
Integrate data captured via XFDL forms into your Notes applications

by Thomas Køcks



Thomas Køcks
Consultant

Thomas Køcks has been developing Lotus Notes and Domino applications for more than 11 years. He is a Domino consultant working in Denmark. His current projects include Web applications and WSE configurations for the IBM subsidiary EDB Gruppen A/S. Thomas is a certified Lotus Professional through Release 6 and a certified Java developer. He can be reached at tkn@lotusdomino.dk.

Businesses, governments, and other organizations use forms for driving requests, budget agreements, approvals, and many other critical processes. To run smoothly, organizations need control over their forms, including rapid development, ease of dissemination or deployment, ease of maintenance, and more. In support of this crucial business need, IBM took advantage of its purchase of PureEdge Solutions last year to bring out IBM Workplace Forms. You don't need IBM Workplace to use IBM Workplace Forms; it's a standalone family of products (available through the Business Partner Program) that developers, business process managers, or administrators can use to create, deploy, and maintain e-forms that are capable of driving workflow processes across systems.

Any tool that works with XML (databases, Web services, etc.) can work with a form created in Workplaces Forms because it uses the standard Extensible Markup Language (XML) syntax for forms, Extensible Forms Description Language (XFDL).¹ The World Wide Web Consortium developed and maintains XFDL, which allows you to store a form template with a pixel-precise layout, graphics, data, integrated calculations, input validation, enclosures, online help, digital signatures, and more in the same file. By combining format, content, actions, and encryption in one XFDL file, you can create forms that collect their own auditable transaction records as they move through a workflow process.

The front end that IBM Workplace Forms provides for developing and displaying XFDL forms is based on service-oriented architecture. It's actually a suite of three tools: Workplace Forms Viewer, Workplace Forms Designer, and Workplace Forms Server that includes an application programming interface (API), deployment support, and a WebForm server.

¹ For more information about the XFDL specification, see the following Web site: <http://www.w3.org/TR/NOTE-XFDL>.

For convenience from here on, I use XFDL forms as a generic term for forms that were created in Workplace Forms Designer.

Domino developers may be asking, “Why use IBM Workplace Forms? I can manage forms pretty well with Notes and Domino Designer.” There are some key advantages to using forms created in Workplace Forms Designer over forms created in Domino Designer:

- **Ease of porting forms to any environment.** By separating data and design, Workplace Forms Designer allows you to make infrastructure changes at will. You can move an XFDL form from Notes/Domino to IBM Workplace or WebSphere Portal with no redesign effort and without the end users noticing. You just handle the data collection in Notes.
- **Design precision.** You can control the placement of form items and design elements, such as lines or color, at the pixel level.
- **Ease of conversion.** You can use third-party tools to convert a library of existing forms in any format to the IBM Workplace Forms format, from which your company can continue to migrate forms to other platforms.
- **Audit trails.** You can design XFDL forms that handle processing rules and keep track of digital signatures.

In this article, I provide a utility that you can use to gather data entered in XFDL forms in a browser and store it in a Note database. The utility uses an efficient, reusable, dual-agent technique that avoids creating a lot of Notes forms or maintenance hassles.

First, I introduce you to IBM Workplace Forms and give you step-by-step instructions to begin composing your own XFDL forms with Workplace Forms Designer.

Then, I show how to store the data collected via the XFDL form in Notes documents, where you can manipulate it like any Notes data. I’ll wrap up by discussing a few other integration techniques you could try. Along the way, I’ll point out resources available for further learning.

You don’t need to know XFDL syntax to get started. To demonstrate how the solution works, I provide a downloadable database with reusable Domino elements and an example XFDL form (created with Workplace Forms Designer 2.6) attached to a Notes document.² If you want to create your own XFDL forms, you need Workplace Forms Designer, and familiarity with the Eclipse framework is beneficial but not mandatory. You need Workplace Forms Viewer to display the XFDL form in the demonstration properly in a browser.

If you don’t have IBM Workplace Forms, you can read along to become familiar with the dual-agent technique for storing information from an XFDL form in Domino.

The IBM Workplace Forms product family

IBM Workplace Forms is based on open standards and supports Java Specification Request (JSR) 168, JSR 170, Java 2 Platform, Standard Edition (J2SE), Java 2 Platform, Enterprise Edition (J2EE), and Web services. The IBM Workplace Forms product palette includes the following:

- **Workplace Forms Designer.** Workplace Forms Designer resembles Domino Designer. It provides a familiar interface and easy-to-use design features for creating e-forms. Its technology is based on Eclipse, so those of you who know the Eclipse platform will find the Workplace Forms Designer features familiar.
- **Workplace Forms Viewer.** This forms viewer is an optional browser plug-in that offers end users more functionality (such as mouse-over help and so on) than they get when viewing an XFDL form through a standard browser.

² You can download the sample database (wseForms.nsf) at www.eVIEW.com. From the home page, click Search → Browse Issue → November/December 2006 → Integrate data captured via XFDL forms into your Notes applications. Scroll to the bottom of the page for the download file.

- **Workplace Forms Server.** This install contains three components:
 - **Workplace Forms Server - API** provides integration capabilities.
 - **Workplace Forms Server - Deployment Server** is an installation system for deploying Workplace Forms Viewer to desktops.
 - **Workplace Forms Server - WebForm Server** is the server-side component that translates the XFDL form into Hypertext Markup Language (HTML)/JavaScript, a readable browser format so the client side isn't burdened with translations.

Any application that supports XFDL, such as GTE, CommerceOne, and Verisign, can directly use an XFDL form. Workplace Forms Designer can also export the XFDL form as an XForm, a specification of Web forms from the Worldwide Web Consortium. For the dual-agent integration technique in this article, with just a couple of property settings in Workplace Forms Designer, you can make any XFDL form compatible with Domino.

Domino handles attachments and Multipurpose Internet Mail Extensions (MIME)³ types natively, which makes it a perfect place to store XFDL forms. The features of IBM Workplace Forms allow developers to exchange form templates and completed XFDL forms among users and systems, without losing the forms' logic, layout, or signatures. Storing XFDL forms in Domino supports the following capabilities:

- Access rights at the field level
- Data extraction at the field level
- Separation of content and presentation in forms
- Collaborative exchange of form templates or completed forms

By using IBM Workplace Forms for forms presentation, Domino developers not only gain control

of the presentation of forms at the pixel level, they also have fewer challenges maintaining the forms' presentation as a separate layer or exchanging forms between Domino and non-Domino systems. When the presentation layer is XFDL, organizations can give the presentation task to specialists, like Web designers, and leave the data manipulation and handling to the developers. Because XFDL is independent of data layers, it's possible to reuse the XFDL presentation layer in many systems, so developers can exchange forms easily between Domino and non-Domino systems, which greatly eases workflow processing.

Setting up the demonstration

In the demonstration scenario, simple Notes documents store XFDL forms as MIME. End users can open an XFDL form using a link in a Notes view in a browser, which invokes Workplace Forms Viewer to display the form. The objective of the Domino agents is to store in Notes the values entered into the XFDL forms by browser users with Workplace Forms Viewer installed. When the data is captured in a Notes document, you can manipulate it like any Notes data. I don't extend the demonstration to manipulate the stored data — developers know how to do that — but I do provide an agent that shows the Notes document content from another view in a browser, just so you can see that the data made it from an XFDL form into Notes documents.

The demonstration requires installation of only two of the IBM Workplace Forms components. Workplace Forms Viewer must be installed on the end-user clients to present the XFDL form for input. Workplace Forms Designer provides the client for creating XFDL forms. This article assumes that those following the demonstration have installed these products on a desktop client, where forms are generally designed and used. The installation of Workplace Forms Designer and Workplace Forms Viewer are straightforward, so I won't repeat the documentation details in this article.⁴

³ MIME is an Internet standard that began as an extension for e-mail to support non-text attachments and non-ASCII character sets. Now MIME content types are also important for communications protocols like HTTP for the Web.

⁴ To locate downloadable instructions and documentation, see the following Web site: <http://www-128.ibm.com/developerworks/workplace/documentation/forms/>

In a more complex scenario, you might use Workplace Forms Server instead of Workplace Forms Viewer for displaying XFDL forms in a browser. However, installing Workplace Forms Server requires configuring a portlet or servlet on an IBM Portal server, which in turn includes determining IP ports to use and signing servlets with certificates. None of this setup is necessary when using Workplace Forms Viewer, so it's easier to use in an introductory demonstration.

Note!

If you plan to roll out IBM Workplace Forms in a large company, be sure to read up on the benefits and limitations of using Workplace Forms Server versus Workplace Forms Viewer.⁵ For example, although Workplace Forms Server takes more to set up, you can use it to circumvent distributing Workplace Forms Viewer to individual desktops.

The Domino side of the demonstration is designed to run in an environment with the following features:

- The Domino server must be release 7.0.1 because the examples require the new version of the Common Gateway Interface (CGI) variable Request_Content to handle an unlimited amount of content. Prior to 7.0.1, this variable can handle up to 64K of content.⁶ If you are using an earlier version of the Domino server, you can continue with the integration examples in this article, but the content that the demonstration's Domino agents can retrieve from Request_Content is limited to 64K.

- Microsoft XML (MS XML) must be installed on the Domino server so an agent can use it to parse the values from fields in an XFDL document. There are other ways of achieving the same result, but I've found this technique to be the easiest and most stable way to do it. MS XML is typically available in the Windows environment, but if not, you can download it for free from the Microsoft site.⁷

To check that MS XML is installed correctly, open an agent that requires it in Domino Designer and look for it under OLE Classes. For example, in **Figure 1**, I opened a LotusScript agent (agGenerateData), selected OLE Classes on the Reference tab, and scrolled to find the Microsoft XML entry. If the entry is listed as a reference, you know that it's available.

When these requirements are met, you're set to go. Let's begin by taking a look at the features of Workplace Forms Designer.

Introducing Workplace Forms Designer

Workplace Forms Designer is a designer environment that has the look and feel of an Eclipse-based development tool. Those of you who already know Rational Application Developer (RAD) or any Eclipse-based development tool will recognize Workplace Forms Designer's features.

For those of you new to Eclipse, becoming familiar with it is beneficial because the Eclipse environment is here to stay. Future versions of Notes will be based on the Eclipse platform. Take a moment to examine a few of the Workplace Forms Designer elements that are similar to Eclipse (see **Figure 2**):

⁵ Refer to the IBM Workplace Forms Server — Webform Server Best Practices Guide at the following Web site for more information: <http://publibfp.boulder.ibm.com/epubs/pdf/32525960.pdf>.

⁶ See the IBM support "Using REQUEST_CONTENT with web agents and large POST data" tech note for more information: <http://www-1.ibm.com/support/docview.wss?rs=899&uid=swg21240370>.

⁷ For an overview of the Microsoft XML methods and properties, see the following Web site: <http://www.microsoft.com/downloads/details.aspx?FamilyID=3144B72B-B4F2-46DA-B4B6-C5D7485F2B42&displaylang=en>

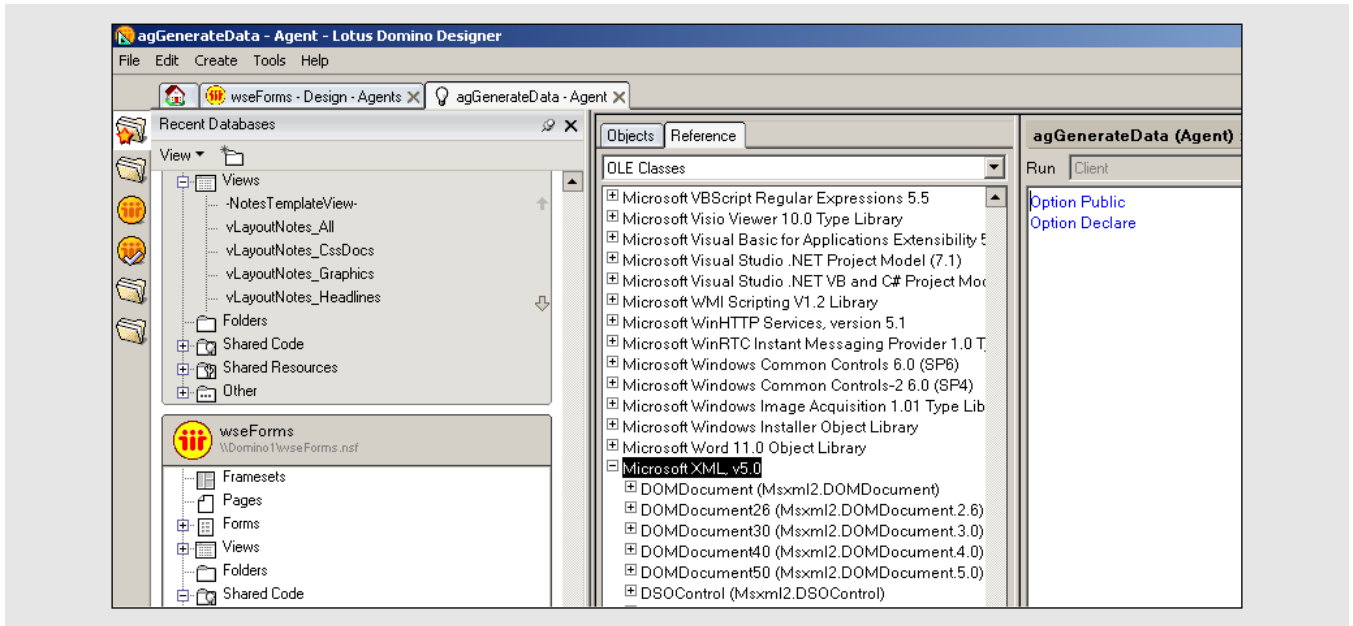


Figure 1 Verifying that Microsoft XML is present on the Domino server

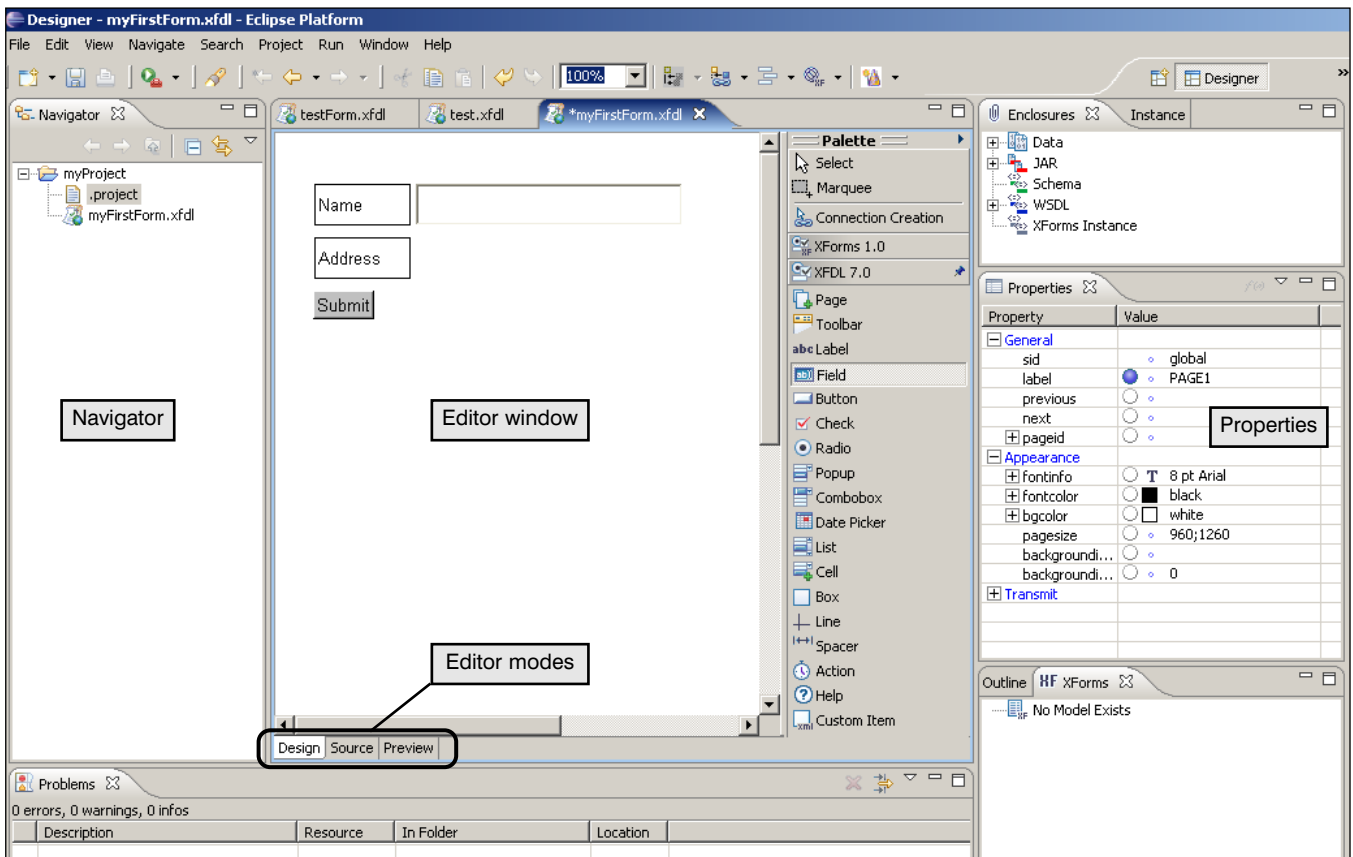


Figure 2 Workplace Forms Designer looks and feels like Eclipse.

- **Navigator.** As in Domino Designer, here you see the projects listed, plus the XFDL forms and elements that you use in your project or form.
- **XFDL editor window.** The contents of the middle pane change depending on the mode you select from the tabs at the bottom of the XFDL editor window. Each tab opens a different mode:
 - **Design** is where you design your form.
 - **Source** is where you work with the source code (XFDL) of your form.
 - **Preview** shows you what the form that you are designing looks like in a browser.
- **Palette.** This view appears to the immediate right of the editor window and is where you choose design elements, such as fields and buttons, to add to your form.
- **Properties.** This view is in the middle of the right side of the screen. You can use it to set the names, links, sizes, and values of items that you select from the palette.

These are just a few of the many Eclipse-like features in Workplace Forms Designer. If you are new to the Eclipse environment, I recommend reading additional resources⁸ to familiarize yourself more thoroughly with Workplace Forms Designer.

Now let's see how to create an XFDL form with Workplace Forms Designer.

Creating an XFDL form

I've supplied an XFDL form named "myFirstForm" attached to a Notes document in the download files, but the agents in the demonstration database can work with any XFDL form. To get you started creating your own forms, I'll go through the process for creating a new form in Workplace Forms Designer, using myFirstForm in the figures. If you decide to create your own XFDL form, you can

⁸ Refer to the resources at the bottom of the Web page for a list of references: <http://www-128.ibm.com/developerworks/workplace/documentation/forms/#6>

adapt the integration exercise to use your own form instead of myFirstForm.

First, create a project for the example. In the Navigator view, right-click and select New → Project. (You can also make the same selection from the File menu.) Using the New Project wizard, create a dynamic Web project and name it "myProject." In the Navigator view, a myProject folder appears with a .project XML file for capturing the project description.

Then, create the XFDL form. Right-click anywhere in the Navigator view and choose New → New Workplace Form, as shown in **Figure 3**. The New Workplace Form wizard opens (**Figure 4**). Enter "myProject" as the parent folder and name the file "myFirstForm." Checkmark the Open Form(s) after Finishing option and click Finish. The myFirstForm file automatically opens in the XFDL editor window in Design mode, as shown in **Figure 2**.

Now you're ready to build the form by adding elements from the palette. To display the Palette view,

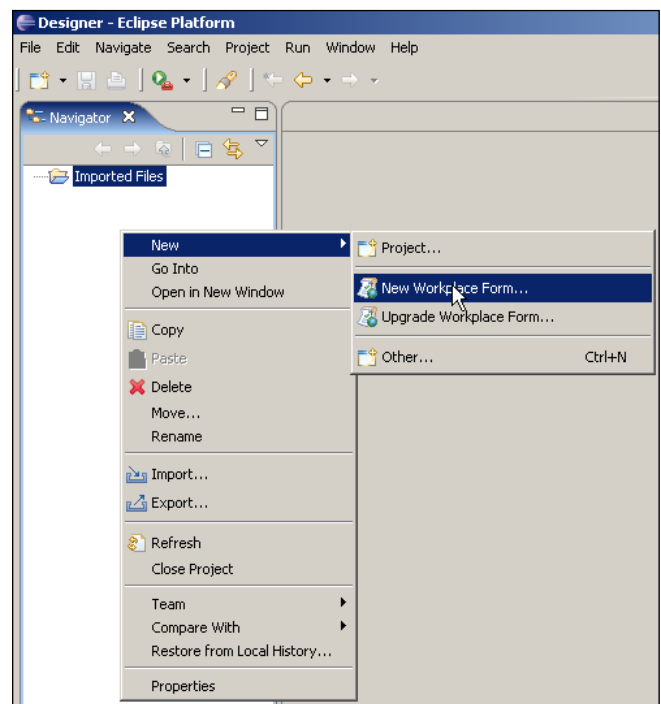


Figure 3 Opening the New Workplace Form wizard

select Window → Show View → Other → Palette. The example form contains two text fields and a Submit button. To add a field, click Field in the palette and then click the location in the editor canvas where you want to place the field. Note that the placement

operation is point and click rather than drag and drop. Always add a Submit button, which I show you how to configure later, to any XFDL form you want to integrate with Domino using the dual agents.

In the Properties pane of the currently selected element (as in **Figure 5**), you can define specific properties of the element, such as the exact coordinates for the element's placement on the form, name, font, etc. Another property allows you to set the background image of your form. This feature is handy if you want to scan a form with particular colorations, icons, or other visual features, use the scanned form as background, and then just add field elements that simulate the original fields on that background so the users can enter data.

As you might expect, the properties listed under Appearance affect how the button looks — the current settings are simply the defaults. Any property settings you adjust are saved when you save the form.

Examine the different elements available in the palette and try adding a few to your form. Most of the features are fairly straightforward, but mouse-over help is available for each feature, and the Workplace Forms Designer documentation provides additional help. It's possible to create automated form features or more complex elements, but to stay on track for the integration exercise, I recommend keeping your form simple.

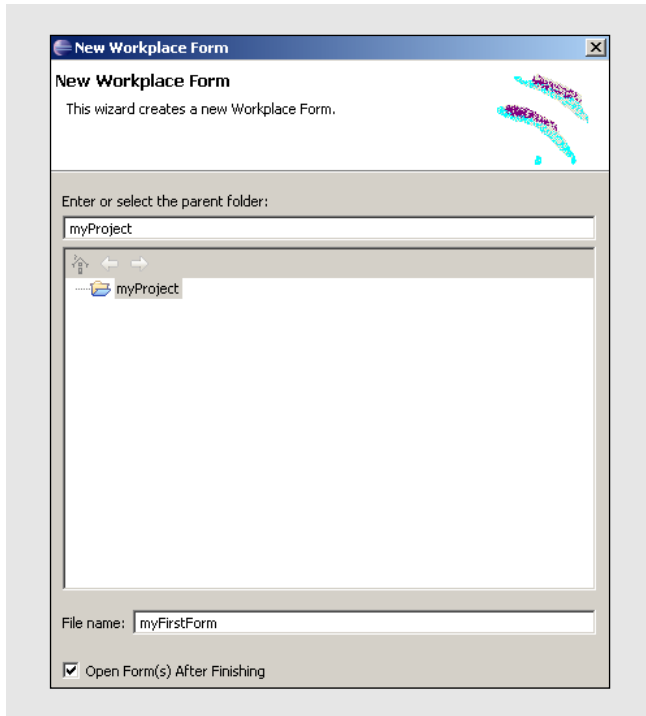


Figure 4 The first panel of the New Workplace Form wizard

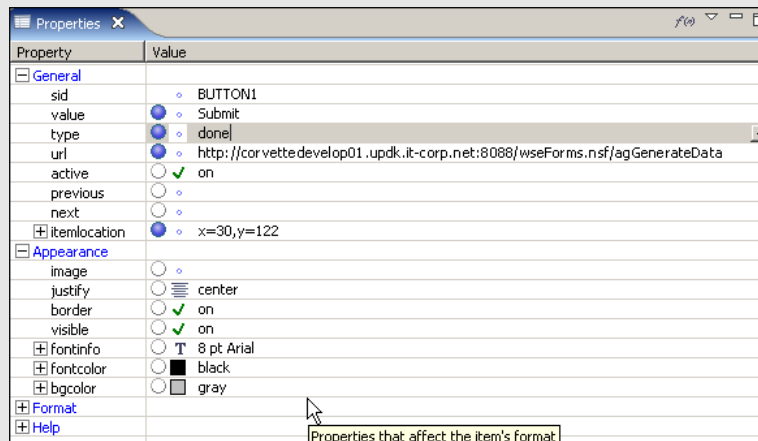


Figure 5 Properties for a Submit button

At any time, you can switch from Design mode to Source mode to view the XFDL source code of the form that you're creating, or select the Preview tab to see what the form looks like in a browser.

Okay, now that you have the basics for creating a form in Workplace Forms Designer, it's time to see how to use wseForms.nsf for storing the data users enter in your XFDL forms in a Notes database.

Integrating data from an XFDL form with Domino

If you haven't already downloaded the wseForms.nsf database, do so now. Sign the database with an Administrator ID, and check that the access control list matches your environment.

It doesn't matter what the example or your XFDL form looks like — the steps to integrate an XFDL form with the demonstration database are the same. To keep the developmental process straight, it helps to have a roadmap of activities.

The integration process

The steps below give you a high-level perspective of the integration process a developer (or another trained person) take to integrate data from an XFDL form with the wseForms.nsf database. Along the way, I indicate the corresponding design elements provided in the demonstration database to help you stay oriented. I've taken all of these steps already for the demonstration once, using myFirstForm.xfdl, so you have an example in wseForms.nsf.

Here are the steps:

1. Open Workplace Forms Designer and create the XFDL form you want to collect information from browser users, or open an existing XFDL form. Set the form parameters — the Submit button properties and the title — that are necessary to integrate with Domino (covered in the next section). Export the XFDL file to your local directory.
2. Open wseForms.nsf in the Notes client and, for each XFDL form you want to integrate, create a

separate foFormsTemplate document. Choose foFormsTemplate from the Create menu (see **Figure 6**). Attach one XFDL file in the Attachment field. In the 'Template name' field ("Workplace Form 1" in the example), name this Notes document in a way that you can remember the XFDL form attached. (See **Figure 7**.)

3. Save the document (i.e., press the Escape key and click Yes). At this point, the demonstration database returns to the viAllFormTemplates view (see **Figure 8**). This view has minimal features to simply get you started with the integration process. In Notes, the Template Name field appears as a URL to the Notes document with the XFDL file attached.

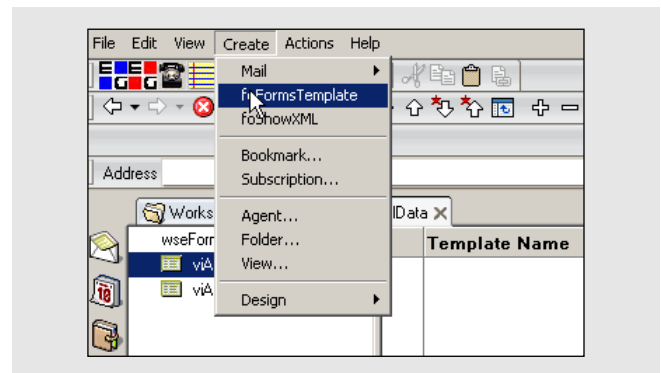


Figure 6 Creating a foFormsTemplate document in the Notes client

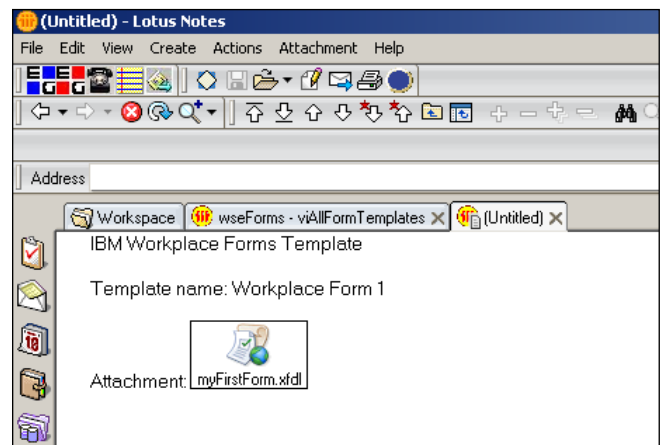


Figure 7 Attaching an XFDL file to the foFormsTemplate document

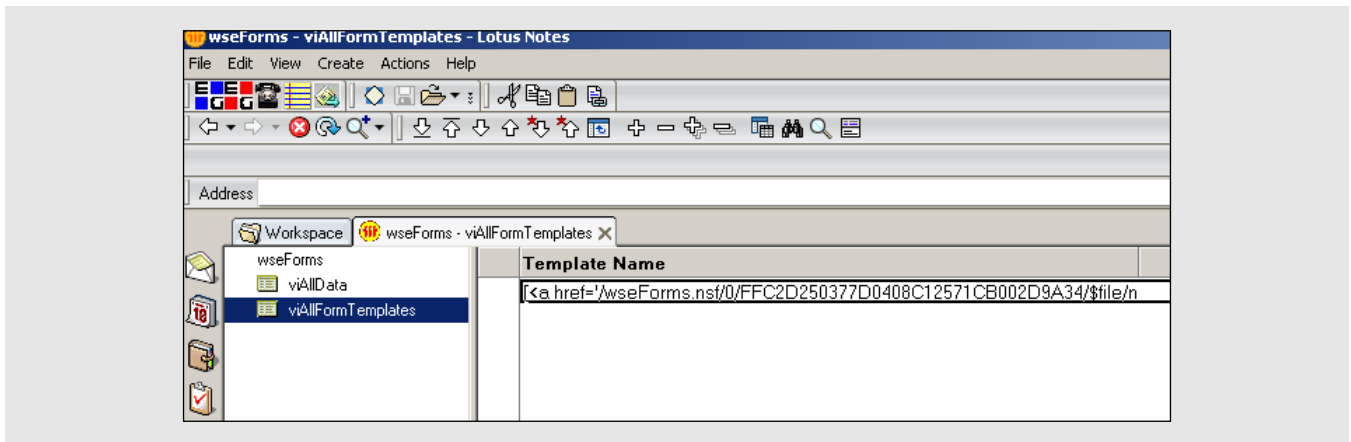


Figure 8 The viAllFormTemplates view in the Notes client

Now test saving data from the XFDL attachment as if you were an end user:

4. Open the wseForms.nsf database from a browser. The URL to your copy of the database is `http://servername:port/databasename`. (In my case, for example, the URL is `http://corvettevelop01.updk.it-corp.net:8088/wseForms.nsf`.) The opening browser window should show links to the wseForms.nsf database's two views (**Figure 9**). The viAllFormTemplates view was just discussed, and the viAllData view (shown later) lists all Notes documents users automatically generated for storing data when they submitted XFDL forms.
5. Still in the browser, open the viAllFormTemplates view (the browser view is shown later in **Figure 16**). In a browser, the values in the Template Name column are links to the XFDL forms you've saved as attachments to foFormTemplate documents. The links are named with the text you typed into the 'Template name' fields when creating these Notes documents.
6. Click the link with the template name you saved in step 3 (in the example, Workplace Form 1). The link presents the XFDL form to you (or any other users looking at this view) in the browser using Workplace Forms Viewer. For example, the link to Workplace Form 1 displays that Notes document's XFDL attachment, myFirstForm.xfdl, as shown in **Figure 10**.

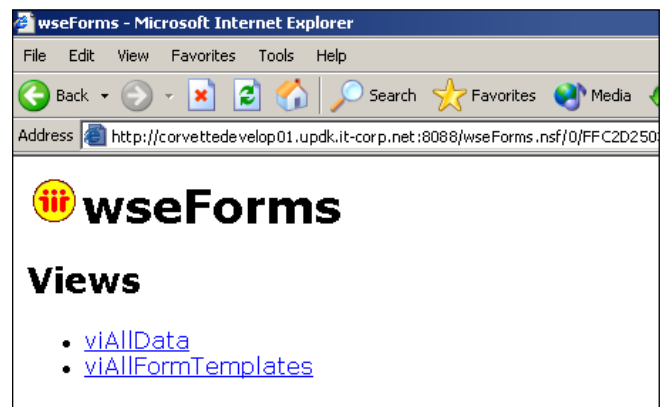


Figure 9 The wseForms.nsf database opened in a browser

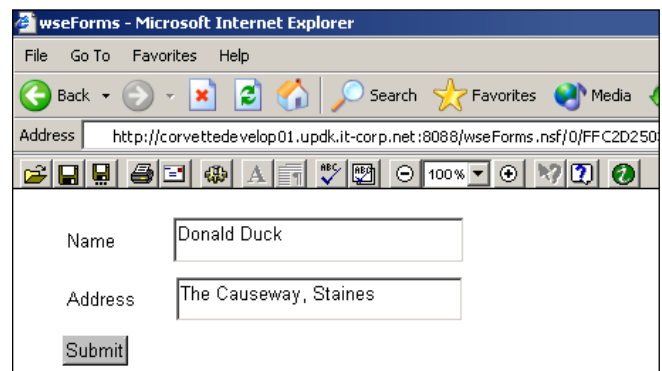


Figure 10 The Workplace Form 1 link in the viAllFormTemplates view in a browser opens myFirstForm.xfdl via Workplace Forms Viewer.

Note!

When installed on the client machine, Workplace Forms Viewer automatically handles the presentation of XFDL forms — no additional coding necessary!

7. Type some data in the form and click the Submit button. You should get a message that says “Thank you for submitting the form!” An agent (agGenerateData) activated by a POST request in this action button processes the content in the XFDL form and saves it as normal fields in a new Notes document. The agent names the new document with the name of the XFDL form used to capture the data.
8. Open the demonstration database from a browser again, as you did in step 4, and click the viAllData

link. In the viAllData view shown in **Figure 11**, the XFDL form used to submit data was myFirstForm. You can tell the entries apart by the time, date, and author fields. (For a production system, you would probably make enhancements that distinguish documents at a glance).

9. Select a document link displayed in the viAllData view. The link activates another agent (agShowData) and takes you to a browser presentation of data submitted from the XFDL form. For example, the agShowData agent displays the data submitted from the XFDL form in **Figure 10** as the Notes document in a browser that you see in **Figure 12**.

When a developer, administrator, or other person completes all of these steps, he or she should have one or more XFDL forms that were created in Workplace Forms Designer. In the Notes database, the developer should have attached each XFDL file on a separate Notes document. End users with Workplace Forms Viewer installed on their client machines should be able

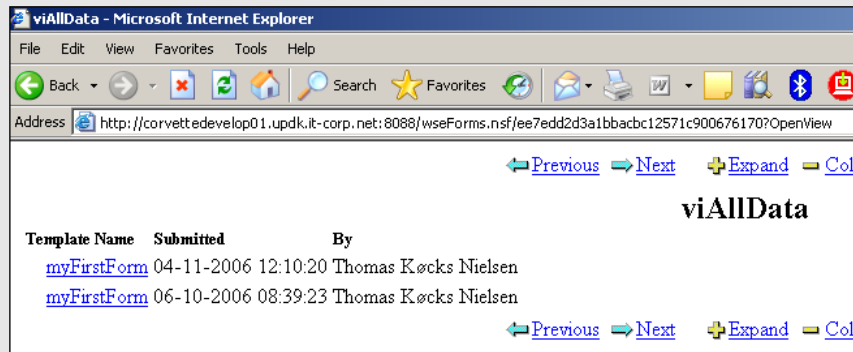


Figure 11 Notes documents created by browser users submitting myFirstForm.xfdl

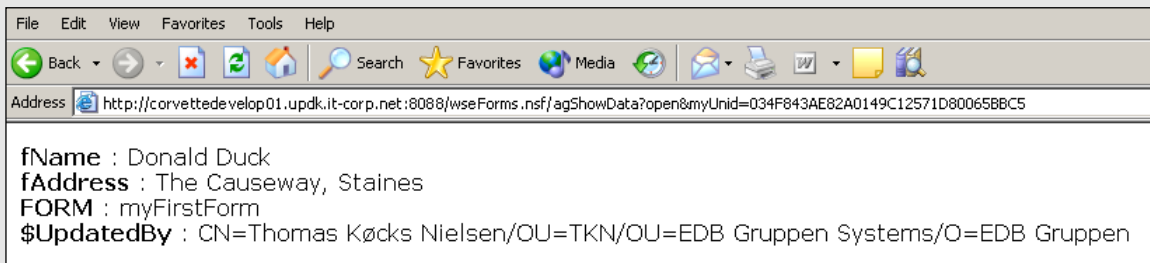


Figure 12 Notes data that a user submitted from myFirstForm.xfdl in a Notes document in a browser

to access the XFDL form from the viAllFormTemplates view in a browser and submit the data they enter. Each submitted form's data becomes available in a Notes document that appears in the viAllData view.

Now that you have the big picture, let's look at some of the details. In step 1, in Workplace Forms Designer, I show you the few properties needed in order for any XFDL form to work with the agents in the demonstration database. Then, in Domino Designer, I show you how the Domino elements are constructed to work together.

Note!

Keep in mind that if you include an extra element (e.g., a background image) on your XFDL form's design, that element must also be available from the client machine (e.g., a common network drive). The reference to the file, such as an Hyper Text Transfer Protocol (HTTP) path, should be the same for all end users of Workplace Forms Viewer.

Setting up an XFDL form for integration with Domino

The settings in this section are required for using any XFDL form with the Domino integration agents supplied in the demonstration database. Again, I've already taken these steps for the demo XFDL file, myFirstForm.xfdl.

Set the value and url properties of the Submit button

If your XFDL form is missing a Submit button, create it now. In Workplace Forms Designer, modify the properties of this button element appropriately (refer to **Figure 5**). The value property should be the button name — Submit. The url property should point to the agGenerateData agent in wseForms.nsf. (I mentioned that this agent handles the data posted by the Submit button and displays the return values in the user's browser. We'll look at the agent a bit later.)

Tip!

In Workplace Forms Designer, you can expand a view, such as Properties, to the full screen by double-clicking on the heading. Double-click the heading again to return the view to its previous size.

Give the XFDL form a title

Set the title property (the template name that appears in the viAllData view to end users) to the name of the form (in the example, myFirstForm). There are two ways to change the title:

- One is from the Outline view that is typically located in the lower-right corner of the Workplace Forms Designer screen (see **Figure 13** on the next page). In the Outline view, expand globalpage and double-click Form Global. When you open Form Global, another panel opens in the Properties view and the words "Form Global" mark the editor window in Design mode. The properties are designed to help you to keep track of a form's pages, each with their own fields, labels, and properties. Now in the Properties view, expand General → formid. Locate the title property and change the value.
- The other way to set the title of the form is to switch to Source mode, scroll to the top of the page, and enter the name between the <title> tags.

Note!

Systems treat XFDL in the same manner as they deal with XML because the syntax is the same. To learn more about XFDL, examine its specification on the Worldwide Web Consortium Web site (refer to footnote 1).

Now that your XFDL form is ready for integration with Domino, it's time to switch to

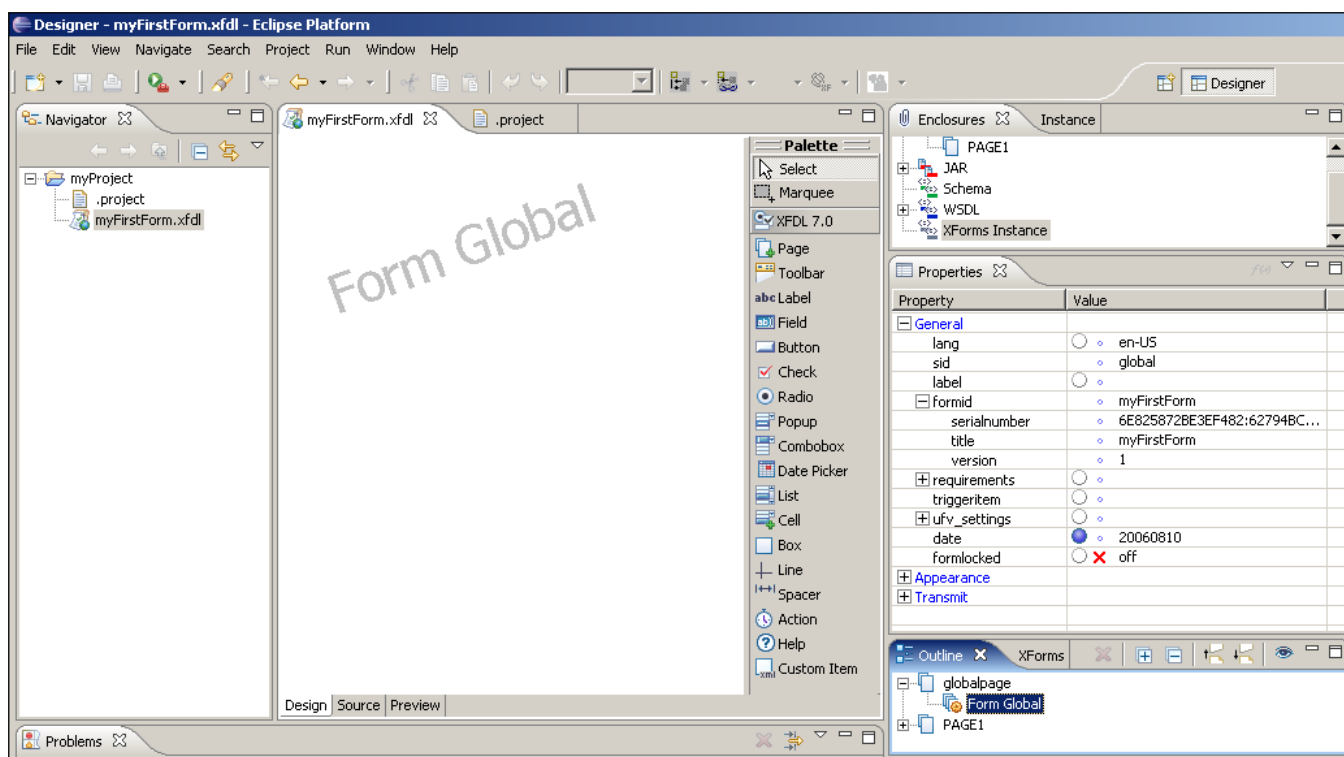


Figure 13 Setting the title of myFirstForm.xfdl as a Form Global property

Domino Designer and become familiar with the design elements in wseForms.nsf that enable this integration.

The wseForms.nsf database elements

In order to keep the demonstration easy to follow, I kept the elements in wseForms.nsf to a bare minimum and as simple as possible. Open the database in Domino Designer and examine the elements described in this section.

The foFormsTemplate form

Use this Notes form for creating documents that store your XFDL forms. As **Figure 14** shows, it contains only two fields. Body is a rich text field where you can attach an XFDL form, and templateName is a text field for naming the attachment.

The viAllData view

Open this view in a browser to see a listing of the data that users submit from the XFDL form or forms (see **Figure 11**). The view shows the templateName field value as a link in the first column.

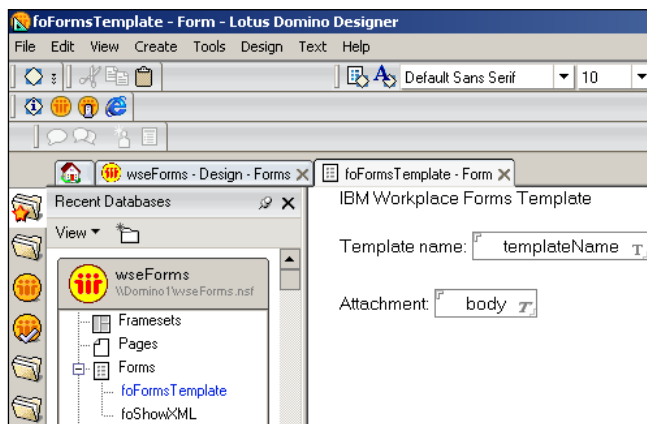


Figure 14 The foFormsTemplate form in Domino Designer

Figure 15 shows the same view in Domino Designer. The Template Name column's formula is:

```
"[<a href='/' + @Subset(@DbName;-1) + "/agShowData?open&myUnid=" + @Text(@DocumentUniqueID) + "'>" + Form + "</a>]"
```

This formula makes a URL link (href) to the agShowData agent, which simply shows browser users the data captured from the XFDL form in a readable format, without using a regular Notes form. (We'll look at this agent a bit later.) The URL passes to the agent the myUnid parameter, which takes the unique identifier (UNID) of the current document (@DocumentUniqueID). Form is the title (part of the formid property — refer to **Figure 13**) of the XFDL form, which this formula uses as the clickable link.

The viAllFormTemplates view

You can view all of the documents created with the foFormsTemplate form in wseForms.nsf in this view (see **Figure 16**). The formula in the Template Name column is:

```
"[<a href='/' + @Subset(@DbName;-1) + "/0/" + @Text(@DocumentUniqueID) + "/"$file/" + @AttachmentNames + "'>" + templateName + "</a>]"
```

This statement makes the templateName value a link to the file attachment (the XFDL file created in Workplace Forms Designer) stored as MIME on a Notes document created with the foFormsTemplate form. For example, when a browser user clicks on the Workplace Form 1 link in **Figure 16**, the link presents

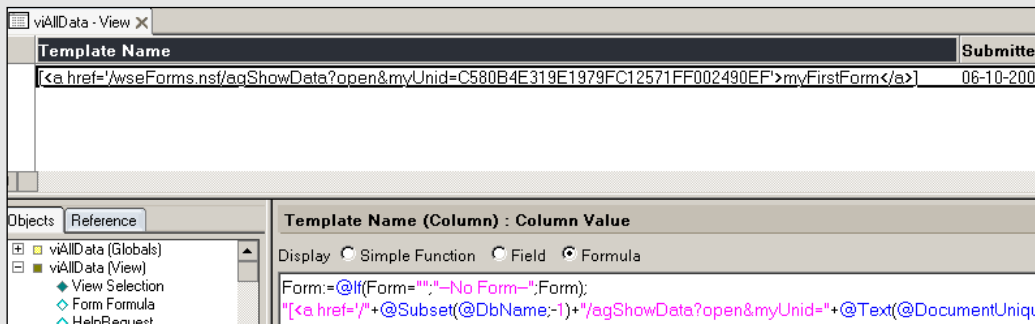


Figure 15 The viAllData view in Domino Designer



Figure 16 The viAllFormTemplates view in a browser

the attached XFDL file in the browser, via Workplace Forms Viewer, as in **Figure 10**.

The agShowData agent

The agShowData agent has two purposes, depending on how it's called:

- When called by links in the first column of viAllData view in a browser (refer to the column formula), this agent simply shows all of the field values from the selected Notes document to the browser user. The reason to use an agent instead of a Notes form to present the data to the user is that it's a generic, adaptable solution. If you depend instead on a Notes form to show the values, you'd have to create a separate Notes form for every XFDL form you create.
- When called by the agGenerateData agent (described next), the agShowData agent generates XML data based on the presence of the Request_Content variable. Request_Content is a CGI variable built into Domino and is available on all Notes documents shown in a browser. When a browser user submits data from an XFDL form to Domino via the POST command, Domino automatically assigns the standard Request_Content CGI variable with all of the data being submitted. Domino agents can read the Request_Content variable that captures the current context (in this case, the document context, not a stored field).

Note!

The agent handles all Request_Content fields that are available on the DocumentContext document. However, recall that the Request_Content size is limited before Domino 7.0.1. Those of you implementing the demonstration database in a Domino 6 environment cannot include as many fields on the XFDL form as you could if using Domino 7.0.1 or later. If you put too many fields on the XFDL form in a Domino 6 environment, you won't get all of the XFDL form field values on the Notes document.

The LotusScript code for the agShowData agent is shown in **Figure 17**.

Lines 1 – 5 are declarations.

Lines 6 – 7 set the current document and current database.

Line 8 sets a string variable to documentuniqueid, the UNID of the document to show in the browser. Note that the code assumes that the agent is passing only one documentuniqueid parameter to the browser. Also note the lack of error handling, such as checking whether the size of the string being passed is a legitimate UNID. I omitted error handling to keep the examples simple.

1. Dim domDoc As notesdocument
2. Dim domDataDoc As notesdocument
3. Dim domSes As New notessession
4. Dim domDb As notesdatabase
5. Dim domUnid As String
6. Set domDb=domSes.CurrentDatabase
7. Set domDoc=domSes.DocumentContext
8. domUnid=Right(domDoc.query_string_decoded(0),32)

Continues on next page

Figure 17 The LotusScript code for the agShowData agent

```

9. Set domDataDoc=domDb.getDocumentbyunid(domUnid)

10. Print "Content-Type:text/plain"
11. Print "Content-Type:text/html"

12. If domDataDoc.Form(0)="foShowXML" Then
13.   Forall domField In domDataDoc.Items
14.     If Left(Ucase(domField.name),15)="REQUEST_CONTENT" Then
15.       Print domField.text
16.     End If
17.   End Forall
18. Else
19.   Forall domField In domDataDoc.Items
20.     Print "<B>"+domField.name+"</B> : " +
           domField.text+"<BR>"
21.   End Forall
22. End If

```

Figure 17 (continued)

Line 9 sets a handle to the document the agent should show.

Lines 10 – 11 print the content type to the browser. This precaution ensures that Domino doesn't add any tags to the HTML that displays in the browser.

Lines 12 – 17 handle parsing the document's data if the document on **line 8** is of the type foShowXML. In such cases, the agent assumes that it should print the data in the Request_Content variable to the browser. The agent loops through all fields in the current document and prints out the ones named with Request_Content. Since Domino 7.0.1 automatically generates the Request_Content variables, there is no way to predict how many field items to expect. If there are more than one, Domino 7 names them request_content_000, request_content_001, and so forth.

Lines 19 – 21 take over if a browser user triggers the current document by clicking a link in the viAllData view. The agent simply loops through all items on the current document, printing them out to the browser in HTML format, with the field name in bold (), followed by the value.

The agGenerateData agent

The POST command in the code for the Submit button in myFirstForm.xfdl activates the agGenerateData agent to generate a Notes document containing the values that the user typed into the form. When the agGenerateData agent runs, its document context (not a physical document but a document context in memory) includes a sequence of fields from the submitted XFDL document, the ones the user typed in, marked with the CGI variable Request_Content. In other words, this agent captures everything typed by the user as Request_Content fields and stores it in the document context temporarily as XML. The agGenerateData agent parses the contextual document with the Microsoft XML Parser⁹ and stores the values in fields in a Notes document created with a virtual form, foShowXML.

⁹ Learn more about how to use the MS XML implementation of the XML Document Object Model in your applications at the MSDN Web site: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/d051f7c5-e882-42e8-a5b6-d1ce67af275c.asp>

Note!

I provided the contents of the wseForms.nsf database only for educational and testing purposes. This database does not contain error handling, such as checks for more than one attachment or what to do if the UNID in the agShowData agent is not valid. The code has been kept to a minimum. Do not use it as-is for production purposes.

Figure 18 shows the LotusScript code for the agGenerateData agent.

Lines 1 – 7 make declarations.

Lines 8 – 9 set the context — the current document and the current database.

Lines 10 – 11 create two new documents, one to use as the Notes document that stores the field data from the XFDL form (domNewDoc), and one temporary document to hold all of the XFDL/XML data the browser user submitted in the XFDL form (domTmpDoc). The domTmpDoc document supports

```

1. Dim domNewDoc As notesdocument
2. Dim domTmpDoc As notesdocument
3. Dim domDoc As notesdocument
4. Dim domSes As New notesession
5. Dim domDb As notesdatabase
6. Dim msXML As Variant
7. Dim sourcefile As String

8. Set domDoc=domSes.DocumentContext
9. Set domDb=domSes.CurrentDatabase
10. Set domNewDoc=domDb.createdocument
11. Set domTmpDoc=domDb.createdocument
12. Set msXML=CreateObject("MsXML2.DOMDocument")

13. Call domDoc.CopyAllItems(domTmpDoc)
14. domTmpDoc.form="foShowXML"
15. Call domTmpDoc.save(True,False)
16. msXML.async=False

17. sourcefile="http://" + domDoc.server_name(0) + ":" + domDoc.server_port(0) +
    "/" + domDb.FilePath + "/agShowData?open&myUnid="
    + domTmpDoc.universalid
18. msXML.load(sourcefile)
19. Call loopXMLTags(domNewDoc, msXML)

20. Call domNewDoc.Save(True,False)
21. Call domTmpDoc.Remove(True)

22. Print "<B><H1>Thank you for submitting the form !</H1></B>"

```

Figure 18 The LotusScript code for the agGenerateData agent

the agShowData agent that's introduced in **line 17** and then can be removed when the agent no longer needs it.

Line 12 sets a handle to the MS XML parser for looping through the XFDL/XML tags in the temporary document.

Line 13 copies all items from the current XFDL document in the browser to the temporary document.

Line 14 sets the form field of the temporary document so that it can be identified later.

Line 15 saves the temporary document so that the agent can reference its document unid property during processing.

Line 16 sets the MS XML object to be asynchronous so that the agent loads the entire XFDL document before continuing, not just a part of it. Partial loading would require different parsing techniques to know when the parser was finished.

Line 17 sets the sourcefile variable to the URL of the agShowData agent, passing along the temporary document's UNID as a parameter. In other words, when the agGenerateData agent invokes this URL, it passes the entire XFDL file that a user submitted to the agShowData agent for parsing to the MS XML

object. The agShowData agent generates the XML data based on the presence of the Request_Content variable.

Line 18 loads the temporary document into the MS XML object.

Line 19 calls the loopXMLTags() function, which loops through the tags in the temporary XFDL/XML document and sets field values in the new Notes document. (The next subsection describes this function.)

Lines 20 – 21 save the new Notes document. At this point, the agent has parsed the XML in the XFDL form (with loopXMLTags), so the agent saves the data in ordinary fields in the Notes document and removes the temporary document.

Line 22 prints a message to the browser to tell the user that the form was submitted.

The loopXMLTags() function

Figure 19 lists the code for the custom LotusScript loopXMLTags() function, which is part of the agGenerateData agent. The purpose of this function is to loop through tags in a temporary XML document

```

1. Sub loopXMLTags(domNewDoc As notesdocument, msXML As Variant)
2. Dim msXMLTags As Variant

3. Set msXMLTags=msXML.getElementsByTagName("field")

4. Forall msFieldTag In msXMLTags
5.     Forall msAttrib In msFieldTag.Attributes
6.         Forall msChildTag In msFieldTag.childnodes
7.             If msChildTag.nodeName="value" Then
8. Dim newBody As New NotesRichTextItem(domNewDoc,msAttrib.text)
9.         Call newBody.AppendText(msChildTag.text)
10.        End If
11.    End Forall
12. End Forall
13. End Forall

```

Continues on next page

Figure 19 The custom LotusScript loopXMLTags() function

```

14. Set msXMLTags=msXML.getElementsByTagName("formid")

15. Forall msFormTag In msXMLTags
16.     Forall msChildFormTag In msFormTag.childnodes
17.         If msChildFormTag.nodeName="title" Then
18. Call domNewDoc.ReplaceItemValue("form",msChildFormTag.text)
19.         End If
20.     End Forall
21. End Forall
22. End Sub

```

Figure 19 (continued)

and set field values for a Notes document (based on the foShowXML form). As you just learned, the agGenerateData agent generates both of these documents. I'll describe the function action using the example case, but you could substitute other XML and Notes documents in your own adaptations.

Line 1, the function header, takes as parameters the new Notes document and the MS XML object that the agGenerateData agent sets when invoked.

Line 3 sets the msXMLTags variable to contain all field nodes in the temporary XML document.

Lines 4 – 13 find the data to store in the new Notes document. The function's premise is that all field values found should be stored in the Notes document. For all the field nodes in the temporary document, the function loops through the attributes and finds the child nodes that contain values. To store each value it finds, the function creates a new rich text field (RichTextItem) on the Notes document, using the name of the field node in the temporary document. The function uses the .text property of childnodes to append the value to the RichTextItem object. Using RichTextItem enables developers to avoid size limits on the XFDL/XML data the function can handle.

Line 14 sets the msXMLTags variable to contain all the formid nodes in the temporary XML document. The <formid> tag contains a <title> tag, which you may recall was defined in myFirstForm.xfdl as the tag where you inserted the form name.

The last code loop ensures that the temporary XML document has only one formid element. **Lines 15 – 20** loop through any formid elements in the document and find the ones containing a title node. When it finds a title node, the function uses .text again to add that value to the form field of the Notes document.

Now you've seen how this dual-agent technique works for integrating data collected from XFDL forms with Domino, I'll mention a few others.

Other ways of integrating XFDL forms

You could handle the POST data in Domino at least a couple of other ways:

- Using a servlet
- Using a Web service

If you have a version of Domino prior to version 7.0.1, to work around the Request_Content limitations, you can use a technique presented in the IBM Redbook "IBM Workplace Forms: Guide to Building and Integrating a Sample Workplace Forms Application."¹⁰

¹⁰ <http://www.redbooks.ibm.com/redbooks.nsf/e9abd4a2a3406a7f852569de005c909f/aa84c61e8b4e82a7852571650055768e?OpenDocument>

Regardless of your Domino environment, you might also be interested in the solution presented in this Redbook because it's much more complex than the one presented in this article. Using a sales-quotation-approval scenario, it shows you how to capture data in an XFDL form and apply business logic and workflow to gain approval. The solution introduces integration points with WebSphere Portal, IBM DB2 Content Manager, and Lotus Domino along the way. For Domino, the integration point is a servlet that parses the POST data.

Publishing to WebSphere Portal or IBM Workplace Services Express (WSE)

If you have a mixed Web server environment containing Domino servers and either WebSphere Portal servers or WSE servers, you should consider using IBM Workplace Forms Integrator,¹¹ which is a collection of portlets and servlets released as a server add-on. This suite of tools enables you to use IBM Workplace Forms to create e-forms and submit them to WebSphere Portal Document Manager. From there, WSE or WebSphere Portal users can import the e-forms into their applications with very few steps. With this solution, you combine the benefits of WebSphere Portal Document Manager and the ease of IBM Workplace Forms.

And more ...

With IBM Workplace Forms, you can use XML text files to replace manual paper processes. XFDL forms have a great future and are the way to go if you want to convert paper-based forms to e-forms for browser applications. You can integrate XFDL forms with customer relationship management, enterprise resource planning, or database tools in the supply chain management workflow. Many third-party software companies (e.g., StarOffice) have already acknowledged and currently support XFDL, so rest assured that your investment in IBM Workplace Forms will have a long life.

¹¹ <http://www-128.ibm.com/developerworks/workplace/library/forms-integrator/>

Conclusion

You've seen the wizard-driven approach IBM Workplace Forms Designer uses for creating XFDL forms, and you've learned how to use a couple of agents to deliver the data collected in an XFDL form into Domino and reuse it. You can use XFDL forms in many of the same ways as Notes forms. The main difference for handling XFDL forms in Notes applications is that you handle the form design separately from the data users submit from it. The separate design layer has the added benefit of leaving the appearance to those who can make it precise to the pixel, looking exactly the same as (or better than) the original paper forms.

And the solution is fast. For example, say that you want to make a paper-based customer survey from last year reusable on the Web. The survey taker uses a score of 1 through 5 for answering each question. All you have to do is the following:

- Scan the form and use it as a background image for the survey form in Workplace Forms Designer.
- Create the score fields as fields on the XFDL form.
- Apply the agents discussed in this article.

Have Domino add up the data from each survey and present a graph to the user in either a portlet or as a summary page in Domino, so that every time a user completes a survey, your application automatically updates the graph or summary page. Notice that the only design that you do is to create the form in IBM Workplace Forms. You don't need to think about field types, default values, formula or script languages, and so on for designing a new form that stores survey data in Domino.

Another powerful advantage is that if you design your forms and schemas using IBM Workplace Forms, you can easily move them from one type of server to another (e.g., from Domino to IBM Portal Server) without changes. XFDL forms created with Workplace Forms Designer can help you to integrate business processes not only in Domino but also through Document Manager in IBM Portal Server and across other platforms that can handle XML data. You're ready to go deeper and practice manipulating data from XFDL forms in other ways in Domino and elsewhere, such as by creating a statistics module or a portlet.

References

Documentation and support

XFDL specification:

<http://www.w3.org/TR/NOTE-XFDL>

IBM Workplace Forms documentation:

<http://www-128.ibm.com/developerworks/workplace/documentation/forms/>

“Using REQUEST_CONTENT with web agents and large POST data,” IBM software support tech note:

<http://www-1.ibm.com/support/docview.wss?rs=899&uid=swg21240370>

IBM Workplace Forms Server — Webform Server Best Practices Guide “Differences Between Webform Server and Workplace Forms Viewer”:

<http://publibfp.boulder.ibm.com/epubs/pdf/32525960.pdf>

MS XML object model:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/d051f7c5-e882-42e8-a5b6-d1ce67af275c.asp>

Publications

“IBM Workplace Forms: Guide to Building and Integrating a Sample Workplace Forms Application,”

IBM Redbook abstract, John Bergland, et al:

<http://www.redbooks.ibm.com/redbooks.nsf/e9abd4a2a3406a7f852569de005c909f/aa84c61e8b4e82a7852571650055768e?OpenDocument>

“IBM Workplace Forms 2.5 Integrator for WebSphere Portal Document Manager,”

Jaime Solari and Angela Douglas:

<http://www-128.ibm.com/developerworks/workplace/library/forms-integrator/>

MSXML 4.0 Service Pack 2

MS XML download:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=3144B72B-B4F2-46DA-B4B6-C5D7485F2B42&displaylang=en>