

**Database Forms:
Adding Persistence to Web-based Applications**

By Larry Smith, Co-founder, Nouveau Systems

Contents

Introduction	2
Problem Statement	2
Previous Options	2
Our Solution	2
Implementation	3
Installation	16
Demonstration	17
Customization	20
Summary	23

Introduction

This white paper describes how to create persistent, web-based applications with a minimum amount of effort.

Problem Statement

The problem is how do you solicit and store information from a web page and provide web-accessible query and update capabilities to use and maintain this data.

The main challenge of this problem is to provide a solution that supports the modeling features of persistence, but is packaged in way to facilitate its use by business process modelers versus information technology developers.

Previous Options

Currently, persistence is typically added to web-based applications via PHP or ASP technologies.

PHP is a reflective programming language originally designed for producing dynamic web pages, but is now used mainly in server-side scripting

Active Server Pages (ASP) is a server-side script engine from Microsoft used to support dynamically-generated web pages.

Generally, these implementations involve imposing technology and implementation-specific expertise.

Our Solution

With Nouveau Alliance, you can select an arbitrary HTML web page containing a form and have that form's elements automatically mapped to an embedded RDBMS database. The resulting database is subsequently populated, queried and updated via web pages backed by Nouveau Alliance workflow-enabled servlets.

Three features of Nouveau Alliance are highlighted in this paper:

Powerful Modeling

Nouveau Systems' FlowSpace Developer provides a library of primitives that allow you to specify an application in a simple, yet powerful way. If some customization is needed, you can add to or modify these primitives via their properties or augment their programming via Java or a myriad of supported scripting languages. When complete, these customizations can be added to the library of built-in primitives to define new applications.

Web-based Servlet Interface

You can enter your application logic graphically by defining workflows with FlowSpace. You then can use a Workflow Servlet to create a web application. This enables domain experts to define and modify business logic without extensive knowledge of web programming.

Embedded RDBMS

You can store and retrieve relational data from a full-featured, fully integrated RDBMS. Using the database access primitives of FlowSpace, you can seamlessly integrate the database access in your survey.

Implementation

The implementation of Database Forms is best described in the context of each of its five operations: Create, Populate, Query, Display, and Update the database.

The first operation is facilitated by a Java Application which imports, presents and exports SQL table definitions. The remaining four operations are implemented via workflow-enabled servlets.

Define the Database

This operation is responsible for taking an HTML form and mapping it to a table in an RDMS database.

It is implemented via a Java application named "form2sql" that is invoked using Java WebStart.

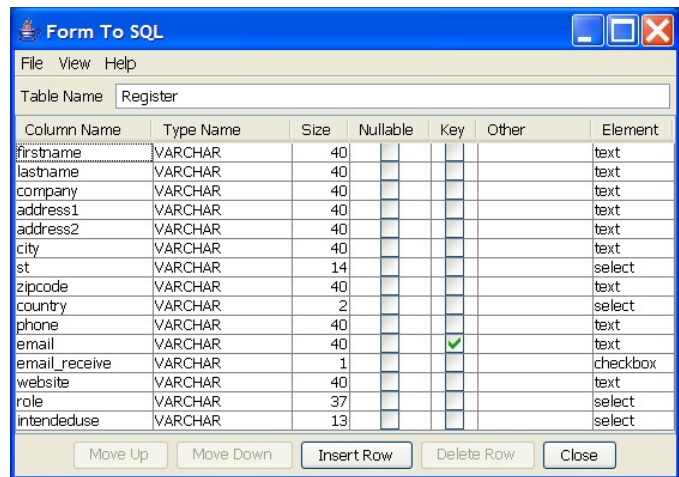


Figure 1 – form2sql application

This application imports a HTML form and displays a table that mirrors a RDBMS table. You can then add, delete, modify and reorder the fields in the table. When you are satisfied with the content of the table, you can create the table directly in the target database.

In order to support a coupling to the Database Forms workflow-enabled servlets, form2sql exports a one-line text file containing the name of the table and the names of the table fields. This "names" file has the following format:

```
tableName, [@,#,*]memberName[*]....
```

Each of the member names can be tagged as follows:

Leading @ - signifies a Select element

Leading # - signifies a Checkbox element

Leading * - signifies a Radio Button element

Training * - indicates a key field

This information enables the workflows to generate the appropriate SQL statements for the various operations.

Populate the Database

This operation is responsible for exacting data from an HTML form and storing the corresponding fields in the database. The following figure shows the Populate web page:

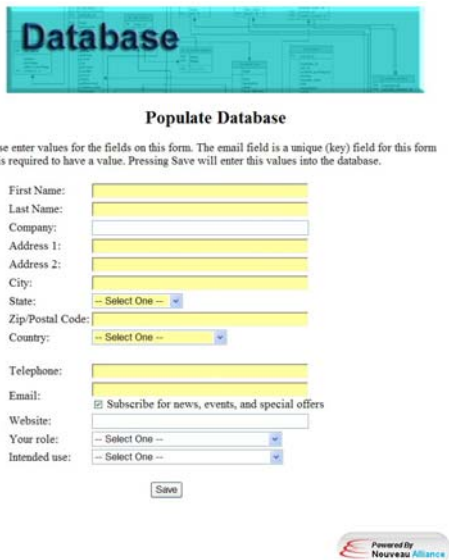


Figure 2 – Populate page

This page form’s action is the PopulateDatabase servlet with its corresponding workflow.

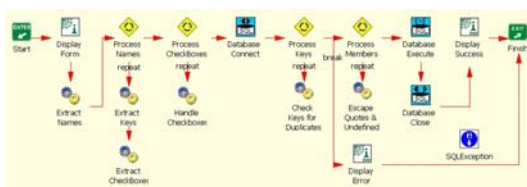


Figure 3 – PopulateDatabase workflow

For a better understanding, the workflow has been broken into two parts, with a listing of its constituent workflow nodes, activities and properties.

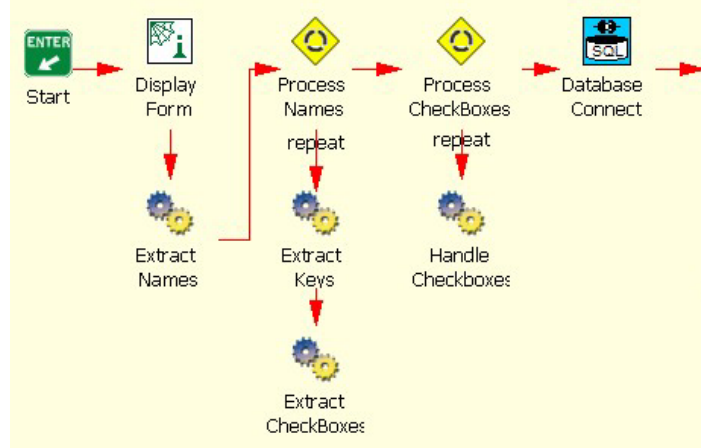


Figure 4 -- PopulateDatabase (front)

Display Form

This node is used to display the Populate web page. The URL for this page template is specified as an INSTANCE property in the Database Forms servlet configuration file.

```
SetPropertyFromTextFileActivityImpl
URLspecifier=$INSTANCE.templateFile
targetPropertyName=unformattedText
```

```
SetPropertyActivityImpl
formattedText="$unformattedText"
SERVLET.RESPONSE.print=$formattedText
close=$SERVLET.RESPONSE.close
```

```
WaitActivityImpl
waitTime=NULL
```

Note: The set property sequence of “SERVLET.RESPONSE.println and \$SERVLET.RESPONSE.close, followed by the indefinite Wait, causes the templated page to be displayed in the user’s default web browser. From this point on, this sequence of activities will be abbreviated by the aggregate activity label “DISPLAY_WEB_PAGE_ACTIVITES” and their custom property settings.

Extract Names

This node is used to read the “names” file created by the “form2sql” application and extract the table name, the form field names and a notation for the form’s values. These properties are used by the workflow to construct the needed database access statements. The INSTANCE.names property retains the name tags added by form2sql to distinguish key names, and element types (radio, checkbox, and combobox).

```
SetPropertyFromTextFileActivityImpl
URLspecifier=$INSTANCE.namesFile
targetPropertyName=INSTANCE.names
```

```
SetPropertyActivityImpl
firstComma=$INSTANCE.names.indexOf(",")
INSTANCE.tableName=
$INSTANCE.names.substring(0,$firstComma)
INSTANCE.memberNames=
$INSTANCE.names.substring($firstComma + 1)
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
INSTANCE.values="\$INSTANCE.$INSTANCE
.memberNames.replaceAll(",","","\$INSTANCE.")\"
```



Process Names

This node is used to iterate on the names extracted by the Extract Names nodes. This iterator is used to extract key names and checkbox names.

```
SetPropertyActivityImpl
INSTANCE.keyNames=[]
INSTANCE.instanceKeyNames=[]
INSTANCE.checkboxNames=[]
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
names=$INSTANCE.names
```



Extract Keys

This node is part of the repeat block for the Process Names node. It populates lists of table key names and INSTANCE key names by checking for names tagged with a trailing “*”.

```
CompareActivityImpl
value1=$CONTEXT.names.endsWith("")
value2=false
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
untaggedName=$CONTEXT.names
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
ADD_KEY=$INSTANCE.keyNames.add(
$untaggedName)
ADD_INSTANCE_KEY=$INSTANCE.instanceKeyNames.add("INSTANCE.$untaggedName)
```



Extract CheckBoxes

This node is part of the repeat block for the Process Names node. It populates a list of checkbox element names by checking for names tagged with a leading “#”.

```
CompareActivityImpl
value1=$CONTEXT.names.startsWith("#")
value2=false
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
untaggedName=$CONTEXT.names
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
ADD_CHECKBOX=$INSTANCE.checkboxNames.add(
$untaggedName)
```



Process CheckBoxes

This node is used to iterate on the list of checkbox element names extracted in the Extract CheckBoxes node.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
checkboxNames=$INSTANCE.checkboxNames
```



Handle Checkboxes

This node extracts a comma –separated list of values for each checkbox and sets each checkbox value to this CSV String. This String will be stores per checkbox name “grouping”.

```
SetPropertyActivityImpl
INSTANCE.$CONTEXT.checkboxNames=
$SERVLET.REQUEST.parameterValues[$CONTEXT.checkbox
Names].toCSV
```

```
CompareActivityImpl
value1=$INSTANCE.$CONTEXT.checkboxNames)
value2=true
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
INSTANCE.$CONTEXT.checkboxNames=""
```



Database Connect

This node is used to establish a connection to the database specified by driver connection URL, user and password and save the connection as an INSTANCE property for later use by other database access nodes in the workflow. All of the database parameters are specified by INSTANCE properties provided in the Database Forms servlet configuration file.

```
DBConnectActivityImpl
driver=$INSTANCE.databaseDriver
connectionProperty=INSTANCE.DB_CONNECTION
connectionURL=$INSTANCE.databaseConnectionURL
connectionUser=$INSTANCE.databaseUser
connectionPassword=$INSTANCE.databasePassword
```

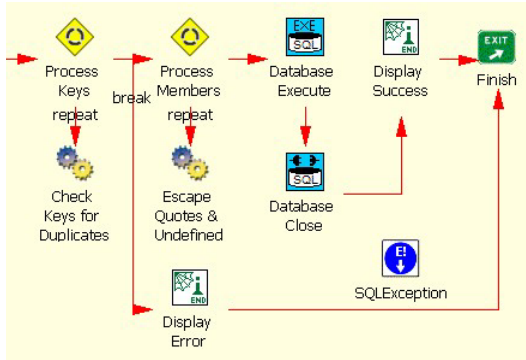


Figure 5 -- PopulateDatabase (back)



Process Keys

This node is used to iterate on the list of database key names extracted in the Extract Keys node. It also specifies the name of the “break” link to traverse if the return status id set to “true”.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
breakLabel=break
keyNames=$INSTANCE.keyNames
```



Check Keys for Duplicates

This node is used to check the current form for existing key values in the database before attempting to save the current entry. If the given SELECT statement detects a duplicate, the “breakStatus” property is set and the repeat loop execution is broken.

```
SetPropertyActivityImpl
breakStatus=false
keyName=$CONTEXT.keyNames
checkSelect="SELECT $keyName FROM
$INSTANCE.tableName WHERE $keyName =
${INSTANCE.$keyName}"

DBExecuteQueryActivityImpl
executeString="$checkSelect"
connectionProperty=$INSTANCE.DB_CONNECTION
resultSetProperty=INTERPRETER.DB_RESULT_SET

DBResultSetActivityImpl
resultSetProperty=$INTERPRETER.DB_RESULT_SE
T
propertyName=keyfield
returnStatusProperty=breakStatus
```



Process Members

This node is used to iterate on the list of database member names extracted in the Extract Names node.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
memberName=$INSTANCE.memberNames
```



Escape Quotes & Undefined

This node is used to check each member name for single quotes in their values and “escapes” them by replacing them with two single-quotes (to allow quotes to be used in SQL statement strings). In addition, If the property is undefined (via \$? Operator), such as for “unchecked” check boxes, a ‘0’ value is substituted.

```
CompareActivityImpl
value1=$?{INSTANCE.$CONTEXT.memberName}
value2=true
comparison=0
skipCount=1
```

```
SetPropertyActivityImpl
INSTANCE.$CONTEXT.memberName=0
```

```
SetPropertyActivityImpl
propertyName=
$INSTANCE.$CONTEXT.memberName
propertyValue="$propertyValue"
newPropertyValue=$propertyValue.replaceAll("'", "'")
INSTANCE.$CONTEXT.memberName=
"$newPropertyValue"
```



Display Error

This node is used to display a web page informing the user of a duplicate key name is present in the form.

```
DISPLAY_WEB_PAGE_ACTIVITES
INTERPRETER.message=There is already someone registered
with the email address ($INSTANCE.email). No entry made in
the database.
URLspecifier=
$INSTANCE.webhost/doc/examples/DatabaseForms/templates/p
opulate_fail_template.html
```



Database Execute

This node is responsible for creating and executing the SQL INSERT statement used to populate the database.

```
SetPropertyActivityImpl
INSTANCE.insertStatement=INSERT INTO
$INSTANCE.tableName VALUES($INSTANCE.values)
```

```
DBExecuteActivityImpl
executeString="$INSTANCE.insertStatement"
connectionProperty=$INSTANCE.DB_CONNECTION
returnStatusProperty=INTERPRETER.DB_RETURN_STATUS
```



Database Close

This node is used to close the connection opened by the Database Connect node.

```
DBCloseActivityImpl
connectionProperty=$INSTANCE.DB_CONNECTION
```



Display Success

This node is used to display a success message if the database was populated successfully.

```
DISPLAY_WEB_PAGE_ACTIVITES
INTERPRETER.message=Database INSERT
successful.
URLspecifier=$$INSTANCE.webhost/doc/examples/Data
baseForms/
templates/populate_succeed_template.html
```



SQLException

This node is used to catch any SQLExceptions throw in the workflow and send email to the Database Forms administrator with a description of the problem and its location. The Installation of Nouveau Alliance prompts for the email address of the administrator and sets a SYSTEM server property “com.nouveausystems.alliance.server.adminEmail” in the etc/config.ini file. The Send Email activity uses this value for its sender and recipient as shown below.

```
SendEmailActivityImpl
EmailRecipient=
$SYSTEM.com.nouveausystems.alliance.server.ad
minEmail
EmailSender=
$SYSTEM.com.nouveausystems.alliance.server.ad
minEmail
EmailSubject=Exception in workflow
PopulateDatabase
EmailMessage=Workflow:$WORKFLOW.name\nNode:
$exceptionCursor.currentWorkflowNodeReference.l
abel\nActivity#:$exceptionCursor.currentActivityInde
x\nException:$exception.toString\nException
StackTrace:\n$exception.stackTrace.toList.toString
```

Query the Database

This operation is responsible for exacting selection data from an HTML form and searching for entries with the selected fields in the database. The following figure shows the Query web page:



Query Database

This page allows you to Query Registrants by attribute (first/last names, address, email address...) and then update selected entries from the resulting Output. The values that you specify will use the 'LIKE' comparison operator.

First Name:

Last Name:

Company:

Address 1:

Address 2:

City:

State:

Zip/Postal Code:

Country:

Telephone:

Email:

Subscribe for news, events, and special offers

Website:

Your role:

Intended use:

Query



Figure 6 – Query input page

This page form’s action executes the QueryDatabase servlet with its corresponding workflow.

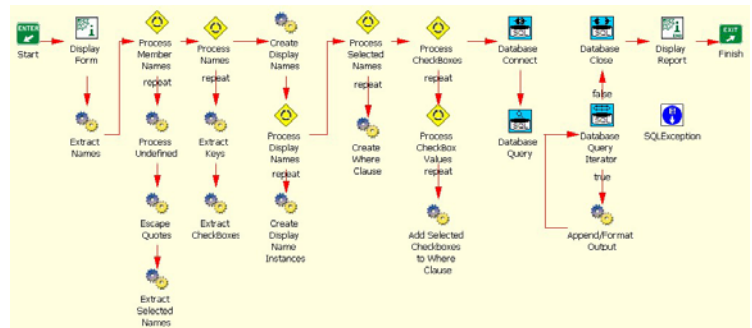


Figure 7 – QueryDatabase workflow

For a better understanding, the workflow has been broken into two parts, with a listing of its constituent workflow nodes, activities and properties.

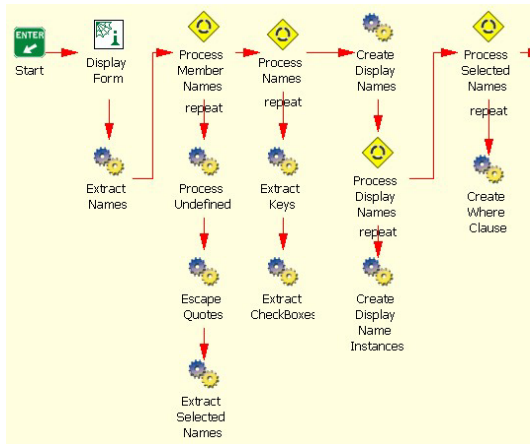


Figure 8 -- QueryDatabase (front)



Display Form

This node is used to display the Query web page. The URL for this page template is specified as an INSTANCE property in the Database Forms servlet configuration file.

```
DISPLAY_WEB_PAGE_ACTIVITES
URLspecifier=${INSTANCE.templateFile}
```



Extract Names

This node is used to read the “names” file created by the “form2sql” application and extract the table name, the form field names and a notation for the form’s values. This node is the same for all workflows in Database Forms.



Process Member Names

This node is used to iterate on the list of database member names extracted in the Extract Names node.

```
SetPropertyActivityImpl
INSTANCE.selectedNames=[]
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
memberName=${INSTANCE.memberNames}
```



Process Undefined

This node is used to check if a member name is undefined. If the property is not defined (detected via \$? Operator), such as for “unchecked” check boxes, a ‘0’ value is substituted.

```
CompareActivityImpl
value1=${INSTANCE.$CONTEXT.memberName}
value2=true
comparison=0
skipCount=2
```

```
SetPropertyActivityImpl
INSTANCE.$CONTEXT.memberName=0
```



Escape Quotes

This node is used to check each member name for single quotes in their values and “escapes” them by replacing them with two single-quotes (to allow quotes to be used in SQL statement strings).

```
SetPropertyActivityImpl
propertyName=
"$INSTANCE.$CONTEXT.memberName"
propertyValue="$${propertyName}"
INSTANCE.newPropertyValue=
${propertyValue.replaceAll("'", "'')}
INSTANCE.$CONTEXT.memberName=
"$INSTANCE.newPropertyValue"
```



Extract Selected Names

This node is used to check for and accumulate any member that has a value as a “selected” name to be used in the selection expression of the query.

```
CompareActivityImpl
value1=${INSTANCE.newPropertyValue.toString.length}
value2=0
comparison=0
skipCount=1
```

```
SetPropertyActivityImpl
ADD_SELECTED_NAME=
${INSTANCE.selectedNames.add(
$CONTEXT.memberName)}
```



Process Names

This node is used to iterate on the names extracted by the Extract Names nodes. This iterator is used to extract key names and their INSTANCE forms.

```
SetPropertyActivityImpl
INSTANCE.keyNames=[]
INSTANCE.instanceKeyNames=[]
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
names=${INSTANCE.names}
```



Extract CheckBoxes

This node is part of the repeat block for the Process Names node. It populates a list of checkbox element names by checking for names tagged with a leading “#”.

```
CompareActivityImpl
value1=$CONTEXT.names.startsWith("#")
value2=false
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
untaggedName=$CONTEXT.names
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
ADD_CHECKBOX=$INSTANCE.checkboxNames.add($untaggedName)
```



Extract Keys

This node is part of the repeat block for the Process Names node. It populates lists of table key names and INSTANCE key names by checking for names tagged with a trailing “*”. This node is the same for all workflows in Database Forms.



Create Display Names

This node is used to collect key and selected names in a Set of “display” names. The display names are used in the selection expression of the query.

```
SetPropertyActivityImpl
INSTANCE.displayNames=$java.util.LinkedHashSet()
ADD_KEY=$INSTANCE.displayNames
.add($INSTANCE.keyNames)
ADD_SELECTION=$INSTANCE.displayNames
.add($INSTANCE.selectedNames)
```



Process Display Names

This node is used to iterate on the “display” names extracted in the previous node.

```
SetPropertyActivityImpl
INSTANCE.instanceDisplayNames=$java.util.LinkedHashSet()
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
displayNames=$INSTANCE.displayNames
```



Create Display Name Instances

This node is used to create INSTANCE annotated forms of names on the “display” list.

```
SetPropertyActivityImpl
ADD_DISPLAY_INSTANCE=$INSTANCE.instanceDisplayNames.add($INSTANCE.$CONTEXT.displayNames)
```



Process Selected Names

This node is used to iterate on the selected names. It is used to populate the WHERE clause of the SQL QUERY statement.

```
SetPropertyActivityImpl
INSTANCE.whereSet=$java.util.Vector()
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
whereNames=$INSTANCE.selectedNames
```



Create Where Clause

This node is used to create the WHERE clause of the SQL QUERY statement.

```
SetPropertyActivityImpl
whereToken=$CONTEXT.whereNames
$INSTANCE.comparisonOperator
\'$INSTANCE.$CONTEXT.whereNames\'
ADD_WHERE=$INSTANCE.whereSet
.add($whereToken)
```

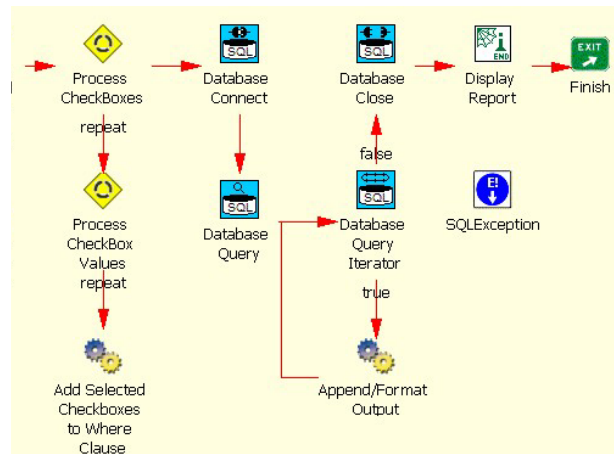


Figure 9 -- QueryDatabase (back)



Process CheckBoxes

This node is used to iterate on the list of checkbox element names extracted in the Extract CheckBoxes node.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
checkboxNames=$INSTANCE.checkboxNames
```



Process CheckBox Values

This node is used to iterate on the list of checkbox values per checkbox.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
cbName=$CONTEXT.checkboxNames
checkboxValues=
  $SERVLET.REQUEST.parameterValues[
  $CONTEXT.checkboxNames]
```



Add Selected Checkboxes to Where Clause

This node is used to generate SQL WHERE clause statements for each checkbox value. The statements are of the form “checkboxName comparisonOperator checkbox ValueExpression”.

```
CompareActivityImpl
value1=false
value2=?CONTEXT.checkboxValues
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
whereToken= $CONTEXT.cbName
$INSTANCE.comparisonOperator
\'%$CONTEXT.checkboxValues%\'
ADD_WHERE=
  $INSTANCE.whereSet.add($whereToken)
```



Database Connect

This node is used to establish a connection to the database. This node is the same for all workflows in Database Forms.



Database Query

This node is used to execute the SQL SELECT statement derived from the Query form.

```
SetPropertyActivityImpl
INSTANCE.whereClause=""
```

```
CompareActivityImpl
value1=0
value2= $INSTANCE.whereSet.size
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
INSTANCE.whereClause= WHERE
  $INSTANCE.whereSet.toCSV.replaceAll(", ", " AND ")
```

```
SetPropertyActivityImpl
INSTANCE.results=<table border=1
  cellpadding=6><th>$INSTANCE.displayNames
  .toCSV.replaceAll(",
  ", "</th><th>")</th><th>details</th></th>
```

```
selectStatement=SELECT $INSTANCE.displayNames.toCSV
FROM $INSTANCE.tableName WHERE
  $INSTANCE.whereClause
```

```
DBExecuteQueryActivityImpl
executeString="$selectStatement"
connectionProperty=$INSTANCE.DB_CONNECTION
resultSetProperty=INTERPRETER.DB_RESULT_SET
```



Database Query Iterator

This node is used to iterate on the result set created by the query, accumulating the fields specified by the INSTANCE display names.

```
DBResultSetActivityImpl
resultSetProperty=$INTERPRETER.DB_RESULT_SET
propertyNames=$INSTANCE.instanceDisplayNames.toCSV
returnStatusProperty=TARGET
```



Append/Format Output

This node is used to produce the query output as rows in an HTML table. For each result row, a hypertext link is created to allow for the display of a populated form corresponding to the row.

```
SetPropertyActivityImpl
findName=$INSTANCE.displayNames.iterator.get(0)
findValue=$INSTANCE.instanceDisplayNames.iterator.get(0)
link=<a
  href=$INSTANCE.webhost/servlet/DisplayDatabaseEntry?findName=
  $findName&amp;findValue=$findValue>More...</a>
tableRow=<tr><td>$INSTANCE.instanceDisplayNames.toCSV
  .replaceAll(", ", "</td><td>\$")</td><td>$link</td></tr>
label=$TARGET
```



Database Close

This node is used to close the connection opened by the Database Connect node. This node is the same for all workflows in Database Forms.



Display Report

This node is used to display a web page with the results of the query.

```
DISPLAY_WEB_PAGE_ACTIVITES
INSTANCE.message=WHERE: "$INSTANCE.whereClause"
INSTANCE.results=$INSTANCE.results</table>
URLspecifier=$INSTANCE.webhost/doc/examples/DatabaseForms/
  templates/query_end_template.html
```



SQLException

This node is used to catch any SQLExceptions throw in the workflow and send email to the Database Forms administrator with a description of the problem and its location. This node is the same for all workflows in Database Forms.

Display Database Entries

This operation is responsible for finding entries in the database and displaying the complete data associated with the selection. The input web page for this operation is the output page from the database Query:

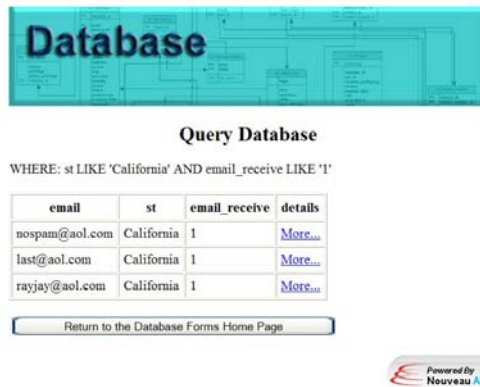


Figure 10 – Query output page

The “More...” links are implemented via the DisplayDatabaseEntry servlet with its corresponding workflow.

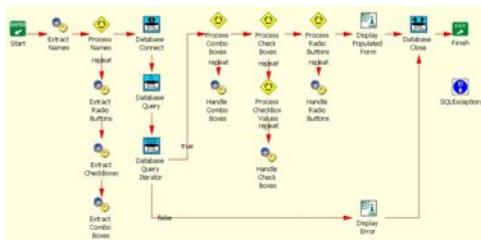


Figure 11 – DisplayDatabaseEntry workflow

For a better understanding, the workflow has been broken into two parts, with a listing of its constituent workflow nodes, activities and properties.

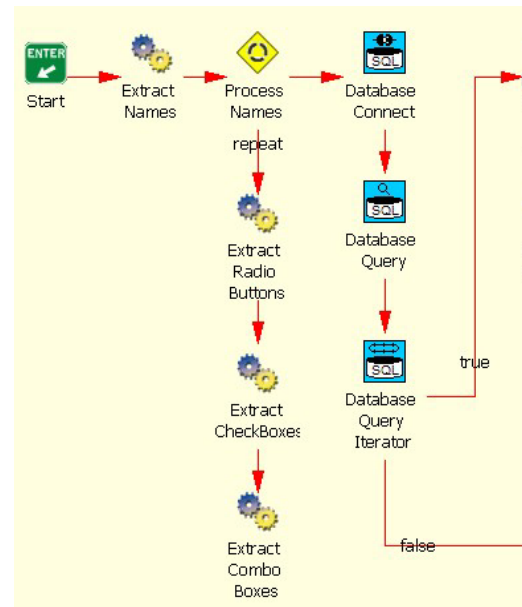


Figure 12 -- DisplayDatabaseEntry (front)

Extract Names

This node is used to read the “names” file created by the “form2sql” application and extract the table name, the form field names and a notation for the form’s values. This node is the same for all workflows in Database Forms with the addition of extracting INSTANCE forms of the member names.

Process Names

This node is used to iterate on the names extracted by the Extract Names nodes. This iterator is used to extract radio, checkbox and combobox elements.

```
SetPropertyActivityImpl
INSTANCE.radioNames=[]
INSTANCE.checkboxNames=[]
INSTANCE.comboNames=[]
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
names=$INSTANCE.names
```

Extract Radio Buttons

This node is part of the repeat block for the Process Names node. It populates lists of radio button names by checking for names tagged with a leading “*”.

```
CompareActivityImpl
value1=$CONTEXT.names.startsWith("*.")
value2=false
comparison=0
skipCount=1
```

```
SetPropertyActivityImpl
untaggedName=$CONTEXT.names
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
ADD_RADIO=$INSTANCE.radioNames
.add($untaggedName)
```



Extract CheckBoxes

This node is part of the repeat block for the Process Names node. It populates lists of checkbox button names by checking for names tagged with a leading “#”.

```
CompareActivityImpl
value1=$CONTEXT.names.startsWith("#")
value2=false
comparison=0
skipCount=1
```

```
SetPropertyActivityImpl
untaggedName=$CONTEXT.names
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
ADD_CHECKBOX=$INSTANCE.checkboxNames
.add($untaggedName)
```



Extract Combo Boxes

This node is part of the repeat block for the Process Names node. It populates lists of combobox names by checking for names tagged with a leading “@”.

```
CompareActivityImpl
value1=$CONTEXT.names.startsWith("@")
value2=false
comparison=0
skipCount=1
```

```
SetPropertyActivityImpl
untaggedName=$CONTEXT.names
.replaceAll("[*]", "").replaceAll("[#]", "")
.replaceAll("[*]", "").replaceAll("[@]", "")
ADD_COMBO=$INSTANCE.comboNames
.add($untaggedName)
```



Database Connect

This node is used to establish a connection to the database. This node is the same for all workflows in Database Forms.



Database Query

This node is used to execute the SQL SELECT statement used to find the database element and extract data for the Update form.

```
SetPropertyActivityImpl
selectStatement="SELECT
$INSTANCE.memberNames FROM
$INSTANCE.tableName WHERE
$INSTANCE.findName = '$INSTANCE.findValue'"
```

```
DBExecuteQueryActivityImpl
executeString="$selectStatement"
connectionProperty=$INSTANCE.DB_CONNECTION
resultSetProperty=INTERPRETER.DB_RESULT_SET
```



Database Query Iterator

This node is used to iterate on the result set created by the find database entry query, accumulating the fields specified by the INSTANCE display names.

```
DBResultSetActivityImpl
resultSetProperty=$INSTANCE.DB_RESULT_SET
propertyName=$INSTANCE.instanceMemberNames
returnStatusProperty=TARGET
```

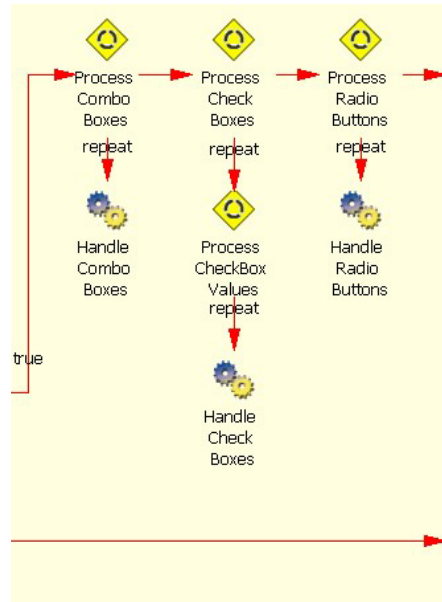


Figure 13 -- DisplayDatabaseEntry (middle)



Process Combo Boxes

This node is used to iterate on the results obtained by the Extract Combo Boxes node.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
comboNames=$INSTANCE.comboNames
```



Handle Combo Boxes

This node is used to create a Map of the selected combo box option with the value “selected” This Map is referenced in the Update HTML template by each Select Option statement. At runtime, only the chosen option will be found in this Map and have its value (“selected”) substituted in the populated Update HTML form.

```
SetPropertyActivityImpl
mapName=INSTANCE.#{CONTEXT
.comboNames}_Map
$mapName=$java.util.HashMap()
ADD=${#{mapName}.put(
#{INSTANCE.#{CONTEXT.comboNames}},
selected)}
```



Process Check Boxes

This node is used to iterate on the results obtained by the Extract Checkboxes node.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
checkboxNames=$INSTANCE.checkboxNames
```



Process CheckBox Values

This node is used to iterate on the list of checkbox values per checkbox creating HashMaps for each checkbox.

```
SetPropertyActivityImpl
mapName=
INSTANCE.#{CONTEXT.checkboxNames}_Map
$mapName=$java.util.HashMap()
checkboxValues=
#{INSTANCE.#{CONTEXT.checkboxNames}}
```

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
mapName=$mapName
checkboxValues$checkboxValues
```



Handle Check Boxes

This node is used to create a Map of the selected checkbox with the value “checked” This Map is referenced in the Update HTML template by each checkbox statement. At runtime, only the chosen checkboxes will be found in this Map and have its value (“checked”) substituted in the populated Update HTML form.

```
SetPropertyActivityImpl
#{CONTEXT.mapName}[#{CONTEXT.checkboxValues}
=checked
```



Process Radio Buttons

This node is used to iterate on the results obtained by the Extract Radio Buttons node.

```
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
radioNames=$INSTANCE.radioNames
```



Handle Radio Buttons

This node is used to create a Map of the selected radio button with the value “checked” This Map is referenced in the Update HTML template by each radio box statement. At runtime, only the chosen radio boxes will be found in this Map and have its value (“checked”) substituted in the populated Update HTML form.

```
SetPropertyActivityImpl
mapName=INSTANCE.#{CONTEXT
.radioNames}_Map
$mapName=$java.util.HashMap()
ADD=${#{mapName}.put(
#{INSTANCE.#{CONTEXT.radioNames}}, checked)}
```

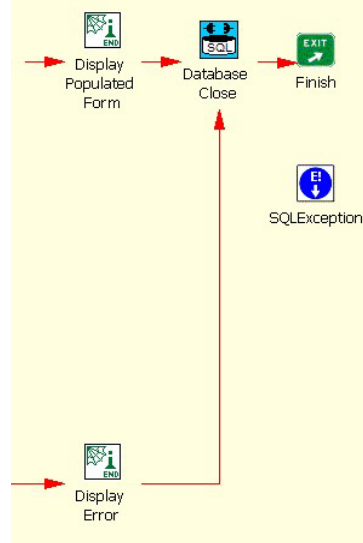


Figure 14 -- DisplayDatabaseEntry (back)



Display Populated Form

This node is used to display the Update web page. The URL for this page template is specified as an INSTANCE property in the Database Forms servlet configuration file. Each member in the form is initialized with its corresponding value in the database (substituted in via INSTANCE properties).

```
DISPLAY_WEB_PAGE_ACTIVITES
URLspecifier=$$INSTANCE.templateFile
```



Display Error

This node is used to display an error web page if the specified entry was not in the database.

```
DISPLAY_WEB_PAGE_ACTIVITES
INTERPRETER.message=No entry found for
$INSTANCE.findName = $INSTANCE.findValue
URLspecifier=$$INSTANCE.webhost/doc/examples/ServletNodes/
templates/info_end_template.html
```



Database Close

This node is used to close the connection opened by the Database Connect node. This node is the same for all workflows in Database Forms.



SQLException

This node is used to catch any SQLExceptions throw in the workflow and send email to the Database Forms administrator with a description of the problem and its location. This node is the same for all workflows in Database Forms.

Update the Database

This operation is responsible for exacting data from an HTML form and updating the corresponding fields in the database. The following figure shows the Update web page:

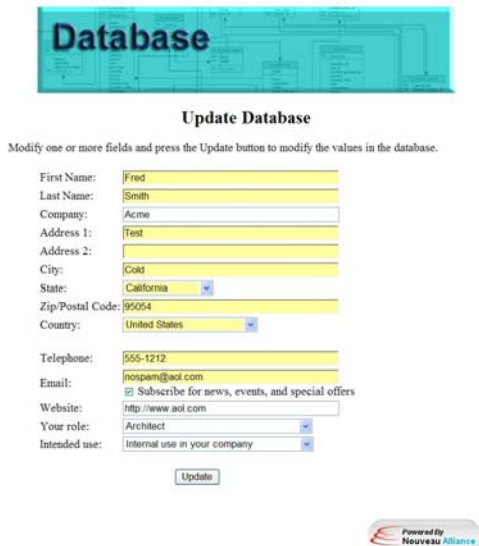


Figure 15 – Update input page

This page form’s action is the UpdateDatabase servlet with its corresponding workflow.

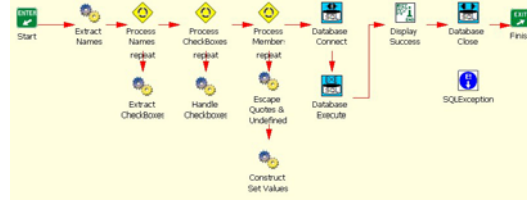


Figure 16 – UpdateDatabase workflow

For a better understanding, the workflow has been broken into two parts, with a listing of its constituent workflow nodes, activities and properties.

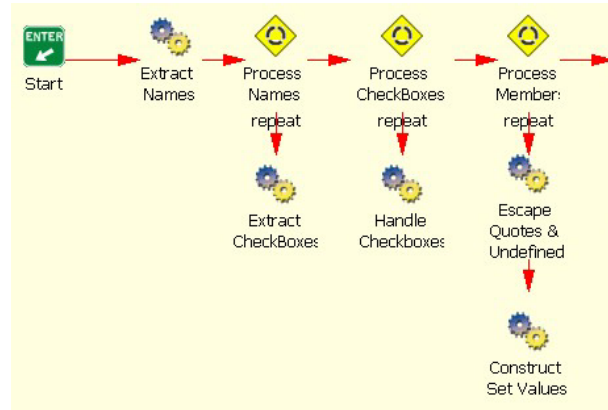


Figure 17 -- UpdateDatabase (front)



Extract Names

This node is used to read the “names” file created by the “form2sql” application and extract the table name, the form field names and a notation for the form’s values. This node is the same for all workflows in Database Forms.



Process Names

This node is used to iterate on the names extracted by the Extract Names nodes. This iterator is used to extract checkbox elements.

```
SetPropertyActivityImpl
  INSTANCE.checkboxNames=[]
```

```
RepeatActivityImpl
  repeatTopology=1
  repeatCount=0
  repeatLabel=repeat
  names=$INSTANCE.names
```



Extract Checkboxes

This node is part of the repeat block for the Process Names node. It populates lists of checkbox button names by checking for names tagged with a leading

“#”. This node is the same for all workflows in Database Forms.



Process CheckBoxes

This node is used to iterate on the results obtained by the Extract Checkboxes node. This node is the same for all workflows in Database Forms.



Handle Checkboxes

This node is used to check if the checkbox is unchecked (its corresponding property is undefined) and set its value to “”. This allows the update process to detect its unselected state when storing its value to the database. The value is stored as a comma-separated String of checkbox “group” values.

```
SetPropertyActivityImpl
INSTANCE.$CONTEXT.checkboxNames=
  $SERVLET.REQUEST.parameterValues[
    $CONTEXT.checkboxNames].toCSV
```

```
CompareActivityImpl
value1=true
value2=${INSTANCE.$CONTEXT.checkboxNames}
comparison=EQUAL
skipCount=1
```

```
SetPropertyActivityImpl
INSTANCE.$CONTEXT.checkboxNames=""
```



Process Members

This node is used to iterate on the list of database member names extracted in the Extract Names node.

```
SetPropertyActivityImpl
INSTANCE.setValues=""
RepeatActivityImpl
repeatTopology=1
repeatCount=0
repeatLabel=repeat
memberName=${INSTANCE.memberNames}
```



Escape Quotes & Undefined

This node is used to check each member name for single quotes in their values and “escapes” them by replacing them with two single-quotes (to allow quotes to be used in SQL statement strings). In addition, if the property is undefined (via \$? Operator), such as for “unchecked” check boxes, a ‘0’ value is substituted. This node is the same as the one used in the Populate Database workflow.



Construct Set Values

This node is used to accumulate the Set expression for the SQL UPDATE statement.

```
SetPropertyActivityImpl
entry=${CONTEXT.memberName =
  \"${INSTANCE.$CONTEXT.memberName}\"
INSTANCE.setValues=${INSTANCE.setValues, $entry}
```

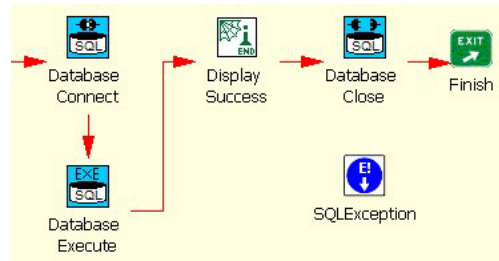


Figure 18 -- UpdateDatabase (back)



Database Connect

This node is used to establish a connection to the database. This node is the same for all workflows in Database Forms.



Database Execute

This node is responsible for creating and executing the SQL UPDATE statement used to update the database.

```
SetPropertyActivityImpl
updateStatement=UPDATE ${INSTANCE.tableName} SET
  ${INSTANCE.setValues.substring(1)} WHERE
  ${INSTANCE.findName} = \"${INSTANCE.findValue}\"
```

```
DBExecuteActivityImpl
executeString=\"$$updateStatement\"
connectionProperty=${INSTANCE.DB_CONNECTION}
returnStatusProperty=TARGET
```



Display Success

This node is used to display a success message if the database was updated successfully.

```
DISPLAY_WEB_PAGE_ACTIVITES
INTERPRETER.message=Database Update completed
successfully.
URLspecifier=${INSTANCE.webhost/doc/examples/DatabaseForms/
  emplates/update_succeed_end_template.html}
```



Database Close

This node is used to close the connection opened by the Database Connect node. This node is the same for all workflows in Database Forms.



SQLException

This node is used to catch any SQLExceptions throw in the workflow and send email to the Database Forms administrator with a description of the problem and its location. This node is the same for all workflows in Database Forms.

Installation

The Nouveau Alliance Database Forms example can be located in three locations:

- A pre-installed workspace called “examples”.
- A workflow **template** called “examples”.
- A ZIP archive file named DatabaseForms.zip that is available from the Nouveau Systems website.

The specific installation instructions depend on the location.

Pre-installed Workspace

If the “examples” workspace has already been installed, with workflow definitions (PopulateDatabase, QueryDatabase, DisplayDatabaseEntry and UpdateDatabase), no additional installation is required. If these workflow definitions do not exist, please follow the ZIP Archive instructions presented below.

Workspace Template

To install the “examples” workspace from a template, start FlowSpace and select the New Workspace option from the File menu and select the “examples” template. Once the workspace is created, confirm that the Database Forms workflows are present (see list above). If not, then

follow the ZIP Archive instructions presented below.

ZIP Archive

The Database Forms ZIP archive file can be downloaded from here:

<http://download.nouveausystems.com/whitepapers/DatabaseForms.zip>

To install this zip file:

1. Change directory to your Alliance root directory
2. Unzip the ZIP archive file in this directory. The output of the ZIP file will be placed in the following directories:
 - doc/examples/DatabaseForms
 - etc/servlets
3. Start FlowSpace and open or create a workspace called “examples”.
4. Into this workspace, import the workflow definition archive files in this Database Forms xml directory:

PopulateDatabase.zip
QueryDatabase.zip
DisplayDatabaseEntry.zip
UpdateDatabase.zip

These XML-archive files specify workflow definitions that contain all of the Database Forms operations.

To load these definitions into the “examples” workspace, select the “Import New Workflow” option from the FlowSpace section page number pop-up menu in the lower-right corner of the display. You can select more than one xml archive file to import at a time by using the control or shift keys while making the selection.

Required License and Version

A Professional or Department license for FlowSpace version 3.2 and later is required to use the Database Forms. If you're

currently using Personal Edition, try FlowSpace Professional Edition *free* for 30 days now! Just point your web browser at your Nouveau Alliance server's home page, and click "Get Your Upgrade Evaluation!" to get started.

With the appropriate license, and a running Nouveau Alliance server, you can demonstrate the use of Database Forms.

Demonstration

To demonstrate the use of Database Forms

1. View the Database Forms main web page in your web browser. The location of the web page is:

http://YOUR_HOST:8081/doc/examples/DatabaseForms/index.html.

where YOUR_HOST is the name of your server host.

This page looks like this:

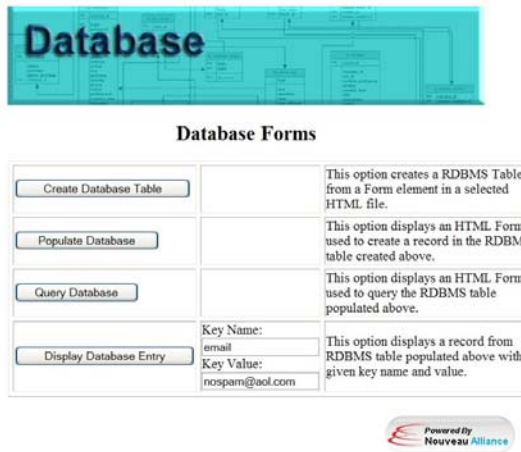


Figure 19 – Database Forms Main Page

2. Locate / Create an HTML page that contains a form that you want to map to a RDBMS. For this demonstration we will be using the following file:

http://YOUR_HOST:8081/doc/examples/DatabaseForms/templates/populate.html

This file contains a form named “Register” that contains personal data fields normally associated with registering a product or service.




Figure 20 – Populate Form

3. Run form2sql to create an SQL table in the RDBMS corresponding to the HTML form. To start the form2sql application, select the “Create Database Table” on the Database Forms main page. This button initiates a Java WebStart invocation of this application. Once the application starts, open the HTML file “populate.html” (described in the previous step) with the File menu’s “Open HTML File...” option.

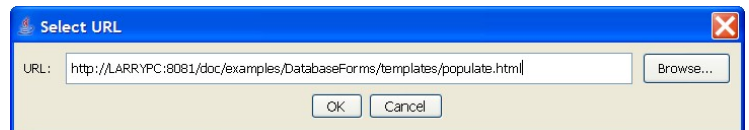
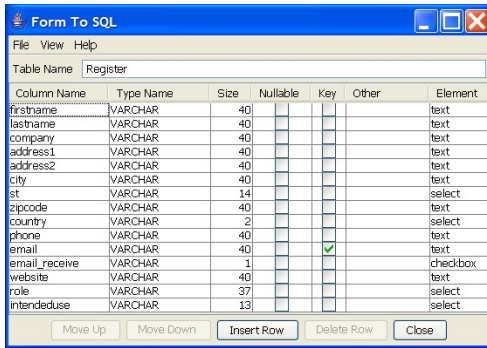


Figure 21 – Open HTML File (via URL)

The form2sql application will update the table display with this page’s form:



Column Name	Type Name	Size	Nullable	Key	Other	Element
firstname	VARCHAR	40				text
lastname	VARCHAR	40				text
company	VARCHAR	40				text
address1	VARCHAR	40				text
address2	VARCHAR	40				text
city	VARCHAR	40				text
st	VARCHAR	14				select
zipcode	VARCHAR	40				text
country	VARCHAR	2				select
phone	VARCHAR	40				text
email	VARCHAR	40		<input checked="" type="checkbox"/>		text
email_receive	VARCHAR	1				checkbox
website	VARCHAR	40				text
role	VARCHAR	37				select
intendeduse	VARCHAR	13				select

Figure 22 – form2sql application

This application extracts the fields of the form contained in the HTML file and displays them in tabular presentation. A table name is derived from the name of the form and the table members from each form elements.

Default values are derived for type and size. You can modify these values as well as specify where the table field will be nullable, is a key, or has other SQL/Database specific attributes (e.g., auto-incrementing).

Make the **email** field a key by checking the "Key" checkbox in the 'email' row for this example. The email field is used to locate entries in subsequent workflows.

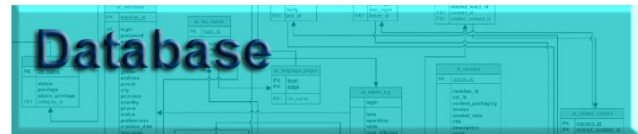
When you are satisfied with the table modeling, select the “Save As SQL Table” option of the File menu to create/replace a table in your database.

- Use form2sql to create a “names” file that contains information needed by the Database Forms workflows to configure the SQL statements that they use. In order to avoid modification of the Database Forms servlet configuration file, please save the “names” file as:

`http://YOUR_HOST:8081/
doc/examples/DatabaseForms
/database/names.txt`

where YOUR_HOST is the name of your server host.

- In a web browser, from the Database Forms main page, select the “Populate Database” button to start a servlet that displays the Populate Database page.



Populate Database

Please enter values for the fields on this form. The email field is a unique (key) field for this form and is required to have a value. Pressing Save will enter this values into the database.

First Name:

Last Name:

Company:

Address 1:

Address 2:

City:

State:

Zip/Postal Code:

Country:

Telephone:

Email: Subscribe for news, events, and special offers

Website:

Your role:

Intended use:



Figure 23 – Populate Database page

To insert an entry in the database, complete the fields of this form and select the “Save” button. This will continue the PopulateDatabase servlet to commit the data to the database.

Repeat the execution of this option to add a few more entries in the database using the “Populate Another Entry” button.

- In a web browser, from the Database Forms main page, select the “Query Database” button to start a servlet that displays the Query Database page.



Query Database

This page allows you to Query Registrants by attribute (first/last names, address, email address...) and then update selected entries from the resulting Output. The values that you specify will use the LIKE comparison operator.

First Name:
 Last Name:
 Company:
 Address 1:
 Address 2:
 City:
 State:
 Zip/Postal Code:
 Country:
 Telephone:
 Email:
 Subscribe for news, events, and special offers
 Website:
 Your role:
 Intended use:



Figure 24 – Query Database page

This page is identical to the Populate Database page except that it interprets each file as an optional query “LIKE” predicate.

The LIKE predicate compares the value in a string expression with a character string pattern which may contain wildcard characters. For example, `_` stands for any single character and `%` stands for any sequence of zero or more characters.

To test the query, type in the name of one of the entries that you added to the database and press the “Query” button. For example, if I had entered many “Bill’s” to the database, my web display will look like Figure 25.



Query Database

WHERE: st LIKE 'California' AND email_receive LIKE '1'

email	st	email_receive	details
nospam@aol.com	California	1	More...
last@aol.com	California	1	More...
rayjay@aol.com	California	1	More...

[Return to the Database Forms Home Page](#)



Figure 25 – Query output

Note that the display consists of a list of “hits” in the database. On each line, the key fields and the fields that you query are displayed along with a details link that you can select to see the full record for each “hit”.

- Select the “details” link for one of the resulting entries from the query. This link will display a form containing the data for the selection.



Update Database

Modify one or more fields and press the Update button to modify the values in the database.

First Name:
 Last Name:
 Company:
 Address 1:
 Address 2:
 City:
 State:
 Zip/Postal Code:
 Country:
 Telephone:
 Email:
 Subscribe for news, events, and special offers
 Website:
 Your role:
 Intended use:



Figure 26 – Database Entry display

You can use this form to modify the data for the selection.

8. Change a value of the selected entry and select the “Update” button. This button will run the UpdateDatabase servlet to save the changes.
9. You should return to the main page of Database Forms to run another query with the QueryDatabase servlet to verify that the change was saved.

This demonstration has given you an understanding of how Database Forms is used. The next section describes how you can customize it for your application.

Customization

This section provides four ways that you can modify Database Forms for you specific application.

Change Forms

The Demonstration used a sample form contained in the populate.html file. You can see the utility of Database Forms when you select one of your forms. To change forms:

1. Select an HTML file containing the new form. The HTML file that you choose must contain at least one named form with elements that are also named.
To select an HTML file, open the Database Forms main page with your web browser. On this page, select the “Create Database Table” button to invoke the form2sql WebStart application. From this application’s File Menu, select the “Open HTML File...” option and chose your HTML file. The form2sql application display will show a tabular display corresponding to the RDMS table that will be

created. If you want to make any changes to the table’s attributes (type, size,...) modify this table.

2. Create a new table in the database from the new HTML file.
After making any changes to the form2sql’s table, select the “Save As SQL Table” File menu option.
3. Create a new “names” interface file. Each form requires a corresponding “names” file containing information about the form needed by the workflow. To create the “names” file, select the “Save Names...” option from the form2sql File menu. This file is intended to be saved as:

```
http://YOUR_HOST:8081/doc/  
examples/DatabaseForms/  
database/names.txt
```

If you place the “names” file in another location, you will need to change the Database Forms servlet configuration file’s “namesFile” entries.

4. Create Populate/Query HTML file from the new HTML file.
The Populate and Query operations share a common HTML template file. This is accomplished by starting with your form’s HTML file and replacing the form’s action parameter with a property place holder:

```
$SERVLET.REQUEST.requestURI
```


This property expression expands to the appropriate servlet invocation: PopulateDatabase or QueryDatabase. In the Database Forms servlet configuration file, modify the PopulateDatabase and QueryDatabase templates file to the location of this shared template file.
5. Make the Populate/Query HTML file (and its servlets) resumable. This is done by adding two hidden elements to the HTML form:

```
<input type="hidden" name =  
"workflowInstanceName"  
value="$INSTANCE.name">  
<input type="hidden" name =  
"workflowInterpreterName"  
value=  
"$INTERPRETER.name">
```

These two form elements will contain the names of the instance and interpreter to resume when you click on the form's submit button.

6. Copy the Populate/Query HTML file and save it as the Update HTML template file.

The Update HTML template file differs from the Populate/Query template file in the selection of the database record to update and in the initialization of the form's fields.

7. In order to specify which database record is to be updated, you need to add two hidden elements to the Update template form:

```
<input type="hidden"  
name="findName"  
value="$INSTANCE.findNa  
me">
```

```
<input type="hidden"  
name="findValue"  
value="$INSTANCE.findVal  
ue">
```

The "findName" input element specifies that name of the key field to locate the database entry. The "findValue" input element specifies the value of the key field that corresponds to the database record to update.

For each text field in the Update template form, add a value attribute:

```
<input size=45 name=firstname  
type="text"  
value="$INSTANCE.firstname"/>
```

This property expression allows you to initialize this form field from a property value extracted from the database.

8. Add value fields for each text and password fields.
For each text or password input elements in the Update template, add the value attributes as shown in the previous step.

9. Update any textarea fields to add a value string.

You will need to modify any textarea fields, you will need to make the following additions:

```
<textarea name="story" cols="30"  
rows="6">$INSTANCE.story  
</textarea>
```

10. Add Map references for SELECT OPTIONS to support selection.
For each Select element, you will need to modify each Option element as follows:

```
<option ${INSTANCE.XXX_Map  
["YYY"]} value="YYY">  
label</option>
```

where XXX is element name (e.g., "country") and YYY is options name (e.g., "United States"). The workflow that uses this template creates this Map and adds one entry corresponding to the extracted "country" value from the database with a value of "selected". This Map reference will be replaced by "selected" for the value selected from the database and null by those options that are not selected. The net result is the processed Update template shows the appropriate selection.

11. Add Map references for radio and checkbox buttons to support selection.

In the same manner as above, radio and checkbox buttons are initialized to their appropriate state. For example, the following checkbox is set or unset by the Map reference:

```
<input name="subscriber"
type="checkbox"
value="${INSTANCE.
subscriber}" ${INSTANCE.
subscriber_Map[
${INSTANCE. Subscriber
]}>
```

The workflow that uses this template sets the “subscriber_Map” to 1 or 0 based upon the extracted value of subscriber.

12. Update the Database Forms servlet configuration file to reference the new Update template file. In step 5, you created the Update template file. In this configuration file, you need to modify the “DisplayDatabaseEntry” configuration’s template file to reflect the location of the file.

Change Database

The Demonstration uses the Derby RDBMS to provide persistence for the forms. You can switch to a different database by modifying two configuration files.

1. Update the Database Forms servlet configuration file to modify the following fields:

databaseDriver

The name of the database driver class. For Derby the value is

org.apache.derby.jdbc.EmbeddedDriver.

databaseConnectionURL

The URL to connect to the database.

For Derby the value is

```
jdbc:derby:$WORKFLOW.workspace.
workspaceName/defaultDB
;create=true.
```

databaseUser

The database username to be used for the connection.

databasePassword

The password for the database user.

comparisonOperator

The comparison operator used in the database query. For Derby, the comparison operator is ‘LIKE’. For MySQL, the comparison operator is REGEXP.

Message

The message to be displayed on the query form. This message usually contains information particular to the query such as the comparison operator that will be used.

2. Update the Form2SQL WebStart file to add arguments to the invocation of form2sql. Specifically, you need to modify the “application-desc” definition to include all of your database options:

```
<application-desc main-class=
"com.nouveausystems.databaseforms.Application
1">
<argument>
  -databaseConnectionURL
</argument>
<argument>
  YOUR_DATABASE_CONNECTION_URL
</argument>
<argument>
  -databaseDriver
</argument>
<argument>
  YOUR_DATABASE_DRIVER
</argument>
<argument>
  -databaseUser
</argument>
<argument>
  YOUR_DATABASE_USER
</argument>
<argument>
  -databasePassword
</argument>
<argument>
  YOUR_DATABASE_PASSWORD
</argument>
</application-desc>
```

For example, to use MySQL, your database connection URL would be:

```
jdbc:mysql://YOUR_HOST
:3306/DB
```

and your database driver will be:

```
com.mysql.jdbc.Driver.
```

Change Workflows

As with any Nouveau Alliance Workflow-enabled Web Application, the Database Forms applications can be customized by modifying its associated workflow. You can specify additional nodes process or filter the captured data.

One such node that you can add could filter the data entered by web site “spammers”. For example, you could add a “Spam Check” node which checks for the validity of the fields before committing the data to the database. The following node definition verifies that the “zipcode” field has all numbers or a dash.



Spam Check

```
Compare Activity
value1 = $INSTANCE.zipcode
value2 = [0-9,-]*
comparison = MATCH
skip = 1
```

```
Throw Exception Activity
type = Exception
message = "Illegal "zipcode" value."
```

Another example of customizing the workflows would be to add an email node at the end of each workflow, notifying interested parties of database activity.

Change the Look and Feel

The Database Forms HTML template files are intentionally simple in order to not overshadow the interface concepts.

You can modify your HTML files to use your custom style sheets, frame sets, background images, and verbiage to match your standards without affecting the operation of its database interface.

Summary

This white paper describes how to build persistent, web-based applications with Nouveau Alliance. Some of the technologies used include:

- Flexible Workflow Modeling
- Workflow Enabled Servlets
- Embeddable RDBMS

These applications can be further customized and extended to create other applications. The paper demonstrates that you do not need a large, complex framework, or specialized skills, to rapidly construct persistent web-based applications.