim mailowner As String
Dim doc As notesdocument

'PARAMETERS TO CODE
'Indicate mail subdirectory name
mailpath = "mail\"
'Server name (in canonical format):
servername = "boston/ITS0bank"

Dim dbdir As New NotesDbDirectory(servername)
'Cycle through databases on the server
Set db = dbdir.GetFirstDatabase(DATABASE)
While Not db Is Nothing
'Skip databases which you don't have access to
On Error 4060 Goto Error4060
'Check to see if this database is in the mail directory
pos = Instr(db.FilePath, mailpath)
If pos = 1 Then
Call db.Open(servername, db.FilePath)

Set doc = db.GetProfileDocument("inotesprofile")
count = "1"
doc.SametimeAwareness = count
Call doc.save(False,False)

End If
GetNextDb:
Set db = dbdir.GetNextDatabase()
Wend
Exit Sub
Error4060:
'If the code reaches here then the user does not have access rights.
Resume GetNextDb

End Sub

```

Attention: The person executing this code must have manager rights to the mail files. If a single user does not have manager rights to all the mail files then the agent can be executed by an administrator with full access.



Administrative customizations

This chapter highlights some areas of Domino Web Access customization that can be performed with common administration techniques. This section include:

- ▶ Customizing Domino Web Access for different groups of users
- ▶ Key notes.ini variables
- ▶ Domino Web Access redirect

6.1 Different customizations for different audiences

Different groups of users in an organization may have different Domino Web Access customization needs. It is possible to meet the requirements of different groups by having each group access its own Domino server. This can be resource intensive so a second approach is to use different forms7 files and point them to different Domino Web Access mail templates.

6.1.1 How to use different forms databases

Forms7.nsf introduces a new NotesVar named s_FF that retrieves from the server the current forms file name. This is a significant improvement in Release 7 because there is no concern that the forms file can be replaced during the upgrade process. Example 6-1 demonstrates how to change the value of the \$FormsTemplateFile item within the icon note of both the mail template and the forms file to refer to a different name. This change will be automatically picked up by Domino Web Access.

Important: It is fundamental to use a forward slash (/) as the path separation character.

Example 6-1 Renaming the forms file name in the icon note

```
Sub Initialize
```

```
    Dim sess As New NotesSession
    Dim db As NotesDatabase
    Dim agent As NotesAgent
    Dim doc As NotesDocument
    Dim item As NotesItem
    Dim n As String

    Set db = sess.CurrentDatabase
    n = "FFFF0010"
    Set doc = db.GetDocumentByID(n)
    Set item = doc.ReplaceItemValue("$FormsTemplateFile", "iNotes/Custom7.nsf")
    Call doc.save(True,True)
```

```
End Sub
```

In addition, there are some NOTES.INI (see Table 6-1) settings to allow other forms files to be treated as valid by the server.

Table 6-1 NOTES.INI settings to use a custom forms file

INI Settings	Description
iNotes_WA_FormsFiles	Override default list of forms files supported on server. Example: iNotes_WA_FormsFiles=iNotes/Custom7.nsf,iNotes/Forms7.nsf,iNotes/Form6.nsf,iNotes/Forms5.nsf

INI Settings	Description
iNotes_WA_DefaultFormsFile	Override the default forms file to use when a DWA mail file specifies a forms file that does not exist on the server. Example: iNotes_WA_DefaultFormsFile=iNotes/Custom7.nsf

6.2 notes.ini variables affecting Domino Web Access

There are several notes.ini parameters that can be implemented on the Domino Web Access server to configure the Domino Web Access user environment. Most Domino Web Access notes.ini parameters came into being as a result of customer requests for the functionality provided. In this section we examine the notes.ini parameters.

Important: notes.ini parameters implemented on the server will not be pushed down to Domino Offline Services. The pre-packaged notes.ini for DOLS is under n_Dolbase.exe and needs to be modified to include the new notes.ini parameters. Also, the Version string under n_Dolbase.inf needs to be incremented. These files can be modified and packaged in a similar manner as discussed in 3.1.1, “Domino Web Access Customization in Domino 6.5” on page 18, for n_Shimmer7_en.exe and n_Shimmer7_en.inf.

There are a bunch of notes.ini parameters for Domino Web Access. Most of the parameters up to Version 6.5.4 are collected and documented in technote 1089521:

<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21089521>

6.2.1 Configuration settings versus notes.ini parameters

In the server configuration document for your Domino Web Access server you will find a tab labeled Domino Web Access. This tab is where the server administrator configures much of the Domino Web Access environment. Below, various Domino Web Access configuration options on this tab are discussed in terms of customization.

Some notes.ini settings used for configuring Sametime integration with Domino Web Access have been replaced by settings in the configuration settings document in Domino 7. To configure users with the DWA7 mail template, use the appropriate settings on the Domino Web Access tab of the configuration settings document instead of these variables.

Although you cannot use these notes.ini settings for Domino 7, they have not been removed and are still valid for users who have the iNotes6 mail template.

Important: In a mixed version environment with both iNotes6.ntf and DWA7.ntf mail users, the notes.ini setting will apply to iNotes6 users, but the corresponding configuration settings will override notes.ini settings for DWA7 users.

Table 6-2 notes.ini settings and corresponding configuration settings document fields

notes.ini setting	Configuration settings document field
iNotes_WA_Chat	Instant messaging features
iNotes_WA_LiveNames	Online awareness

notes.ini setting	Configuration settings document field
iNotes_WA_SametimeJavaConnect	Prefer Sametime Java Connect for browsers
iNotes_WA_NoLocalArchive	Local archiving
iNotes_WA_SametimeServer	Set an Instant Messaging server host name for all Domino Web Access users
iNotes_WA_SametimeToken	Allow secrets and tokens authentication
iNotes_WA_STLinksLocal	Loading \stlinks from Domino application server

6.2.2 Obsolete notes.ini variables in Domino Web Access 7

Every release of Notes and Domino contains a list of the obsolete INI settings that can and should be checked. Technote 1207338 describes notes.ini settings that are obsolete for Notes and Domino 7:

<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21207338>

6.2.3 New or modified notes.ini variables in Notes and Domino 7

With each release of Domino new notes.ini parameters are introduced allowing customers the flexibility to add and remove functionality. In a future release some of these ini settings may be integrated with the configuration document. The following are the new or modified notes.ini variables in Notes and Domino 7.

iNotes_WA_Areas

The value is a set of 6 bits, which can be 0 or 1. Each bit represents one functional area to display or not to display in the following way:

iNotes_WA_Areas=ABCDEF

Where:

- ▶ A: Welcome
- ▶ B: Mail
- ▶ C: Calendar
- ▶ D: ToDo
- ▶ E: Contacts
- ▶ F: Notebook
- ▶ 0: Disable Functional area
- ▶ 1: Enable Functional Area

For example, iNotes_WA_Areas=010010 would make only Mail and Contacts available.

iNotes_WA_NamePickerSearchAccentInsensitive

Defaults:

- ▶ R6.x: iNotes_WA_NamePickerSearchAccentInsensitive=0
- ▶ R7: iNotes_WA_NamePickerSearchAccentInsensitive=1

Values:

- ▶ iNotes_WA_NamePickerSearchAccentInsensitive=0 - Name Picker search becomes accent sensitive
- ▶ iNotes_WA_NamePickerSearchAccentInsensitive=1 - Name Picker search becomes accent insensitive

The default behavior of the Name Picker search is changed to accent (diacritical mark) insensitive in Release 7. In Release 6.5.5 and later, the default behavior is accent sensitive.

► **iNotes_WA_PrintUserStyleSheet**

The parameter `iNotes_WA_PrintUserStyleSheet` is used to specify a user-defined style sheet for printing. (For example, a specific Linux® environment where the display resolution is set to lower than the default, and print output would be larger than expected.) Specify the relative URL of the user-defined style sheet file (*.css) for printing under Domino's html directory. For example, if the file `print.css` is put under `data\domino\html`:

```
iNotes_WA_PrintUserStyleSheet=print.css
```

Important: Stats should not be on all of the time because they could affect performance. Also, making the cache size too small could affect performance.

iNotes_WA_Cache_Stats_On

Set to 1 to enable stats, but note that this will affect performance.

iNotes_WA_FormsCache_Size

Set the max number of entries in the orms cache (default 256).

iNotes_WA_SubformsCache_Size

Set the max number of entries in the subforms cache (default 256).

iNotes_WA_SkinGroupsCache_Size

Set the max number of entries in the skin groups cache (default 256)

iNotes_WA_QueryAgents

Set to "1" to enable the ability to customize Domino Web Access to invoke Web Query Open or Web Query Save agents.

iNotes_WA_debug

Setting this variable is very useful for JavaScript error testing and debugging. Instead of the standard error message `Sorry we are unable to process....`, a call stack of the current JavaScript on the server side is displayed. This helps diagnose where a particular problem resides. The variable is documented in technote 1164431:

<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21164431>

iNotes_WA_DefaultFormatPlainText

When this is set to 1, two things happen:

- (For new users) the default mail format preference is set to plain text.
- When the mail format preference is set to prompt me when sending, the default format on the prompt UI is set to plain text.

iNotes_WA_ReadAttachments

When set to 0, Domino Web Access users will not be able to read attachments. When set to 1, Domino Web Access users will be able to read attachments. The default is 1. There is also functionality in Domino 7 to disable the ability to read attachments via an argument on the URL. If `&ra=0` is placed on the URL, the Domino Web Access user will not be able to read

attachments. The URL argument might be used to block particular users via a reverse proxy rule or custom Web page.

iNotes_WA_WriteAttachments

When set to 0, Domino Web Access users will not be able to write attachments. When set to 1, Domino Web Access users will be able to write attachments. The default is 1. There is also functionality in Domino 7 to disable the ability to write attachments via an argument on the URL. If &wa=0 is placed on the URL, the Domino Web Access user will not be able to write attachments.

iNotes_WA_ReuseChildWindows=1

When this feature is set, Domino Web Access caches and then reuses portions of the Mail and Calendar forms that do not change. For example, when a user opens a message, the Read Message window opens. The Read Message form is also cached without the message text. The form is then used to display any additional messages that the user opens in the same window. This eliminates the time it takes to close the first window and open a subsequent message in a new one.

iNotes_WA_DisableReuseChildWindows

When this notes.ini value is set to 1 it turns off the Reuse Child Windows UI option and disables the feature globally.

6.3 Domino Web Access redirect personal options

Using the Domino Web Access redirect database adds value in terms of customization. The out-of-the-box database already supports some administrative features to offer personal options or specific startup customizations to Domino Web Access users.

6.3.1 Configuration document options

The Domino Server settings for the default page that is loaded for a Domino Web Access user is set in the configuration document. This setting can be changed per server and also in the default configuration document. For example, the default template leads the user to the Welcome page of Domino Web Access, as shown in Figure 6-1.

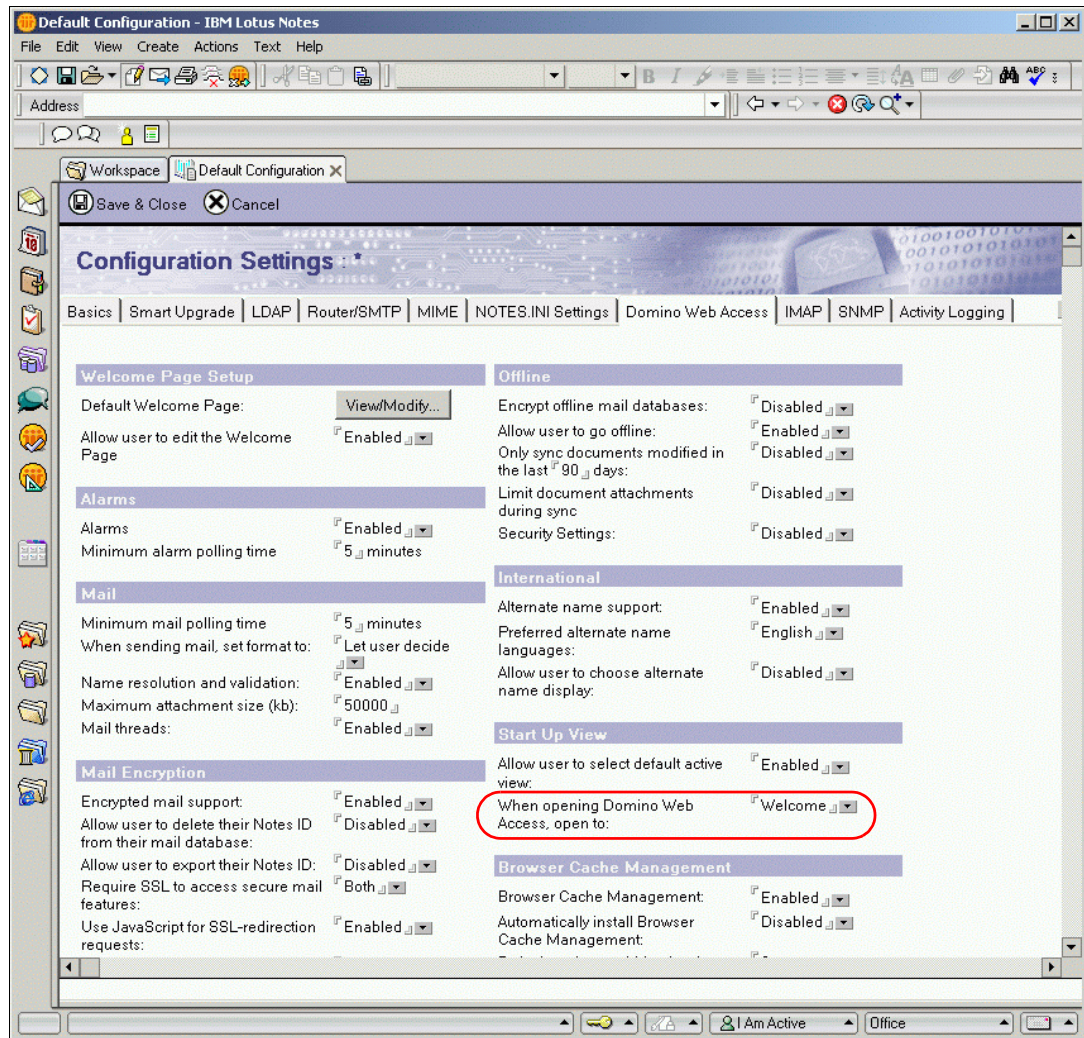


Figure 6-1 Configuration settings document

This default can be changed by the administrator in the configuration document to point all users to a different page. The options available are shown in Figure 6-2.

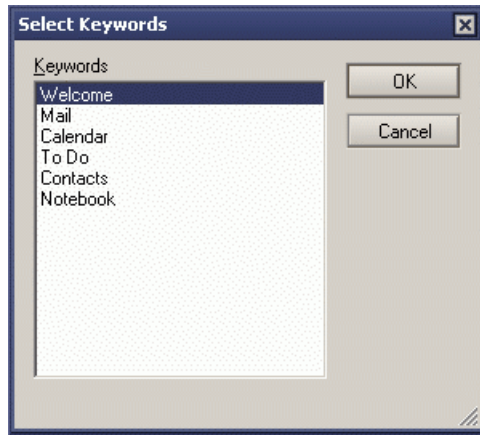


Figure 6-2 Startup screen options in configuration document

6.3.2 DWA redirect options

The administration of the Domino Web Access redirect database allows a customer to set additional options to customize what users can select and how the redirection is displayed. Additionally, Figure 6-3 shows the Domino Web Access Redirect configuration screen with the following options set:

- ▶ Corporate logo set to the ITSO Bank banner
- ▶ Background color set to the corporate standard
- ▶ Enable the personal options for the user being redirected

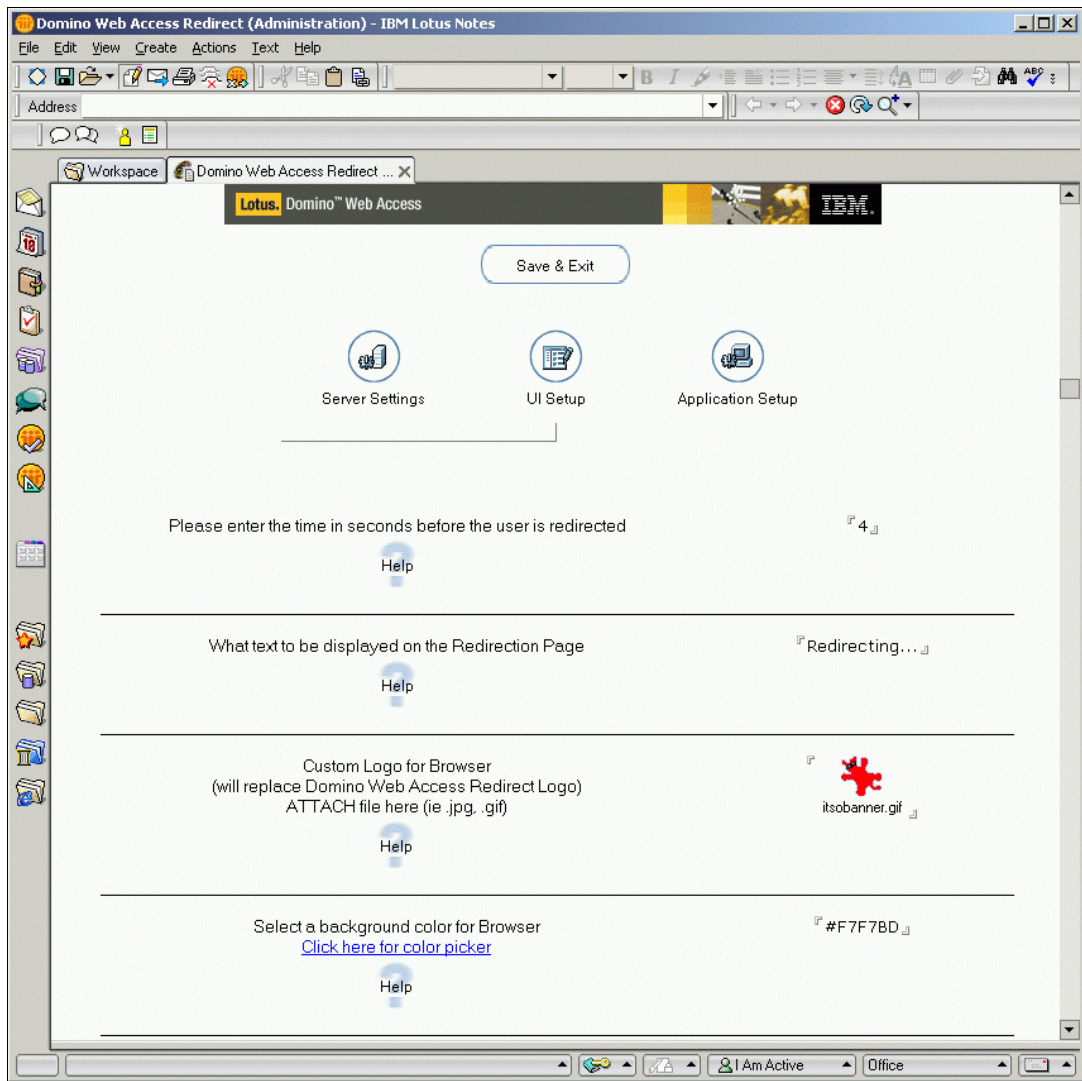


Figure 6-3 Redirect options administration

The Domino Web Access redirect page then displays an additional button for the end user called *Personal Options*, as shown in Figure 6-4.



Figure 6-4 Personal options button in Domino Web Access redirect

Clicking the Personal Options button leads the user to a page where the start page can be selected from a set of predefined options like those in Figure 6-5.

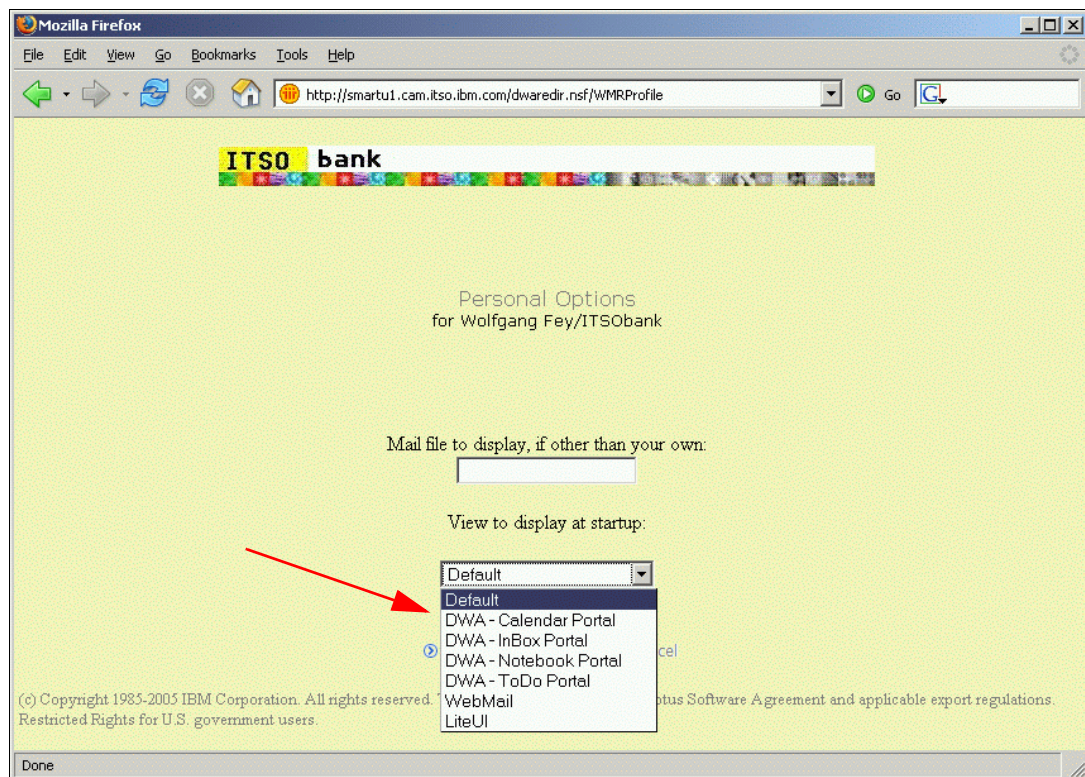


Figure 6-5 Personal options selection window

The default list can also be customized to display different target page layouts or even customized Domino Web Access pages. The place to put the option values and links for the

redirections is the form *WMRProfile*. The options are set in the combobox field called *WMRProfileURL*. This field's values can be extended, modified, or deleted as needed. For an example see Figure 6-6.

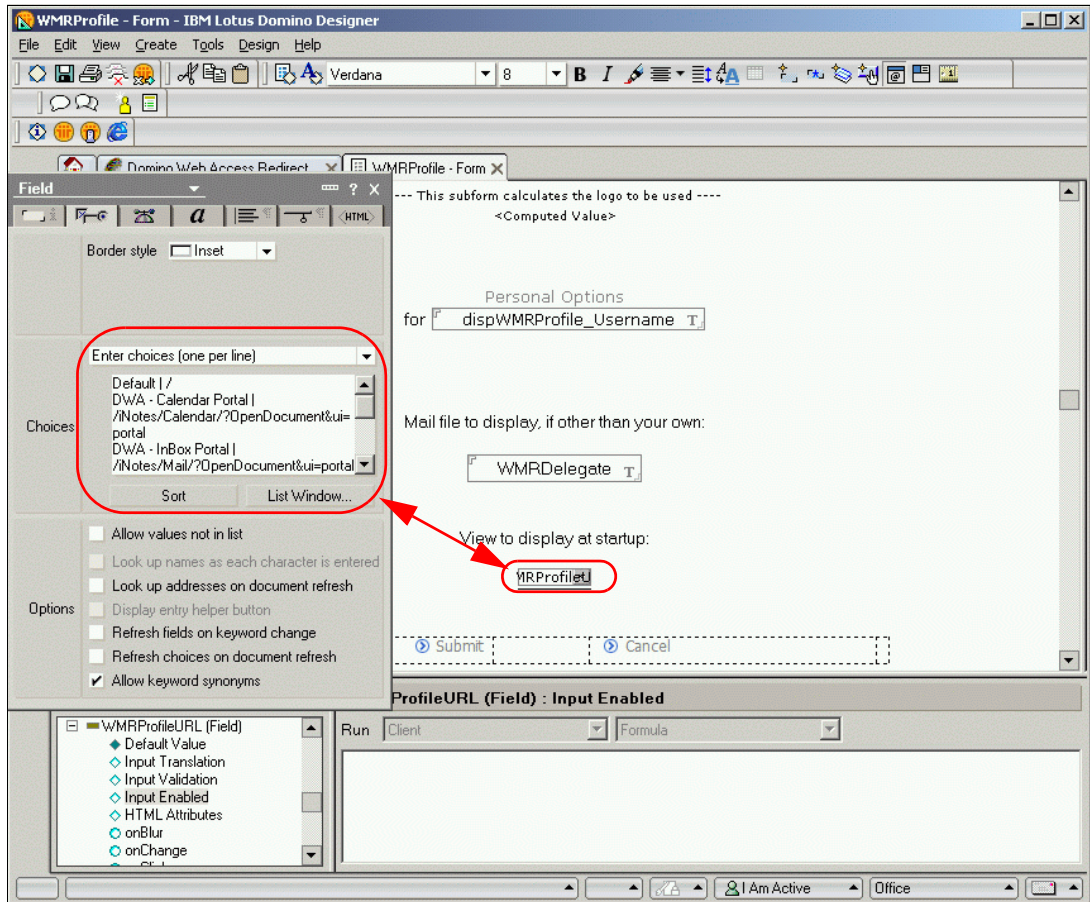
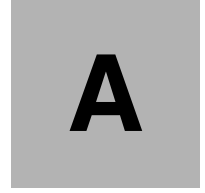


Figure 6-6 *WMRProfile* form with options for *WMRProfileURL*



Additional customization tips

Creating a custom help document in the Domino Web Access help database

The content of all Help pages is stored as a file resource to the iNotes/help70_iwa_en.nsf database. If a new page or dialog is being added or an existing page is modified it might be useful to also customize the help document or to create an additional one. The name of the HTML file has to be put into the IF variable in the page init context, as shown in "Subform s_AccountEdit_Init" on page 82. An easy way to customize or add a help page it is to start with an existing help HTML file in the help database.

Different techniques to include externalized code

There are at least two ways to include the externalized code content of forms or subforms in other forms or subforms to keep them as small as possible.

One option is to use the <includesubform> special DWA tag. This includes all of the HTML and JavaScript code from the subform named in the tag at the location where the tag is placed. This is done by the http server and the result is delivered to the browser.

The other option is to move the code to an external script file by utilizing an HTML script tag with a src value referencing another Domino Web Access form that is set up to contain JavaScript content (\$ContentType NOTESVAR set to "application/x-javascript"). This results in the browser loading the external page when it encounters the tag. The script tag would take the form:

```
<script src="@{s_FF}/iNotes/Proxy/?OpenDocument&Form=s_SomeForm@{s_StaticJSArgs}">
```

The use of a script tag to load an external page (part) is so all the script code is not emitted as part of a top-level page. This top-level page may not be cacheable, and hence every time it is invoked all scripts are also included, increasing bandwidth consumption and slowing down performance of that page. Placing it in an external page part and setting that part to be cacheable avoids increasing the bandwidth consumed on subsequent re-visits to the top page (or similar pages) since the external script file will now be in the browser cache and can be potentially reused. If the external script part is a data-related script and not static code, it cannot be cached, and there is no benefit to having it in an external script file.

If the external script is really a data-related script (contains formulas that evaluate to different values at different times or for different users), unless it will remain static for some period of time and hence can be set up to be cached. There is limited benefit to having it be an external file.

Tools helpful for Domino Web Access Development

The sections below reference helpful tools for Domino Web Access customization.

NotesPeek

In case you want to dig into the documents in forms7.nsf or dwa7.nsf that are not viewable with the Notes client you should consider NotesPeek.

NotesPeek was written for technical Domino administrators who want to see more of a raw view of their Notes databases than the Notes client provides. It presents the information in Notes databases as it is available through the Notes API.

NotesPeek is based on a containment hierarchy: Notes servers have databases, databases have documents, documents have items, and so on. For the most part, this containment hierarchy is that presented by the API. Selecting a line in the tree view on the left displays information about that line in the text view on the right.

NotesPeek displays almost all of the data available through the Notes API. Where data cannot be interpreted because its format is not documented, it is displayed as a hex dump.

NotesPeek is provided *as is* by IBM and can be downloaded at:

<http://www-10.lotus.com/1dd/sandbox.nsf/0/2791869f4e1d3fa385256f2c00432973?OpenDocument>

Figure A-1 shows NotesPeek to display the content of a profile document.

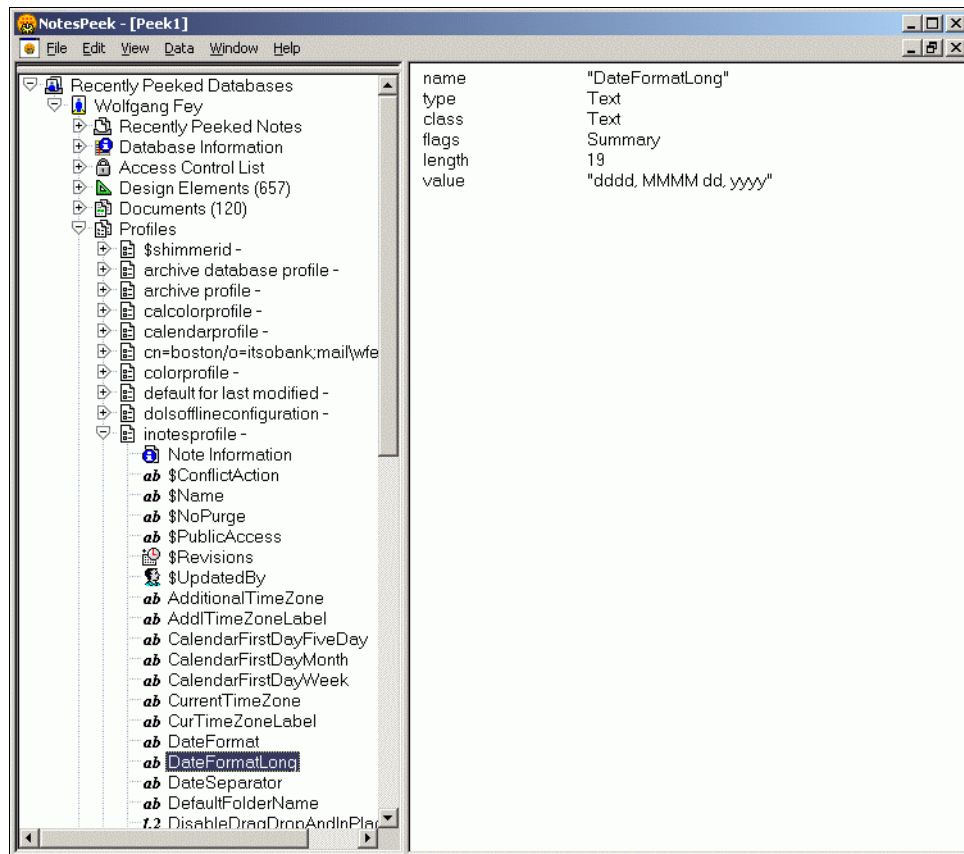


Figure A-1 NotesPeek showing an inotesprofile

Firefox extensions and tools

In this section we discuss Firefox extensions and tools.

DOM Inspector

The DOM Inspector comes with the Firefox distribution and is installed as part of Firefox if the Developer Tools option is selected in the custom installation dialog box. If installed, the DOM Inspector can be called at any time in the browser by pressing Shift+Ctrl+I. Figure A-2 shows the DOM Inspector with a Domino Web Access page displaying the table of contents (TOC) elements of the page.

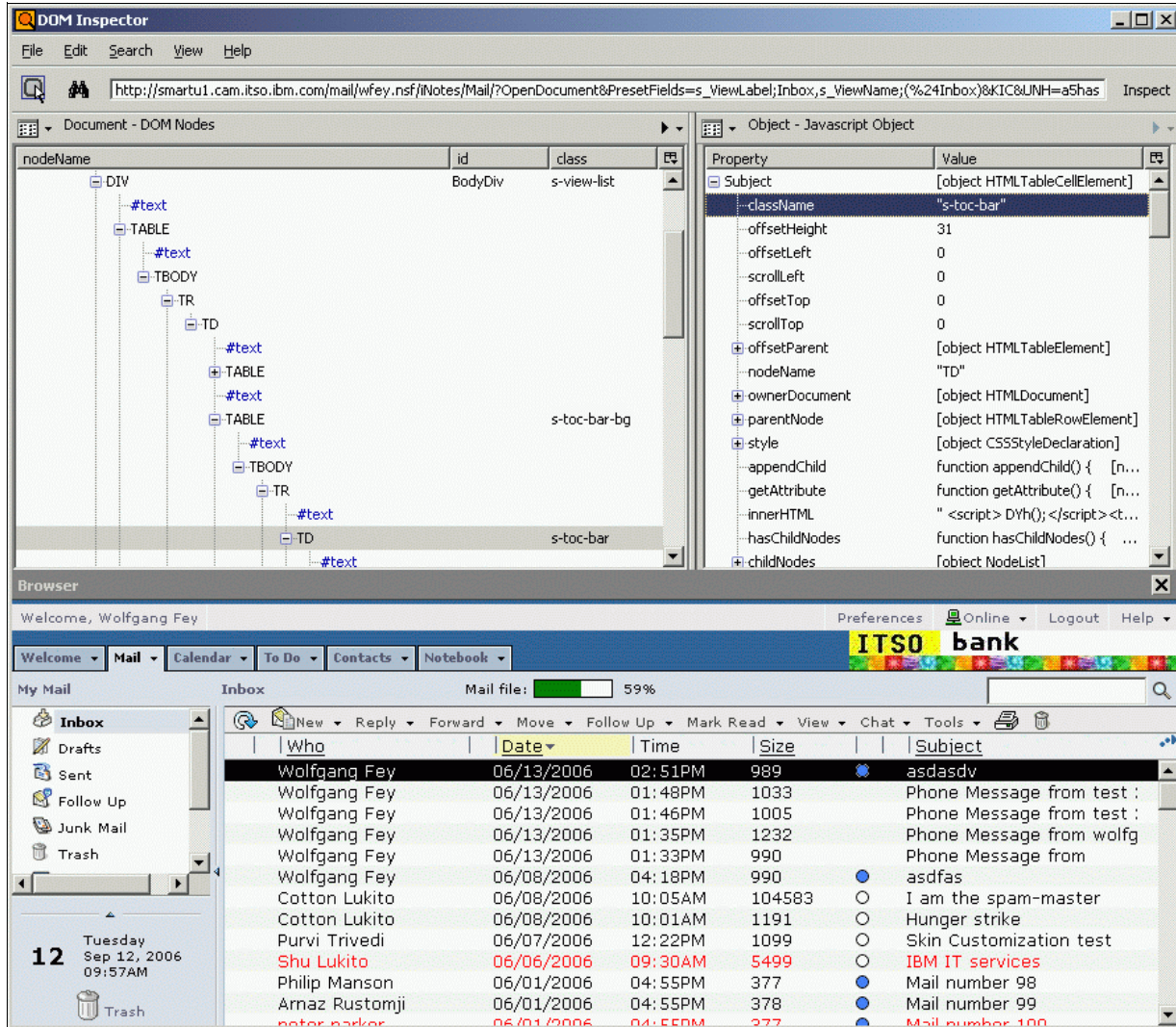


Figure A-2 DOM Inspector displaying a Domino Web Access page

Web Developer

Web Developer helps developers with many very helpful tools included in the toolbar. The download address is:

<https://addons.mozilla.org/firefox/60/>

After installation, the Web Developer toolbar is displayed after the next restart of the Firefox browser. The toolbar is shown in Figure A-3.

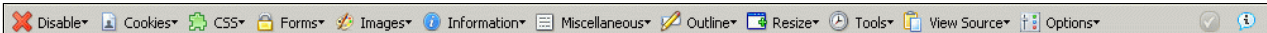


Figure A-3 Web Developer toolbar

For example, a developer can dig into the page structure in terms of outlining tags, editing the CSS code live, so any update displays immediately without affecting the real site online. Figure A-4 shows Web Developer outlining table cells.

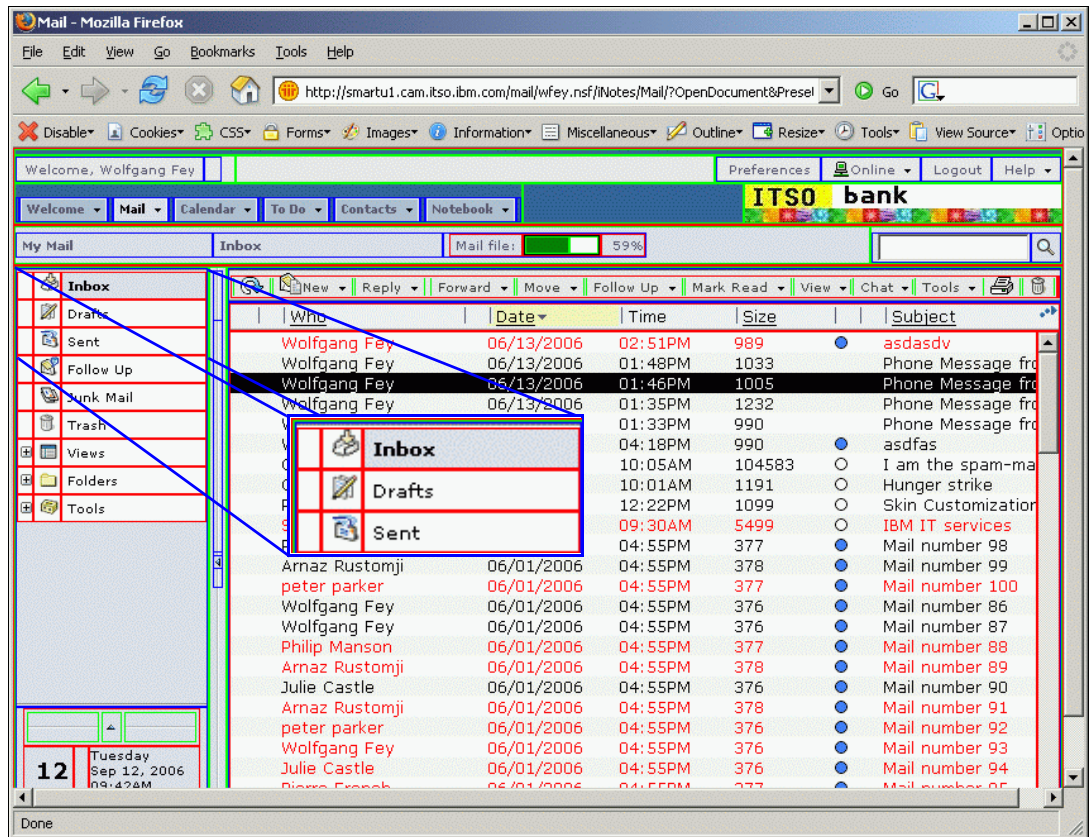


Figure A-4 Web Developer outlining table cells

Another very useful example is the in-place edit of the cascading stylesheet. Modifying a particular style immediately affects the page display. The css is not written back to the server but can be saved locally. Figure A-5 shows an example of the css editor.

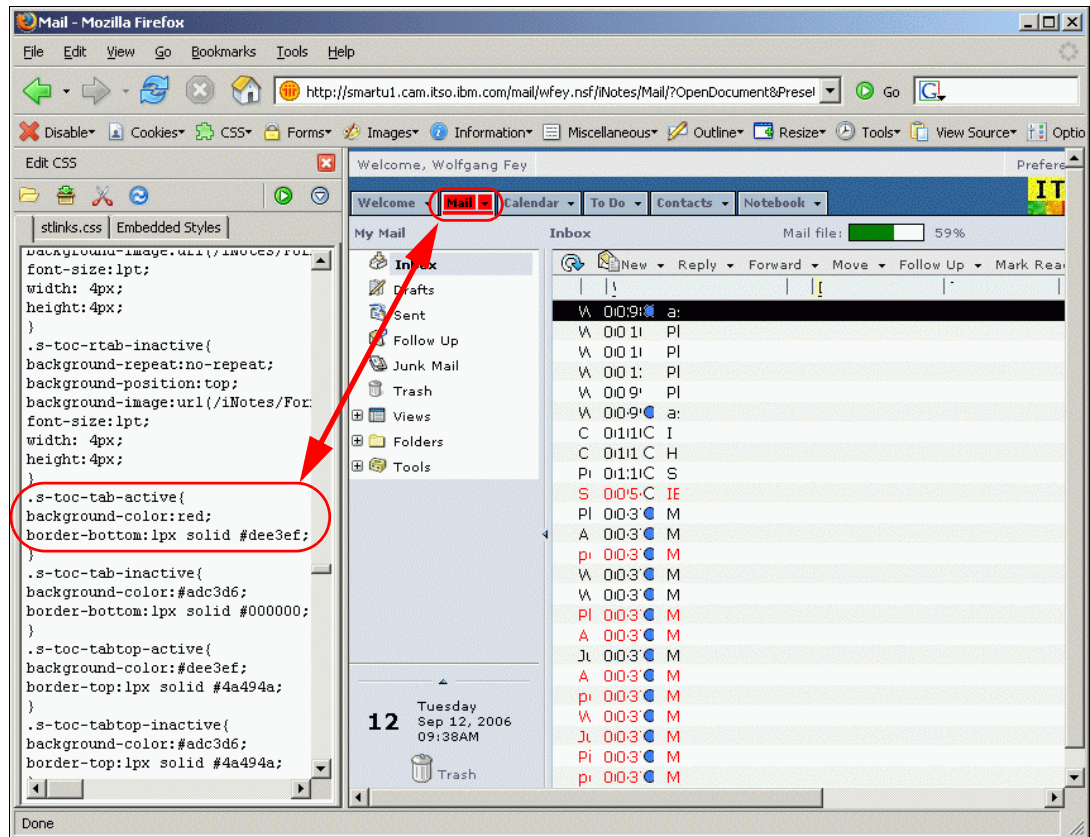


Figure A-5 Web Developer CSS editing sidebar

View formatted source

This tool hooks into the Tools menu of Firefox and also into the right-click menu. Figure A-6 shows the Tools menu. It can be launched from almost every Web page displayed. The source code is displayed in a color-coded formatted view and also displays the css information for each tag. The source code can be folded and expanded with plus signs (+) and minus signs (-) before each parenthesized section like table, tr, td, div, span, and so on.

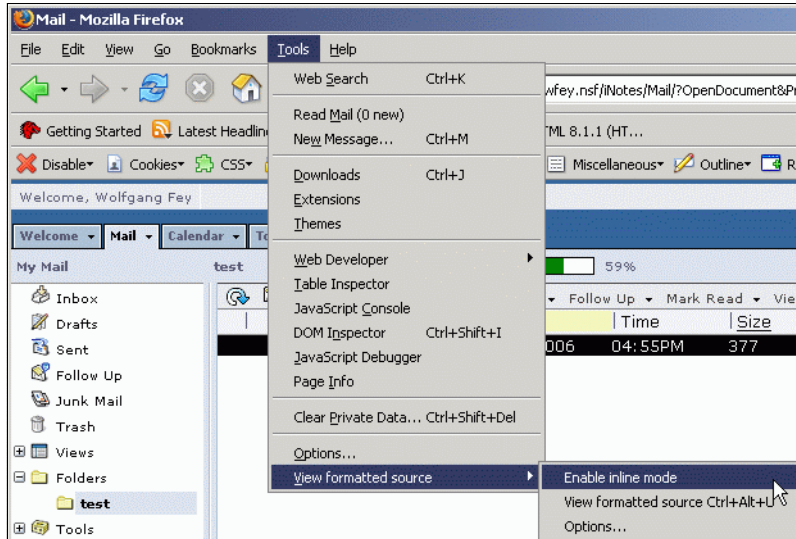


Figure A-6 View Formatted source menu

Also, there is an option to display colored inline selection frames for specific areas of the page. These can be clicked and will display only the source for that particular section of the page. The source code displayed is based on the rendered document so every dynamically modified element, either by DOM or JavaScript, is displayed in the way the browser renders the data.

The tool can be downloaded at:

<https://addons.mozilla.org/firefox/697/>

After installation Firefox needs to be restarted to enable the extension.

Figure A-7 shows the tool in the inline mode displaying the source code of the mail file quota usage bar.

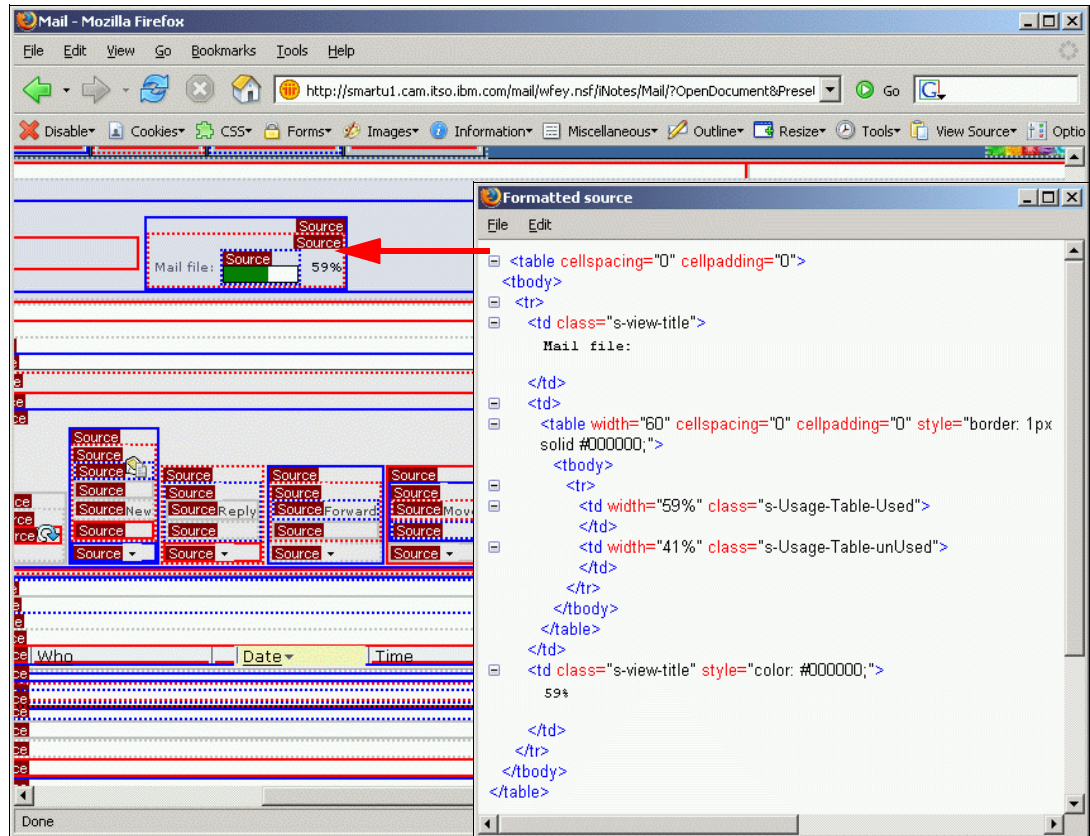


Figure A-7 View source inline mode

View source Chart

This tool produces a colored chart of the currently displayed Web page, using the rendered code, so all includes like JavaScript and DHTML modifications are integrated. The tool can be downloaded at:

<https://addons.mozilla.org/firefox/655/>

Figure A-8 shows a sample output of View Source Chart charting the Domino Web Access mail page. The tool does not have as many other extensions are under the tools menu. It is integrated into the view menu of the Firefox browser.

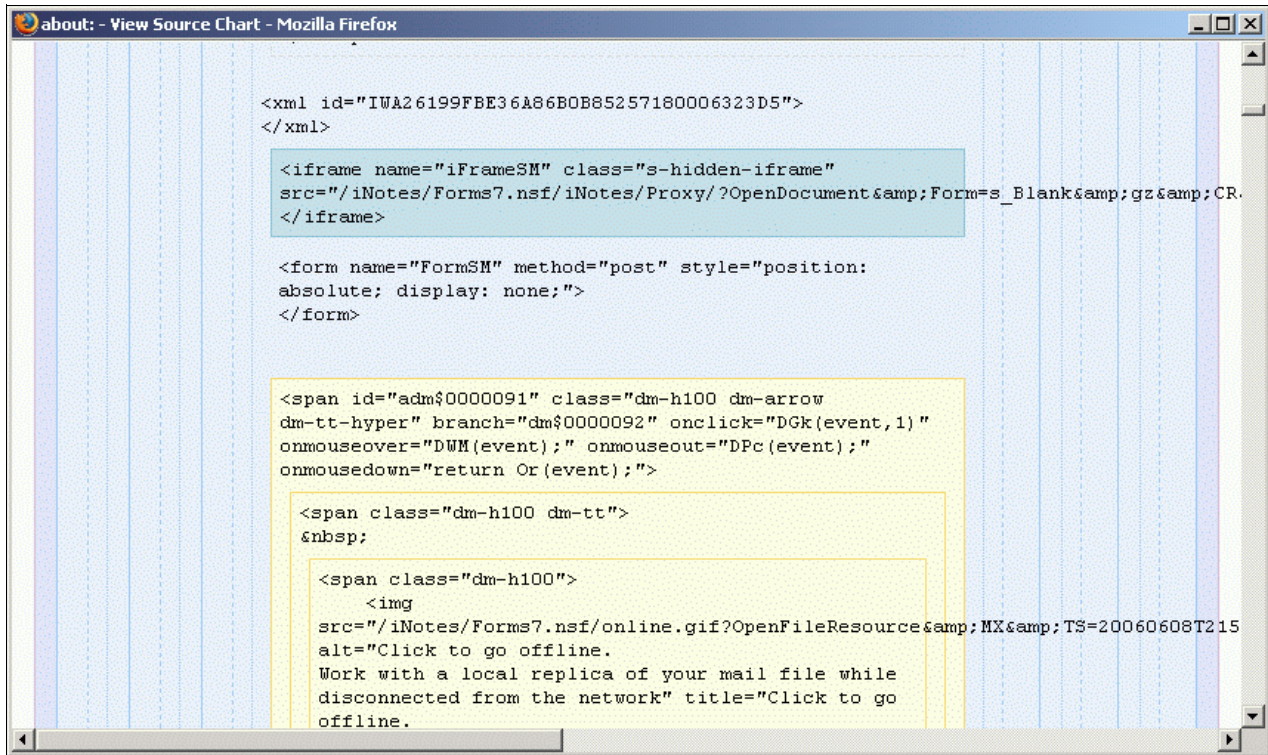


Figure A-8 Sample view source chart output

JavaScript debugger

The Firefox/Mozilla JavaScript debugger is better known by its codename Venkman. It provides a powerful debugging environment for JavaScript code within Mozilla and Firefox. The debugger can be downloaded at:

<https://addons.mozilla.org/firefox/216/>

Internet Explorer tools

Here we discuss Internet Explorer tools.

Internet Explorer Developer Toolbar

At the time of this writing a preview version of the Microsoft IE Developer toolbar is available. The Microsoft Internet Explorer Developer Toolbar provides a variety of tools for quickly creating, understanding, and troubleshooting Web pages.

The download of the Microsoft's IE Developer toolbar is available at:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=e59c3964-672d-4511-bb3e-2d5e1db91038&displaylang=en>

Figure A-9 shows the new toolbar in Internet Explorer.



Figure A-9 Internet Explorer Developer Toolbar

Note: This tool was still a beta version at the time this Redpaper was written.

The DOM Inspector of the IE developer Toolbar is shown in Figure A-10. Note that this tool is a read-only tool, so you can read all objects and properties, but cannot modify them.

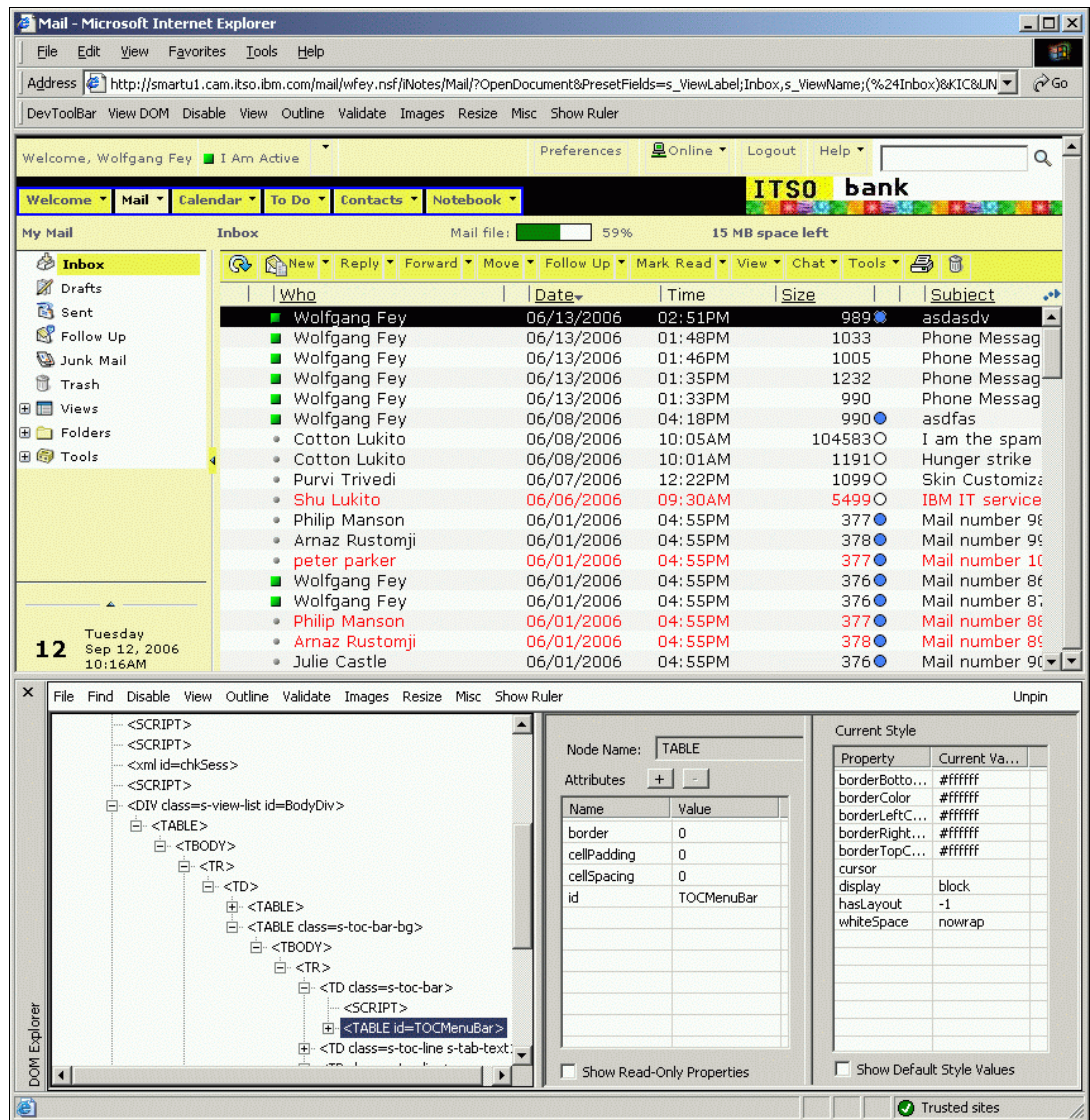


Figure A-10 Internet Explorer DOM Inspector

Documentation

In this section we review documentation.

HTML, CSS, JavaScript, DOM, and DHTML

Developers should be very familiar with JavaScript, CSS, HTML, DHTML, and the DOM implementation of the targeted browser. Some links are provided for your convenience regarding these topics:

- ▶ JavaScript documentation
 - Microsoft MSDN® JScript® Language Reference
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/29f83a2c-48c5-49e2-9ae0-7371d2cda2ff.asp>
 - IE JScript objects
http://www.irt.org/xref/jscript_objects.htm
 - SUN Microsystems Developer Connection JavaScript Resource Center
<http://java.sun.com/javascript/index.jsp>
- ▶ HTML
 - W3C HTML home page
<http://www.w3.org/Markup/>
 - Microsoft MSDN article “HTML and Dynamic HTML Reference”
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml_node_entry.asp
- ▶ DOM
 - W3C Document Object Model (DOM)
<http://www.w3.org/DOM/>
 - Microsoft MSDN article “About the DHTML Object Model”
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml_node_entry.asp
 - Mozilla DOM Reference
http://developer.mozilla.org/en/docs/Gecko_DOM_Reference
- ▶ Cascading Stylesheets (CSS)
 - W3C CSS home page
<http://www.w3.org/Style/CSS/>

Additional tools

In this section we discuss additional tools.

LDAP Browser

If you need to analyze any LDAP issues there is a very good tool called *LDAP Browser* from Softerra, Ltd. that you can use to examine the LDAP environment, LDAP server responses, and so on at a very comfortable level. This can help a lot to get any issues regarding LDAP with Single-Sign On, Sametime integration, Domino Assistance with LDAP, and so on. The product is free of charge and can be downloaded at:

<http://www.ldapbrowser.com/>

CSS Stylesheet Editor

Topstyle from NewsGator Technologies, Inc. is a stylesheet editing tool. It can be downloaded from:

<http://www.newsgator.com/NGOLProduct.aspx?ProdID=TopStyle>

Wrapper functions for action menu operations

As demonstrated by examples provided in Chapter 5, “Feature customization techniques” on page 65, you can see the improvement in action menu operations with Version 7.0.1, as compared to Version 6.5. In this appendix, we have provided you with useful JavaScript wrapper functions to make it easier to manipulate action menus in Version 6.5.

We also provide the comparable functions for Version 7.0.1 to provide you with a more powerful way to work with action menus without having to be concerned with the details of the implementation of the DM functions discussed in 5.2, “Domino Web Access functions for Action menu operations” on page 67. Additionally, these wrapper functions enable you to build an array to add, overwrite, or remove multiple action menus easily.

removeActions function

Here the removeAction function takes two arguments. The first argument is the array passed in as the third argument to Scene_Actions (a_Actions for forms6.nsf and s_TopBranchId for forms7.nsf). The second argument is an array consisting of positions or titles identifying menus to remove.

Table A-1 shows details of the expected properties in this passed in array.

Table A-1 Details of properties passed as parameter array to removeActions function

Parameter	Description
title	Visible text of action entry to remove.
pos	Position number of action entry to remove (only way to remove image-only actions). There is also a check Positions routine below to modify DWA to have it display all the top-level position values.
suboffsets	An array of submenu offsets to be removed. Each entry is a zero-based offset. If such an array is present, then the top-level menu is not removed. (You should not use both this and subtitles within the same object.)
subtitles	An array of submenu visible names to be removed. If such an array is present, then the top-level menu is not removed. (You should not specify both suboffsets and subtitles.)

Removing action menu in Version 6.5

Example A-1 shows the wrapper and helper functions for removing an action menu item in Version 6.5 and the forms6.nsf database.

Example: A-1 Remove action menu wrapper and helper functions in Version 6.5

```
// Helper functions:  
// Routine to add visible position numbers to displayed menu titles. This will  
allow to easily
```

```

// identify positions that you desire to remove. Particularly helpful for
non-textual actions
function checkPositions(a_Actions) {
    var a=[], n=0;
    for(var i=0;i<a_Actions.length;i++) {
        if( typeof a_Actions[i].title == "string" )
            a_Actions[i].title = a_Actions[i].pos + ':' + a_Actions[i].title;
        else
            a_Actions[i].title[0] = a_Actions[i].pos + ':' +a_Actions[i].title[0];
    }
}

// Check the passed in array to see if it applies to each action being processed
function checkRemoval(a, action) {
    for(var i=0; i<a.length; i++) {
        if( typeof a[i].pos != "undefined" ) {
            if( a[i].pos == action.pos )
                return i;
        } else if( a[i].title ) {
            if( typeof action.title=="string" ) {
                if (a[i].title == action.title )
                    return i;
            } else if (a[i].title == action.title[0])
                return i;
        }
    }
    return -1;
}

// Routine to remove subentries based on offsets or titles
function removeSubEntries(action,aSubEntries) {
    var aSubMenu=action.href;
    var aNewMenu=[], n=0;
    var bIsPos = typeof aSubEntries[0] == "number";
    var nEntries=aSubMenu.length/2;
    var j=0;
    for(var i=0;i<nEntries;i++) {
        if(j<aSubEntries.length && ( (bIsPos && i==aSubEntries[j]) || (!bIsPos &&
aSubMenu[i*2].split("|")[0]==aSubEntries[j])) )
            j++;
        else {
            aNewMenu[n++]=aSubMenu[i*2];
            aNewMenu[n++]=aSubMenu[(i*2)+1];
        }
    }
    action.href=aNewMenu;
}

/* The first argument is the array passed in as the 3rd argument to Scene_Actions
* The second argument is an array of numbers or titles indicating top level
actions to remove
* See ActionPositions.txt for actions associated with various DWA actions
* Sample removal array by Position: [
{pos:0,suboffsets:null},{pos:50,suboffsets:[0,1,2,3,4]}}];

```

```

* Sample removal array by Title: [{title:"New", subtitles:null},{title:"Tools",
subtitles:["Out of Office",...]}]
*/
function removeActions(a_Actions,a_ToRemove) {
    var nPos;
    var aRemove=[], n=0;
    var bIsSubMenu = false;
    for(var i=0;i<a_Actions.length;i++) {
        nPos = checkRemoval(a_ToRemove,a_Actions[i]);
        if( nPos != -1 ) {
            bIsSubMenu = typeof(a_Actions[i].href) == "object";
            if(a_ToRemove[nPos].suboffsets && bIsSubMenu) {
                removeSubEntries(a_Actions[i],a_ToRemove[nPos].suboffsets);
            } else if(a_ToRemove[nPos].subtitles && bIsSubMenu) {
                removeSubEntries(a_Actions[i],a_ToRemove[nPos].subtitles);
            } else
                aRemove[n++] = a_Actions[i];
        }
    }
    if(aRemove.length) {
        // BOY == removeItems
        a_Actions.BOY(aRemove,true);
    }
}

```

Removing action by position

The wrapper functions allow you to remove the action menu either by menu label or by position. In some cases, you will not be able to remove the action menu by label. Examples for such cases are non-textual actions like refresh, print, and delete.

The checkPositions function in Example A-1 on page 116 is very useful when you want to remove an action menu by position instead of by menu label. It displays the position next to the menu label, as depicted in Figure A-11.

Who	Date	Time	Size	Subject
Mail Router	06/07/2006	06:15PM	2529	DELIVERY FAILURE
Shu Lukito	03/29/2006	09:51AM	1321	test follow up
Shu Lukito	09/27/2005	08:02PM	1614	test 4

Figure A-11 Action menus with positions displayed in Version 6.5

Example A-2 provides an example of removing an action menu by position.

Example: A-2 Remove menu by position in Version 6.5

```

function Scene_Actions( s_SceneName, o_Window, a_Actions ){
...

    switch (s_SceneName) {
        case 'Mail':
            checkPositions(a_Actions);

            // To remove Tools -> Block Mail from Sender action menu by position
            var a=[{pos:330,suboffsets:[3]}];
            removeActions(a_Actions, a);
    }
}

```



```

// To remove the New top level menu by position value
removeActions(a_Actions, [{pos:0}]);

// To remove the 3-5th submenus within a top level menu named New
var a=[{title:"New", suboffsets:[2,3,4]}];
removeActions(a_Actions, a);

break;

default:
break;
}
...
}

```

Removing action menu by label

Example A-3 is an example of removing an action menu by label.

Example: A-3 Remove menu by label in Version 6.5

```

function Scene_Actions( s_SceneName, o_Window, a_Actions ){
...

switch (s_SceneName) {
case 'Mail':
// To remove Tools -> Block Mail from Sender action menu using label
var a=[{title:"Tools",subtitles:["Block Mail from Sender"]}];
removeActions(a_Actions, a);

// To remove a top level (and any submenus it contains)
removeActions(a_Actions, [{title:"New"}]);

// To remove a "Message" submenu within a "New" top level menu
removeActions(a_Actions, [{title:"New", subtitles: ["Message"]}]);

// To remove a "Message" and "Contact" submenus
// within a "New" top level menu
var a=[{title:"New", subtitles: ["Message", "Contact"]}];
removeActions(a_Actions, a);

break;

default:
break;
}
...
}

```

Removing action menu in Version 7.0.1

Example A-4 shows the comparable wrapper and helper functions for removing an action menu item in Version 7.0.1 and the forms7.nsf database.

Example: A-4 Remove action menu wrapper functions in Version 7.0.1

```
// Routine to add visible position numbers to displayed menu titles.
// This will allow to easily identify positions that you desire to remove.
// Particularly helpful for non-textual actions
// Also provides a way to display menuids by passing true to the second argument.
function checkPositions(branchId, bDisplayIds, bDisplayPos) {
    if(arguments.length < 3)
        bDisplayPos = true;
    if(arguments.length < 2)
        bDisplayIds = false;
    var a_Actions = DZo(branchId); /* DM_getBranch */
    var menuNode, aSubMenu;
    var i,j;
    for(i=0;i<a_Actions.length;i++) {
        if( typeof a_Actions[i] == "string" )
            menuNode = DPN(a_Actions[i])
        else
            menuNode = a_Actions[i];
        menuNode.label = (bDisplayPos && typeof menuNode.pos == "number" ?
menuNode.pos : '') +
            (bDisplayIds ? ('[' + menuNode.$id + ']') : '') + ':' + menuNode.label;
        if(menuNode.$branchId && bDisplayIds) {
            checkPositions(menuNode.$branchId, bDisplayIds, false);
        }
    }
}

// Routine to remove subentries based on offsets or titles
function removeSubEntries(action,aSubEntries) {
    var i;
    var menuNode;
    if( typeof aSubEntries[0] == "number" ) {
        var aSubMenu= DZo(action.$branchId); /* DM_getBranch */
        var nEntries=aSubMenu.length;
        var j=0;
        var aRemoval = [], n=0;

        for(i=0;i<nEntries;i++) {
            if(j<aSubEntries.length && i==aSubEntries[j]) {
                aRemoval[n++] = aSubMenu[i];
                j++;
            }
        }
        for (i=0;i<aRemoval.length;i++)
            DGa(aRemoval[i]);
    }
    else {
        // Find by title
        for(i=0; i<aSubEntries.length; i++) {
            menuNode = CyE(action.$branchId, aSubEntries[i]);
            if(menuNode)
```

```

        DGa(menuNode.$id);
    }
}

/* The first argument is the branch id passed in
 * as the 3rd argument to Scene_Actions
 * The second argument is an array consisting of positions
 * or titles identifying menus to remove
 * Sample removal array by Position:
 * [{pos:0,suboffsets:null},{pos:50,suboffsets:[0,1,2,3,4]}];
 * Sample removal array by Title:
 * [{title:"New", subtitles:null},{title:"Tools", subtitles:["Out of
Office",...]}]
 * The array can contain a mixture of these techniques if desired.
 */
function removeActions(branchId,a_ToRemove) {
    var action;
    var bIsSubMenu = false;
    for(var i=0;i<a_ToRemove.length;i++) {
        if(typeof a_ToRemove[i].pos != "undefined")
            action = Dap(branchId, a_ToRemove[i].pos);
        else if (a_ToRemove[i].title)
            action = CyE(branchId, a_ToRemove[i].title);
        if(action) {
            bIsSubMenu = action.$branchId ? true : false;
            if(a_ToRemove[i].suboffsets && bIsSubMenu) {
                removeSubEntries(action,a_ToRemove[i].suboffsets);
            } else if(a_ToRemove[i].subtitles && bIsSubMenu) {
                removeSubEntries(action,a_ToRemove[i].subtitles);
            }
        }
        else
            DGa(action.$id);
    }
}
}
}

```

Removing action by position

As discussed earlier, the wrapper functions allow you to remove the action menu either by menu label or by position. In order to remove non-textual action like refresh, print, and delete, you will need to remove the action by position.

The checkPositions function is useful when you want to remove an action menu by position instead of by menu label. If you set the second argument bDisplayIds to true, it will display the IDs (for example, dm\$0000006). Similarly, setting the third argument bDisplayPos to true will display the position (for example, position 1 for New). See Figure A-12 for a visual representation after running this function.

Who	Action	Size	Subject
Shu Lukito	[dm\$0000008]:Message	PM	test 23
Shu Lukito	[dm\$0000009]:Phone Message	PM	test 22
Shu Lukito	[dm\$0000010]:Calendar Entry	PM	test 21
Shu Lukito	[dm\$0000011]:To Do	AM	test 20
Shu Lukito	[dm\$0000012]:Contact	AM	test 19
Shu Lukito	[dm\$0000013]:E-mail Group	AM	test 18
Shu Lukito	[dm\$0000014]:Notebook Page	AM	test 17
Shu Lukito	[dm\$0000016]:Folder	AM	

Figure A-12 Action menus with positions and ID displayed in Version 7.0.1

Example A-5 is an example of removing an action menu by position.

Example: A-5 Remove action by position in Version 7.0.1

```
function Scene_Actions( s_SceneName, o_Window, a_Actions ){
...
    switch(s_SceneName) {
        case 'Mail':
            checkPositions(s_TopBranchId, true, true);

            // Remove the "Tools\Block Mail from Sender" action using position
            removeActions(s_TopBranchId, [{pos:330, suboffsets:[5]}]);

            // To remove a top level menu with a particular position value
            removeActions(s_TopBranchId, [{pos:1}]);

            // To remove the 3-5th submenus within a top level menu named New
            var a=[{title:"New", suboffsets:[2,3,4]}];
            removeActions(s_TopBranchId, a);

        default:
            break;
    }
...
}
```

Removing action by label

Example A-6 is an example of removing an action menu by label.

Example: A-6 Remove action by label in Version 7.0.1

```
function Scene_Actions( s_SceneName, o_Window, a_Actions ){
...
    switch(s_SceneName) {
        case 'Mail':
            // Remove the "Tools\Block Mail from Sender" action using label
```

```

removeActions(s_TopBranchId, [{title:"Tools", subtitles:["Block Mail from
Sender"]}]);

// To remove a top level (and any submenus it contains)
removeActions(s_TopBranchId, [{title:"New"}]);

// To remove a "Message" submenu within a "New" top level menu
removeActions(s_TopBranchId, [{title:"New", subtitles: ["Message"]}]);

// To remove a "Message" and "Contact" submenus
// within a "New" top level menu
var a=[{title:"New", subtitles: ["Message", "Contact"]}];
removeActions(s_TopBranchId, a);

default:
break;
}
...
}

```

addActions function

The addActions function used here takes two arguments. The first argument is the third argument passed into Scene_Actions (a_Actions for forms6.nsf and s_TopBranchId for forms7.nsf). The second argument is an array of objects with a special structure to support inserting and updating entries without having to worry about specific menu APIs.

Table A-2 shows details of the expected properties for each top-level menu object in this passed in array.

Table A-2 Details of properties passed as parameter array to addActions function

Parameter	Description
title	Visible text to display for the new or updated action.
pos	Position number for where this action should appear. Lower positions are to the left and higher positions are to the right.
href	The URL to invoke on a click of the action. This can be a JavaScript URL referring to a custom function provided by the customizer.
helpText	Hover text displayed when a user hovers over the action.
img	URL to an image to display for the action. The image will appear to the left of any title specified.
img_width	Numeric value representing the width of the image specified by the img property.
img_height	Numeric value representing the height of the image specified by the img property.
submenus	An array of submenu objects to be inserted.
find	An object with properties related to finding an existing action to facilitate replacing properties of the top-level action or inserting submenus at a particular location within the submenu.

Table A-3 displays details of the expected properties for each submenu object in the submenus array.

Table A-3 Details of properties passed as the submenu structure

Parameter	Description
title	Same description as for top-level object.
href	Same description as for top-level object.
helpText	Same description as for top-level object.
isDivider	Set to 1 or true to add a divider. (Do not specify other properties for this submenu.)

The addAction function also supports the ability to modify (or overwrite) existing menus via its find property. Table A-4 shows details of the possible properties for the find object.

Table A-4 Properties of the find object

Parameter	Description
title	Visible text to use to find the action object we wish to update (should specify only this or position).
pos	Position number of the action object we wish to update (should specify only this or title).
replace	Set to 1 to replace an existing submenu entry.
subtitle	Visible text for subtitle before which to insert specified submenus (or name of single submenu to update if replace property is specified) (should specify only this or suboffset).
suboffset	Zero-based offset for submenu before which to insert specified submenus (or location of single submenu to update if replace property is specified) (should specify only this or subtitle).

Adding new action menu in Version 6.5

Example A-7 shows the wrapper and helper functions for adding an action menu item in Version 6.5 and the forms6.nsf database.

Example: A-7 Add new action wrapper functions for Version 6.5

```
// utility routine to insert submenus into submenu array or replace one entry
function insertSubmenus(a, submenus, offset) {
  var bReplaceOne = (arguments.length >= 3);
  var n = bReplaceOne ? offset : a.length;
  if( bReplaceOne && !submenus[0].isDivider ){
    /* If some info is missing, then use prior existing info */
    var aTmp = a[n].split('|');
    if( typeof submenus[0].title != "string" )
      submenus[0].title = aTmp[0];
    if( typeof submenus[0].helpText != "string")
      submenus[0].helpText = aTmp[1] ? aTmp[1] : "";
    if( typeof submenus[0].href != "string" )
      submenus[0].href = a[n+1];
  }

  for(var j=0; j<submenus.length; j++) {
    if( submenus[j].isDivider ) {
```

```

        a[n++] = "DIVIDER";
        a[n++] = "";
    } else {
        a[n++] = submenus[j].title + "|" + submenus[j].helpText;
        a[n++] = submenus[j].href;
    }
    if( bReplaceOne) break;
}
}

// Find specified action and then update it
function updateAction(a_Actions, findInfo, updateInfo) {
    var i, action, j;

    // First find the offset for this action
    for(i=0; i<a_Actions.length; i++) {
        action = a_Actions[i];
        if(findInfo.title) {
            if( typeof action.title == "string" ) {
                if( findInfo.title == action.title ) break; else continue;
            } else if ( findInfo.title == action.title[0] ) break; else continue;
            return i;
        } else {
            if( findInfo.pos == action.pos )
                break;
        }
    }

    if( i >= a_Actions.length )
        return;

    // action is now the found entry
    if( updateInfo.submenus ) {
        var aSubMenu=action.href;
        var nEntries=aSubMenu.length/2;
        if( typeof findInfo.subtitle == "string" ){
            for(j=0; j<nEntries; j++) {
                if(aSubMenu[j*2].split("|")[0]==findInfo.subtitle ) {
                    findInfo.suboffset = j;
                    break;
                }
            }
        }
    }
    if( typeof findInfo.suboffset == "number" ) {
        if( findInfo.replace ) {
            insertSubmenus(action.href, updateInfo.submenus, findInfo.suboffset);
        } else {
            // support inserting submenus at an offset
            var aNewMenu=[];
            var nInsertOffset = Math.min(findInfo.suboffset,nEntries);
            if( nInsertOffset > 0 )
                aNewMenu = aSubMenu.slice(0,nInsertOffset*2);
            insertSubmenus(aNewMenu, updateInfo.submenus);
            if( nInsertOffset < nEntries )

```

```

        aNewMenu =
aNewMenu.concat(aSubMenu.slice(nInsertOffset*2,nEntries*2));
        action.href=aNewMenu;
    }
} else {
    // insert all new submenus
    action.href = [];
    insertSubmenus(action.href, updateInfo.submenus);
}
}

// update position
if( typeof updateInfo.pos == "number")
    action.pos = updateInfo.pos

// update title
if( typeof updateInfo.title == "string") {
    if( typeof action.title == "string" )
        action.title = updateInfo.title;
    else
        action.title[0] = updateInfo.title;
}

// update href
if( typeof updateInfo.href == "string") {
    if( typeof action.title == "string" )
        action.href = updateInfo.href;
    else
        action.title[1] = updateInfo.href;
}
}

// Function to add array of specified actions
// [{title:"New action", pos: 30, href:"javascript:...", helpText:"hover help
text", img:..., img_width:13,
//   img_height: 15, submenus: [{title:"submenu1", href:"...", helpText:"some
text"}]}]

function addActions(a_Actions, a_ToAdd) {
    var oNewEntry;
    var sGif;
    var n=0, i, j;
    var title, href, helpText, sWidth, sHeight;
    for(i=0; i<a_ToAdd.length; i++) {
        oNewEntry = a_ToAdd[i];

        if(oNewEntry.find) {
            updateAction(a_Actions, oNewEntry.find, oNewEntry);
            continue;
        }

        if(oNewEntry.img) {
            /* First two chars represent img width and following two img height */
            sWidth = oNewEntry.img_width + ' ';

```



```

        if(sWidth.length > 2) sWidth = sWidth.substring(0,1);
        if(sWidth.length < 2) sWidth = '0' + sWidth;
        sHeight = oNewEntry.img_height + '';
        if(sHeight.length > 2) sHeight = sHeight.substring(0,1);
        if(sHeight.length < 2) sHeight = '0' + sHeight;
        sGif = sWidth+sHeight+oNewEntry.img;
    }
    if( oNewEntry.img || oNewEntry.submenus ) {
        title = [oNewEntry.title,oNewEntry.href,oNewEntry.helpText,sGif];
        href = '';
        helpText = '';
    }
    else {
        title = oNewEntry.title;
        href = oNewEntry.href;
        helpText = oNewEntry.helpText;
    }
    if( oNewEntry.submenus ) {
        href = [];
        insertSubmenus(href,oNewEntry.submenus);
    }
    a_Actions[a_Actions.length] = {pos:oNewEntry.pos, title:title, href:href,
    BT0:helpText};
    }
}

```

Add new action

Example A-8 shows an example of adding an action menu using addActions wrapper function.

Example: A-8 Add menu in Version 6.5

```

function Scene_Actions( s_SceneName, o_Window, a_Actions ){

...
    switch(s_SceneName) {
        case 'Mail':
            // To add New → Account submenu
            var a=[{find:{title:"New", suboffset:9}, submenus:[{title:"Account",
href:"javascript:openNewShimmerDoc('$ToDo'),'Account'", helpText:"Create a new
account"}]}];
            addActions(a_Actions, a);

            // To add a top level menu with no submenus:
            var a=[{title:"Hello world", pos: 30, href:"javascript:alert('Hello
world!')", helpText:"some hover text"}];
            addActions(a_Actions, a);

            // To add a top level menu with submenus:
            var a=[{title:"Sub menu", pos:31, href:"javascript:alert('submenus')",
helpText:"submenu help", submenus: [{title:"submenu1",
href:"javascript:alert('submenu1 clicked')", helpText:"click me"},
{isDivider:true}, {title:"submenu2", href:"javascript:alert('submenu2 clicked')",
helpText:"click me"}]}];
            addActions(a_Actions, a);
    }
}

```

```

        // To add a new top level image-based action
        var a=[{title:"", pos:30, href:"javascript:launchCorporatePortal()",
helpText:"Open corporate portal",
img: "/iNotes/Forms6.nsf/h_ResourcesByName/corpportal.gif?OpenElement",
img_width:12, img_height:19}];
        addActions(a_Actions, a);

        // To find and update a top level menu named "New" and replace it with the
text "Create"
        var a=[{find:{title:"New"}, title:"Create"}];
        addActions(a_Actions, a);

        break;
    }
    ...
}

```

Adding new action menu in Version 7.0.1

Example A-9 shows the wrapper and helper functions for adding an action menu item in Version 7.0.1 and the forms7.nsf database.

Example: A-9 Add new action wrapper and helper function in Version 7.0.1

```

// utility routine to insert submenus into submenu array
function insertSubmenus(menuNode, submenus, subpos) {
    var j, submenuNode;
    for(j=0; j<submenus.length; j++) {
        submenuNode = Dbc(menuNode.$id);
        with(submenuNode) {
            if(submenus[j].isDivider)
                CrR = true; /* isDivider */
            else {
                label = submenus[j].title;
                onclick = submenus[j].href;
                DFB = submenus[j].helpText;
            }
            if(subpos && subpos != -1)
                pos = subpos;
        }
    }
}

function updateMenuNode(mn, o) {
    // update position
    if( typeof o.pos == "number")
        mn.pos = o.pos

    // update title
    if( typeof o.title == "string")
        mn.label = o.title

    // update href
    if( typeof o.href == "string")
        mn.onclick = o.href;
}

```

```

// update tooltip
if( typeof o.helpText == "string")
    mn.DFP = o.helpText;
}

// Find specified action and then update it
function updateAction(branchId, findInfo, updateInfo) {
    var i, action, aSubMenu, nEntries, nInsertOffset, nAdj, nSubPos=-1;

    // First find the offset for this action
    if( typeof findInfo.pos != "undefined")
        action = Dap(branchId, findInfo.pos);
    else if (findInfo.title)
        action = CyE(branchId, findInfo.title);

    // action is now the found entry
    if(!action)
        return;

    if( updateInfo.submenus ) {
        aSubMenu= DZo(action.$branchId); /* DM_getBranch */
        nEntries=aSubMenu.length;

        // If title specified as insert point, find offset
        if (typeof findInfo.subtitle == "string") {
            menuNode = CyE(action.$branchId, findInfo.subtitle);
            if(menuNode) {
                for(i=0;i<nEntries;i++) {
                    if( aSubMenu[i] == menuNode.$id ) {
                        findInfo.suboffset = i;
                        break;
                    }
                }
            }
        }
        if( typeof findInfo.suboffset == "number" ) {
            nInsertOffset = Math.min(findInfo.suboffset,nEntries);
            nAdj = 100;

            for(i=nInsertOffset;i<nEntries;i++) {
                menuNode = DPN(aSubMenu[i]);
                if( findInfo.replace ) {
                    updateMenuNode(menuNode,updateInfo.submenus[0]);
                    break;
                } else {
                    if(nSubPos===-1)
                        nSubPos = menuNode.pos + 50;
                    menuNode.pos += nAdj;
                }
            }
        } else {
            // Remove all existing submenus
            for(i=0;i<nEntries;i++)
                DGa(aSubMenu[i]);
        }
    }
}

```

```

    }
    // insert all new submenus
    if( !findInfo.replace )
        insertSubmenus(action, updateInfo.submenus, nSubPos);
    }

    updateMenuNode(action, updateInfo);
}

// Function to add array of specified actions
// [{title:"New action", pos: 30, href:"javascript:...", helpText:"hover help
text", img:..., img_width:13,
//   img_height: 15, submenus: [{title:"submenu1", href:"...", helpText:"some
text"}]}]

function addActions(s_TopBranchId, a_ToAdd) {
    var oNewEntry;
    var n=0, i, j;
    var menuNode, submenuNode;
    var subTitle;
    for(i=0; i<a_ToAdd.length; i++) {
        oNewEntry = a_ToAdd[i];
        if( oNewEntry.find ) {
            updateAction(s_TopBranchId, oNewEntry.find, oNewEntry);
            continue;
        }

        menuNode = Dbc(s_TopBranchId);
        with(menuNode) {
            pos = oNewEntry.pos;
            label = oNewEntry.title;
            onclick = oNewEntry.href;
            DFB = oNewEntry.helpText;
            if( oNewEntry.img )
                icon = oNewEntry.img + "|" + oNewEntry.img_width + "|" +
oNewEntry.img_height;
        }
        if( oNewEntry.submenus )
            insertSubmenus(menuNode, oNewEntry.submenus)
    }
}

```

Add new action

Example A-10 is an example of adding an action menu using the addActions wrapper function.

Example: A-10 Add new action in Version 7.0.1

```

function Scene_Actions( s_SceneName, o_Window, s_TopBranchId ){

...

    switch(s_SceneName) {
        case 'Mail':

```

```

        // Add a menu under the "New" action
        var a=[{find:{title:"New", suboffset:9}, submenus:[{title:"Account",
href:"javascript:openNewShimmerDoc('$ToDo','Account')", helpText:"Create a new
account"}]}];
        addActions(s_TopBranchId, a);

        // To add a top level menu with no submenus:
        var a=[{title:"Hello world", pos: 30, href:"javascript:alert('Hello
world!')", helpText:"some hover text"}];
        addActions(s_TopBranchId, a);

        // To add a top level menu with submenus:
        var a=[{title:"Sub menu", pos:31, href:"javascript:alert('submenus')",
helpText:"submenu help", submenus: [{title:"submenu1",
href:"javascript:alert('submenu1 clicked')", helpText:"click me"},
{isDivider:true}, {title:"submenu2", href:"javascript:alert('submenu2 clicked')",
helpText:"click me"}]}];
        addActions(s_TopBranchId, a);

        // To add a new top level image-based action
        var a=[{title:"", pos:30, href:"javascript:launchCorporatePortal()",
helpText:"Open corporate portal",
img:"/iNotes/Forms6.nsf/h_ResourcesByName/corpportal.gif?OpenElement",
img_width:12, img_height:19}];
        addActions(s_TopBranchId, a);

        // To find and update a top level menu named "New" and replace it with the
text "Create"
        var a=[{find:{title:"New"}, title:"Create"}];
        addActions(s_TopBranchId, a);

        break;

    default:
        break;
}

...
}

```

Overwriting action menu in Version 6.5

The addAction wrapper function also supports updating or overwriting menu actions. Use the find property in order to overwrite menu actions.

Example: A-11 Overwrite New and New → Message in Version 6.5

```

function Scene_Actions( s_SceneName, o_Window, s_TopBranchId ){
...

    switch(s_SceneName) {
        case 'Mail':
            // Overwrite top level New and the New -> Message menu

```

```

        var a = [{find: {title:"New", subtitle:"Message", replace:1},
href:"javascript:showHeaderDialog()",
submenus:[{href:"javascript:showHeaderDialog()"}]}]
        addActions(a_Actions, a);
        break;

    default:
        break;
}

...
}

```

Overwriting action menu in Version 7.0.1

The addAction wrapper function also supports updating or overwriting menu actions. Use the find property in order to overwrite menu actions.

Example: A-12 Overwrite New and New → Message in Version 7.0.1

```

function Scene_Actions( s_SceneName, o_Window, s_TopBranchId ){
...

    switch(s_SceneName) {
        case 'Mail':
            // Overwrite the top level New and the New -> Message menu
            var a = [{find: {title:"New", subtitle:"Message", replace:1},
href:"javascript:showHeaderDialog()", submenus:[{href:"
javascript:showHeaderDialog()"}]}]
            addActions(s_TopBranchId, a);
            break;

        default:
            break;
    }

...
}

```



B

Sneak preview: upcoming features

Note: Any future capabilities, features, or products that are discussed within this Redpaper appendix are current IBM plans, and are subject to change in whole or in part by IBM at any time, without notice.

What is in Notes and Domino Version 7.0.2

This section shows the types of enhancements that are being designed for the upcoming version of Domino Web Access. This list is provided to give readers a sense of the general direction in which future releases are heading. This list is not a definitive list of future features and may change at any time.

Domino Web Access support with Firefox 1.5

Macintosh users will be able to use Domino Web Access using the Mozilla Firefox 1.5, as displayed Figure B-1. Beginning with 7.0.2, Mozilla Firefox 1.5 will also be supported for W32 and Linux.

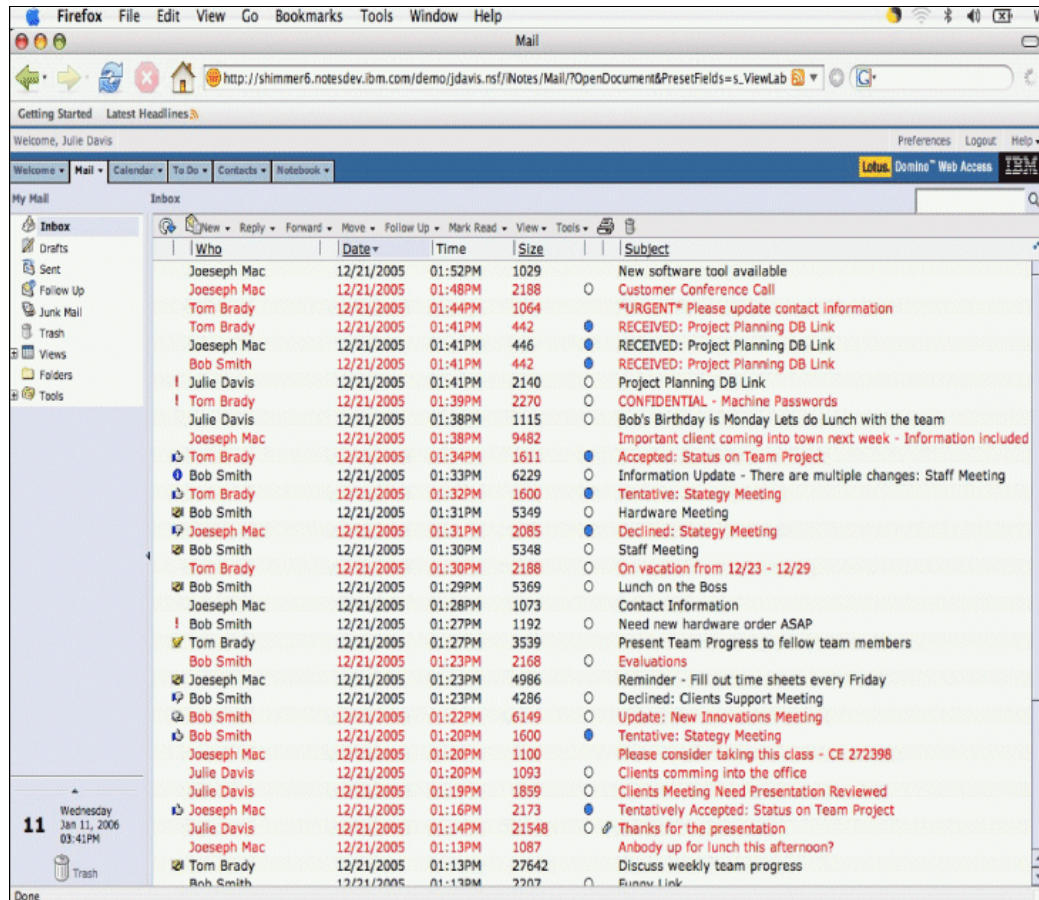


Figure B-1 Domino Web Access through Macintosh client using Firefox

Improved database quota reporting

Domino Web Access 7.0.2 improves the database quota reporting by polling for quota information when it polls for new mail or alarms, thereby updating the indicator. Previously there was a tendency for this information to be accurate only the first time any particular functional area page was displayed during a session.

Thread panel improvements

Domino Web Access 7.0.2 now offers the ability to display the thread panel when using the "Show documents one page at a time" preference setting. The thread panel now supports deleting entries selected within a thread panel list via a right-click action.

UI for attention indicator

Beginning in Version 7.0.2, Domino Web Access users will be able to set attention indicators through preferences under the Message Marking option, as shown in Figure B-2.

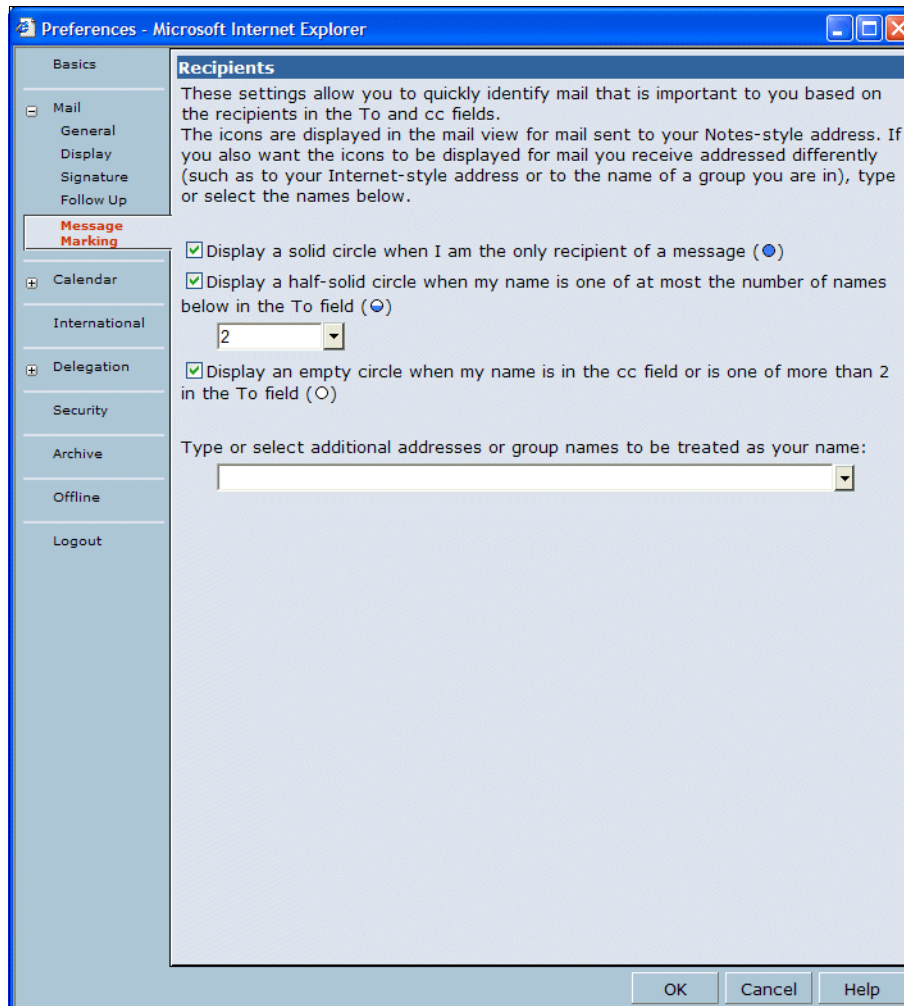


Figure B-2 Setting attention indicator under preferences

Advertise data as Atom and RSS feeds

Note: The Atom and RSS feeds are considered at this time to be only a *tech preview* for Domino 7.02.

Domino Web Access 7.0.2 can advertise data as Atom and RSS feeds for the inbox folder. By default this feature is disabled. Feeds may be turned on by the following server notes.ini settings:

- ▶ iNotes_WA_Feeds=atom,rss (to enable both feed types)
- ▶ iNotes_WA_Feeds=atom (only enable Atom feed)
- ▶ iNotes_WA_Feeds=rss (only enable RSS)

When feeds are enabled, the mail page will be modified to advertise the existence of the feed when someone goes to the inbox folder. Feed readers built into browsers, or browser extensions, will then automatically detect that a link is present. For example, the Sage

extension for Firefox will display the clickable image at the bottom right of the Firefox window, as shown in Figure B-3.



Figure B-3 Firefox displays this clickable image

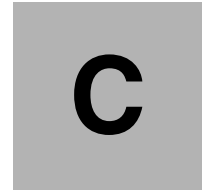
The advertised feed title will be the following format (if only one feed is advertised):

<db title>: <view title>

The advertised feed title will be the following format (if multiple feeds are advertised):

[<FEEDTYPE>] <db title>: <view title>

Where <FEEDTYPE> is either Atom or RSS. So from a feed reader one should be able to see information about new messages that arrive and who they are from, and a link should be offered to go to the document.



Additional material

This Redpaper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/REDP4188>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the Redpaper form number, REDP4188.

Using the Web material

The additional Web material that accompanies this Redpaper includes the following files:

<i>File name</i>	<i>Description</i>
forms7.zip	Forms7.nsf with the modifications shown in Chapter 4, “Skin customization techniques” on page 43, and Chapter 5, “Feature customization techniques” on page 65.
resource.zip	Sample resource Lotus Notes database to store the copies of all e-mails, as referred to in 5.3, “Overwrite New Message action” on page 69.
qsagent.zip	Contains QSMailStoreAgent.lss, as documented in 5.3.4, “Modify s_MailMemoEdit subform” on page 75.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Attention: Before any modification to a production or even a test Domino environment is made, all databases and templates should be backed up.

To use the provided databases you need to import the agent provided as *QSMailStoreAgent* into the mail files of the desired users or into the *dwa7.ntf* template for all users. This agent has to be set as target *none*.

The next step is to copy the provided *forms7.nsf* to the iNotes directory of the Domino server. Also, the repository has to be copied to the data directory on the Domino server.

Recycling the Domino server should now show the modified design and utilize the modified functionality.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redpaper.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 139. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Domino Web Access 6.5 on Linux*, SG24-7060
- ▶ *iNotes Web Access Deployment and Administration*, SG24-6518
- ▶ *iNotes Web Access on the IBM @server iSeries Server*, SG24-6553
- ▶ *Customizing QuickPlace*, SG24-6000

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Lotus Domino Web Access product page on IBM / Lotus site
<http://www-142.ibm.com/software/sw-lotus/products/product1.nsf/wdocs/webaccessshome>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Domino Web Access 7 Customization



Integrate your organization's design into Domino Web Access

Extend and enhance Domino Web Access functionality

Leverage new Version 7 design capabilities

Lotus Domino Web Access 7 gives users the power to create rich text messages, schedule meetings, manage tasks, and collaborate with colleagues, whether they are using their own workstation, an Internet kiosk, or another user's PC. Offline services for the security-rich e-mail, collaboration, and personal information management features of DWA allow a mobile workforce to maintain a high level of productivity by expanding access to key information and applications.

This IBM Redpaper provides you with the knowledge that you need to customize Domino Web Access to meet your organization's needs, whether you are an application developer or an administrator. It contains different customization examples for new and additional functionality, as well as removing and rearranging existing components, to meet the design and compliance needs of a fictitious bank.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks