

Tutorial - Using Domino Collaboration Builders

Overview

WebSphere Portlet Factory contains specialized builders in the Lotus Collaboration Extension feature set that are used to work with data stored in Domino databases. Domino View and Domino View and Form are the two builders that can access Domino databases. These builders can be used in applications and portlets directly or via Service Oriented Architecture (SOA). In this tutorial, you will use both techniques. Portlet Factory Page Automation will then be used to display, create, and modify the Domino data.

View categorization is common in Domino applications. Portlet Factory supports displaying tabular data using categories via the Category View builder. This builder can rely on Domino to categorize the view or it can use a flat Domino view and compute the categories itself. You will use both techniques to become familiar with them.

Document data can be read, edited, created and deleted using Domino View and Form. Simple checkbox options enable these major features and automatically add code, navigation buttons, and all other artifacts needed to make them work properly. You will use all of these features to promote your understanding of how to employ them.

The Lotus Collaboration Extension feature set also includes several other builders to enhance use of Domino data in a web application or portlet. The Domino Keyword Lookup builder is used to retrieve label-value pairs from a Domino view. These label-value pairs are then used the same way a regular Lookup Table builder is used. It is very common to use these pairs in HTML UI elements such as radio buttons and select lists. The Domino Formula builder accepts Lotus Notes formula language expressions to produce dynamic text strings in much the same way computed text or Computed for Display fields are used in Lotus Notes and Domino-rendered web applications. You will use both of these builders in this tutorial.

Domino data can also be accessed using SOA. Many Lotus Notes / Domino applications include features that retrieve lists of user names from the Domino Directory. Since it is common for numerous applications to need this same data, it makes sense to create centralized data services that produce these names. In this tutorial, you will create a Service Provider that accesses the Domino Directory's (`$VIMPeople`) view. You will then add a Service Consumer builder to the application model in order to apply these names as a list of choices in a select list.

An important part of many Domino-rendered web applications is the ability to run Web Query Save agents in conjunction with submitting a new document or a document update. This type of agent gets invoked automatically when the form used to submit the document contains a reference to the agent. The Domino-side mechanism that triggers this agent only functions when the document is submitted via HTTP. Portlet Factory accesses Domino data via DIIOp, therefore, the Domino view and Form builder contains a mechanism for triggering agents during the submit process. Although the procedure is different, the net result is same: submitted documents can be post-processed immediately via agents. Portlet Factory offers a bit more flexibility here as different agents can be run for new documents versus updated ones.

Once you complete this tutorial, you should have a solid understanding of how to use Domino data in web

applications and portlets.

Setup Requirements

To make this tutorial work properly, a certain amount of setup is required. Some tasks must be performed in Portlet Factory and others in Domino. Those requirements are described below.

WebSphere Portlet Factory Setup

1. Complete the tutorials that ship with WebSphere Portlet Factory.

It is necessary to complete the following tutorials so that you have the basic understanding of WebSphere Portlet Factory concepts.

- Creating a Web Application Project
- Creating a Simple Portlet
- Creating Data-Driven Portlets

2. Optionally, create a page in WebSphere Portal. For convenience, create a new page so that it is either a child or a sibling of one of the pages used in an earlier tutorial.
3. Create a new WebSphere Portlet Factory project containing the Lotus Collaboration Extension feature set or add this feature set to an existing project.
4. Set the Domino configuration properties.
 - a. In the project you made for this tutorial, expand the `WEB-INF\config\domino_config` folder and locate the `default_domino_server.properties` file. **Note:** If you connect regularly to several different Domino servers, you can create several of these properties files and switch between them by choosing the correct file in the Domino builders.
 - b. Open this file and set the properties as needed.
 - i. `ServerName=` use the hostname of the Domino server and not the Domino server name itself, i.e., `hostname` instead of `ServerName/Organization`. No port number is needed unless HTTP is running on the Domino server on a port other than the default port of `80`.
 - ii. `UserName=` the common name of a user with permission to use DIIOp on the server and with at least Editor access to the Notes databases you plan to connect to. In this tutorial, you will use `DominoTutorialEmployees.nsf` and `Names.nsf`. By default, the Domino server, DIIOp, and these databases will be accessible by any authenticated user. **Note:** Only the user's common name is needed although the user's fully distinguished name will also work.
 - iii. `Password=` the internet password of the user specified in the `UserName` property.
 - c. Save and close this file.

Domino Setup

1. Ensure the Domino server is running.
2. Ensure there are at least 2 or 3 Person documents in the server's Domino Directory. Having only

one test user in the Domino Directory will be inadequate for the Service Oriented Architecture (SOA) section of this tutorial. If necessary, register a few test users.

3. Put the `DominoTutorialEmployees.nsf` database on the Domino server. This file is in the project you created for this tutorial and is found in the following location: `<projectname>\samples\tutorials\domino\nodeploy`. This database contains employee and department records derived from the sample tables that ship with Cloudscape and DB2 as many people are already familiar with them. **Note:** For consistency with those sample tables, the misspelling of `FirstName` as `FirstNme` was preserved.
4. Sign the `FixApproverName` agent in the `DominoTutorialEmployees.nsf` database via Domino Designer or Domino Administrator. The signer's user name must be listed in the server document on the `Security` tab in the field labeled `Run restricted LotusScript/Java agents` either explicitly or via group membership. The simplest way to sign this agent is to open and save it via Domino Designer while logged in as this user. Alternatively, you can sign the entire database via Domino Administrator.
5. Familiarize yourself with the contents of this database via a Lotus Notes client.
 - a. Views
 - i. `All Employees` - A simple view that will be used by WebSphere Portlet Factory to retrieve employee data for display in a table.
 - ii. `All By Department` - A categorized view containing all of the employee records categorized by department ID.
 - iii. `All By Last Name` - A flat view containing all of the employee records sorted by last name.
 - iv. `Lookup > Departments` - A utility view containing department names and department IDs. It is used programmatically by the Notes application and by Portlet Factory to associate the names with the IDs to gain certain user interface benefits for data entry and display.
 - b. Forms
 - i. `Employee` - A form used for display and data entry of employee data in Lotus Notes and Domino-rendered pages. Portlet Factory will use it to build schemas for data entry and display.
 - ii. `Department` - A form used for display and data entry of department data in Lotus Notes and Domino-rendered pages. This form will not be used by Portlet Factory in this tutorial but the department data will be used.
 - c. Agents
 - i. `FixApproverName` - A hidden agent that uses the `NoteID` that gets passed to the agent when invoked by Portlet Factory to get the document being submitted. Portlet Factory will invoke this agent after employee records have been added or edited. Its function is to canonicalize names in the `Approver` field, i.e. to convert `User Name/Org Name` to `CN=User Name/O=Org Name`.
 - ii. `Remove Approvers` - An agent that runs against all selected documents. This agent removes the data in the `Approver` field from all selected documents and is included as a convenience.
6. Familiarize yourself with the contents of this database via a web browser. You can open this database via the following URL

```
http://<hostname:portnumber>/DominoTutorialEmployees.nsf/  
AllByLastName?OpenView
```

You should see a simple, Domino-rendered web application. Use the action buttons across the top of the page to navigate between views and create new employee records. Use the links in the views to open individual employee records.

Working with Domino View Data

This section focuses on using a Domino View and Form builder to access tables of Domino data found in Domino views. This type of builder examines any given view from a Domino database and builds an XML schema from it. The node names and data types used in the schema are derived from the view column names and the data in the columns. Any view can be used but a flat view without any categorizing works best for generating the schema for working with view data.

Since Portlet Factory builds the schema directly from a view, it can be useful to create a view for this sole purpose. By performing this task, one can easily control the node names, node order, and data types Portlet Factory will use to build the schema. It can also be useful to include column formulas to manipulate the raw Domino data before Portlet Factory consumes it. To keep things simple, however, this section uses a view called `All Employees` which is a flat view containing all of the fields in the employee records using simple column names and no column formulas other than the names of the fields.

Once the Domino data is consumed by a Domino View and Form builder, other builders can be used to enhance the appearance of the data display. For example, a Category View builder can categorize a flat view. The resulting categorization in the web application will look familiar to Domino application developers. This builder also resolves certain traditional shortcomings of Domino-rendered categorized views. For example, a Category View builder allows multiple categories to remain expanded simultaneously.

Creating a Domino-Driven Model


The Portlet Factory Domino integration builders are powerful, high-level builders. An application that displays the data from a Domino view can be made by adding only one of these builders to a model. Either a Domino View or a Domino View and Form builder is all that is needed. In this section, you will build such an application by using a Domino View and Form builder.

While adding a Domino View and Form builder, you will choose one of the views in the Domino database. View names can be chosen using either their full names or their aliases. Hidden views are also supported.

1. Create the new model.
 - a. From the `File` menu, choose `File > New > WebSphere Portlet Factory Model`. The New Model Wizard will open.
 - b. In the Choose Project screen, select the project you created in the Setup Requirements section and click **Next**.
 - c. In the Select Model screen, choose `Factory Starter Models > Empty` and click **Next**.
 - d. Name and save the new model.
 - i. In the Save New Model screen, expand the folder structure and select `models/`


tutorials.

- ii. At the top of the screen, there is a field into which you can enter a new subfolder name. Enter `\domino` so that the full path is `models/tutorials/domino`.
 - iii. In the Model Name field, enter `DominoEmployees` and click **Finish**.
 - e. The New Model Wizard will create this model and store it in the `WEB-INF\models\tutorials\domino` folder in the project. This new model will open automatically.
2. Add a `Domino View & Form` builder call to establish access to the `Domino Tutorial Employees` database.

- a. In the Builder Call List, click , choose `Domino View & Form` and click **OK**.
- b. Fill out the Domino View & Form builder call using the details provided in this table. Click **Apply** to save your changes when finished.

Input Name	Input Value
Name	<code>Employees</code>
Database Name	Click the Get databases and views button then choose <code>DominoTutorialEmployees.nsf</code>
View name	Choose <code>AllEmployees</code> - this value is the alias for the <code>All Employees</code> view
Paged Data Display	Check the box
Rows per page	Enter <code>15</code>

3. Test the model.

- a. Run the model using the  icon.
- b. View the application in the web browser. You should see rows and columns of data from the Notes database.

Categorizing View Data

There are two approaches to categorizing Domino view data. One approach uses a view that is already categorized in Domino and relies on that view to provide the categorization. The other approach uses a flat Domino view and then relies on Portlet Factory to provide the categorization. In either case, one must use a Category View builder to make it all work.

Using a Categorized Domino View


A Domino View and Form builder can display document rows, category rows, or both simultaneously. If the Domino view is already categorized and the document rows must be displayed, all rows must be included. This configuration is quite common when using a categorized Domino view.


In this section, you will temporarily change the view in the Domino View and Form to a categorized view in order to understand the first approach to categorizing view data.

1. Edit the Domino View & Form builder call using the details provided in this table. Click **Apply** to save your changes when finished.


Input Name	Input Value
View name	Choose <code>AllByDept</code> - this value is the alias for the <code>Employees - By Department</code> view
Rows to include	Choose <code>All rows</code>
Paged data display	Deselect the box - Categorizing works better without paging

2. Add a Category View builder to create the expand / collapse features.

- a. In the Builder Call List, click the , choose `Category View` builder and click **OK**.
- b. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.

Input Name	Input Value
Name	<code>Categories</code>
Container Field	Click the  and choose <code>[EmployeesviewDataViewPage] DataPage/ViewData/Row</code>
Categories Selection	Choose <code>Use Domino category rows</code>

3. Test the model.

- a. Run the model using the  icon.
- b. In the web browser, you should see arrows next to each `WorkDept` ID. Click the arrows to expand and collapse categories. Note that more than one category can remain open at a time unlike Domino-rendered HTML views which can only have one category open at a time.

Using a Flat Domino View

Categorizing a flat view is a great feature because it allows the data to be displayed in categories when no Domino view exists with the same categorization. It takes a little bit more work to categorize a flat view but it has some advantages. First, if there is no existing view with the desired categorization, the view can be categorized without creating a new view in the Domino database. Second, it is generally better to use a flat view with Domino View and Form because the generated schema is simpler. The ability to categorize a flat view offers the power of categorization and the convenience of using a flat view. Finally, since the categorization work is being performed by Portlet Factory instead of Domino, there are more opportunities to extend the categorization features via Profiling.

In this section, you will switch back to the `All Employees` view because it is a flat view.


1. Edit the Domino View & Form builder call using the details provided in this table. Click **Apply** to save your changes when finished.

Input Name	Input Value
View name	Choose <code>AllEmployees</code> - this value is the alias for the <code>All Employees</code> view
Rows to include	Choose <code>Document rows only</code>

2. Edit the Category View builder call using the details provided in this table. Click **Apply** to save your changes when finished.

Input Name	Input Value
Categories Selection	Choose - Specify one or more columns by which to group the data
Categories Data	In the <code>Type</code> column on the <code>WorkDept</code> row, choose <code>Category</code>
Categorization	Choose <code>Auto: categorize the data on first load</code>


3. Test the categorization.


- a. Run the model using the  icon.
- b. Click the arrows to expand and collapse categories. Note that although the categorization works, the view does not look as good as it could because the categorization is in a middle column.

4. Add a Data Column Modifier.

As its name suggests, a Data Column Modifier is a modifier builder. Modifier builders are used to make adjustments to the organization and presentation of schema-typed data such as the view data made available via a Domino View and Form builder. It is common to use integration and Page Automation builders to capture data and then use modifier builders to enhance the organization and presentation of that data.


A Data Column Modifier builder can change the display order of the columns, among other critical tasks. This builder can reorder the columns to resolve the problem seen in the last test of the application where the categorization appears in a middle column. Sorting via column headers can also be enabled using Data Column Modifier. **Note:** When using categorization and column sorting, it is best to add sorting to all categorized columns to permit resorting of the categories once the view has been sorted by some other column.

- a. In the Builder Call List, click the , choose `Data Column Modifier` builder and click **OK**.
- b. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.

Input Name	Input Value
Name	<code>mainViewFormatting</code>
Container Field	Click the  and choose [<code>EmployeesviewDataViewPage</code>] <code>DataPage/ViewData/Row</code>

- c. Adjust the column order and add column sorting.
 - i. In the `Settings for Add and Delete Row Operations` section, locate the table below the `Manage Columns` field.
 - ii. In the `Column Name` column, use the mouse to drag the `WorkDept` row to a position immediately above the `EmpNo` row.
 - iii. On the `WorkDept` row, change the value of the `Column Sort Type` to `Case Insensitive String`.
 - iv. On the `EmpNo` row, change the value of the `Column Sort Type` to `Case Insensitive String`.
 - v. On the `LastName` row, change the value of the `Column Sort Type` to `Case Insensitive String`.
 - vi. On the `HireDate` row, change the value of the `Column Sort Type` to `Date`.

5. Test these changes.

- a. Run the model using the  icon. The categorizing and column sorting should work and the `WorkDept` column should be the left-most column.
- b. Click a column header to sort the view on that column's data. Note that the categories still group the data unlike Domino-rendered HTML views.
- c. Click the `WorkDept` column header to reestablish the categorization. Now you should see the benefit of having added sorting to this column.

Working with Domino Document Data

In addition to displaying view data, Domino View and Form can display document data. This powerful builder is remarkably easy to use for reading, editing, creating and deleting Domino documents via web applications and portlets. By enabling document and form support, these features become available by default and require very little additional work on the part of the portlet developer.


Document and form support relies on a schema derived from the Domino form indicated in the `Form name` builder call input. The Domino View and Form builder examines the Domino form and uses all of the fields on that form to build the document schema. This schema is then used for displaying documents for reading

and for constructing forms for creating and editing Domino documents.


Reading Documents

As you have seen already, a Domino View and Form builder can display a table of Domino view data very easily. The next logical step is to add the ability to open individual records and display them using some layout controls. Domino View and Form automatically adds a link to each individual document represented in the view data when document and form support is enabled. These links can be added to any column in the view.

There are two ways to display documents using a Domino View & Form builder. One way is to use the Domino Web Rendering Engine to display the document in a separate window. The other way is to retrieve the raw data from Domino and use Portlet Factory Page Automation to generate JSPs and HTML pages. The objectives of the application design generally determine which method to use. In this section, you will use both methods in order to understand the differences.

1. Use the Domino Web Rendering Engine to display the documents.
 - a. Open the Domino View & Form builder call.
 - b. Locate the `Document and form support` section.
 - c. In the `Document support` input, choose `Open Domino window`.
 - d. In the `Document link column` input, choose `EmpNo`. This option adds links on the values in the `EmpNo` column. When clicked, each of these links opens the indicated document.
 - e. Click **Apply** and save the model.
2. Test these changes.
 - a. Run the model using the  icon.
 - b. Expand any category. You should now see a link on every number in the `EmpNo` column in the view. **Hint:** If you prefer to suppress the categorization, you can right-click the `Category View` builder call in the Builder Call List and choose `Disable`. Some warnings will appear in the `Problems` tab, but the conditions producing those warnings are not critical.
 - c. Click any of these links to open the document. By examining this URL and the content of the pages, you can see clearly that the pages come directly from Domino. **Hint:** Domino URLs contain the Domino server host name, the path and filename to the Domino database, and a reference to the specific document or design element being opened. In this case, the reference is the document's unique ID. For example, `http://hostname/DominoTutorialDatabase.nsf/0/A239DEC0934AF459CC098D7624789FE8`.
 - d. Close any open browser windows and switch back to Portlet Factory Designer.
3. Use WebSphere Portlet Factory Page Automation to display the documents.
 - a. Locate the `Document and form support` section in the Domino View & Form builder call.
 - b. In the `Document support` input, choose `Create pages from Domino form`.
 - c. In the `Form name` input, choose `Employee`.
 - d. In the `Document page type` input, choose `Create fields based on Domino form`, which is the default value. This option tells the Portlet Factory to build schemas for handling the documents based upon the fields defined on the Domino form chosen in the `Form name` input. The other option, `Use fields from imported HTML`, tells the Portlet


Factory to generate schemas based upon the fields defined in a separate HTML page.

- e. In the **View Document** section, there is an input labeled **Document view link**. In this input, choose **EmpNo** in order to add a to the view for each document. These links open individual employee documents.
 - f. Click **Apply** and save the model.
4. Test these changes.
- a. Run the model using the  icon.
 - b. Expand any category. You should now see a link on every number in the **EmpNo** column in the view.
 - c. Click any of these links to open the document. By examining this URL and the content of the pages, you can see clearly that the pages come from WebSphere Portlet Factory Page Automation.
 - d. Close the browser window and switch back to Portlet Factory Designer.

Editing / Creating / Deleting Documents

The schema generated for reading documents is also used for editing and creating them. Portlet Factory Page Automation uses this schema to generate forms with HTML inputs. By now, you should have noticed that documents opened using Page Automation include buttons to edit and delete documents and perform related navigation tasks. These options are easy to add as it requires nothing more than enabling them via checkboxes in the Domino View & Form builder call. Disabling any of these options removes the buttons and the supporting artifacts from the model.

In this section, you will edit, create, and delete employee records using the generated forms.

1. Open the Domino View & Form builder call. Examine the sections for editing, creating, and deleting documents. Notice that these options can be enabled or disabled via checkboxes. Since these features are enabled by default, there is nothing to change at this point.
2. Test the editing capabilities.
 - a. Run the model using the  icon.
 - b. Open any document.
 - c. Click the **Edit** button, make some changes, and save them by clicking the **Save Changes** button.
 - d. After submitting these changes, the document will reload and will contain the changes you just made.
3. Test the creating capabilities.
 - a. Click the **Back to View** button to return to the view.
 - b. Click the **Create new...** button. A form containing the fields defined on the **Employees** form in Domino will appear.
 - c. Fill out this form and save the new document by clicking the **Create Document** button.
 - d. After submitting these changes, the view of employee data will be displayed. Locate the document you just created and open it by clicking its **EmpNo** link. This document should contain the field values you entered earlier. Leave this document open in the browser as you will return to it in a moment.
 - e. Switch back to the Notes client and look for this document in the **Domino Tutorial Employees** database. You may need to press the F-9 key to refresh this view in order to find the document. If you open and examine this document, you will see that it is no different

in structure or design than any other document added to the database via a Lotus Notes client.

4. Test the deleting capabilities.
 - a. Switch back the web browser window containing the newly created document.
 - b. Click the **Delete** button. You should see a confirmation alert window. This alert window is controlled by the **Confirm deletion** option in the **Delete document** section of the **Domino View & Form** builder call. After confirming the deletion, the view of employee data will appear in the browser again. The document will not appear in this view. You can confirm that the document has been removed from the Notes database by examining the **Domino Tutorial Employees** database via a Notes client.

Working with other Domino Collaboration Builders

The Domino View builder and the Domino View and Form builder contain Java helper objects that provide access to the Domino API. When a model contains either of these builders, it is easy to use Method builders and Linked Java Objects to invoke methods to get certain Domino objects including a Domino session object. There are additional Domino Collaboration builders in Portlet Factory that also leverage this access to the Domino API. Among these additional builders are Domino Keyword Lookup and Domino Formula. In this section, you will work with these two builders in order to become familiar with them.

Using Domino Keyword Lookup

It is common in HTML forms to have list fields like select, radio button, and checkbox. It is also common to design these types of fields such that each choice in the list has a plain text label that is meaningful to the end user and a corresponding hidden value that is meaningful to the application code. These label-value pairs are often stored in database tables. Lookup Table builders are used retrieve these pairs and make them available to certain other builders such as Select and Radio Button Group.

A Domino Keyword Lookup builder is a specialized variant of a Lookup Table builder than can retrieve the label-value pairs from a view in a Domino database. Since a Domino Keyword Lookup builder contains a Lookup Table builder, it is used by other builder calls in the same manner in which regular Lookup Table builders are used. For example, the list of label-value pairs can be applied to a field on an input form via a Data Field Modifier builder to produce a drop-down list of choices. These label-value pairs can also be applied to fields on read-only forms in order to translate between the value stored in the document and its corresponding label.


In this section, you will add a Domino Keyword Lookup builder call to the model. Its function is to build label-value pairs from the **Lookup > Departments** view. This view has a column of department names and another column of department IDs. The names will be used for the labels and the IDs will be used for the values.


1. Examine the **Lookup > Departments** view in Lotus Notes.
 - a. Open the **Domino Tutorial Employees** database in a Lotus Notes client.
 - b. Open the view called **Lookup > Departments**. You will see that this view has one column containing the names of the departments (**Dept Name**) and another containing the ID numbers (**Department ID**) of the departments, among other columns. The

Domino Keyword Lookup builder call will use this view to build label-value pairs that will be used by Page Automation for display and input of department names in employee records.


2. Add a Domino Keyword Lookup builder call.

This builder has an input labeled `Use existing view` which refers to the existence of a Domino View or Domino View & Form builder call within the same model. The idea is that if the Domino Keyword Lookup builder gets its label-value pairs from a Domino view being accessed one of these Domino builders, then the `Use existing view` option should be enabled. If the label-value pairs come from a different view in the Domino database, then this option should be disabled. When disabled, additional builder call inputs appear to allow the developer to specify which view in which Domino database contains the source data for the label-value pairs.

- a. In the Builder Call List, click the , choose `Domino Keyword Lookup` and click **OK**.
- b. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.


Input Name	Input Value
Name	<code>lookupDepartments</code>
Data provider	<code>View</code>
Use existing view	Deselect this option. You must choose a different Domino view than what is specified in the Domino View & Form builder call.
Database name	Click the Get databases and views button then click  . Choose <code>DominoTutorialEmployees.nsf</code>
View Name	Choose <code>LookupDepartments</code> - this value is the alias of the <code>Lookup > Departments</code> view
Rows to include	Choose <code>Document rows only</code>
Value Tag	Choose <code>Department_ID</code> - this value refers to the <code>Department ID</code> column
Label Tag	Choose <code>Dept_Name</code> - this value refers to the <code>Dept Name</code> column

3. Apply the Lookup Table to the `WorkDept` field on the create page.

- a. In the Builder Call List, click the , choose `Data Field Modifier` and click **OK**.
- b. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.

Input Name	Input Value
Name	<code>departmentNames</code>
Field Selector	<code>Select by Name</code> - this is the default value
Tool	

Fields


Click  and choose [EmployeesDocumentCreate] EmployeesDocumentCreate/DocumentItems/WorkDept

Field Settings section

Choose lookupDepartments

Lookup Table Used

4. Test the create page.


- a. Run the model using the  icon.
- b. In the web browser, click the **Create new...** button. A form for creating employee records will open. Notice that the `WorkDept` field now has a select list of choices built from the `Lookup > Departments` view in the `Domino Tutorial Employees` database.

5. Examine the HTML for this page.

- a. Right-click the page and choose `View Source` or use whatever command is necessary to view the HTML depending upon which browser you are using.
- b. Find the HTML for the `WorkDept` field. You will see that there is a `<select>` element containing a series of `<option>` elements. In each `<option>` element, you can see the value and the corresponding label. Here are just a few of the `<option>` elements you should be able to see.

```
<option value="D21">Administration Systems</option>
<option value="A00">Computer Services</option>
<option value="D01">Development Center</option>
```

6. Apply the Lookup Table to the other pages in the model.

- a. Open the Data Field Modifier builder call named `departmentNames`.
- b. In the `Fields` input, use the  to add the following page references. In short, you will choose the `WorkDept` field for every remaining page. in the model.

```
[EmployeesDocumentEdit]EmployeesDocumentEdit/DocumentItems/
WorkDept
[EmployeesviewDataViewPage]DataPage/ViewData/Row/WorkDept
[EmployeesDocumentView]EmployeesDocumentView/DocumentItems/
WorkDept
```

- c. Click **Apply** and save the model.

7. Test these changes.

- a. Run the model using the  icon.
- b. Look at the `WorkDept` column in the view. Note that the department IDs are no longer

displayed. Department names are used instead. This behavior is an improvement over Domino-rendered view as keyword synonyms do not work this way.


- c. Open any document and note that the `WorkDept` field displays the department name instead of the ID.
- d. Click the **Edit** button and note that the `WorkDept` field now has a select list of choices that are the same as the select list on the create page.

Using Domino Formula

It is common in Domino form development to use Computed Text and Computed for Display fields to build text expressions that combine values from several fields to enhance the organization and display of documents. For example, a complete mailing address is stored using several fields but is often displayed using a single Computed for Display field by concatenating the values from those fields.

Domino Formula builders are used in much the same way as Computed Text and Computed for Display fields. This builder type supports formulas written using the Lotus Notes formula language. These formulas can access field values on documents given the document's UNID. Domino Formula builder calls are added to named controls on pages to designate where on the page the formula results should appear. The most common use of this builder is to display field values from the currently displayed document. By entering `${Variables/EmployeeSelectedUNID}` into the Document UNID builder call input, the formula will use the field values from the current document.

1. Add a Domino Formula builder call.


- a. In the Builder Call List, click the , choose `Domino Formula`, and click **OK**.
- b. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.

Input Name	Input Value
Name	<code>pageHeader</code>
Location	Choose <code>Relative to Named Tag</code>
Technique	
Page	Choose <code>AllPages</code>
Tag	Choose <code>data</code> - this builder call will be applied only to those pages containing an element named <code>data</code>
Placement	Choose <code>Before</code>
New Tag Name	Leave it blank
Domino connection helper	Choose <code>Employeesview</code>
Document UNID	<code>\${Variables/EmployeeSelectedUNID}</code> - this value should fill in by default

Formula Enter the following formula. This formula will display the title of the Domino database, the full name of the employee followed by the employee's job title. The entire text results will appear boldfaced in the web browser.

```
"<b>" + @DBTitle + " - " + Firstnme + " " +
LastName + " - " + Job + "</b>"
```

2. Test the Domino formula.

- a. Run the model using the  icon.
- b. Open any document. Note that there is now a text string at the top of the page containing the results of the formula. For example,

Domino Tutorial Employees - JAMES WALKER - DESIGNER appears at the top of the employee record for James Walker

- c. Click the **Edit** button. Note that the same text string also appears in the edit mode.

3. Close all open builder calls to clean up the workspace before continuing.

Using Domino with Service Oriented Architecture

As you have seen thus far, Domino View and Form is a very powerful high-level builder. With very little work on the part of the portlet developer, Domino view data can be retrieved and displayed. It is also very easy to include create, read, update, and delete features in a Domino-based portlet.

Domino data can also be retrieved and edited using the Service Oriented Architecture (SOA) capabilities of WebSphere Portlet Factory. SOA offers the advantages of a clear separation between the presentation and data access layers of n-tier enterprise applications. SOA is particularly useful when several applications need to use the same data. In this case, it is far better to create broadly available data services and consume them in separate portlets than it is to put redundant data access code in multiple portlets.

Many Domino applications need lists of user names from the Domino Directory. One approach to retrieving this data would be to include a Domino View builder in every portlet to retrieve these names. A better approach would be to create a Service Provider model that retrieves the names from the Domino Directory and makes those names available via a service. Each portlet needing the names would contain a Service Consumer builder to get the data from the Service Provider. The latter approach is more efficient and easier to manage as there is only one point of access to the back end Domino data. Domino services are conceptually the same as any SOA services which provide services for data retrieved from any other back end data source.


In this section, you will use SOA to get a list of the user names in the Domino Directory by creating a Service Provider model. This model will expose this data as a service. Then you will add a Service

Consumer builder to the DominoEmployees model to retrieve the names from the Service Provider. Finally, you will apply this list of names to a field in the edit and create pages. The results will give the portlet user an easy way to choose names from the Domino Directory.

Creating the Domino Service Provider

In this section, you will create a Domino Service Provider that retrieves a list of fully distinguished user names from the Domino Directory. The Service Provider will use the (`$VIMPeople`) view to retrieve these names. Although all of the columns of this view will be accessible to Service Consumers, you will only use the values found in the FullName column.

You will also have a chance to become familiar with the Service Provider model by using its built-in testing features.

1. Run the New Model Wizard to create Domino Service Provider.
 - a. Choose from the File menu `File > New > WebSphere Portlet Factory Model`.
 - b. Select the project you have been working in during this tutorial and click **Next**.
 - c. Choose the `Service Providers > Domino Service Provider`, wizard and click **Next**.
 - d. Enter `DominoDirectory` for the service name and click **Next**.
 - e. Click  and choose `names.nsf` as the database name. The wizard will query `names.nsf` and return a list of view names. Some of the view names are rather long. As a result, you may have to scroll to the right in the New Model Wizard dialog box in order to choose from this list of view names.
 - f. Choose (`$VIMPeople`) from the list of view names. This view is one of the most commonly used views in the Domino Directory for looking up fully distinguished Notes user names. Click **Next** to continue.
 - g. In the Create Service Operations page of the wizard, enable only the following services:
 - `Operation to read the Domino view`
 - `Operation for reading a Domino document`


Note: It generally does not make sense to allow modification or creation of Person documents in the Domino Directory via non-Domino systems for security and user management reasons, thus, it probably does not make sense to enable services that provide these functions.

- h. In the `Document schema form` input, choose or enter `Person` and click **Next** to continue. **Hint:** You may have to scroll very far down the list to find `Person`. It may be simpler to enter `Person` via typing instead.
- i. In the Save New Model page of the wizard, select the `models/tutorials/domino` folder and enter `DominoDirectory_SP` for the model name.
- j. Click **Finish** to complete the wizard and create the model. After completion, the `DominoDirectory_SP` model will open automatically.

2. Examine the contents of the Service Provider model to become familiar with it. In the next step, you will run it and test it.

A Domino Service Provider model must contain three essential builder calls: Service Definition, Service Operation, and Domino View. Below are brief descriptions of each of these builder calls:

- Service Definition - provides the framework for exposing the services declared in the Service Provider to the Service Consumers. There are options for creating a stub service provider and stand-alone testing support.
- Service Operation - defines an individual service's name and which method of the Domino View builder the service will call when invoked by the Service Consumer. Although the Domino View builder tells the Service Operation builder what the default input and output schemas are, these schemas and their input mappings can be overridden for finer control over the service's interface. Pre- and post-processing of the input/output data can also be declared here. Every service must have a separate Service Operation builder call.
- Domino View - performs the data "get" and "set" tasks with the designated Domino server and database on behalf of the Service Operation builder call. Technically, a Domino View and Form builder can be used but it adds unnecessary overhead.

3. Run the `DominoDirectory_SP` model using the  icon in order to test the services it provides.
 - a. By default, service testing features are enabled. In the web browser, there should be one link for each service in the service provider model. Click the link labeled `DominoDirectoryReadView`. You should see a table of data containing the person documents in the Domino Directory.
 - b. Select the UNID of one of these documents and copy it to the clipboard.
 - c. Click the **Back** button so that you can run the other service.
 - d. Click the link labeled `DominoDirectoryReadDocument`.
 - e. In the `UNID` field, paste the data from the clipboard.
 - f. Click the **Submit Query** button to run the service that retrieves the document with the UNID you just pasted. That document will open and will display all of the fields found on the `Person` form in the Domino Directory.

Note: You will notice that the list of fields is very long. This list is built entirely from all of the fields on the Person form except the Computed for Display fields. In a production environment, you may not need all of these fields. A simple technique for specifying which fields to include in a form schema is to create a separate form in the Domino database and add to it only the fields needed for the schema. Additional computed fields that need to become part of the document can be placed on a computed subform. These fields will compute when saving the document but will not be part of the document schema.


- g. Close the browser window when you are finished testing the services.
3. Switch back to Portlet Factory Designer. Close the `DominoDirectory_SP` model to avoid editing it by accident.


Consuming the Domino Service


Service Provider models access data and make that data available via Service Oriented Architecture (SOA). Service Consumer models invoke the services of the Service Provider, receive the data produced

by these services, and display it usually via Page Automation. In order for a model to function as a consumer, it must contain a Service Consumer builder call. Inside the Service Consumer builder call, one indicates which Service Provider to use, which services the consumer will use, and the sources of inputs for those services that require them.

In this section, you will setup the consumer model so that it invokes the Domino Directory service and uses the results of that service to populate a list of choices for the `Approver` field.

1. Add a Service Consumer builder call to the `DominoEmployees` model.
 - a. Open the `DominoEmployees` model. **Note:** Ensure you are not working in the `DominoDirectory_SP` model.
 - b. In the Builder Call List, click the , choose `Service Consumer`, and click **OK**.
 - c. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.

Input Name	Input Value
Name	<code>approverNames</code> - the service will be used to provide a list of names for the Approver field in the employee records
Provider Model	Click  and choose <code>tutorials/domino/DominoDirectory_SP</code>
Add All Provider Operations	Deselect this option. This consumer only needs the <code>ReadView</code> service as it will only use the list of names from the <code>(\$VIMPeople)</code> view
Operation Name	Choose <code>DominoDirectoryReadView</code>

2. Apply the service results to a Select builder.
 - a. In the Builder Call List, click the , choose `Select`, and click **OK**.
 - b. Use the details provided in this table to fill out this builder call. Click **Apply** to save your changes when finished.


Input Name	Input Value
Name	<code>approversOnCreatePage</code>
Location	Choose <code>On Named tag</code>
Technique	
Page	Choose <code>EmployeesDocumentCreate</code>
Tag	Choose <code>Approver</code>
Lookup Table Used	Choose <code>None</code>
Select Data	Choose <code>\${DataServices/approverNames/DominoDirectoryReadView/results/ViewData}</code>

XML Data

Source

Elements Section


Value Element Enter `Row/FullName`Label Element Enter `Row/FullName`3. Test the `Approver` field on the create page.






- a. Run the model using the  icon.
- b. Click the **Create new...** button to bring up an input page.
- c. Examine the `Approver` field at the bottom of the page. Note that although a select field is present, it does not contain a list of names. There are no names because the service is not being invoked when the model is run. You will fix this problem in a moment.

4. Invoke the `DominoDirectoryReadView` service via the `main` method.

The Domino View and Form builder has an option to generate a `main` method. In any model, the `main` method runs automatically after the model loads. The `main` included in Domino View and Form performs two essential tasks: it runs the service that gets the view data and then loads the `employeesviewDataViewPage` page. This method does not run the `DominoDirectoryReadView` service which provides the list of names for the `Approver` field because that service is outside the scope of what a Domino View and Form builder is supposed to do. To populate the select list in the `Approver` field, however, this service must be invoked as part of the model loading process.

There are several techniques for invoking this service as the model loads. An easy approach is to replace the `main` method wrapped inside the Domino View and Form builder with an Action List called `main`. This Action List must perform the tasks found in the `main` method it replaces and invoke the `DominoDirectoryReadView` service. In this section, you will use this technique to resolve the problem found in the last step where the list of names in the `Approver` field was unpopulated.

- a. Open the Domino View & Form builder call.
 - i. Scroll to the bottom and expand the `Advanced` section.
 - ii. Deselect the option labeled `Generate Main`. You will soon add a new Action List builder call named `main`. A model cannot have two Action List builder calls called `main`, thus, you must disable this option to prevent Portlet Factory from creating an Action List inside the Domino View and form builder call in addition to the one you are about to create.
 - iii. Click **OK** to save and close this builder call.
- b. Add an Action List builder call.
 - i. In the Builder Call List, click the , choose `Action List`, and click **OK**.
 - ii. Name the builder call `main`.

- iii. In the **Actions** input, click the  and choose `DataServices/Employessview/readTable`. This action invokes the service that returns the data used to display the table of employee data.
 - iv. In the **Actions** input, click the  and choose `DataServices/approverNames/DominoDirectoryReadView`. This action invokes the service in the `DominoDirectory_SP` model that returns the list of names in the Domino Directory.
 - v. In the **Actions** input, click the  and choose `EmployeesviewDataViewPage` from the **Pages** folder.
 - vi. Click **OK** to save and close this builder call.
- c. Test again.
- i. Run the model using the  icon.
 - ii. Click the **Create new...** button to bring up an input page. Note that this time the list of names in the `Approver` field is populated.
 - iii. Click the **Cancel** button to return to the view of employee data.
 - iv. Open any document by clicking an `EmpNo` link.
 - v. Click the **Edit** button to view the document in the edit mode. Note that the `Approver` field has no list at all. In a moment, you will add a `Select` builder call that will create a list and apply the names from the Domino Directory.
5. Apply the list of approver names to the edit page.
- a. In the builder call list, locate the `Select` builder call named `approversOnCreatePage`.
 - b. Right-click this builder call and choose **Copy**.
 - c. Right-click this builder call and choose **Paste** to create an exact copy of the `approversOnCreatePage` builder call. You may notice a warning in the **Problems** tab about one builder call overwriting the effects of another builder call. This warning exists because you now have two builder calls that add select lists to the same named element on the same page. You will correct this condition in the next step.
 - d. Edit either of the `Select` builder calls to apply it to the `EmployeesDocumentEdit` page.
 - i. Open one of the `approversOnCreatePage` builder calls.
 - ii. Change the name of the builder call to `approversOnEditPage`.
 - iii. Change the value of the `Page` input to `EmployeesDocumentEdit`. The idea is to recreate the exact same builder on the edit page. When finished, there will be identical select lists on both the create and edit pages.
 - iv. Click **OK** to save these changes. Note that the warnings in the **Problems** tab have gone away.
6. Test the edit page.
- a. Run the model using the  icon.
 - b. Expand any category and then open any document.
 - c. Click the **Edit** button on that document to open it in the edit mode. Note that there is now a list of Notes user names in the `Approver` field.
 - d. Choose a name for the `Approver` field.
 - e. Click the **Save Changes** button to submit this document. You should see this document in

the read mode now. Note that the name you chose is now stored in the `Approver` field.

Running Domino Agents

Domino application developers should be familiar with a concept called Web Query Save agents. A Web Query Save agent acts upon a document during the "submit" process. Generally, Web Query Save agents are fairly simple agents that perform some basic post-processing tasks. Each Domino form can contain a reference to one of these agents. When a document is submitted via HTTP, as is the case when using Domino-rendered web applications, the form used to submit the document knows which agent to run. This process also passes the document context of the document being submitted so that the agent runs against the correct document. WebSphere Portlet Factory accesses Domino via DIOP, therefore, it uses a different mechanism for triggering post-processing agents. The net effect is the same as a Web Query Save agent but the process and the agents are different.

The Domino View and Form builder allows agents to be invoked in conjunction with submission of a new documents and edited documents. There is a bit more flexibility with the Domino View and Form builder than with Web Query Save agents as two different agents can be invoked: one for new documents and a different one for edited documents.

In order for these agents to work properly, they must be designed according to the following specifications:

- The agent must be triggered via "On Event" using "Action menu selection" or "Agent list selection"
- Target documents must be "None".
- Agent must be LotusScript or Java. Formula language agents will not work.
- The NoteID of the document will be passed by Portlet Factory automatically. This ID must be retrieved in the agent using the `ParameterDocID` property.
 - LotusScript - `NotesAgent.ParameterDocID`
 - Java - `NotesAgent.getParameterDocID()`
- Get the document using the `GetDocumentByID` method and not the `GetDocumentByUNID` method.
 - LotusScript - `NotesDatabase.GetDocumentByID(NoteID)`
 - Java - `Database.getDocumentByID(String noteid)`

There is also a security element that must be considered when trying to run agents via DIOP: agent signing. For an agent to run on a Domino server, it must be signed by a user with authority to run that type of agent. The agent signer must also have sufficient rights to the database to perform the tasks contained in the agent. For example, if the agent is signed by a user with only Reader access to the database and the agent is designed to make changes to a document, the agent will not run properly because the agent signer does not have permission to make document changes. In the Domino Setup section at the beginning of this tutorial, you were instructed to ensure that the agent was signed properly.

Cleaning up Documents after Submit

You may notice that the name in the `Approver` field is stored as an abbreviated, fully distinguished name and not as a canonicalized name: `Anita Holiday/ABC` instead of `CN=Anita Holiday/O=ABC`. The Notes datatype of the `Approver` field is `Names`. This datatype automatically stores names in canonical

format when the document is saved via the Notes client or via a Domino-rendered web application. When the document is saved via DIOP, however, names do not get canonicalized even when the `ComputeWithForm` option is enabled which it is by default in WebSphere Portlet Factory.

To get these names canonicalized, there is an agent provided in the Domino Tutorial Employees database called `FixApproverName`. Running this agent automatically after submitting either a new document or a document updated will canonicalize the names in the `Approver` field. In this section, you will modify the Domino View and Form builder call to invoke this agent when submitting new or edited documents. Essentially, the problem is that the documents need additional processing after they are submitted.


1. Understanding the problem.

- a. Open the `Domino Tutorial Employees` database in a Lotus Notes client and locate the employee document you edited in the last section.
- b. Right-click the document to examine its properties.
- c. Click the `Fields` tab and examine the `Approver` field. Note that the value of the field is a name without the `CN=` or `O=` component labels although the field has a `NAMES` flag. **Hint:** Hover your mouse cursor over the tabs in the document properties info box to see the tab names.
- d. Double-click the document to open it.
- e. Click the **Edit Employee** button to switch to the edit mode.
- f. Without making any changes, click the **Save & Close** button to refresh, save, and close this document.
- g. Right-click the document to open the document properties info box again. This time, you should see that the value of the `Approver` field now contains the `CN=` and `O=` component labels. This name has now been canonicalized. **Note:** For a regular Names field, it is usually not much of a problem if the names are not canonicalized. It does matter Readers and Authors fields as these field types are security-related. Consequently, it is important to learn how to resolve this problem.
- h. Switch back to WebSphere Portlet Factory Designer when you are finished examining the `Approver` field.

2. Modify the Domino View & Form builder to run the `FixApproverName` agent when documents are edited or created.

- a. Open the Domino View & Form builder call.
- b. Scroll down to the `Edit Document` section.
- c. In the `Agent to run on save` input, choose `FixApproverName`.
- d. In the `Create Document` section, there is an input labeled `Agent to run on document creation`. For this input, also choose `FixApproverName`.
- e. Click **OK** to save these changes.

3. Test to ensure the agent runs and fixes the names upon save.

- a. Run the model using the  icon.
- b. Open a different document than the one you edited previously. Be sure to check the `Approver` field before editing the document to ensure that there is no value in that field.
- c. Click the **Edit** button to edit this document.
- d. Choose a value for the `Approver` field.
- e. Submit the document by clicking the **Save Changes** button. You should now be able to see the results of the edit. The value in the `Approver` field should now be canonicalized. If not, something is probably wrong with the permissions to run the agent. Any problems with the

agent should appear at the top of the document screen.

Summary

Data in Domino-based applications can be accessed either directly in applications or via Service Oriented Architecture using builders in the Lotus Collaboration Extension feature set. The Domino View and Form builder provides simple options to enable reading, editing, creating, and deleting of documents in Domino databases.

Once retrieved, standard Portlet Factory Page Automation techniques can be used to manipulate and display the data. Modifier builders such as Data Column Modifier apply to Domino data exactly the same way as they apply to data from other sources. View data can be categorized easily using a Category View builder.

There are also a few additional Domino-related builders designed to incorporate Domino data easily into an application. These builders include Domino Keyword Lookup and Domino Formula, among others. The Domino Keyword Lookup builder is a specialized Lookup Table builder used to retrieve label-value pairs from Domino views in order to use these pairs in HTML UI elements such as radio buttons and select lists. The Domino Formula builder uses Lotus Notes formula language expressions to function much like Compute Text or Computed for Display fields do.

Web Query Save agents are very common in Domino-rendered web applications. The Domino View and Form builder includes options to run specific agents as part of the submit process for new and updated documents. Although the mechanism for invoking these agents is different in Portlet Factory applications, the results are the same.