



---

# Integrating Lotus Domino/Notes with Tivoli Directory Integrator

*Written using TDI 6.1.1*

*Document version 1.1*

*Eddie Hartman  
Storyteller, TDI Development team*



# REVISION HISTORY

Date	Version	Revised By	Comments
16 April 2008	1.0	EH	Published vrs. 1.0
25 August 2009	1.1	EH	Corrected information regarding the LBE LDAP Browser

# CONTENTS

---

Revision History.....	3
1. Introduction.....	5
2. Reading and Writing Domino Internet Users .....	9
2.1. Reading CSV.....	9
2.2. Creating The AssemblyLine .....	14
2.3. Adding the LDAP Connector .....	16
2.4. Mapping to dominoPerson Schema .....	21
2.5. Modifying existing users .....	27
2.6. Deleting Internet Users.....	30
Using Hooks to handle exceptions .....	30
3. Domino-specific Connectors.....	35
3.1. Choice of connections.....	35
1.1.1 Local Client .....	36
1.1.2 Local Server.....	37
1.1.3 Remote connections (IIOP).....	37
4. Working with Notes Documents.....	39
4.1. Working with Domino Objects .....	44
5. Provisioning Notes User Accounts .....	50
5.1. Special Attributes for User Registration.....	52
6. Detecting Changes in NSF Databases.....	61
7. Error Messages and Suggested Solutions .....	69

## 1. Introduction

Don't panic at the length of this document – there are lots of screenshots. For your own sake, spend a couple of hours working through the exercises in the TDI Getting Started manual<sup>1</sup> plus the first 6 online video tutorials<sup>2</sup>. Also, if you don't know much about Domino/Notes then I suggest reading up on it as well<sup>3</sup>.

As always, make sure that you are running the correct version of the software you work with. You'll find the recommended patch-level for all TDI versions here: [http://www-1.ibm.com/support/docview.wss?rs=697&context=SSCQGF&dc=DA400&uid=swg27010509&loc=en\\_US&cs=UTF-8&lang=en&rss=ct697tivoli](http://www-1.ibm.com/support/docview.wss?rs=697&context=SSCQGF&dc=DA400&uid=swg27010509&loc=en_US&cs=UTF-8&lang=en&rss=ct697tivoli)

These tutorials have been built using Domino 8.

After this introductory section, the rest of the document is divided into these topics and tutorial exercises:

- Reading and Writing Domino Internet Users
- Configuring the TDI Domino Connectors
- Working with Notes Documents
- Provisioning Notes User Accounts
- Catching changes in Domino Notes

These exercises all use the following CSV data:

```
Alysson A. Angst,aangst@earthlink.net,21 Allen Dr.,Anchorage,Alaska,1-555-432-1234
Christopher Crumpet,chris@crumpet.org,Crescent Lake Apts. #42,Costa Mesa,California,1 (555) 424 2424
Eduardo Echnidea,ed.echo@matador.es,Estrada 4-2,Ebladieblada,España,123 456 789
Theodor Tightly,ted.tight@yoohoo.com,42 Tallahassee St.,Turpid,Texas,(555) 121-2121
```

So your first step is to create a file with this textual data on disk, for example here:

```
<TDI solution-directory4>/Domino_Tutorials/Folk.csv
```

---

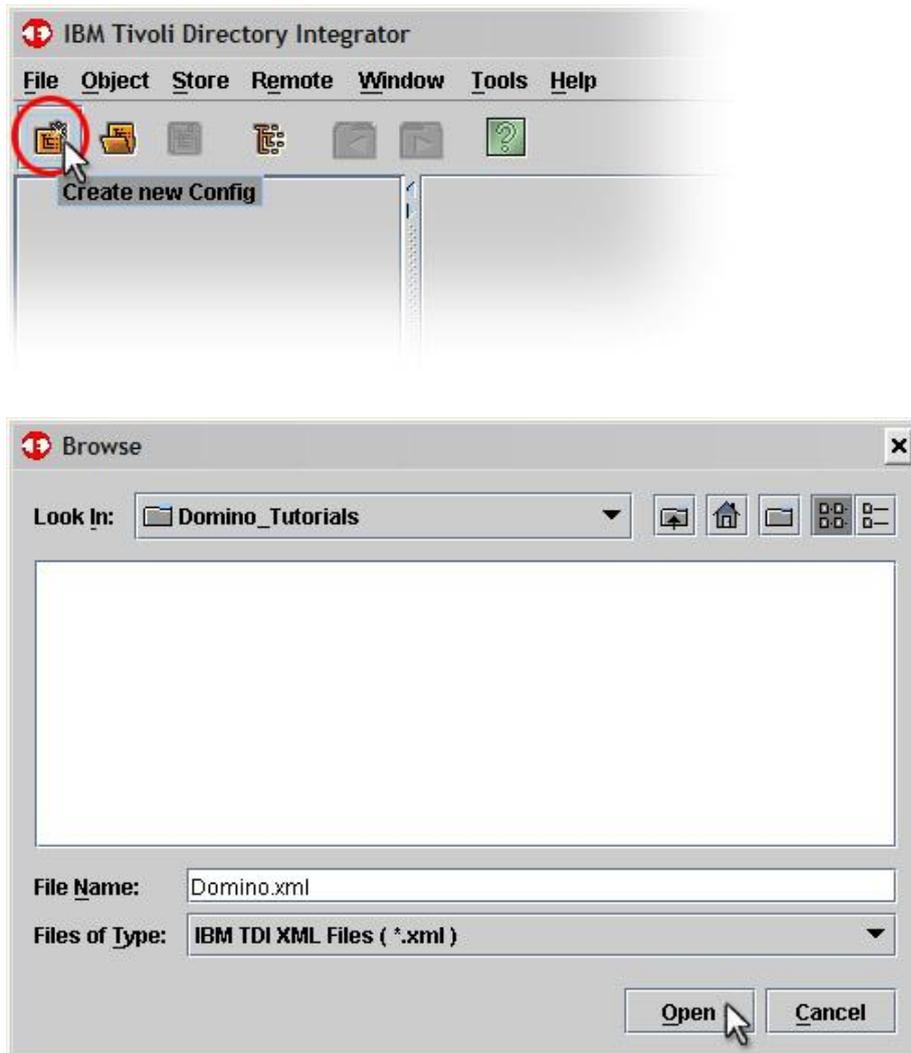
<sup>1</sup> This guide includes a number of tutorial exercises, plus just enough theory to get you going with TDI: [http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/gettingstarted.htm](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/gettingstarted.htm)

<sup>2</sup> Online resources for learning TDI can be found here: <http://www.tdi-users.org/twiki/bin/view/Integrator/LearningTDI>

<sup>3</sup> A good place to start is here: <http://www.ibm.com/developerworks/lotus>

<sup>4</sup> You specified your Solution Directory when you installed TDI. It is recommended that you choose the default options which creates a “tdi” folder in your *home* directory (e.g. My Documents\TDI). This facilitates adding your TDI project assets (Config xml and custom properties files) to whatever backup routine already in place for other user data like documents, images and videos. The TDI Config xml is a small text file, just like all custom properties, making them fast and easy to work with in the Config Editor development environment, it lets you quickly dispatch tasks to a remote TDI Server (as some bundlers do, like Tivoli Identity Manager).

Now it's time to fire up the TDI Config Editor (or 'CE' for short) and create a new Config. Call it "Domino.xml" and save it to the Domino\_Tutorials folder<sup>5</sup>.



A Config is an XML document where your TDI work is saved, and a single Config file contains a library of configured components and the AssemblyLines that use them. In this example, the 'Domino' Config is used to store your tutorial work.

With this out of the way, let's turn our attention to the four different Connectors available for use with Domino/Notes, each designed for different kinds of integration work:

- **LDAP Connector**, giving you access to reading and writing internet user information via Domino's LDAP service. These internet users can then be used for authentication by systems like Portal and Sametime;  
[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide36.htm#dapconnect](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide36.htm#dapconnect)

---

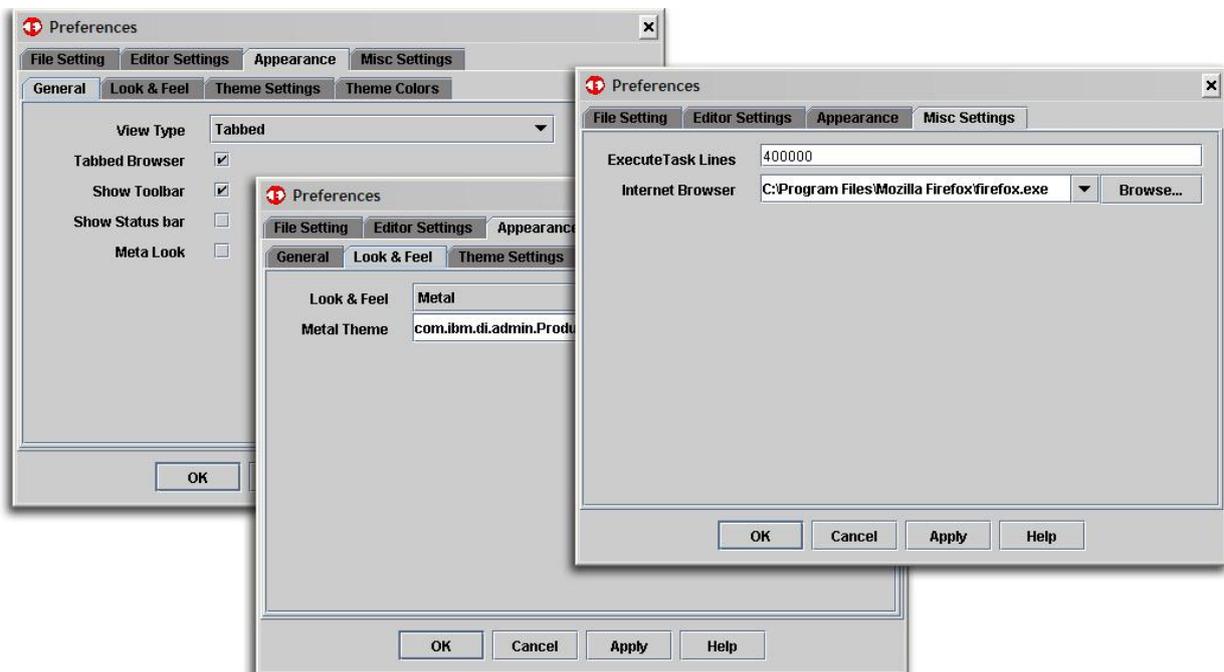
<sup>5</sup> Create sub-folders in your Solution Directory for your various TDI projects; even you first one :)

- **Notes Connector**, for working directly with any type of Notes Documents in any .nsf database;  
[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide18.htm#lotusconnect](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide18.htm#lotusconnect)
- **Domino Users Connector**, for creating, modifying and deleting Notes User information. This is the component you use if you want to provision and manage Notes accounts.  
[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide18.htm#duc](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide18.htm#duc)
- **Domino Change Detection Connector**, allowing you to detect *adds*, *modifies* and *deletes* of Notes Documents, including in names.nsf. This component forms the basis of any synchronization solution that uses Domino as a source;  
[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide18.htm#dcdc](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide18.htm#dcdc)

Unlike the LDAP Connector which only needs the LDAP task running on the Domino Server, the last three components above make use of Java libraries provided by IBM Lotus. Using any of these necessitates telling TDI where these library files are located, and we'll take a close look at how to do this in the second exercise. But first let's go for low-hanging fruit and see how to connect to Domino LDAP.

Also, if you want your TDI screens to look just like those in this doc then do the following:

1. Select **File ► Edit Preferences**
  - a. in the **Appearance ► General** tab:
    - i. Change **View Type** to *Tabbed* from the drop-down;
    - ii. Check the top two checkboxes.
  - b. in the **Appearance ► Look & Feel** tab:
    - i. Change **Look & Feel**<sup>6</sup> to *Metal* from the drop-down.
  - c. in the **Misc Settings** tab:
    - i. Set **Execute Task Lines**<sup>7</sup> to (at least) 4000. This is a Java Swing buffer that should be bigger than just 400.



<sup>6</sup> The *Meta Look & Feel* is a better implementation – at least under MS Windows.

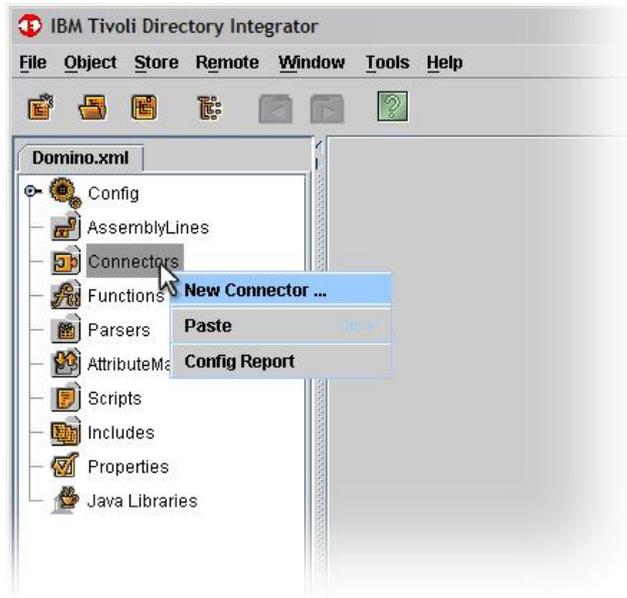
<sup>7</sup> This is a Java Swing buffer size that should be increased.

## 2. Reading and Writing Domino Internet Users

If you want to create, delete and modify Internet Users on your Domino Server, including setting and changing passwords, then the TDI LDAP Connector is all you need<sup>8</sup>. This exercise will see you configuring an LDAP Connector and loading information from your Folk.csv file. However, you first need to set up the Connector for reading this data into your AssemblyLine.

### 2.1. Reading CSV

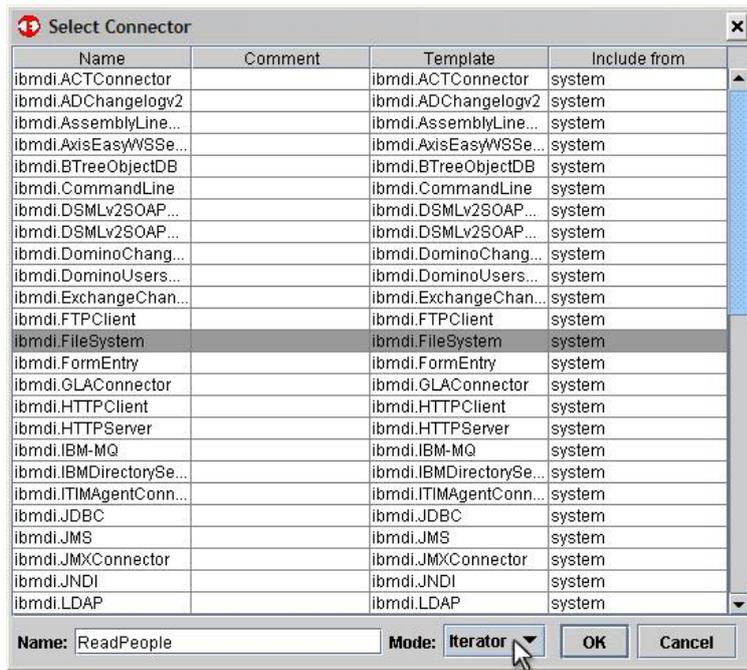
Start by setting up a FileSystem Connector to read the data provided in the first section of this document. Create this component in the Connectors library so that it can be re-used in the other exercises. Do this by right-clicking on the Connectors folder and selecting *New Connector*.



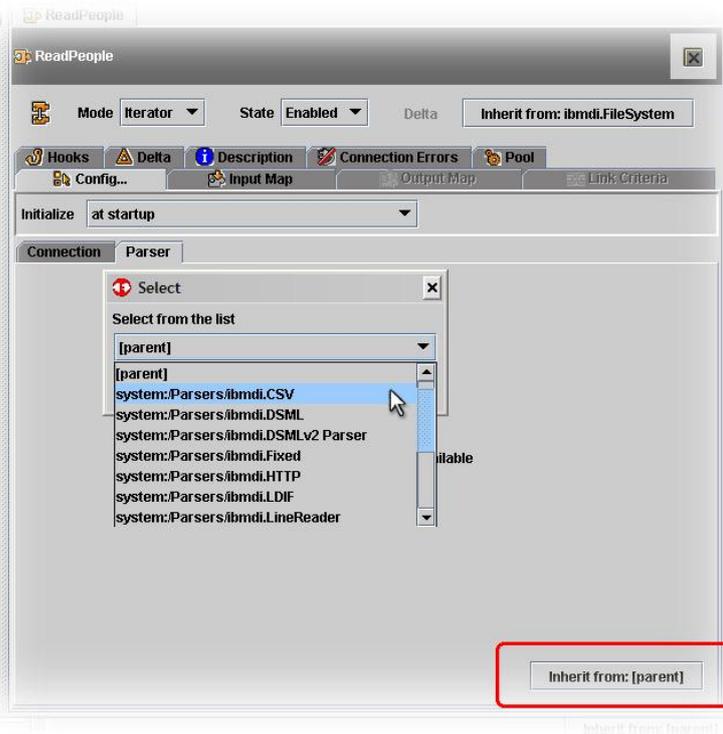
Choose the FileSystem Connector, call it "ReadPeople" and select Iterator mode.

---

<sup>8</sup> TDI also boasts a Password-Sync plugin for Domino that can catch password changes through the standard "Change Password" option, as well as via LDAP add and modify operations.

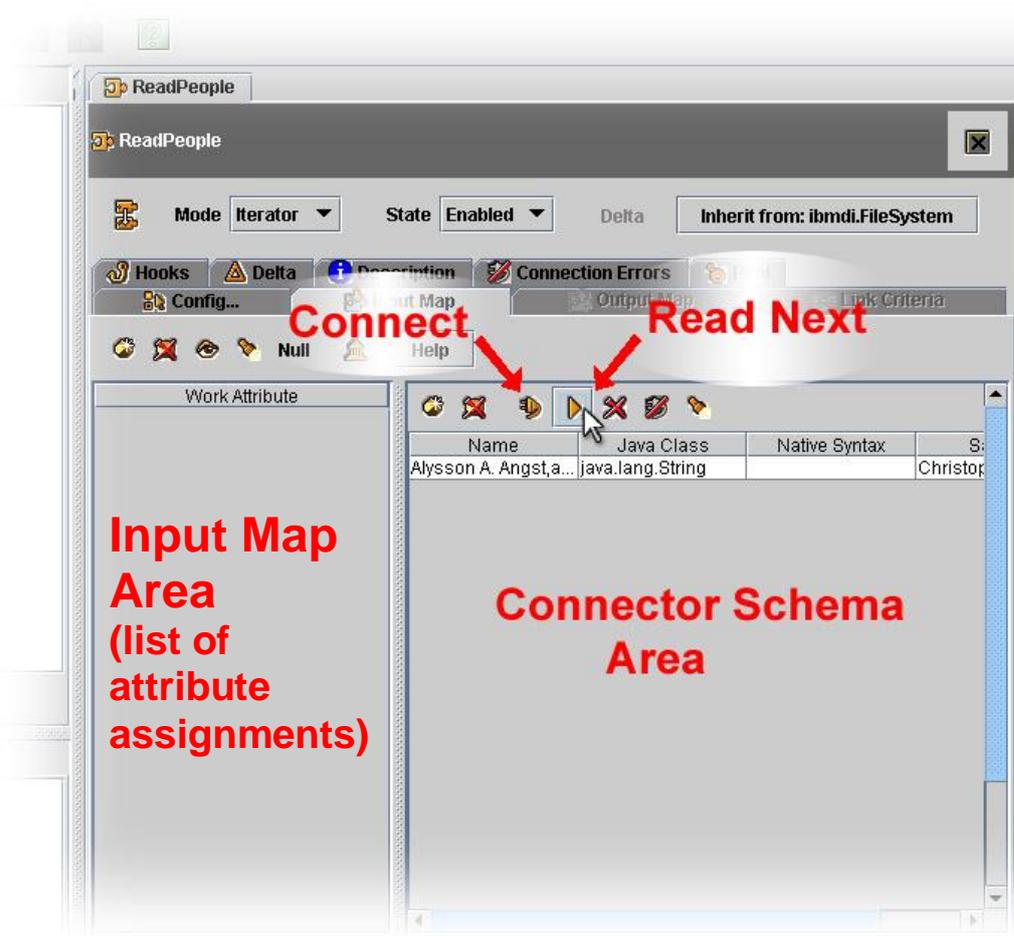


In the **Config** tab for this Connector enter the path to the .csv file you created earlier<sup>9</sup>. Now select the **Parser** tab, click on the **Inherit From** button at the bottom right-hand corner of the Parser tab and select the **CSV Parser** from the selection list.



<sup>9</sup> All filepaths can be specified as relative to your Solution Directory, making your solutions more portable.

You should now be able to select the **Input Map** tab and press the **Connect** button above the Connector Schema area. This fires up the Connector which opens the specified file. Now press the **Read Next** button causing the first line to be read, parsed and presented onscreen<sup>10</sup>.

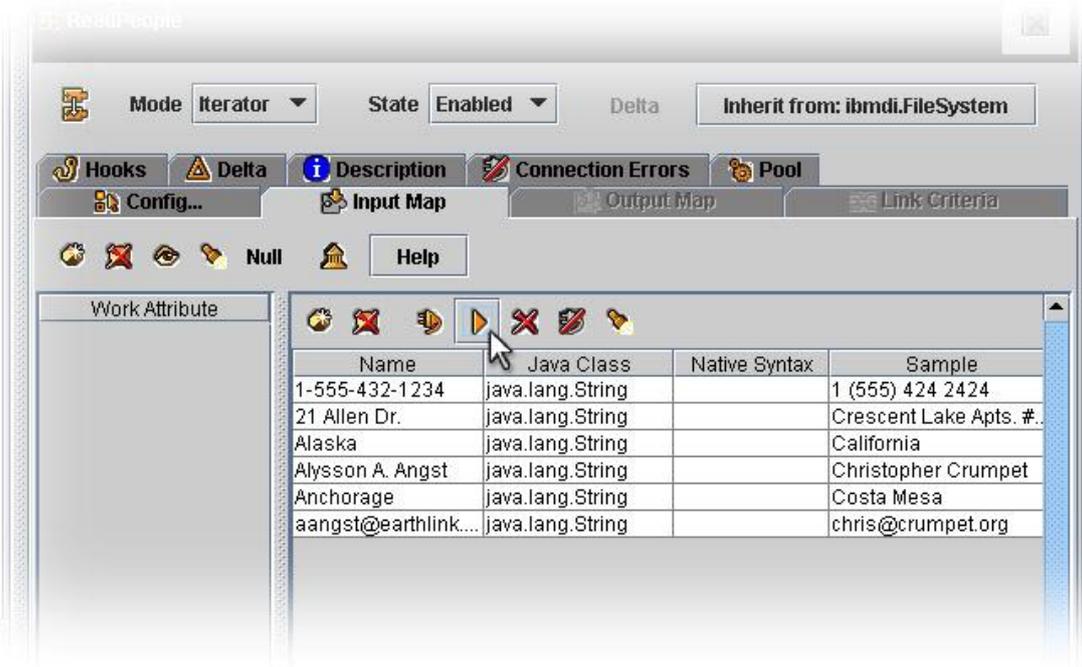


Instead of getting the six attributes you saw in the .csv file, the Parser is returning a single attribute with a name that looks a lot like data. This is because the CSV Parser requires a bit more configuration.

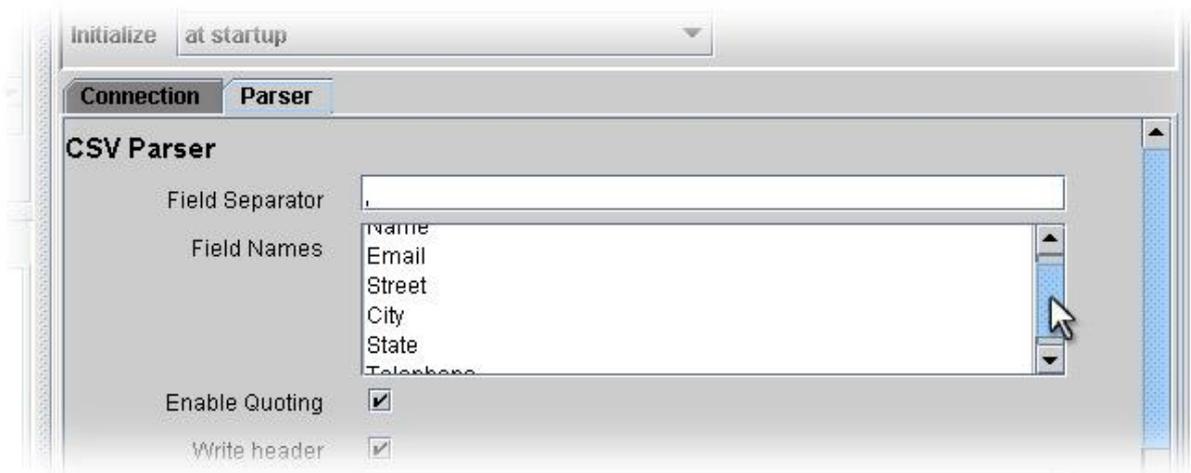
Select the **Parser** tab and you'll notice that the default Field Separator is the semicolon (;). However, if you look at the .csv data you will see that this should be a comma instead. Make this change to your Parser and re-test it by going back to the **Input Map** tab and clicking on the **Connect** and **Read Next** buttons again. Before

<sup>10</sup> This is called *schema discovery* and works the same way for almost all Connectors. Note that some Connectors, like LDAP and JDBC, also support *querying* the schema of the connected system via little flashlight icon at the right end of this button bar.

you do this, use the red **X** button next to **Read Next** in order to clear out previously discovered data<sup>11</sup>.



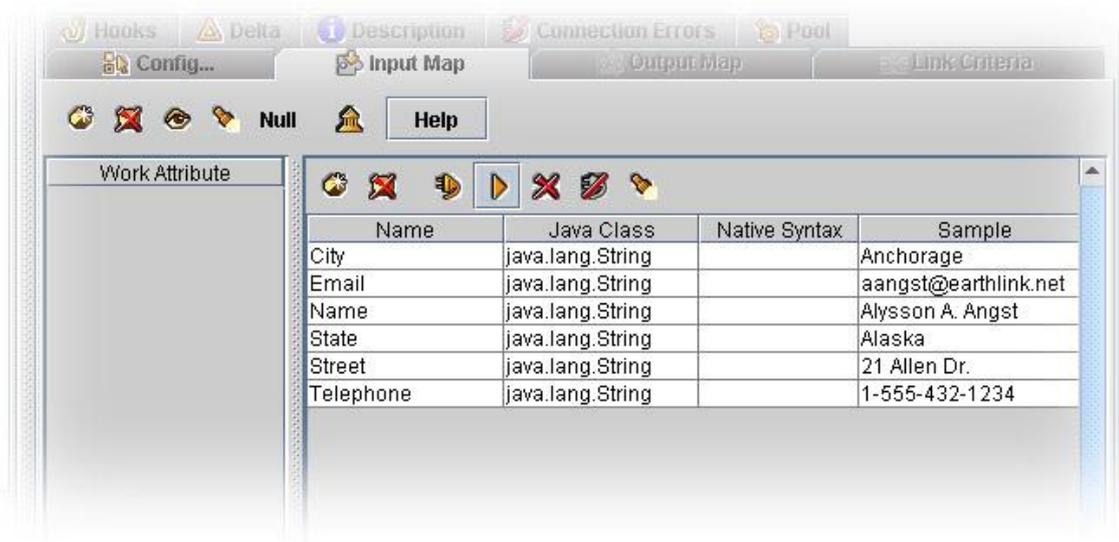
Although you are now getting six separate attributes, the field names are still not right. That's because the CSV Parser expects the first line of our .csv file to provide the names of these fields. Since our file does not, you will have to specify these in the Field Names parameter of the Parser. The fields are *Name*, *Email*, *Street*, *City*, *State* and *Telephone*, in this order<sup>12</sup>.



<sup>11</sup> Otherwise any new attributes read will simply be added to what has already been discovered.

<sup>12</sup> As shown in the screenshot, you can enter each attribute name on a separate line. Alternatively, you can put them all on a single line using the Field Separator specified for the Parser.

Once you've specified the field names then re-discover the data again with the **Connect** and **Read Next** buttons.

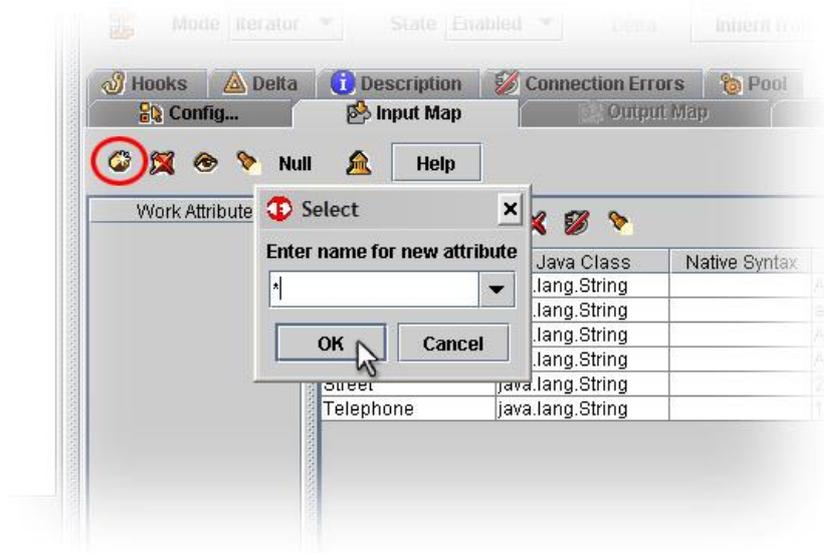


Now that our Connector is configured and the data correctly discovered, the next step is to specify which attributes to bring into the AssemblyLine for processing. You will instruct the Connector to map all these fields into the AssemblyLine by clicking on the **Add new attribute** button at the top of the map and then specifying the wildcard: the asterisk (\*)<sup>13</sup>.

<sup>13</sup> The wildcard map performs an automatic Simple mapping of all attributes found in the source entry (e.g. conn or work) to create corresponding attributes in the target entry. You can combine the wildcard with mapping instructions for specific attributes. For example, if you want to map all but one attribute, add both the wildcard and the desired attribute to the Attribute Map. Then select the attribute, change the mapping Type to *JavaScript* and then enter this snippet:

```
ret.value = null;
```

Default Null Behavior for TDI causes any attributes that return a *null* value to be removed from the target entry (e.g. work or conn). This means that even if an attribute with that name was present in the target entry before attribute mapping took place, it won't be after Null Behavior removes it. There is a video tutorial on the Null Behavior feature here: <http://www.tdi.users.org> under the Learning TDI Online link.



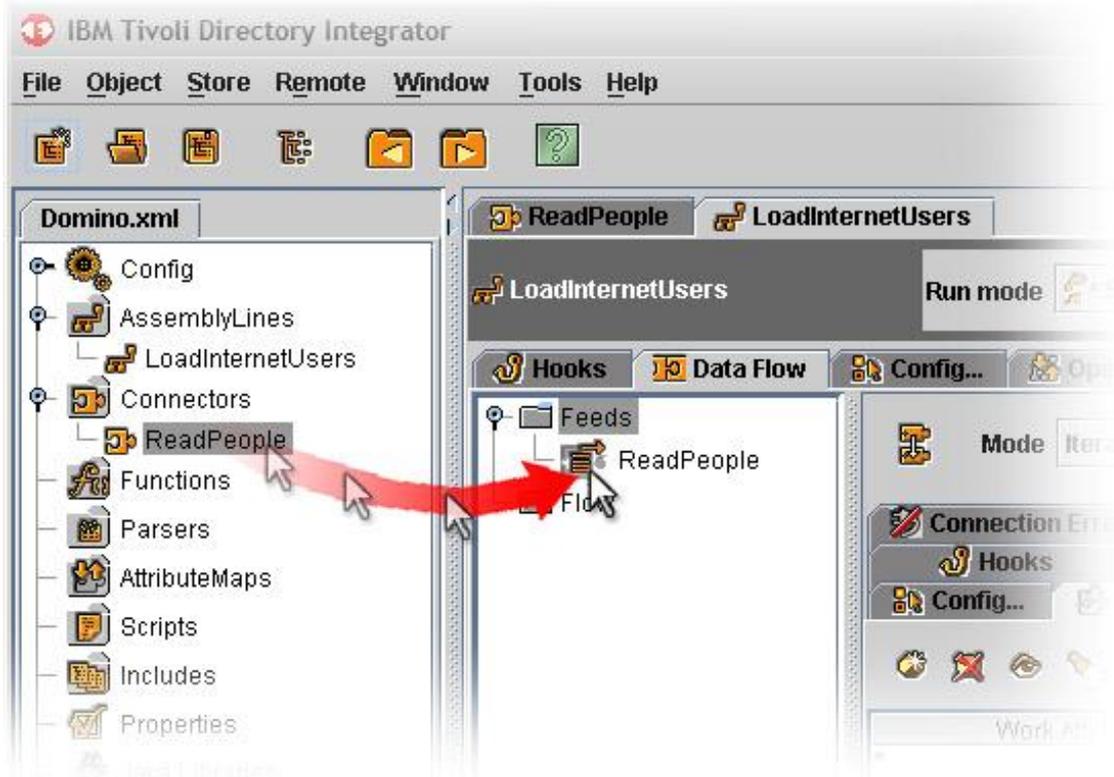
Your 'ReadPeople' library Connector is now in place and ready to use (and re-use).

## 2.2. Creating The AssemblyLine

Now it's time to create an AssemblyLine to make use of your library Connector. Do this by right-clicking on the **AssemblyLine** folder and selecting *New AssemblyLine*.



Call this AL "LoadInternetUsers" and then drag the "ReadPeople" Connector into the Feeds section of the AssemblyLine DataFlow list.

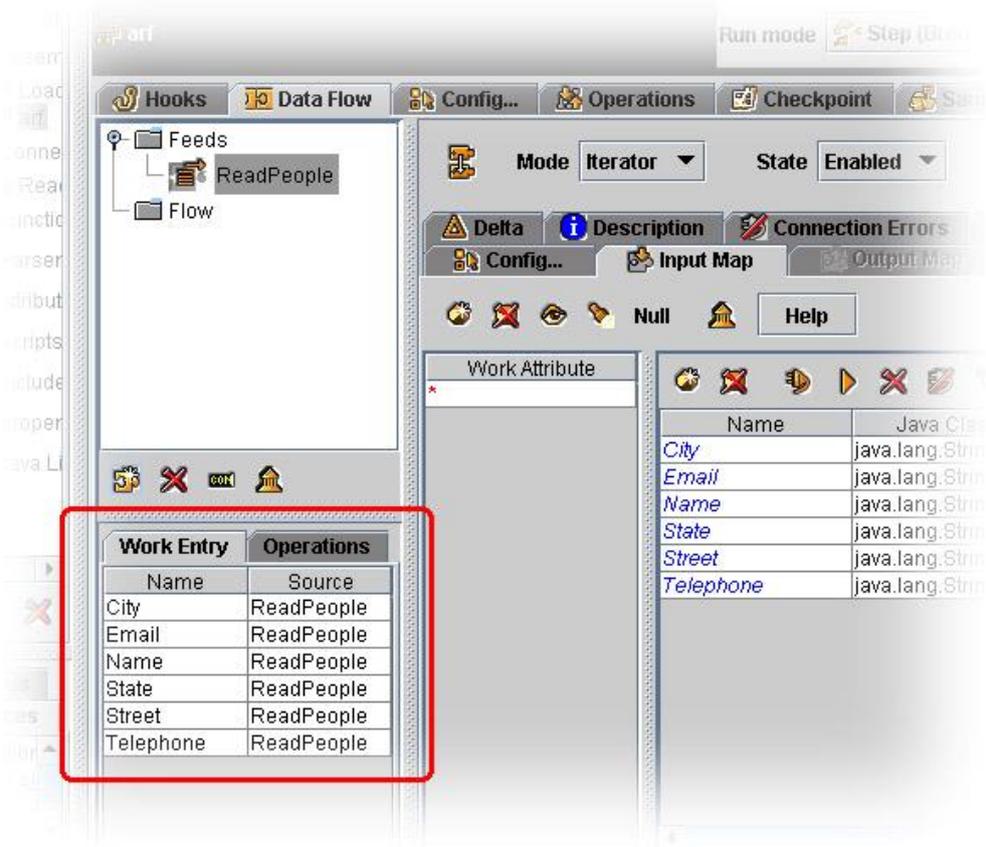


TDI 'remembers' that you dragged this Connector into the AL and will automatically link its configuration to that of the *parent* Connector<sup>14</sup>. In other words, the AssemblyLine Connector *inherits* the parameter values and Parser setup from your library Connector, and therefore requires no further configuration.

The AL Connector also inherits the Input Map of your library Connector. This wildcard map causes all attributes in the Connector Schema area to show up in the Work Entry panel.

---

<sup>14</sup> Of course you can break this inheritance for any configuration parameter or feature. For example, If you look at the **Config** tab of "ReadPeople" in your AL then you'll notice that all the parameters are in blue italics font. This means that they are *inherited* from the parent component. You can break inheritance by typing in the parameter field. You restore inheritance by clicking on the label of the parameter and then on the **Use original value** button in the resultant Parameter dialog.



The Work Entry panel is like a window into your AssemblyLine, showing you at all times which attributes are being processed and where they come from<sup>15</sup>.

### 2.3. Adding the LDAP Connector

As mentioned previously, the LDAP Connector only needs the LDAP task active on a Domino Server in order to get a connection. You will also need a registered Internet User with adequate privileges to access the Domino directory.

The most commonly used parameters in this component are:

**LDAP Url** This is in the format `ldap://<hostname>`  
 <hostname> is the IP address or hostname of the Domino server. You can optionally tag on `:<port number>` after the <hostname>, although if you don't then the default (`:389`) is used.

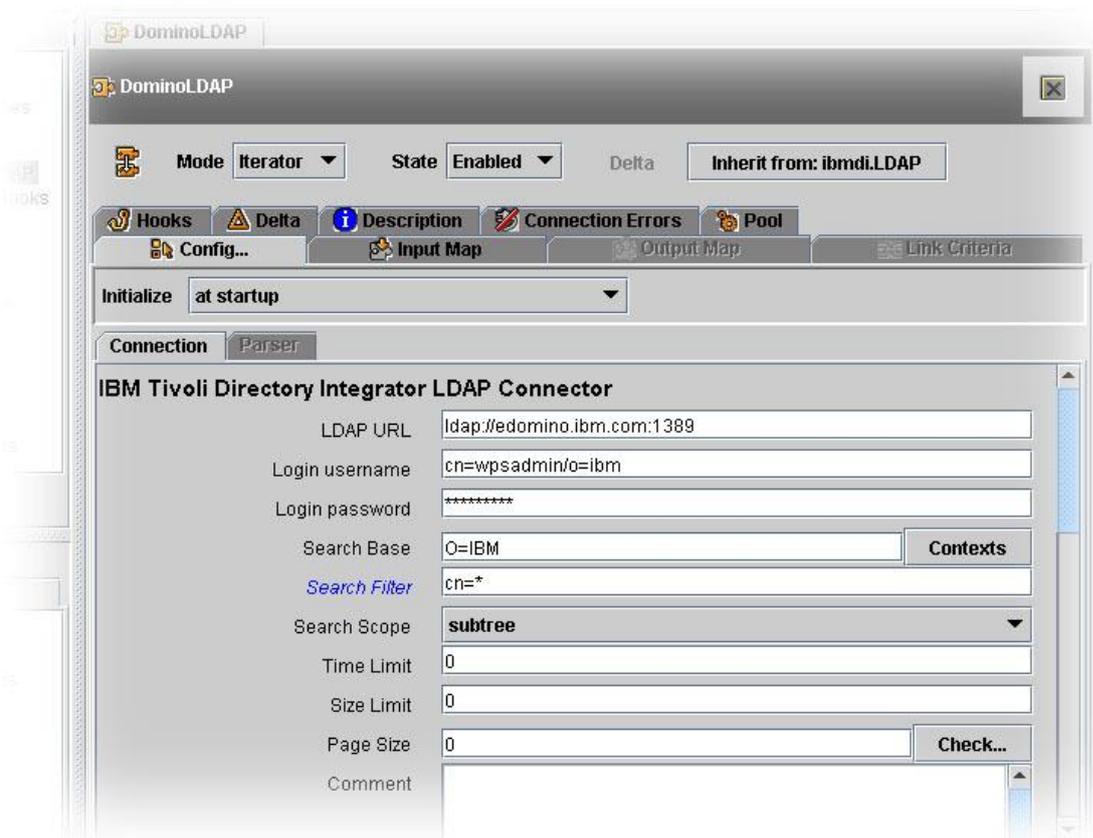
**Login username** The name of a registered Domino internet user, for example `cn=John Andersen/o=Matrix`

**Login** Here you enter the password of the user specified in the

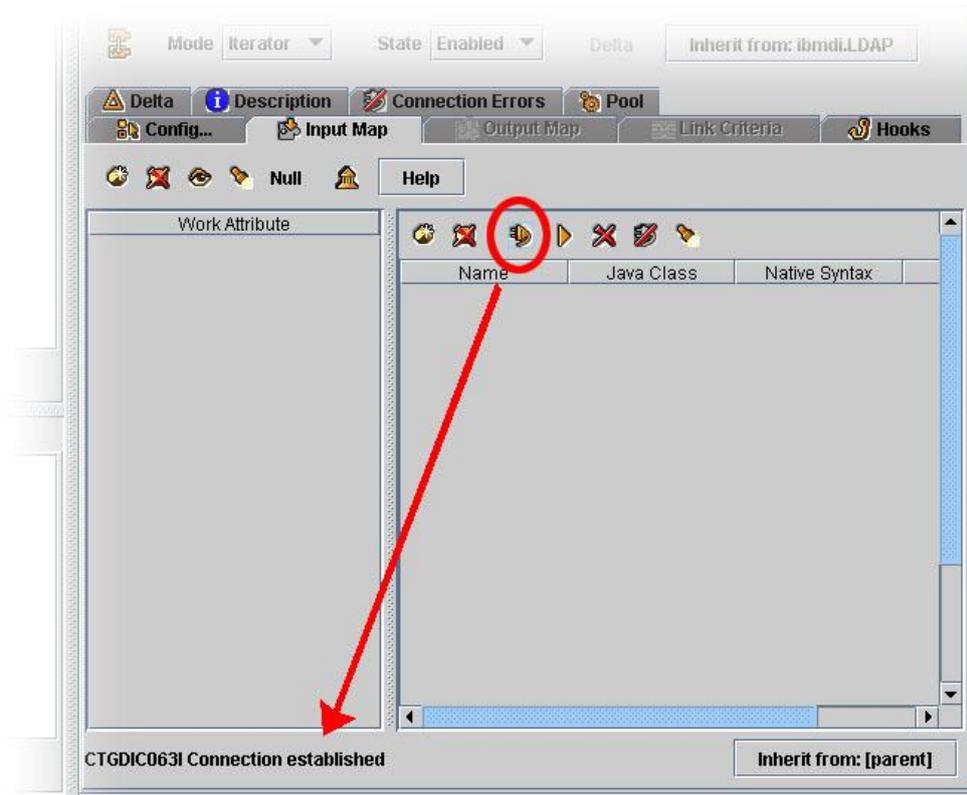
<sup>15</sup> Notice how the Connector Schema is displayed in blue italics font in the screenshot above, indicating that this information is being inherited from your library Connector.

<b>password</b>	above parameter.
<b>Search Base</b>	This parameter sets the point in the directory tree below which you want to search. The value you enter here must be the distinguished name (dn) of an entry in the tree, or a published suffix; For example: <code>ou=Agents,o=Matrix</code>  Note that this parameter is only used for Iterator and Lookup modes.
<b>Search Filter</b>	Here you specify the LDAP search filter to use for Iterator mode.

Add an LDAP Connector to your AL by right-clicking the **Flow** node in the AL DataFlow list, or by pressing the **Add Component** button at the bottom of this list. Choose the *LDAP Connector*, call it 'DominoLDAP' and select **AddOnly** mode. In TDI your Connector Config panel should look something like this:



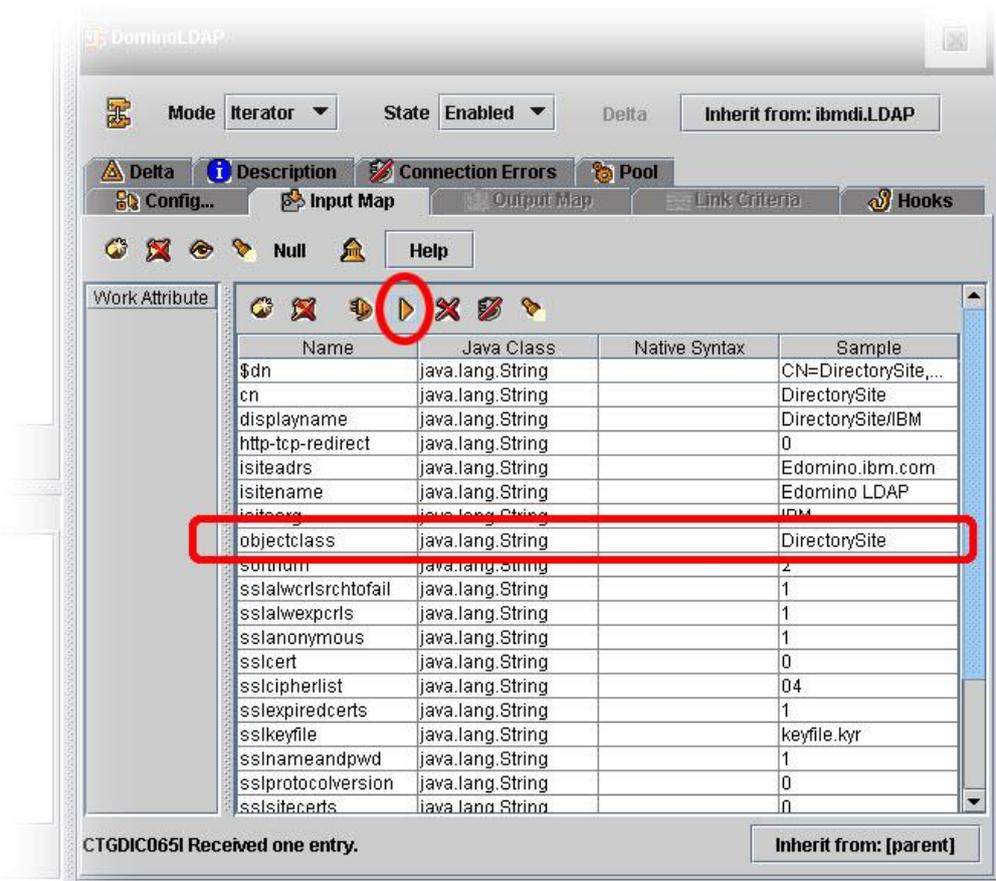
As before, once you've configured your Connector then test your configuration by selecting the **Output Map** tab and pressing the **Connect** button just as you did in the previous section.



The *Connection established* message<sup>16</sup> tells you that the URL and login credentials are all valid. You can now press the **Read Next** button and the attributes of the first entry read appear onscreen.

---

<sup>16</sup> You will notice that TDI gives you a connection status message at the bottom of the Input Map panel, for example *Connection established*. Whenever you have a connectivity problem, you will see an error message appear here. Click on the message in order to get more details. This can be helpful when trying to diagnose the problem or report an incident to TDI support.



Each time you press the **Read Next** button, the attributes from a new entry are added to the list already displayed in the Connector Schema area. If you only want to see the attributes belonging to the current Map entry then press the **Clear** button (red X) before pressing **Read Next**.

As you step through the data in your Domino server you'll probably notice that you can access more than just Internet Users, including directory sites, containers (OU's) and groups. You can tell the type of an LDAP entry by the value of its *objectClass* attribute.

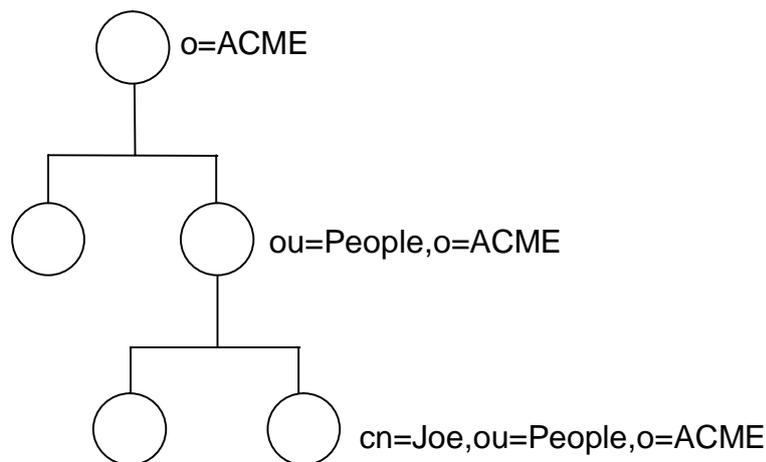
We will just be focusing on the **dominoPerson** type for this exercise, the LDAP objectClass that maps directly to the Form *Person*<sup>17</sup>. Control the scope of entries returned to only those of type **dominoPerson** by entering the following value in the **Search Filter** parameter of your LDAP Connector:

```
objectclass=dominoPerson
```

Now when you connect and step through your Domino directory, you will only see entries representing Internet Users<sup>18</sup>.

Another attribute worth noting here is the one called “\$dn”. It contains the *distinguished name* of the entry. The distinguished name (or *dn* for short) is like the filepath to a file on your PC. As you know, the filepath is not part of the file itself but rather its location on disk. Only the last part of the filepath is information carried by the file (i.e. it’s name), and this part must be different than the names of all the other files found in the same sub-directory.

In the case of a *distinguished name*, it is the first part of the path value that is information carried by the entry and which must be different than that of all other siblings under the same parent node. This part is called the *relative distinguished name* and is based on an attribute value stored in the entry.



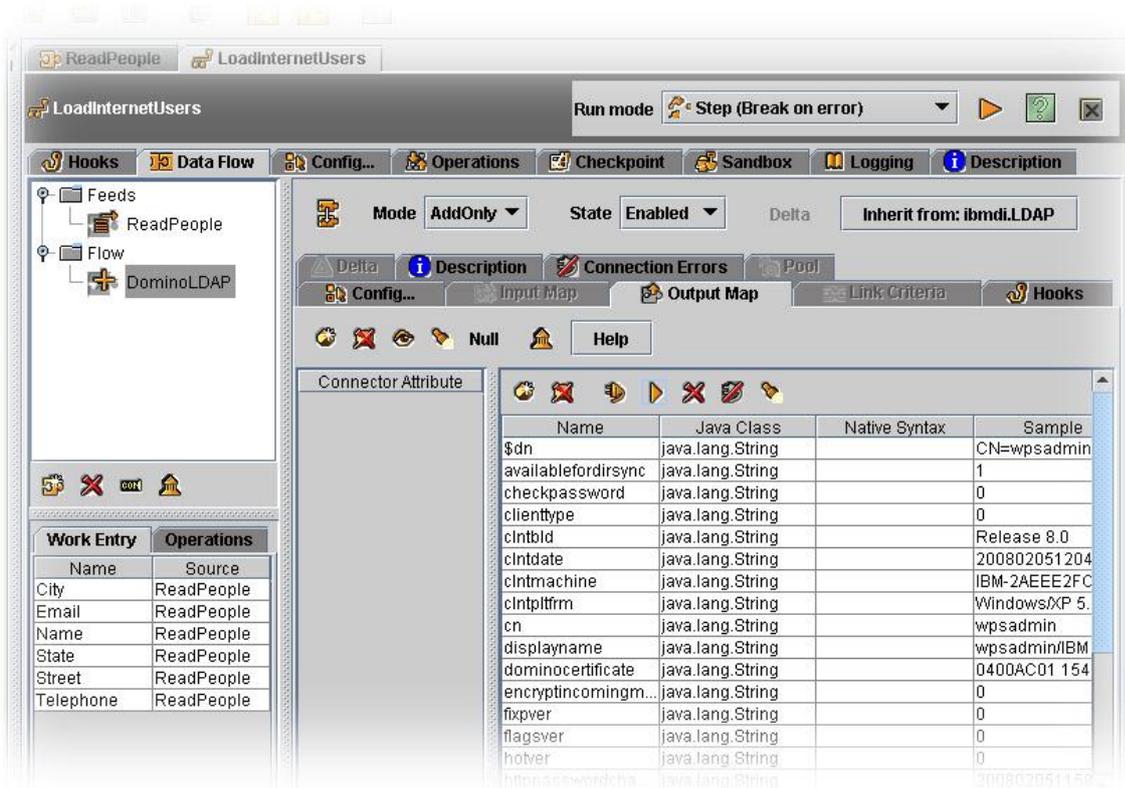
For example, the entry above with dn equal to “cn=Joe,ou=People,o=ACME” must have a “cn” attribute with a value of “Joe” and reside under a parent (ou=People) that hangs under the directory root (o=ACME).

<sup>17</sup> Just as the objectClass **dominoGroup** maps to the Form *Group*.

<sup>18</sup> By simply adjusting relevant configuration settings – for example, the Search Filter or Search Base parameters in our LDAP example – and then using the data discovery buttons (Connect and Read Next) to issue the search and browse the results, TDI becomes a handy tool for exploring all kinds of user data, including Notes .nsf files and relational databases (RDBMS).

Although the dn is not actually contained in the entry, TDI returns it as the specially named attribute *\$dn*. This attribute is always present when you read from a directory, and it must be present when you create new entries as well. We'll look at this later in this tutorial.

Your AL now has two Connectors: one in input mode (Iterator) and the other configured for output (AddOnly).



This should tell you something basic about how TDI works: information is first read into the AssemblyLine using one or more Connectors in an *input* mode (Iterator or Lookup). Once it's in the AL, data is then filtered, transformed and finally output to target systems using Connectors in an *output* mode (AddOnly, Update or Delete<sup>19</sup>).

## 2.4. Mapping to dominoPerson Schema

In order to be able to create new dominoPerson entries in Domino, you must first map your input data to the output attributes defined for the dominoPerson object class<sup>20</sup>. The dominoPerson objectClass has only three mandatory attributes that

<sup>19</sup> There are other Connector modes as well, but these are the most commonly used.

<sup>20</sup> Here is a link to information about the dominoPerson schema in the online docs for Domino R8: <http://publib.boulder.ibm.com/infocenter/domhelp/v8r0/index.jsp>.

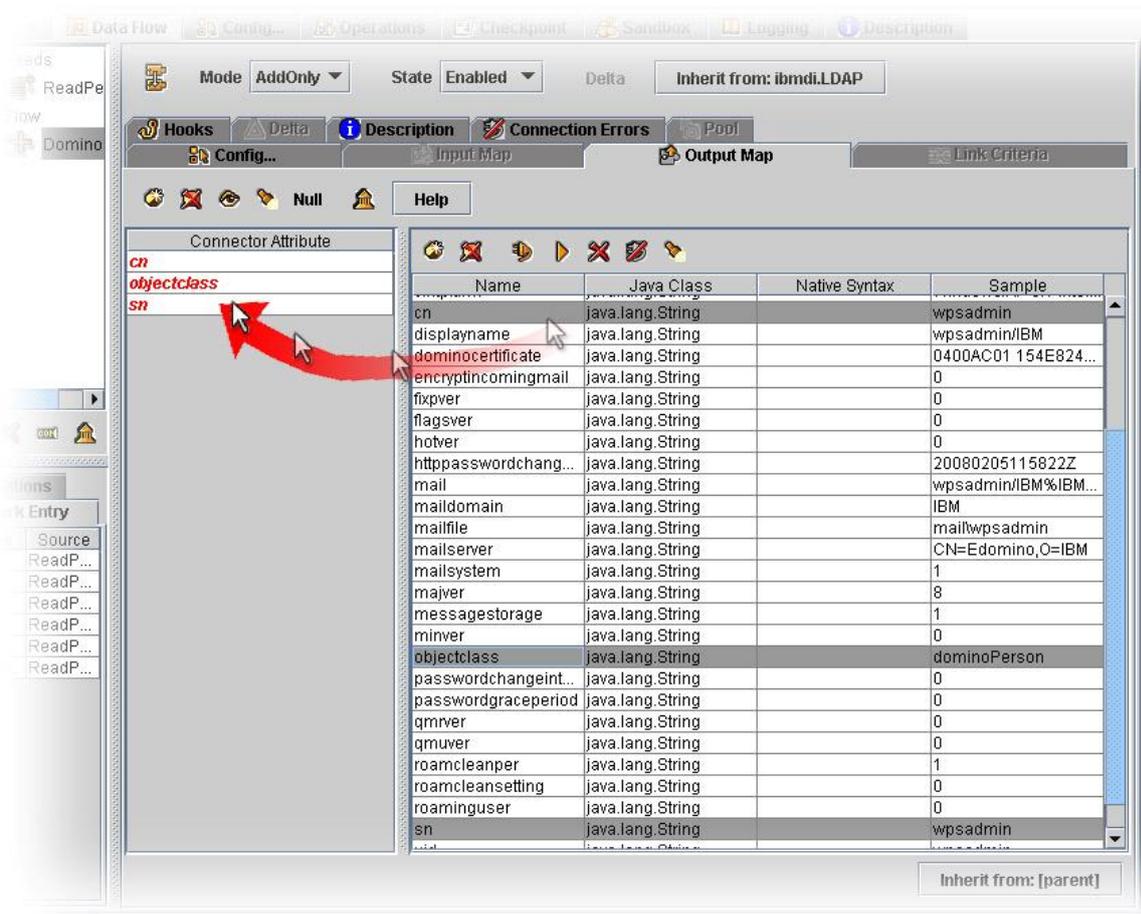
must be present when creating an entry<sup>21</sup>:

**objectClass** corresponds to the *Type* field in a document, and will have the value "dominoPerson"

**cn** which is the *CN* field

**sn** called *LastName* in Domino Person documents

Start by control-clicking to select these fields in the Connector Schema and then dragging them into the Output Map.



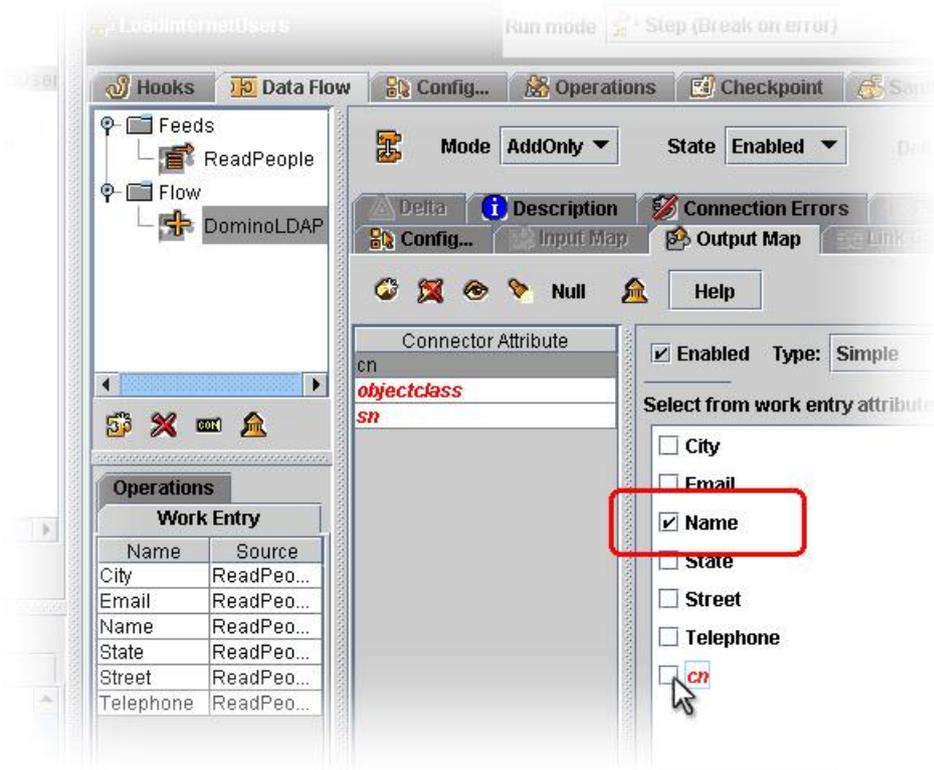
When you drag these into the Output Map then TDI assumes you are mapping values from similarly named attributes in the Work Entry. Since there are no *cn*, *objectClass* or *sn* attributes coming from our "ReadPeople" Iterator<sup>22</sup>, these show up

<sup>21</sup> Of course, if you want to use these entries for authentication purposes (e.g. for Portal or Sametime), you will need to write a userPassword attribute as well.

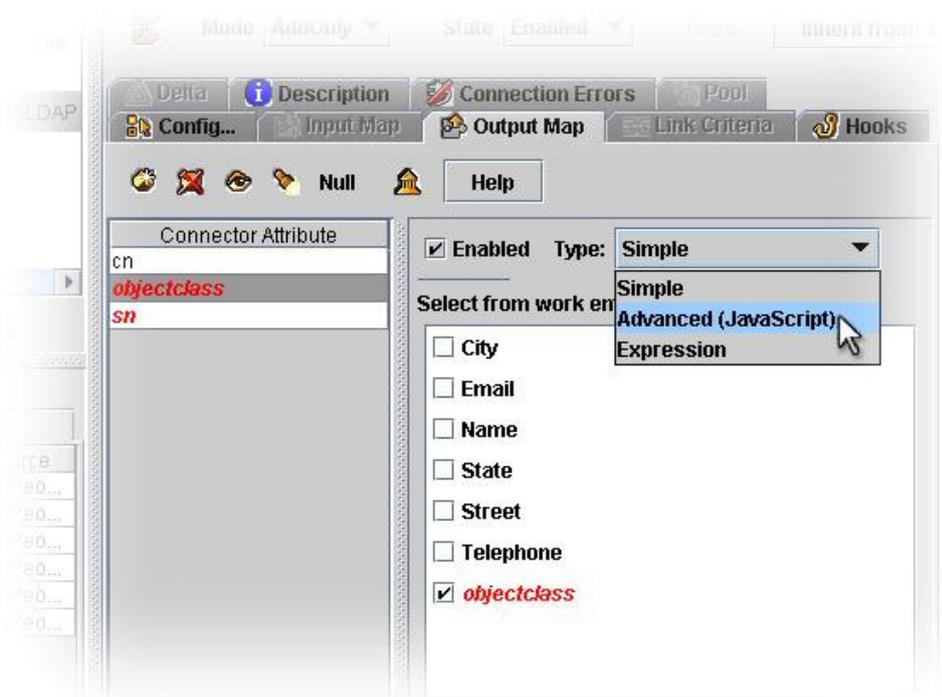
<sup>22</sup> A Connector in Iterator mode is often referred to as an *Iterator*.

red in the map. To fix this you will have to select each attribute and correct the mapping.

Start with the *cn* attribute by clicking on it in the map. The right-side panel changes to show you the list of available Work Entry attributes, plus the assumed (but missing) *cn* choice in red at the bottom. You can correct this by unselecting the invalid *cn* attribute and instead choosing *Name*.



Now to correct *objectClass* you must select it and then change type Mapping Type from Simple to JavaScript.



Enter this snippet in the Script Edit window that appears where the Work Entry attributes list was:

```
ret.value = "dominoPerson";
```

This JavaScript code will return the literal text “dominoPerson” as the value of the attribute named *objectClass*<sup>23</sup>.

Now to fix the mapping of *sn*: again you select this item in the Output Map, change the Mapping Type to *JavaScript* and then enter this script snippet which will grab the last part of the “Name” attribute and return this:

```
// return the Name attribute value as a Java String
name = work.getString("Name");
// split into an array of Strings based on spaces (" ")
parts = system.splitString(name, " ");
// return the last item in this array
ret.value = parts[parts.length-1];
```

In addition to including at least the three mandatory attributes discussed above, the LDAP add-operation must also specify *where* in the directory tree the new entry is to be located. This is done by mapping out to the “\$dn” attribute.

<sup>23</sup> Even though the dominoPerson objectClass is derived from inetOrgPerson, Domino does not require that you enter the entire hierarchical class structure. This is still easy to do with TDI as you can see from this example mapping script:

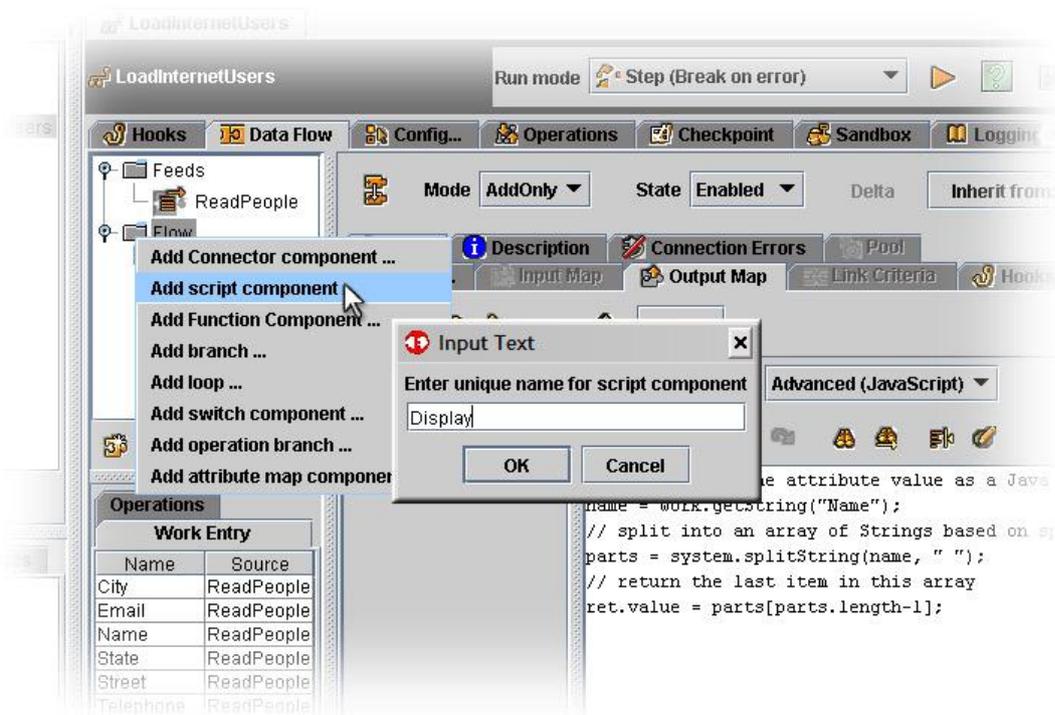
```
ret.value = ["top","person","organizationalPerson","inetOrgPerson","dominoPerson"];
```

To do this, either drag the \$dn attribute from the Connector Schema into the Output Map list, or press the **Add a new attribute...** button  at the top of the Output Map and call this new attribute "\$dn". Switch the mapping type from Simple to JavaScript and then enter this code:

```
ret.value = "cn=" +
            work.getString("Name") +
            "ou=People,O=ACME";
```

This will use the cn attribute as the *relative distinguished name* and place the entry under the ou=People container below the o=ACME root suffix. You will of course have to modify this path value to fit the structure of your Domino directory.

Although our AL is ready to use now, it could use with a little polish. So add a Script component (also known as an SC) and call it "Display".



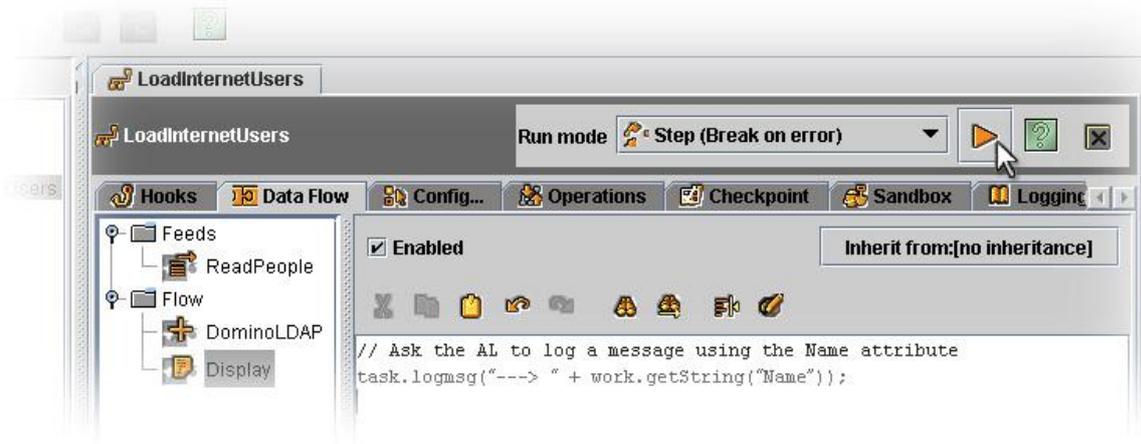
Enter the following script code into the Script Edit window for this SC:

```
// Ask the AL to log a message using the Name attribute
task.logmsg("--->" + work.getString("Name"));
```

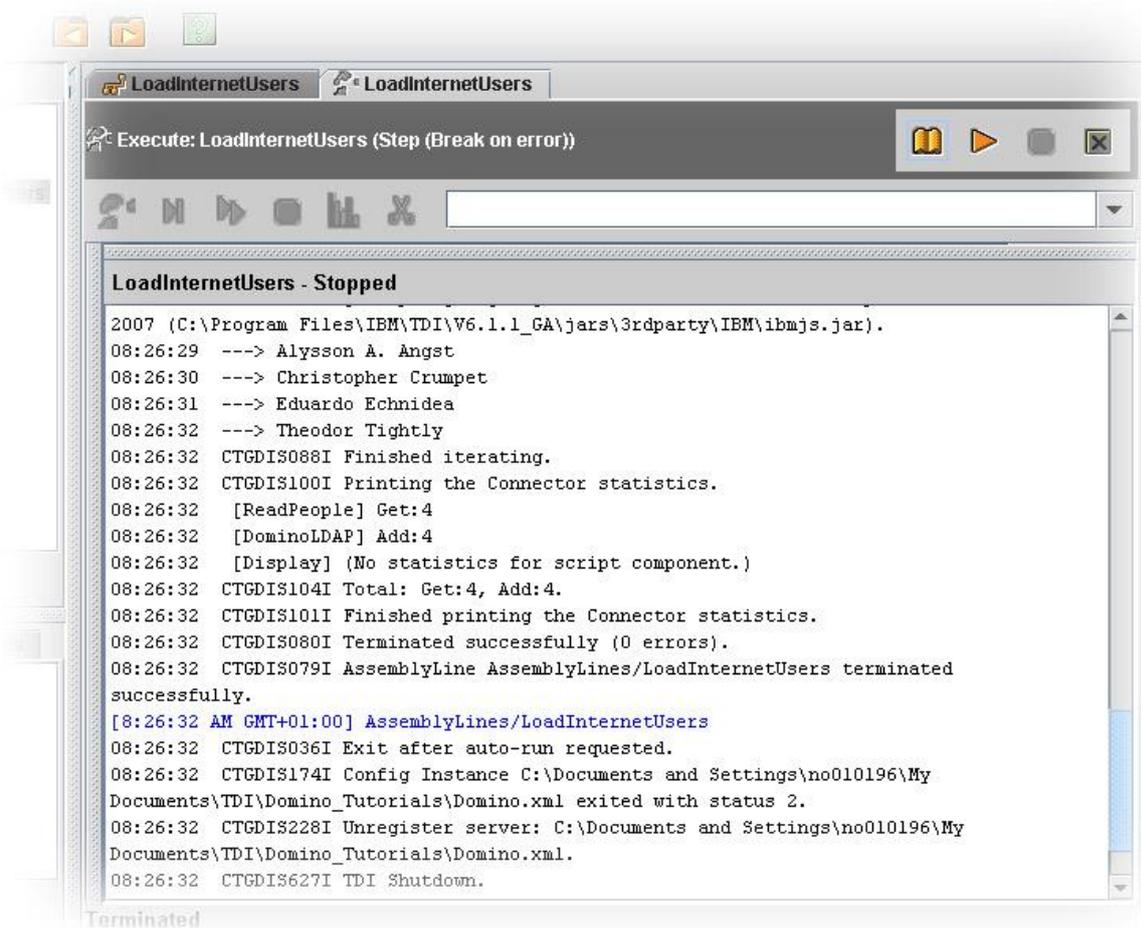
Now you will get status information on the progress of your AL<sup>24</sup>.

<sup>24</sup> It doesn't matter where the Script component is located in your AL Flow section. If you drag it above your DominoLDAP Connector then the message will be logged *before* the actual write operation; if you leave the SC at the bottom of the AL then the logging will happen *after* the entry is added to Domino.

Once this is in place you are now ready to run your AssemblyLine. Do this by pressing the **Run** button at the top right-hand corner of the AL Detail pane.



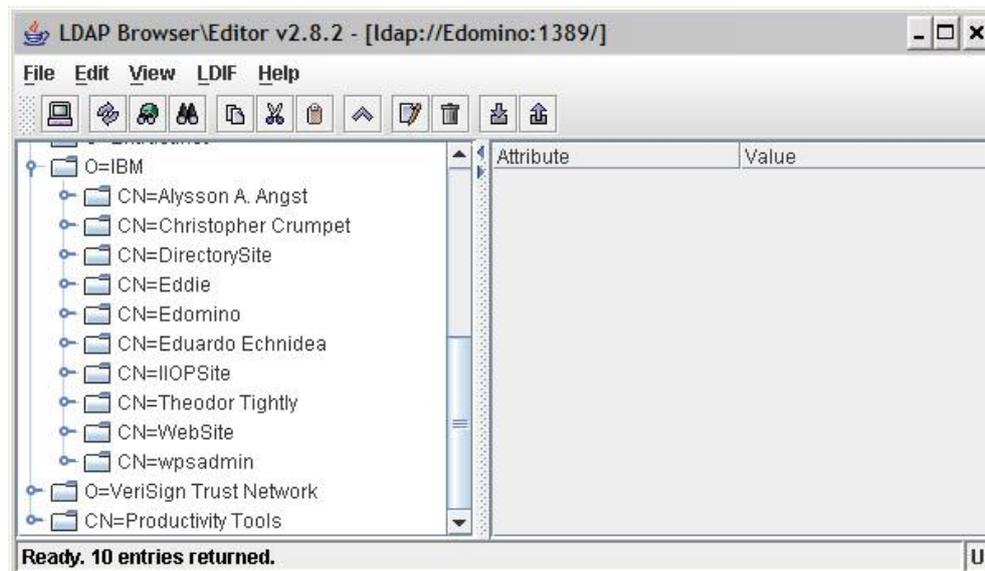
This causes the CE to launch a new instance of the TDI Server, connect to it, pipe over the Config that you are working on, and finally to instruct this Server to run your AssemblyLine and then stop. The Config Editor then catches all log output and displays it onscreen in a *Run window*. Once the AL completes, the launched Server shuts down and the icon in the Run window tab becomes grayed.



From the log output we can see our own status messages as well as component statistics:

```
[ReadPeople] Get: 4
[DominoLDAP] Add: 4
[Display] (No statistics for script component)
Total: Get:4, Add:4.
```

You can confirm that there are four new entries in our Domino directory either by using your DominoLDAP Connector (Output Map → Connect + Read next buttons) or with an LDAP Browser<sup>25</sup>.



## 2.5. Modifying existing users

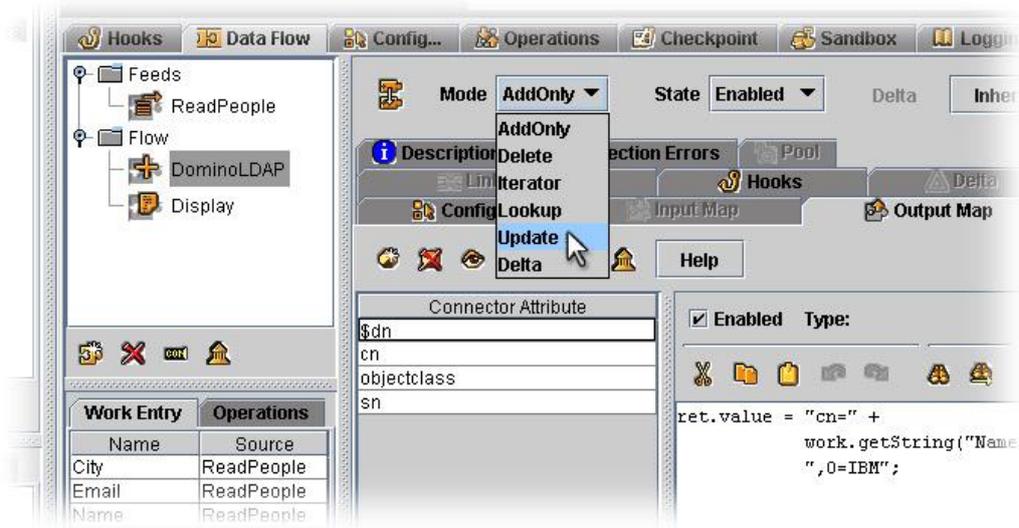
Using AddOnly mode in the DominoLDAP Connector will only allow us to create new entries in the directory. If you want to be able to also modify existing users as well then you will need to change the mode of this Connector to *Update*.

Update mode first performs a lookup-operation based on search criteria defined by you. If a matching entry is found then it is modified; if no match is found then the Output Map data is used to create a new entry – just like AddOnly mode.

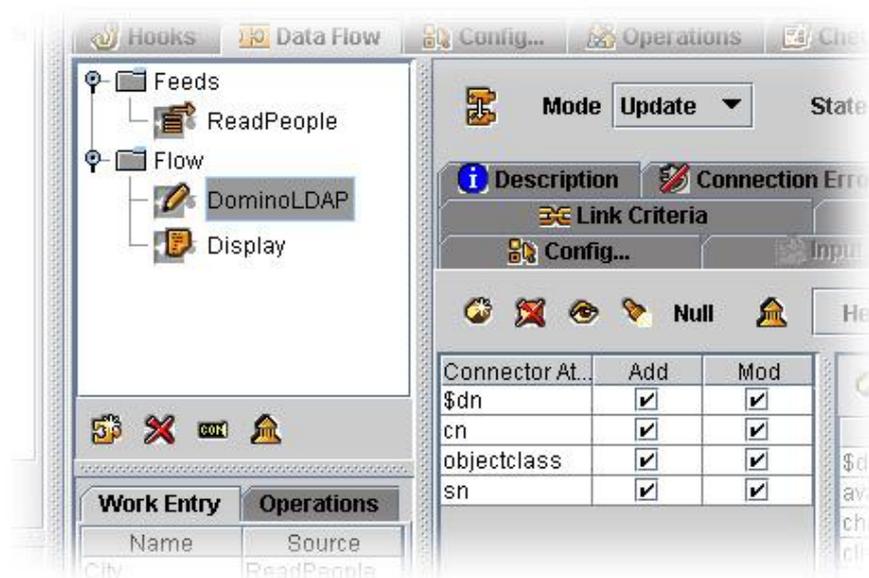
Change the mode of your DominoLDAP Connector from AddOnly to Update.

---

<sup>25</sup> Note: You can modify your LDAP configuration using various tools. In our scenario, we use the LDAP Browser/Editor (LBE) from Argonne National Laboratory. It is commercially available at the following Web site: [http://www.anl.gov/techtransfer/Software\\_Shop/LDAP/LDAP.html](http://www.anl.gov/techtransfer/Software_Shop/LDAP/LDAP.html)



Now close your AL window by pressing Ctrl-W or clicking on the Close Window button at the top right-hand corner of the AL details screen. Then re-open it and select the DominoLDAP Connector again<sup>26</sup>. You should now see that two rows of checkboxes have appeared in the Output Map.



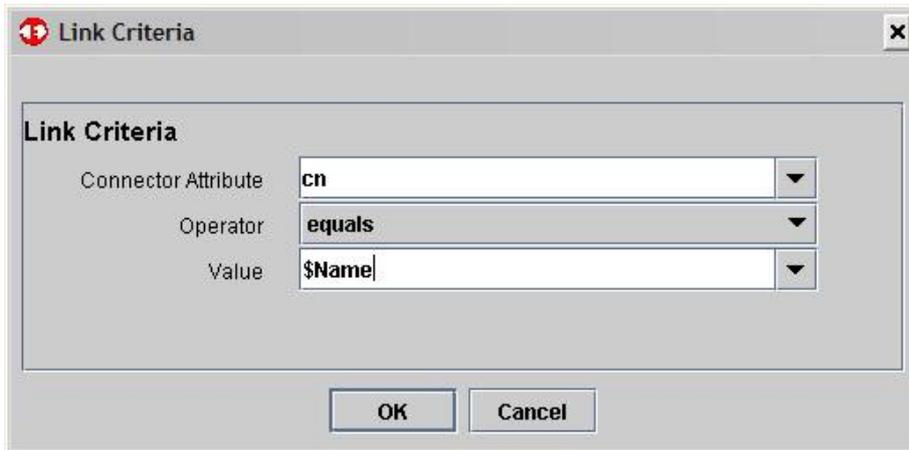
These checkboxes allow you to enable/disable the mapping of attributes for either the add- or modify-operation, depending on whether the lookup found a match or not.

Uncheck the Mod boxes for the \$dn, cn and objectClass attributes. Although we need these when we create a new entry, these will not be changed by a simple modify-operation. Changing the values of an entry's \$dn or cn (it's rdn) is the same as renaming the user, and this must be done using special calls to the Domino api.

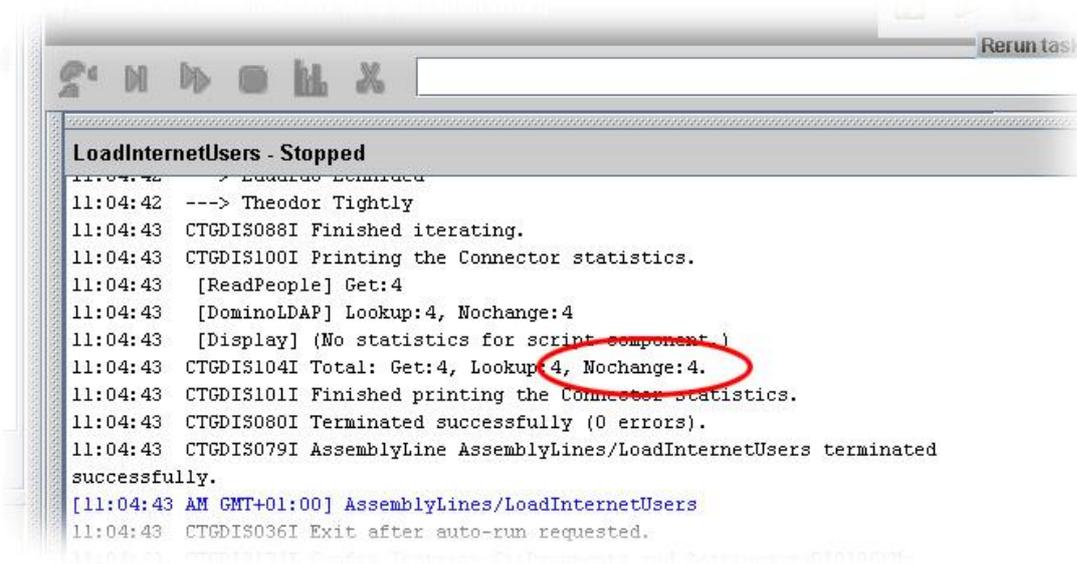
<sup>26</sup> This is to force TDI to redraw the Output Map.

Likewise, objectClass will not be changing either, so the only attribute left enabled for modify is *sn*.

Before your Update-mode Connector will work, you must also define the search criteria used to find which Internet User to modify. This is done in the LinkCriteria tab by adding a simple LinkCriteria that searches the *cn* attribute in Domino LDAP for the value in the *Name* attribute<sup>27</sup>.



Run your AL again to make sure it is working as desired and you will notice something new:



The statistics claim that 4 entries were not actually changed. This is a feature of Update mode called *Compute Changes*.

<sup>27</sup> That's what the dollar-sign means: use the value of the named attribute in the Work Entry. So '\$Name' means 'The value of the *Name* attribute'.



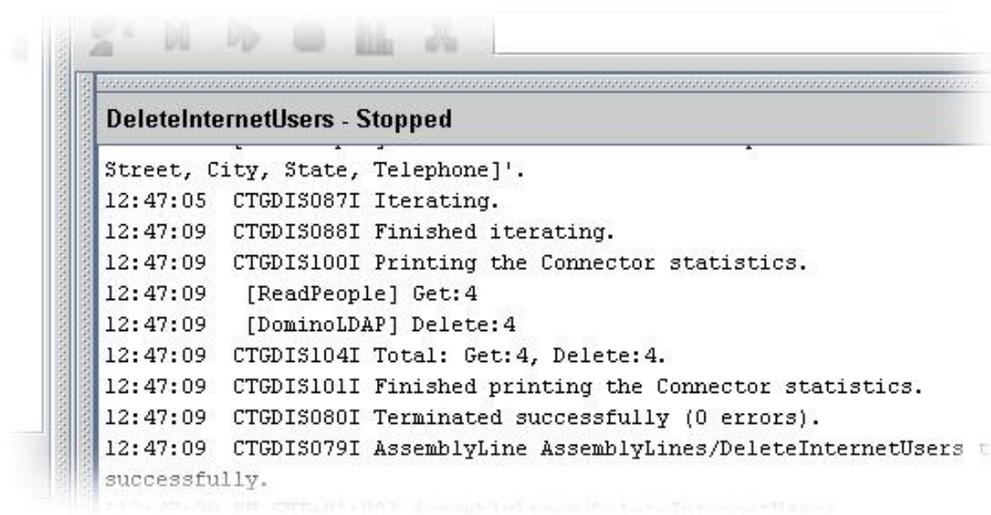
Compute Changes instructs the Connector to compare the attribute mapped out with those returned by the lookup-operation. If any differences are detected then the modify-operation is carried out, otherwise no write is performed.

## 2.6. Deleting Internet Users

Create a new AL and call it “DeleteInternetUsers”. Use the ReadPeople Iterator in the Feeds section and then drag in the “DominoLDAP” Connector and set it to *Delete* mode.

Delete mode also requires Link Criteria. If you had added Link Criteria to the library Connector (in the Connectors folder of the Config Browser) then when you dragged the component into your AL, this setting is inherited automatically. If you instead configured Update mode in the AssemblyLine Connector of your *LoadInternetUsers* AL then you need to re-create this Link Criteria again: `cn equals $Name`.

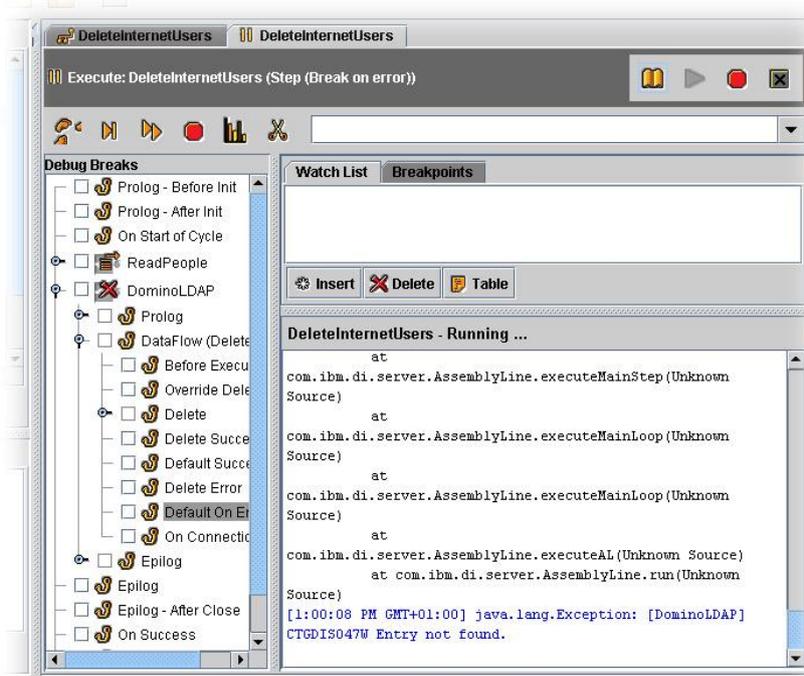
You now have an AL to remove these test users from your Domino directory.



A side note here is that you should probably rename the *DominoLDAP* Connector to something that provides more meaning to someone viewing your AssemblyLine; for example, “*DeleteInternetUser*”.

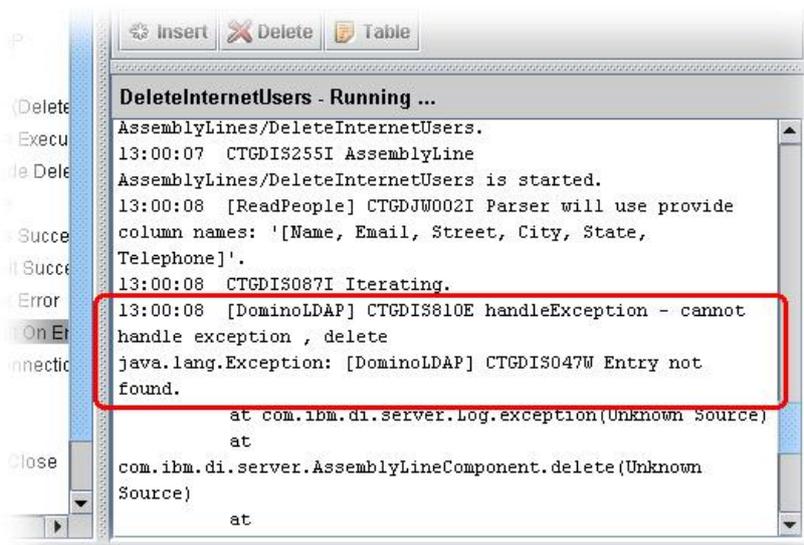
## Using Hooks to handle exceptions

If you run this AL a second time then you will see that it aborts with an exception:



By default, AssemblyLines are run in *Step (Break on Error)* mode, which means that in the case of an exception, the Debugger pauses the AL and gives you a set of powerful tools for uncovering and correcting the error<sup>28</sup>.

Whenever you get an error situation like this, your first step is to scroll up the Log Output window to the top of the very first Java stack-dump. Here you will find info about the cause of the error:



<sup>28</sup> For a quick intro to the AL Stepper/Debugger see video tutorial #6 on this page:

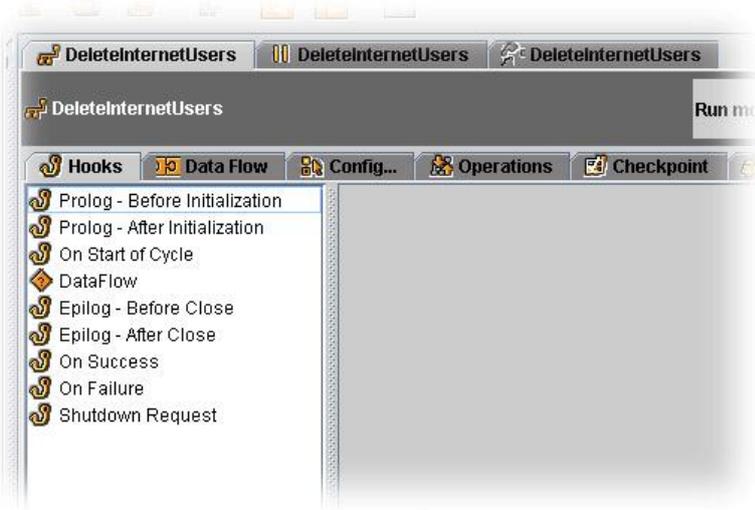
<http://www.tdi-users.org/twiki/bin/view/Integrator/LearningTDI>

There is also a write-up on the various Error Handling concepts and features in TDI here:

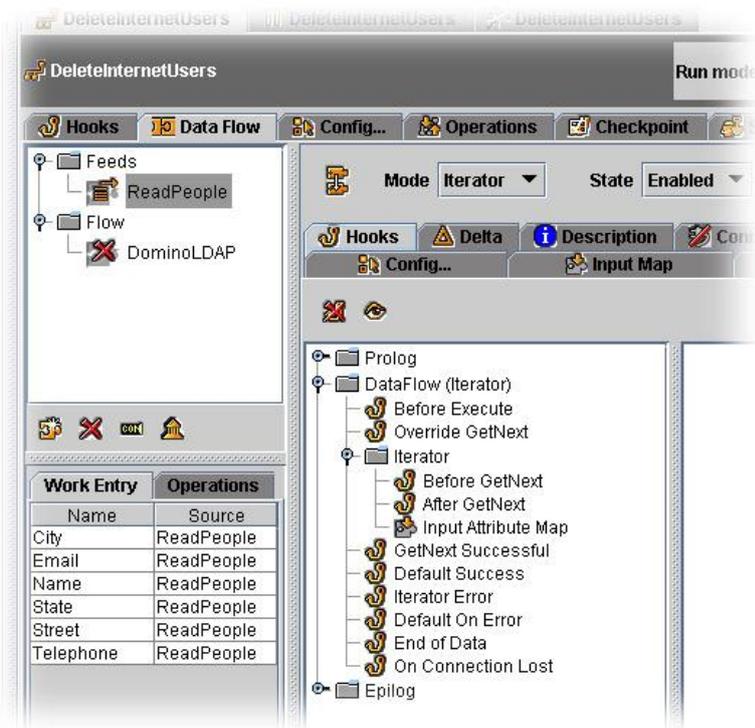
[http://www.tdi-users.org/twiki/pub/Integrator/HowTo/HowTo\\_HandleErrors.pdf](http://www.tdi-users.org/twiki/pub/Integrator/HowTo/HowTo_HandleErrors.pdf)

From the above message you can see that the DominoLDAP Connector failed during the delete operation. The actual error message is *Entry not found*. This means that the lookup-operation performed prior to the delete failed to find a matching entry. In order to customize the handling of these exceptions, you will need to use a *Hook*.

Hooks are waypoints in the built-in behavior of the AssemblyLine and its components where you can write JavaScript. Your code will then extend or even override the built-in behavior at this point. Hooks for an AL are found under the **Hooks** tab at the top of the AssemblyLine details panel.

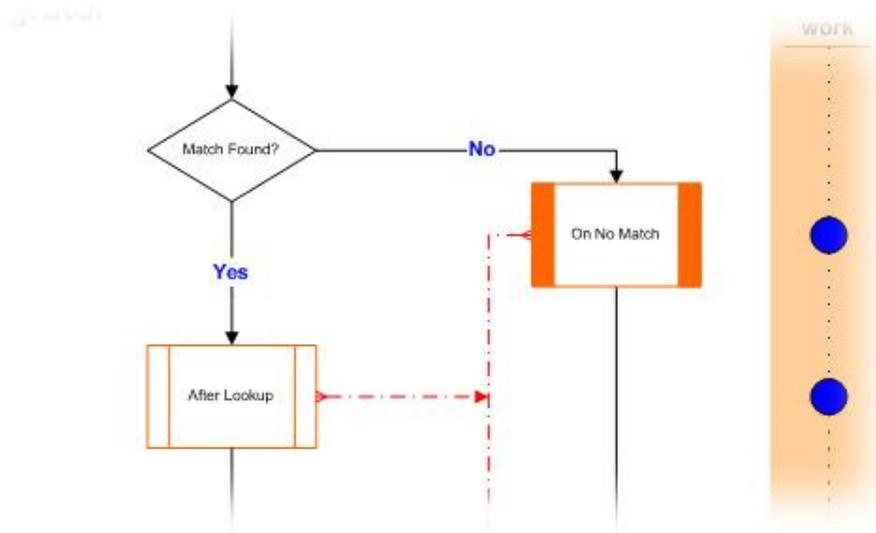


Hooks for Connectors and Function components are available under the **Hooks** tab for that component's Details screen.



These built-in behaviors are documented in the TDI Flow Diagrams<sup>29</sup>, and the Debugger allows you to step from Hook to Hook and explore how these workflows behave.

If you bring up the second page for Delete mode then you can see that if no matching entry is found, control is passed to the *On No Match* Hook.



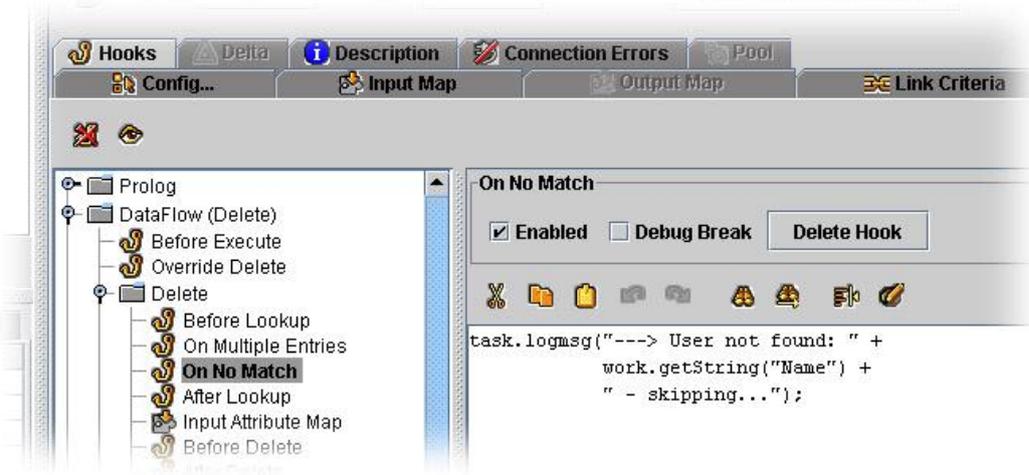
This Hook is drawn with solid side-bars, indicating that it is a *mandatory* Hook. Whenever the flow ends up at a mandatory Hook then this Hook must at least be enabled, otherwise the AL aborts with an error.

Enhance your AL to handle this situation by entering the following code in the *On No Match* Hook:

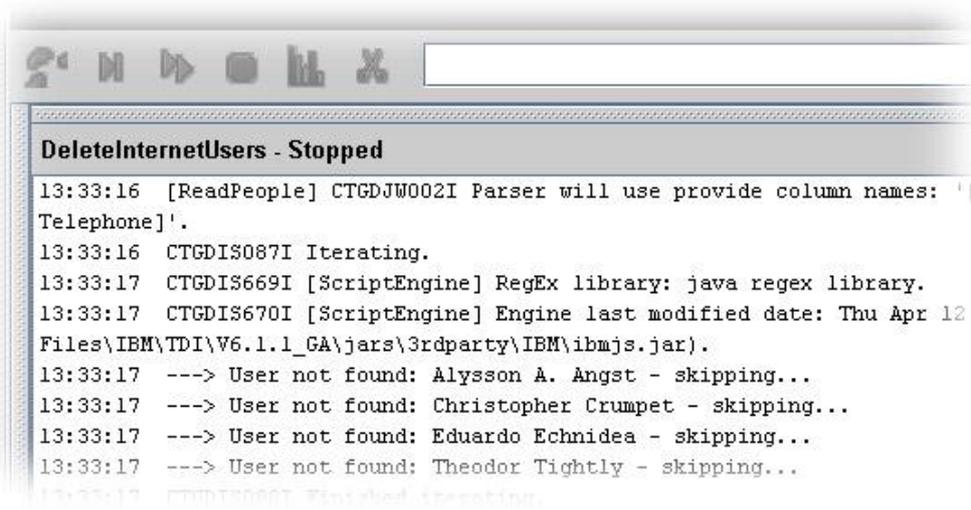
```
task.logmsg("---> User not found: " +
            work.getString("Name") +
            " - skipping...");
```

---

<sup>29</sup> These diagrams are part of the TDI documentation, and can be found here:  
[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide148.htm#connectmode](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide148.htm#connectmode)  
The diagram for Delete mode is here:  
[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide152.htm#wq904](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide152.htm#wq904)



Now if your AL tries to delete a non-existent user, you will see this message in the log output:



Now you should be able to expand on your solution to include the other input attributes, mapping these as you see fit to the dominoPerson schema.

### 3. Domino-specific Connectors

While the LDAP Connector provides easy access to data in the Domino directory, you will need to use the other Domino-specific components if you want to provision Notes accounts, work with other .nsf databases than `names.nsf` or handle Document fields not available via LDAP (e.g. *Form*).

#### 3.1. Choice of connections

With the exception of the Domino Change Detection Connector (which must connect through a local Notes client), the TDI Domino components support two connection choices: local or remote. This means that either you have TDI installed on the same machine as a Notes client or as the Domino server itself (*connecting locally*), or TDI connects across the network using a protocol called *IIOp* to Domino running on a different machine (*remote connection*). Deciding which option to use will depend on the topology of your infrastructure, security constraints, server capacity considerations and similar factors. It will also hinge on what you want to do.

The Notes and Domino Change Detection Connectors provide both options for connecting either through a local Notes client or to a remote server via IIOp. The Notes Connector has an additional option for connecting to a locally installed Domino Server. The Domino Users Connector however relies on features not available in the IIOp interface and so can only connect via a local Notes client or directly to a local Server.

Here is an overview of the connection types supported by the various Domino/Notes Connectors.

Connectors	Session types	Local Client	Local Server	IIOp
Domino Change Detection Connector		Yes	No	Yes <sup>30</sup>
Domino Users Connector		Yes	Yes	No
Notes Connector		Yes	No	Yes

It is recommended that you use IIOp when possible while developing your solution since this works best when running ALs from the Config Editor, and does not require any additional setup of a Notes client. When you are ready to deploy your TDI work,

---

<sup>30</sup> The TDI 6.1.1 documentation shows this option as deprecated for the Domino Change Detection Connector. However, this is incorrect and the IIOp option is completely viable for this component. Note that unlike the Notes Connector, you must include a Domino server installation path in the PATH setting as described in this section for *Local Server* connections.

simply switch the connection type for your Connectors to Local Client and run your Config using the `ibmdisrv` command-line script/batch-file.

### 1.1.1 Local Client

The Notes, Domino Users and Domino Change Detection Connectors support this connection option. In order to prepare for connecting through a local client, you must have a Notes client installed. Furthermore, you have to point your Notes client at the server you want to connect to.

The next step is to give TDI access to the `Notes.jar` file found in the `<Notes install directory>/jvm/lib/ext` folder. This can be done in a couple of ways:

- Either copy the `Notes.jar` file to `<TDI install directory>/jars`<sup>31</sup>;
- or set the `com.ibm.di.loader.userjars` property to point to the `Notes.jar` file or the directory where it is located<sup>32</sup>, e.g.:

```
com.ibm.di.loader.userjars=c:/notes/jvm/lib/ext/Notes.jar
```

That takes care of giving TDI access to the local api. However, the Java loader also needs to know where library files are located. You configure this by editing the two batch-files/scripts that are used to start the TDI Config Editor (CE) and the TDI Server: `ibmditk.bat(.sh)` and `ibmdisrv.bat(.sh)`, respectively.

These files are nearly identical and you will be editing the line near the top of each that sets the system path so that it includes the directory where the Notes client libraries are located. Here is an excerpt from the `ibmditk.bat` of a Windows installation:

```
set PATH=C:\Notes;C:\Program Files\IBM\TDI\V6.1.1\...
```

Finally, you have to start your Notes client and log in to the Domino server that you want to connect to. You will also need to restart the TDI Config Editor after making these changes.

As mentioned above, in order to use the Local Client option you have to point your Notes client at the desired Domino server. Configuring your TDI component is done by entering the id-file password and the `.nsf` database you want to work with. These are the only parameters that are used for this connection option, except for with the Domino Change Detection Connector which also requires you to enter the name of the Domino server.

---

<sup>31</sup> Or some sub-folder here, since TDI scans this directory and all sub-directories looking for `.jar` and `.zip` files.

<sup>32</sup> This property is found in your `solution.properties` file (e.g. `My Documents/tdi`), or in `global.properties` found under `<TDI installation directory>/etc` if you do not have a solution directory.

## Special considerations

If you have any experience with TDI then you are used to connecting and stepping through data directly from the Config Editor. This will not work with the Domino/Notes components using Local Client mode. Instead, if you want to look at the data in your Domino database then you will need to configure the Connector for Iterator mode and then place it in an AssemblyLine, allowing it to run and display the contents of the Work Entry: e.g. `task.dumpEntry(work)`.

Another issue that you may have noticed when you started this AL and components initialize is that you get a popup login window for entering the id-file password. You can suppress this popup by bringing up the Domino Notes **User Security > Security Basics** panel and selecting the option labeled “*Don’t prompt for a password from external Notes-based programs*”.

### 1.1.2 Local Server

Both the Notes and Domino Users Connectors support Local Server connections.

As with a Local Client connection, you must first ensure that you have made the `Notes.jar` file available to TDI. If you don’t have a Notes client installed then you can find this library file under the `jvm\lib\ext` folder of your Lotus Domino installation. Unlike the Local Client option, connection via Local Server does not require you to have a Notes client installed.

Similar to setting up a Local Client connection, you have to edit the path setting in the `ibmditk/ibmdisrv` files. However, instead of including a path reference to the Notes client folder, you instead point these to your Domino server installation, e.g.:

```
set PATH=C:\Program Files\Lotus\Domino;C:\Program Files\IBM\TDI\V6.1.1\...
```

The advantage of using a Local Server connection is that you do not need a Notes Client installed, and if you do have one then you do not have to have it logged into the same server as TDI is integrating.

### 1.1.3 Remote connections (IIOP)

Both the Notes Connector and the Domino Change Detection Connector support connecting via IIOP. In contrast to both of the local options described above, connecting via IIOP requires a different library file: the remote one called `ncso.jar` which is found under the `data\domino\java` directory of your Domino installation.

The same steps are necessary for making this library available to TDI as those outlined above for preparing local connections. However, these two libraries (`Notes.jar` and `ncso.jar`) are mutually exclusive; You cannot have TDI using both at the same time since they contain identically named classes and will conflict with each other.

Since additional libraries are not required when using the IIOP option, no modification to the path setting in the TDI startup batch-files/scripts is necessary. However, you

must ensure that the DIIOP task is running on the target Domino server and that you have set up an Internet document to allow access to this service over IP.

When it comes to configuring your Notes or Domino Change Detection Connectors you now have a couple of options:

- Enter the ip address or hostname of the target Domino server in the **Hostname** parameter, allowing the Connector to perform an HTTP GET in order to retrieve the IOP connection details. As a result you must have the HTTP task running on the Domino server, as well as having an Internet Document defined to provide access to it.
- Alternatively, you can take the contents of the `diiop_ior.txt` text file (located in the `data/domino/html/` folder of your Domino installation) and paste it into the **Hostname** parameter of the TDI Notes Connector<sup>33</sup>. Doing this bypasses the need to have HTTP running on the Domino server, although you will need to update this string value in the parameter setting if DIIOP settings are changed on the Domino server.

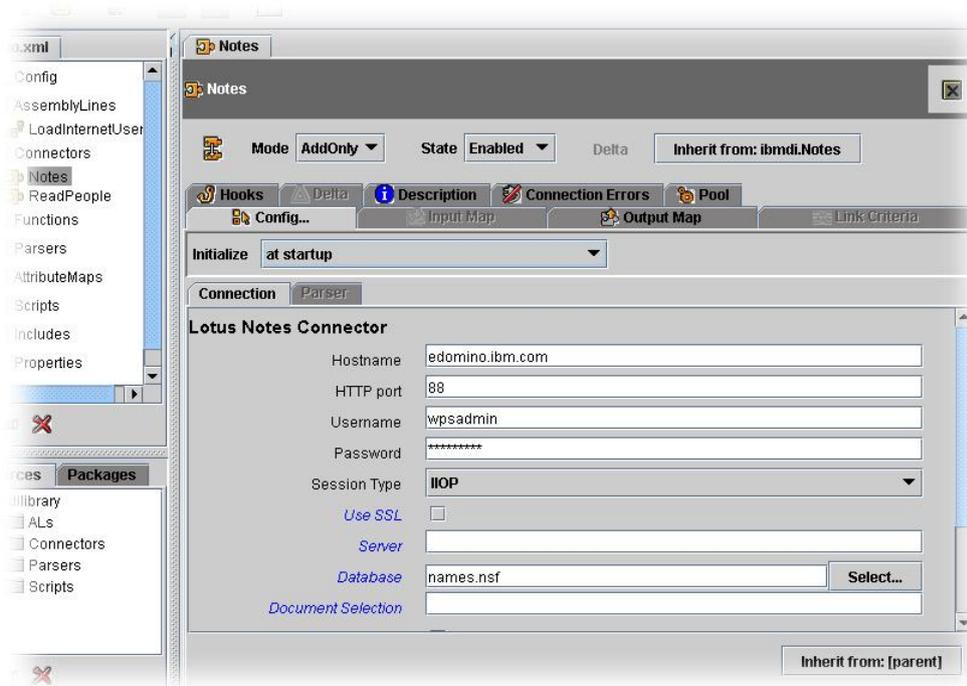
Connecting via DIIOP allows you to use the *Connect + Read next* technique to discover and browse data in the target nsf.

---

<sup>33</sup> This text string should be with "IOR:".

## 4. Working with Notes Documents

In this exercise you will be creating Notes documents directly. For this you will need to set up a Notes Connector using the desired connection type as outlined in the previous section. Here is a shot of one set up for IIOp:



The necessary parameters for connection with IIOp are:

**Hostname** This is either the hostname/ip address of your Domino server, or just copy in the contents of the `diio_pior.txt` file from the server's html directory.

**HTTP port** If you want to dynamically retrieve the `diio_pior.txt` file from the server instead of pasting its contents directly in the Hostname parameter then you must enter the Domino server's configured TCP port here – the one serviced by the HTTP task.

**Username** Internet User name.

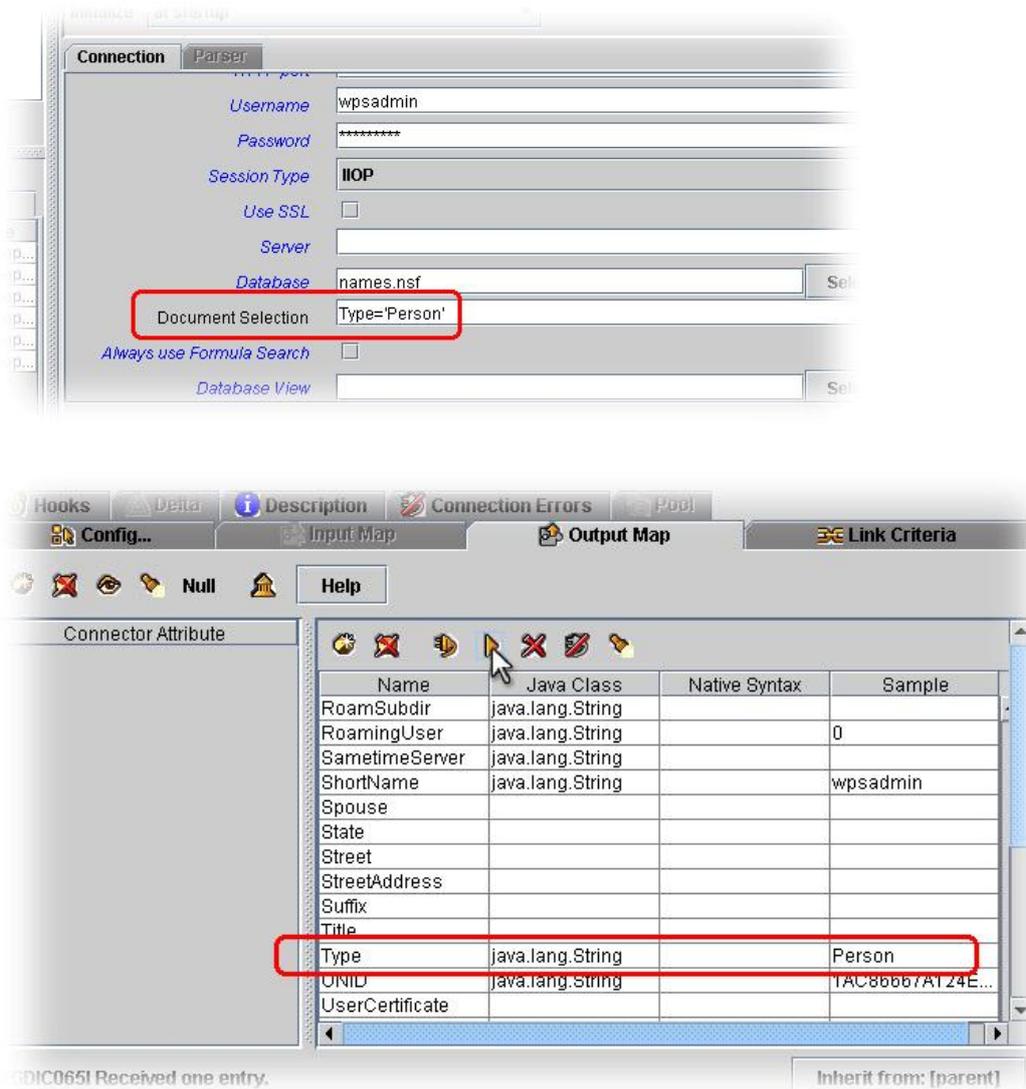
**Password** The Internet User's password.

**Session Type** Connection type: local or remote (IIOp).

**Database** Path to the nsf database that you want to work with.  
**Note:** The path is relative to the Domino server's *Data* sub-directory.

There are a couple of more relevant parameters that apply regardless of the connection type: *Document Selection*, *Database View* and the *Always Use Formula Search* switch. These can be used to determine which documents are returned by the Connector for Iterator mode, just like we used *Search Base* to control which directory entries were returned by the LDAP Connector in the previous section.

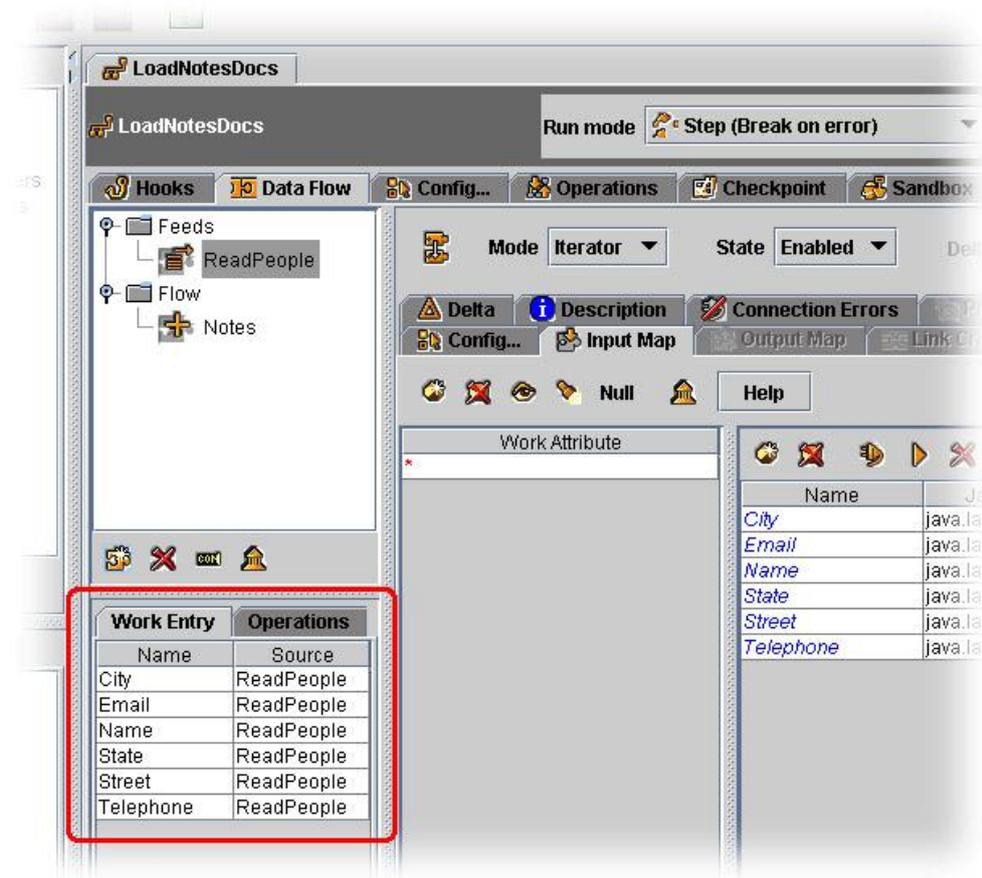
Test your Connector (like you do all Connectors) by bringing up an Attribute Map and using the **Connect + Read next** buttons to validate the configuration and explore your data. If you put the Notes search filter *Type='Person'* in the *Document Selection* parameter, then the Connector will only return documents of type *Person*.



As you saw before, this is a technique that you can use with other TDI Connectors as well (like LDAP and JDBC) giving you a general-purpose approach to searching for information in different types of systems<sup>34</sup>.

<sup>34</sup> You need to understand the syntax of the connected system's search filter in order to do this. The above example used a Notes search string. If you wanted to do this with an LDAP Connector then you

Once you have your Notes Connector working, create a new AssemblyLine. Call it "LoadNotesDocs" and drag in your *ReadPeople* Connector (in Iterator mode) to the Feeds section of your AL, then drag your newly configured *Notes* Connector to the Flow section (in AddOnly mode). The Input Map of your *ReadPeople* library Connector is being inherited into the AL Connector in this AssemblyLine. As a result, the Work Entry window of your AL should be filled with attributes.



The last step is to set up the Output Map of the Notes Connector so that is writing to valid document fields.

would need to enter a valid LDAP Search Filter. In the case of JDBC you must type in the SELECT statement that the Connector will issue when selecting its result set for Iteration.

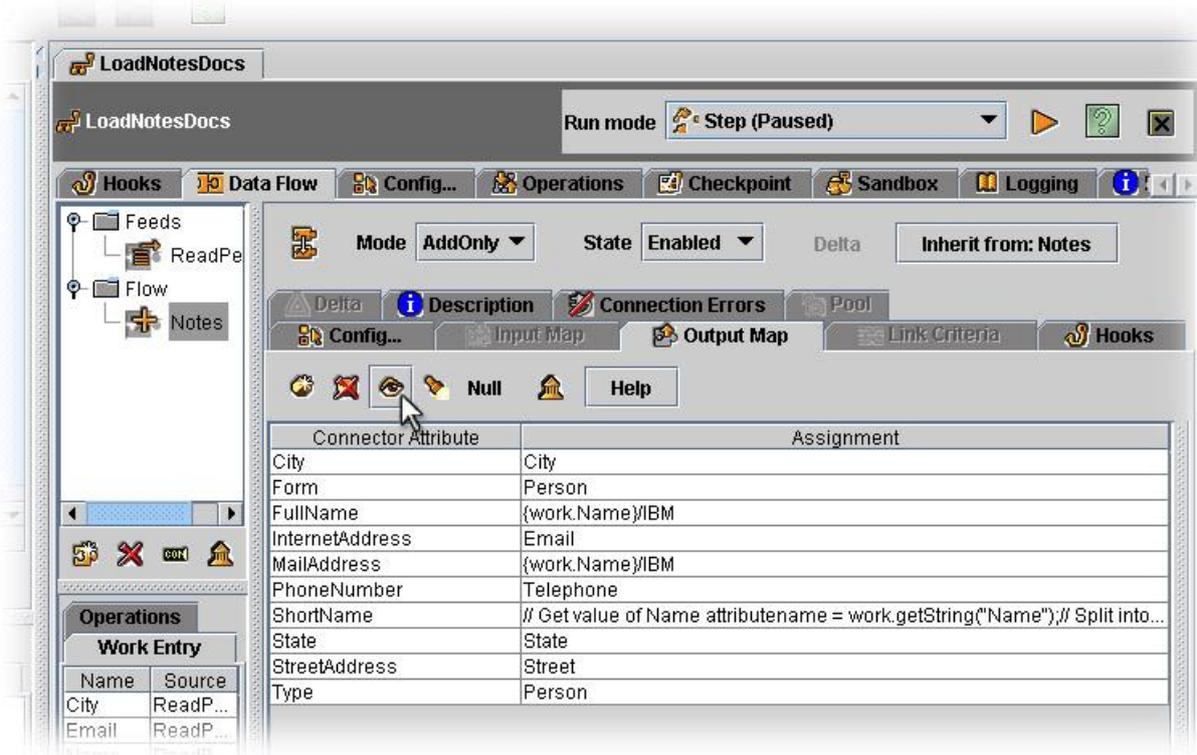
You can either drag in from the schema you discovered in the previous step, or enter these manually using the **Add new attribute...** button at the top of the Output Map. Either way, set up the following Output Map attributes:

<b>Attribute</b>	<b>Type</b>	<b>Map details</b>
<b>City</b>	simple	City
<b>Form</b>	expression <sup>35</sup>	Person
<b>FullName</b>	expression	{work.Name}/IBM
<b>InternetAddress</b>	simple	Email
<b>MailAddress</b>	expression	{work.Name}/IBM
<b>PhoneNumber</b>	simple	Telephone
<b>ShortName</b>	javascript	<pre>// Get value of Name attribute name = work.getString("Name"); // Split into array (" ") parts = system.splitString(name, " "); // Return the first part ret.value = parts[0];</pre>
<b>State</b>	simple	State
<b>StreetAddress</b>	simple	Street
<b>Type</b>	expression	Person

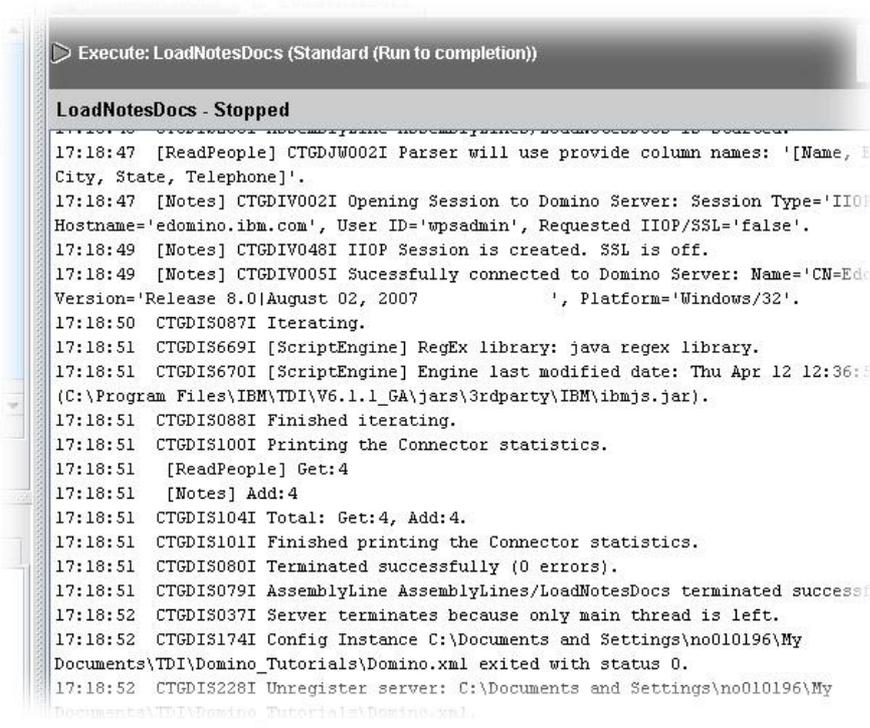
You can use the **Switch...** button at the top of any Attribute Map to switch between showing the maps in List, Detail and Schema view. Note that you can edit both the Attribute Name and the mapping details in List view, although you must select Detail view to change the mapping Type (e.g. Simple, JavaScript or Expression).

---

<sup>35</sup> Expression maps are literal text that can optionally contain substitution tokens. For example, this expression '{work.FullName}/IBM' evaluates to the string value of the FullName attribute in the Work Entry followed by '/IBM'.



Once you have this Output Map in place then you can test your AssemblyLine, which should create four new Person documents in the Server's `names.nsf`.



The statistics show that things went as expected: four *gets* and four *adds*.

You can get this AL to output more information about the data being handled just like you did in earlier exercises by dropping in a Script component. You already have one set up in “LoadInternetUsers” AssemblyLine – let’s reuse it.

First you must take this AL component and create a copy in your library. You do this by bringing up the AL, selecting this Script component (SC) and then pressing the **Copy to Library** button.



Once you have this component in the Library then simply drag it into the LoadNotesDocs AL<sup>36</sup>.

#### 4.1. Working with Domino Objects

A TDI Connector has three basic jobs to do: First it must provide one or more data access operations into the connected system. Which data access operations are actually implemented determines which modes this Connector supports. For example, AddOnly mode requires only the *create* operation implemented, while Delete mode necessitates both *search* and *delete*<sup>37</sup>.

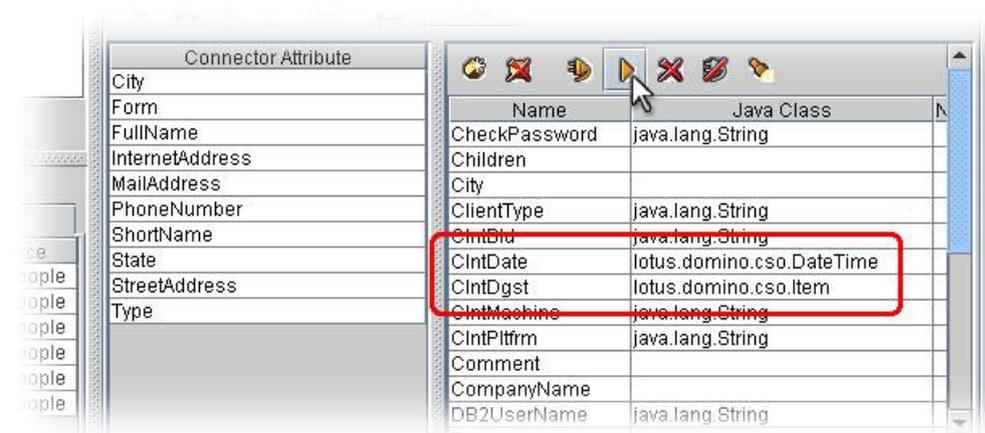
<sup>36</sup> Note that although the SC in the LoadNotesDocs AssemblyLine is inheriting from your new Library component, the original one in LoadInternetUsers is not. The **Copy to Library** button simply cloned the selected component to the appropriate Library folder; it did not change the source component in any way. To set up similar inheritance for it as well, drag the Library component onto the top right-most **Inherit From** box of the original one.

<sup>37</sup> These operations are implemented through a set of Connector Interface functions. For example, AddOnly mode uses only the putEntry() method, while Iterator needs two: first selectEntries() which is used during initialization to build the result set for iteration, and then getNextEntry() which is called on each AL cycle to return the next entry from this set.

Secondly, the Connector wraps these datasource-specific operations in the generic functionality of the AssemblyLine. This lets you work with technically diverse systems in a consistent and predictable way. Finally, each Connector understands how data is stored in its target system and is responsible for converting between these native types and the Java objects used to represent this data in TDI.

To see this in action, go to the Output Map of your Notes Connector. As you step through your data (e.g. **Connect + Read next**) the column entitled *Java Class* shows how this component is marshalling the data. For the most part your Notes Connector is returning standard Java types like *java.lang.String* and *java.lang.Integer*. These have equivalent JavaScript types making them straight-forward and easy to use in script.

However, you will also see that some types returned are Domino-specific objects.



In order to make sense of these objects you will need documentation on the various Domino Java classes, like this one for Domino 7:

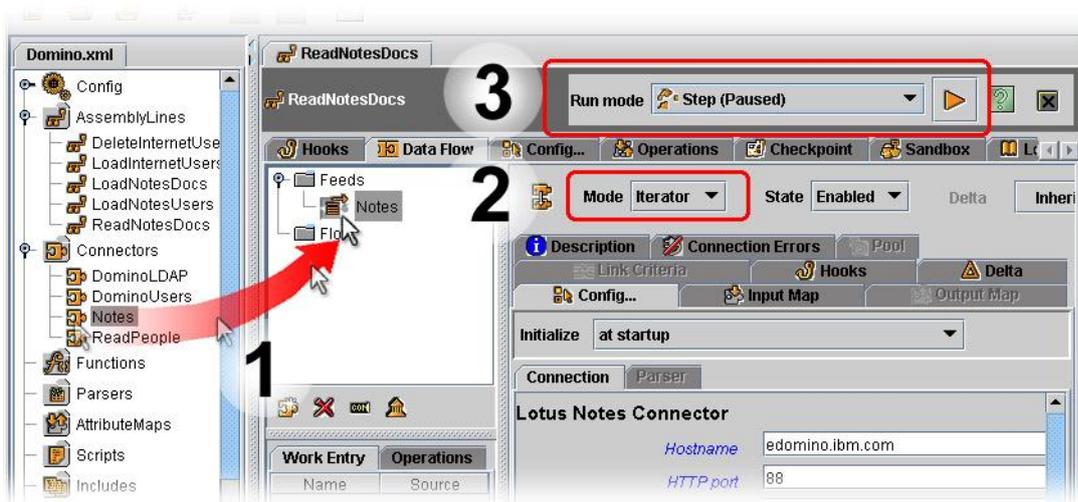
[http://www-12.lotus.com/ldd/doc/uafiles.nsf/docs/DESIGNER70/\\$File/prog3.pdf](http://www-12.lotus.com/ldd/doc/uafiles.nsf/docs/DESIGNER70/$File/prog3.pdf)

The best way to explore these (or any) datasource-specific Java objects from TDI is through the AssemblyLine Stepper/Debugger. To do this, create a new AL and call it "ReadNotesDocs". Now you perform the following steps: 1) drag in the "Notes" Connector from your library, 2) set it to Iterator mode and 3) selecting first *Step (Paused)* from the **Run mode** drop-down, start your AL.

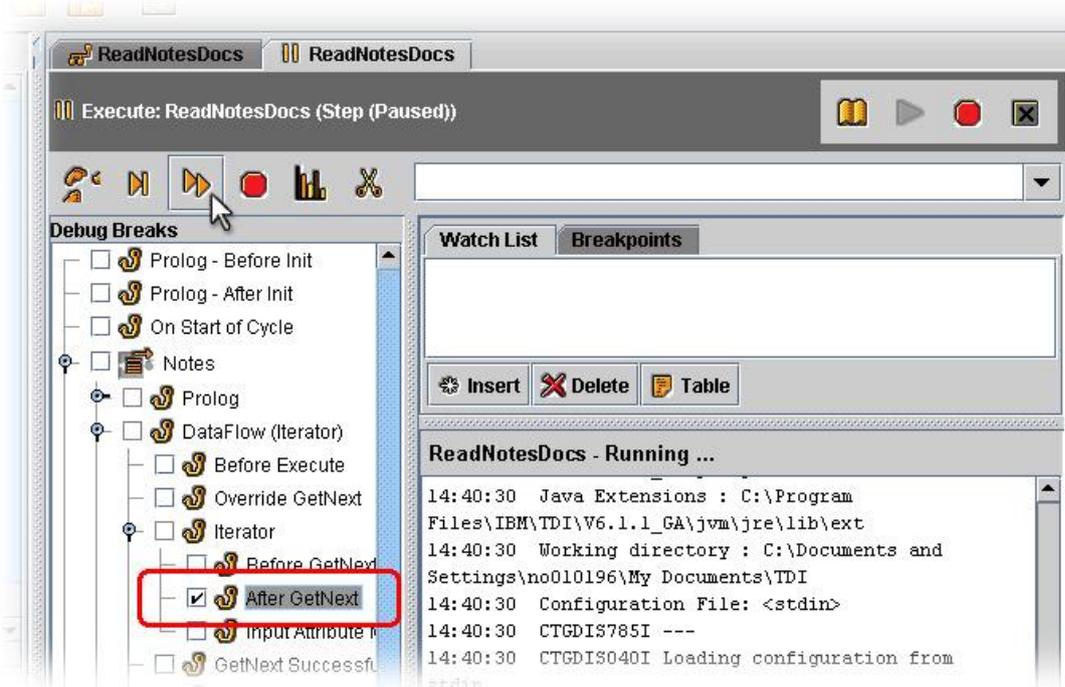
To learn more about how to make your own TDI components, follow these links:

<http://www.tdi-users.org/twiki/bin/view/Integrator/HowTo> (Writing Custom TDI Component section)

[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc\\_6.1.1/referenceguide172.htm?resultof=%22%70%75%74%45%6e%74%72%79%22%20%22%70%75%74%65%6e%74%72%69%22%20](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide172.htm?resultof=%22%70%75%74%45%6e%74%72%79%22%20%22%70%75%74%65%6e%74%72%69%22%20)



This opens up the Debugger window, launches your AssemblyLine and then pauses execution as soon as the TDI Server has created the AL in memory and is ready to initialize it. Set a breakpoint at the **After GetNext** Hook of the “Notes” Connector and then press the Continue button.

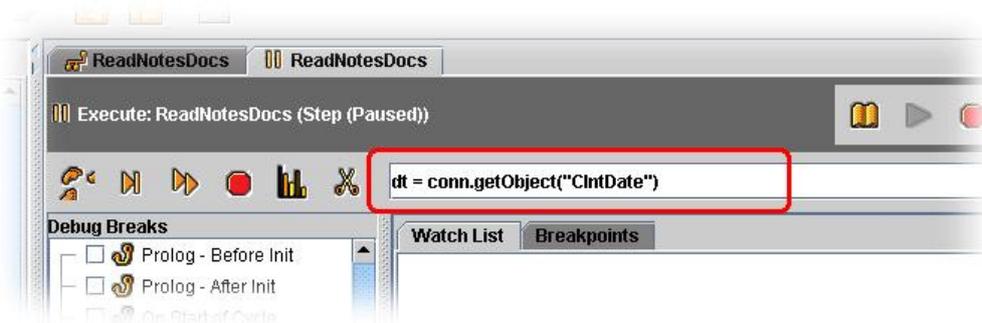


This causes the AL to first initialize and then begin cycling. Just after the first Notes Document has been read into the Connector’s cache, the AssemblyLine pauses at your breakpoint and we can use a snippet or two of JavaScript to access the data in this cache<sup>38</sup>.

<sup>38</sup> Remember to use the **Document Search** field to limit documents return to Type='Person'. Other document types may not contain the attributes used in this exercise.

The local cache of a Connector is available as the pre-registered script variable called *conn*. This is an Entry object, just like the Work Entry, and you can type in the following code in the JavaScript Command Line and then press Enter<sup>39</sup>.

```
dt = conn.getObject("ClntDate")
```



When you hit the Entry key, this snippet is executed in the running TDI Server where it creates a variable called “dt” that references the object used to store the value of the “ClntDate” attribute. The result of the JavaScript statement is displayed using blue text in the Log Output window.



The screenshot above shows that the result of the expression is a formatted date value, which is how the underlying *lotus.domino.cso.DateTime* object represents itself as a string. If you enter this code into the JavaScript Command Line then you can discover the classname of this object:

```
dt.getClass()
```

Now you should see `dt.getClass() -> class domino.lotus.cso.DateTime` in the Log Output. This approach can be used with any type of attribute value since all Java objects implement the `getClass()` function. Once you know the class name you can look it up in your reference materials. The *lotus.domino.cso.DateTime* class is described on page 379 of the PDF linked earlier in this section. Here you will find information about the various fields and accessor methods (`get/set`) available for this object. For example, calling the `toJavaDate()` method returns the value of this object as a standard `java.util.Date` object, making date comparisons and

<sup>39</sup> You have previously been using the `getString()` method which returns a string representation of the attribute value, regardless of the type of value. If you want the actual Java object used to hold this value then you use the `getObject()` function instead.

computations simple to script.

```
javaDT = dt.toJavaDate()
```

Sometimes you also need to know how to create a datasource-specific object in situations where the Connector does not do this for you automatically during write operations. The Domino Class reference PDF explains on page 380 that in order to create a *lotus.domino.cso.DateTime* object you need to use the `createDateTime()` method of the Domino Session (*lotus.domino.Session*) object.

Fortunately, your Notes Connector already has a valid Domino Session set up that you can request using following script snippet, for example in an Attribute Map:

```
domSession = Notes.getConnector().getDominoSession();
```

Now you have a new variable called “domSession” that references this Session object<sup>40</sup>, described on page 955 in the same PDF. This class offers three versions of the `createDateTime()` method as detailed on page 971: one which accepts a formatted date string as an argument, one that accepts a *java.util.Date* and finally one based on a *java.util.Calendar*.

[Session](#)

### Syntax

```
public DateTime createDateTime(String date)
    throws NotesException

public DateTime createDateTime(java.util.Date date)
    throws NotesException

public DateTime createDateTime(java.util.Calendar date)
    throws NotesException
```

**Note:** This signature is new with Release 6.

### Parameters

String date

The date and time you want the object to represent. See [DateTime](#) for formats. An invalid date-time or empty string results in an “Invalid date” exception.

java.util.Date date

Chapter 2. Java Classes A-Z 971

Although you don’t have to be a Java developer, learning how to navigate and read Java class documentation is an advantage when doing more than just simple integration work. TDI accelerates the initial creation of integration solutions through its rich set of components and automated behaviors. By simultaneously giving you direct access to the underlying libraries and objects native to the connected system, you are empowered to take your solution beyond TDI’s built-in functionality. The TDI

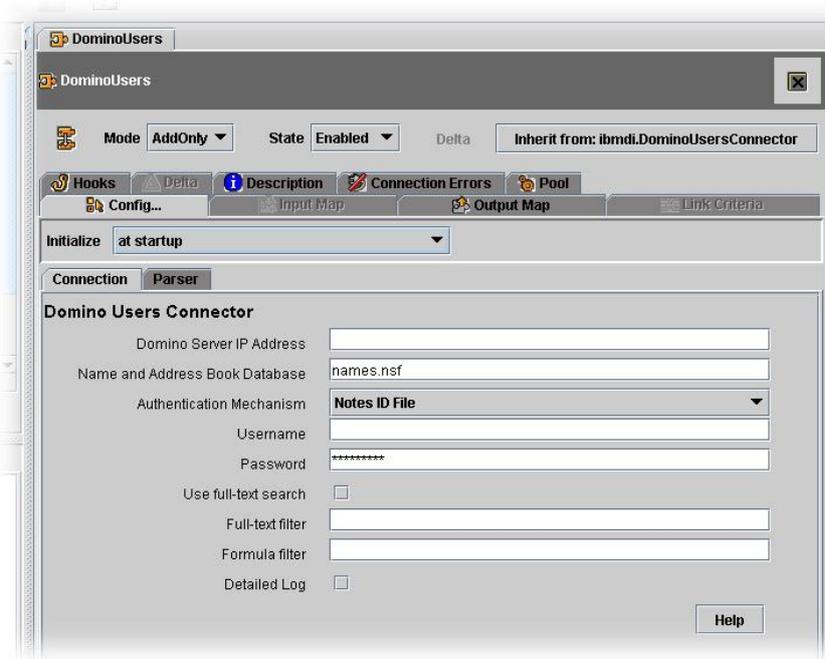
<sup>40</sup> For a complete list of the functions and features available in the Notes Connector, select **Help ► Low Level API** in the main menu and then locate the DominoConnector class in the TDI JavaDocs. Hint: to find out which Connector class this is, just type `Notes.getConnector().getClass()` in the JavaScript Command Line.

AL Stepper/Debugger provides a visual framework for interactively exploring your solution and its connected systems.

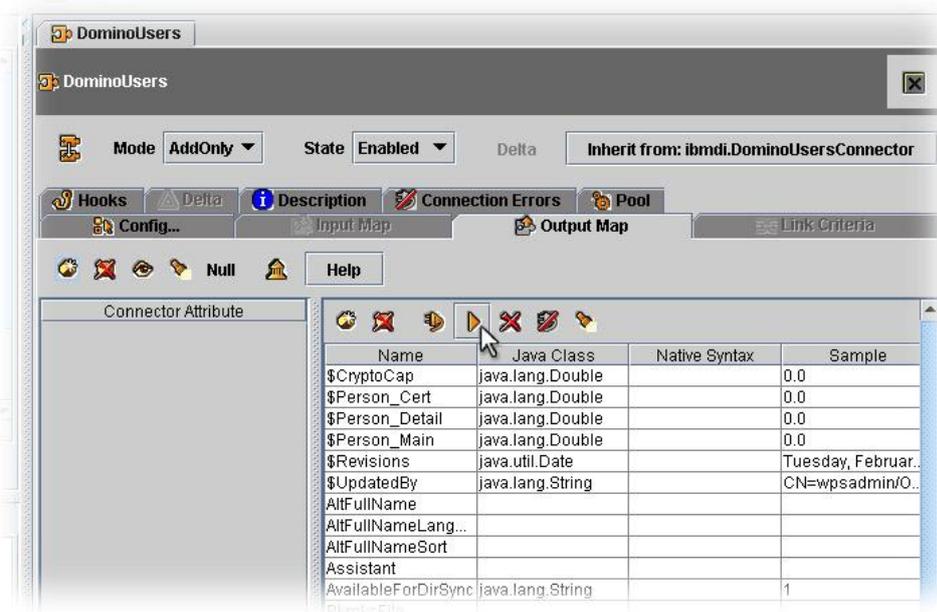
You'll be using this technique in the next section to provision Notes User accounts using the Domino Users Connector.

## 5. Provisioning Notes User Accounts

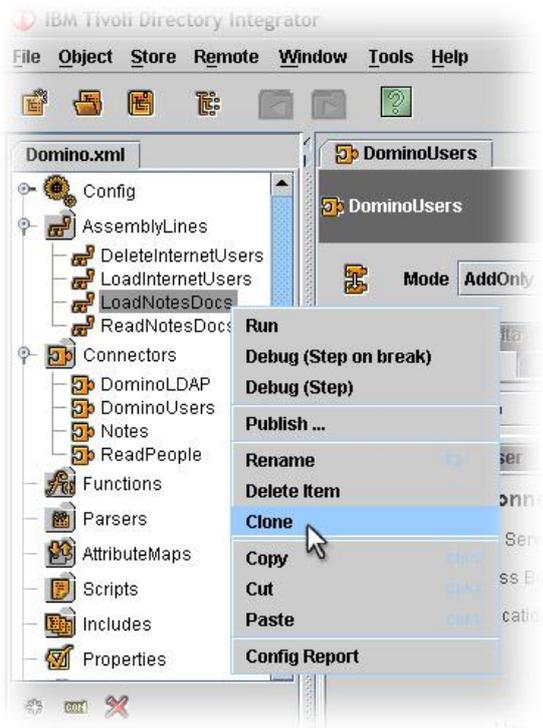
Create a Domino Users Connector (aka *DUC*), called it “DominoUsers” and set it up using one of the connection methods described in section 3. Here is an example of a DUC set up for the Local Client connection option (called “Notes ID File” onscreen).



The only parameters you need to fill out for a Local Client connection are the NAB database filename and the ID File password. You can now bring up the Output Map (or Input Map, depending on the mode of the Connector) and use the **Connect + Read next...** buttons to view existing user entries



This time instead of creating a new AssemblyLine from scratch, right-click on the “LoadNotesDocs” AL and select **Clone**. Call this cloned AL “LoadNotesUsers”.



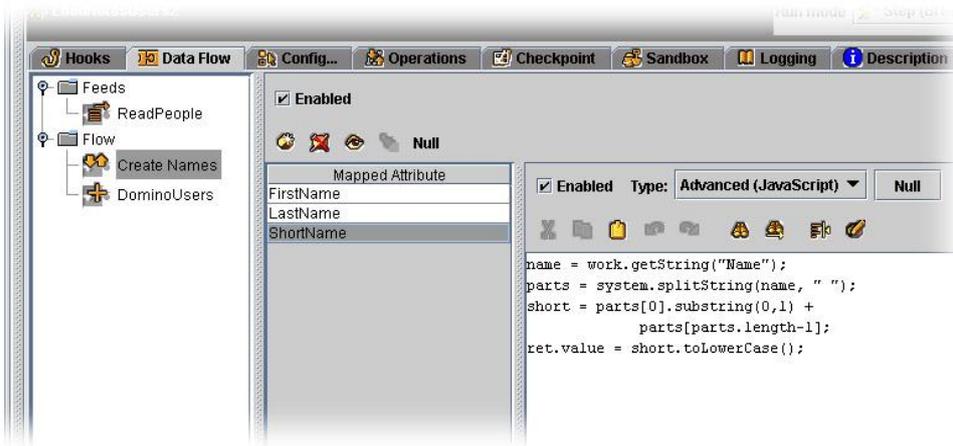
TDI has now created a new AL identical to the one you cloned. Open this new AL and delete the Connector called “Notes” and drag your new “DominoUsers” Connector in its place in the Flow section (also in AddOnly mode).

Now add an AttMap component to your AL, call it “Create Names” and place it just before the DominoUsers Connector. Here add a new attribute called “ShortName” with this JavaScript mapping code:

```
name = work.getString("Name");
parts = system.splitString(name, " ");
short = parts[0].substring(0,1) +
        parts[parts.length-1];
ret.value = short.toLowerCase();
```

Then add two more attribute: FirstName and LastName, both JavaScript maps.

<b>Attribute</b>	<b>Map details</b>
<b>FirstName</b>	<pre>name = work.getString("Name"); parts = system.splitString(name, " "); ret.value = parts[0];</pre>
<b>LastName</b>	<pre>name = work.getString("Name"); parts = system.splitString(name, " "); ret.value = parts[parts.length-1];</pre>



This is done here so that we can use these elsewhere in mapping operations. Note that we cannot count on the attributes in an Attribute Map being evaluated in any specific order. This means that we can't have one attribute that depends on the result of another in the same Attribute Map. That's why we create the *name* attributes in a separate AttMap component so we can use them later in other maps.

### 5.1. Special Attributes for User Registration

As you've seen before, a vital step in building an AL is to set up the mappings for your output Connector so that new entries are successfully created. In our case, we want new user accounts correctly provisioned in Notes, which means that in addition to creating new Person documents in the NAB, you also have to pass some instructions to the Domino Administration Process (affectionately called '*AdminP*') on how to set up this account.

Fortunately you won't need to script all this since the Notes Connector lets you pass commands to AdminP via specially named attributes, all starting with "REG\_". The list of available registration attributes is found in the online docs here: [http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.IBMDI.doc\\_6.1.1/welcome.htm](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.IBMDI.doc_6.1.1/welcome.htm), plus these three additions for TDI 6.1.1:

<b>REG_MailTemplateFile</b>	String	<i>optional</i>	The filename/path to a Notes template database which the Connector will use to create the user mail file.  If this Attribute does not exist then the default mail template is used.
<b>REG_MailTemplateServer</b>	String	<i>optional</i>	The IP address of the Domino server machine on which the mail template database (e.g. specified by "REG_MailTemplateFile") resides.  If this Attribute does not exist the local Domino server machine is used.
<b>REG_MailDbInherit</b>	Boolean/ String	<i>optional</i>	"true" - the user mail database to be created will inherit any changes to the mail template database design  "false" - the user mail database to be created will <b>not</b> inherit any changes to the mail template database design  If this Attribute is missing, a default value of "false" will be assumed.

We will proceed with this example exercise by leveraging a subset of these special attributes. Use the following as a guide to setting up your own "Registration Settings" AttMap (these are all maps of type *Expression*).

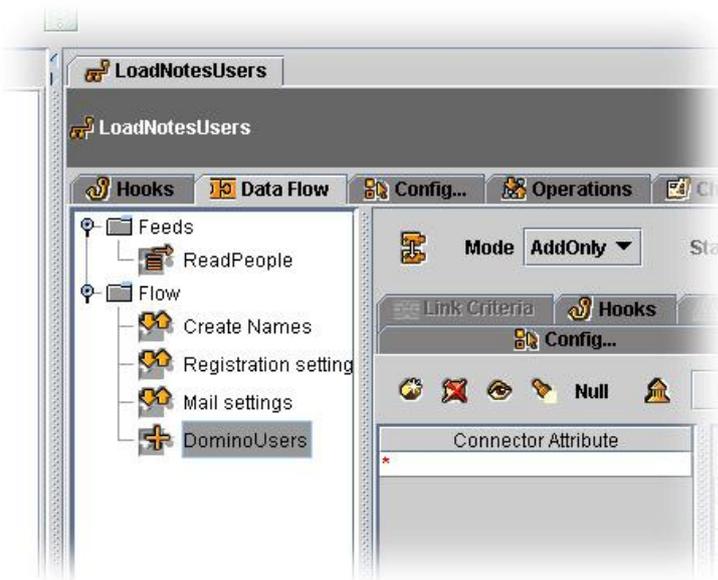
<u>Attribute</u>	<u>Map details</u>
<b>REG_CertPassword</b>	secret123
<b>REG_CertifierIdFile</b>	C:\Program Files\Lotus\Domino\data\cert.id
<b>REG_IdFile</b>	{work.ShortName}.id
<b>REG_Perform</b>	true
<b>REG_RegistrationServer</b>	CN=Edomino/O=IBM
<b>REG_Server</b>	CN=Edomino/O=IBM
<b>REG_StoreIdInAddressBook</b>	true
<b>REG_UserPw</b>	newpw123

These represent the bare minimum of settings that have to be passed to Domino in order to get our users created.

Just after the above AttMap create another one and call it "Mail Settings". This will contain commands to control the mail configuration for our users. Add the following maps (map type = *Expression*):

<b><u>Attribute</u></b>	<b><u>Map details</u></b>
<b>MailAddress</b>	{work.Name}/IBM
<b>MailDomain</b>	IBM
<b>MailFile</b>	c:/program files/lotus/domino/mail/_new_{work.ShortName}.nsf
<b>MailServer</b>	Edomino/IBM
<b>MailSystem<sup>41</sup></b>	1
<b>InternetAddress</b>	{work.ShortName}@mymail.com

You will also need to include all these attributes in the Output Map of your DominoUsers Connector. This is easily done by using the wildcard map (\*) since the DUC will ignore attributes that do not conform to the schema for creation of new Notes users.

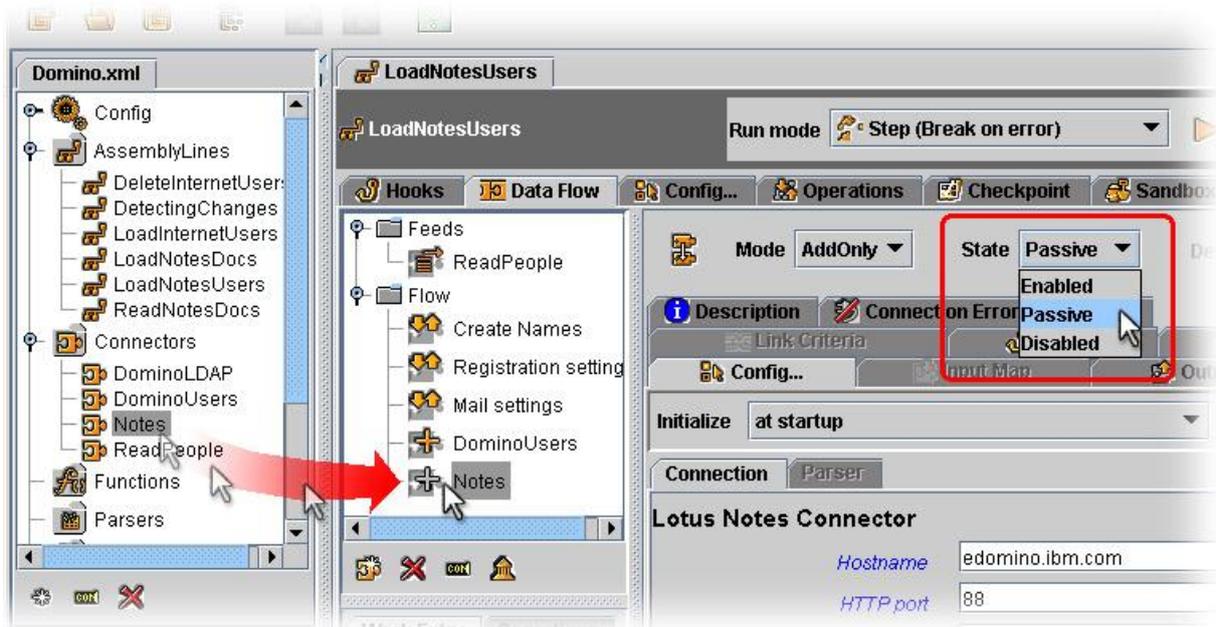


In addition to the wildcard, add a new Attribute to the Output Map called "myREG\_MailQuotaSizeLimit" with the Expression-mapped value of 250000 (this value is in bytes). This is an example of a user-defined registration attribute which will

<sup>41</sup> MailSystem = 1 means that Notes is the mail system for this user.

be handled by script code executed *after* the user has been registered<sup>42</sup>. You must add this code now.

Instead of setting up a Domino Session in script, you will exploit a feature in TDI called *Passive Connectors*. So drag in the “Notes” Connector you created in your library, configure it for Local Client (like your DUC) and then set it to Passive State.



The *State* of a Connector determines its level of participation in the AssemblyLine lifecycle. By default, all Connectors are in *Enabled* State, causing them to be initialized when the AL starts, executed during AssemblyLine cycling and closed when processing is completed. Conversely, setting a component to *Disabled* State means that it will not be initialized/closed or used in any way during AL processing.

*Passive* State provides a very handy middle ground. A Connector in Passive State will be initialized and shutdown when the AL starts and stops, but it will not be used by the built-in AL logic to do any work. However, this won't stop you from using it via JavaScript. Since Passive State Connectors do not participate in AL cycles, it doesn't matter where you put this Connector in the AL component list. I dropped mine at the end of the AL, as shown in the previous screenshot.

Once the Notes Connector is in place, configured for Local Client and working – remember to use the **Connect** button in the Input/Output Map to ensure correct configuration – you can now put some code in the Default Success Hook of your Domino Users Connector. This Hooks is only executed if the add operation was successful.

Here is the self-documenting code to drop into the Default Success Hook<sup>43</sup>:

---

<sup>42</sup> This technique was first shown to me by Jason Williams of the pre-eminent TDI L2 Support team.

```
// You might need to have TDI wait a few seconds for the mail
// file to be created before continuing. I did not need any
// delay for my test scenario, but this may differ for you
// depending on your Domino Server configuration and load.
//
// system.sleep(3); // sleep 3 seconds

task.logmsg("=====");
task.logmsg("Begin custom configuration of mail db for " +
            work.getString("Name"));

// Unlike the DUC, the Notes Connectors provides a handy *get*
// method for retrieving the open Domino Session. Note that I
// am referencing the Notes Connector using the name I gave it
// in the AL (namely "Notes"). Then I grab the Connector
// Interface with the getConnector() call. This returns
// the actual DUC (which is the interface for the AL
// Connector named "DominoUsers").
// Finally, I retrieve the open Session object from the DUC.
//
var session = Notes.getConnector().getDominoSession();

// Now retrieve some Attribute values. I am grabbing these
// from the Connector Interface's cache object, the
// conn Entry.
//
var MailFile = conn.getString("MailFile");
var MAddress = conn.getString("MailAddress");
var UserName = work.getString("Name"); // This one is not
// mapped to conn
var Server = conn.getString("REG_RegistrationServer");
var NewMailQuota = conn.getString("myREG_MailQuotaSizeLimit");

// Now open the newly created mail db
task.logmsg("Opening the db...");
var db = session.getDatabase(Server,MailFile,false);
task.logmsg("Got db: " + db);
```

---

<sup>43</sup> . Note that if you would rather put this code in a separate Script component *following* the Domino Users Connector instead of using the Default Success Hook then you must change the thisConnector reference to the name of your DUC:

```
var DUC = thisConnector.getConnector();
```

All AL components are registered as script variables, as well as being available upon request via the task.getConnector() method, which despite its name returns whatever type of AL component you specify.

```
// Set the title
task.logmsg("Setting the title to " + UserName);
db.setTitle(UserName);

// Grab the access control list associated with this db
var acl = db.getACL();

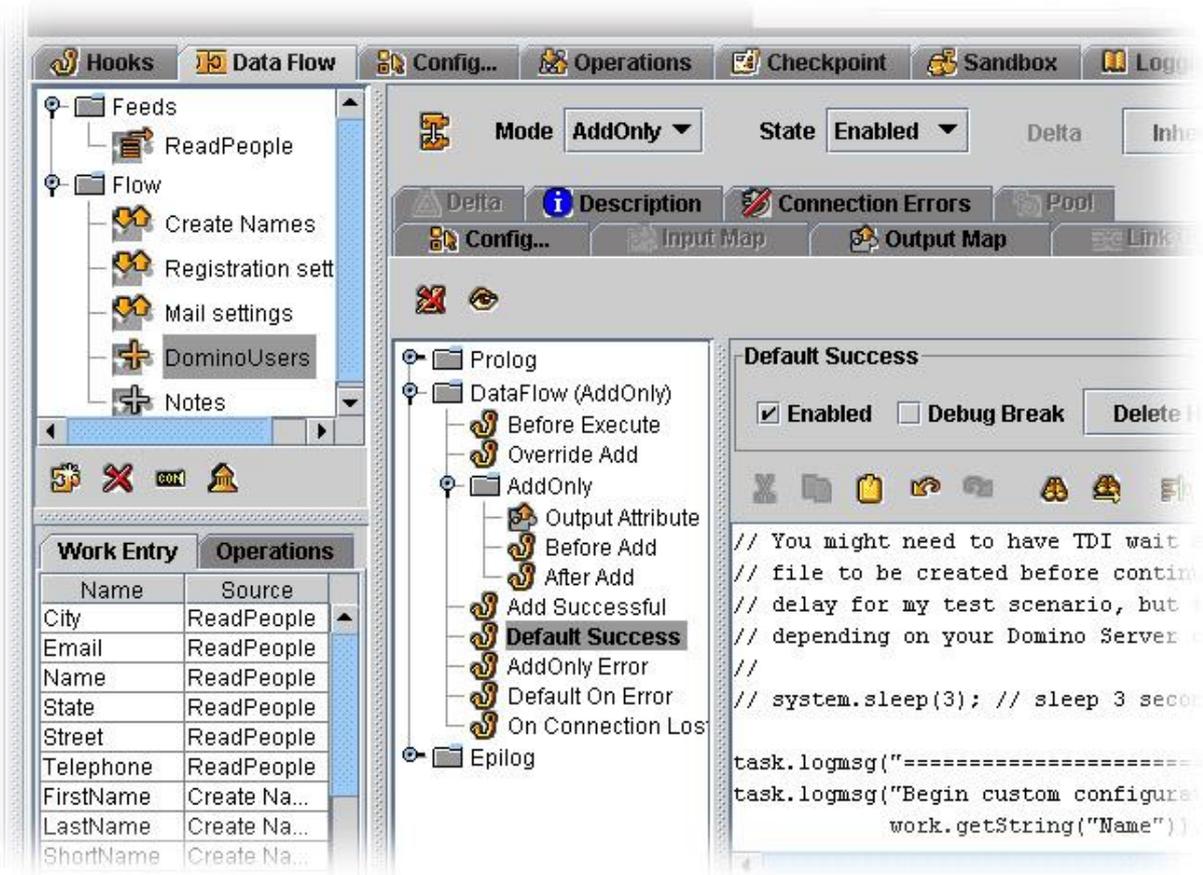
// create a new ACL Entry
var ACLEntry = acl.createACLEntry(MAddress,
lotus.domino.ACL.LEVEL_EDITOR);

// set user type equal to Person
ACLEntry.setUserType(lotus.domino.ACLEntry.TYPE_PERSON);
ACLEntry.setCanCreatePersonalFolder(true);
ACLEntry.setCanCreateSharedFolder(true);

// Save the access control list
task.logmsg("Changing the ACLs...");
acl.save();

// Here you set the size quota based on the Attribute value
// retrieved earlier in this script.
//
task.logmsg("Setting the mail quota size to " +
            NewMailQuota +
            " byte(s)");
if (system.isValidInt(NewMailQuota))
    db.setSizeQuota(system.toInt(NewMailQuota))
else
    task.logmsg("** Invalid mail quota setting: " +
                NewMailQuota);

task.logmsg("Finished for " + work.getString("Name"));
```



Now you should be able to run your AL and add new Notes users to the Domino server, set the mail db title and ACLs, and even modify the quota.

LoadNotesUsers - Stopped

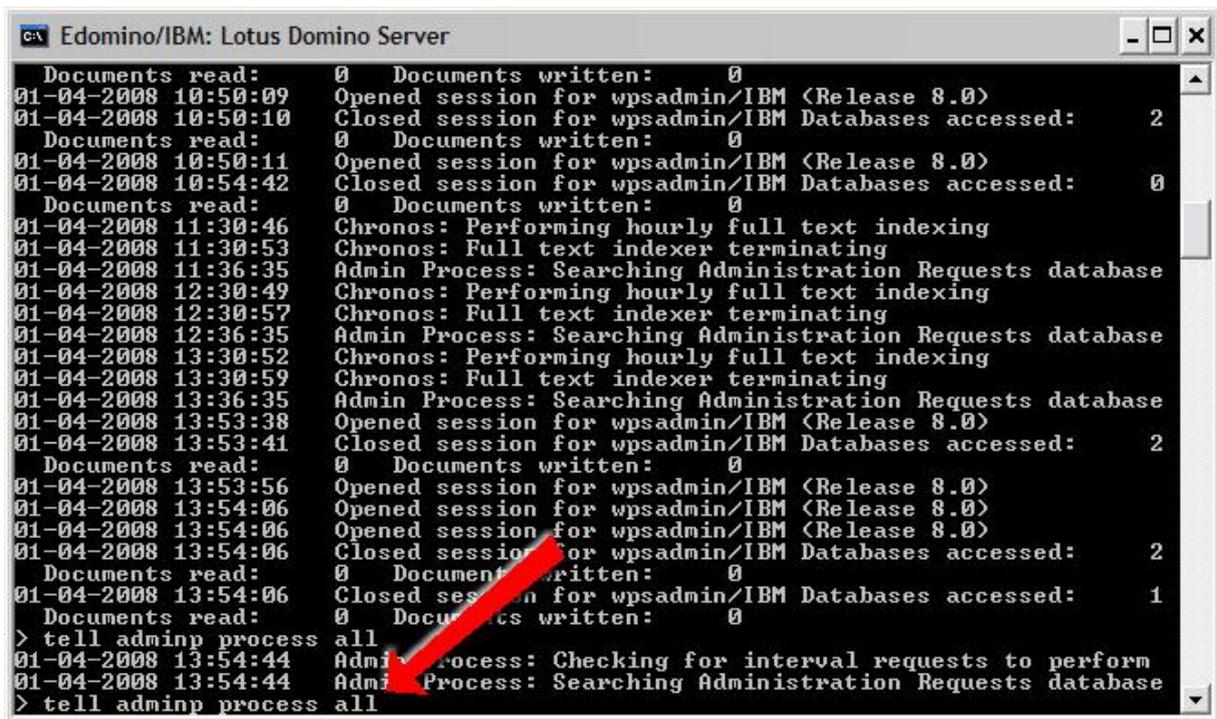
```
Files\IBM\TDI\V6.1.1_GA\jars\3rdparty\IBM\ibmjs.jar).
15:20:30 =====
15:20:30 Begin custom configuration of mail db for Alysson A. Angst
15:20:30 Opening the db...
15:20:30 Got db: mail\_new_aangst.nsf
15:20:30 Setting the title to Alysson A. Angst
15:20:30 Changing the ACLs...
15:20:30 Setting the mail quota size to 250000 byte(s)
15:20:30 Finished for Alysson A. Angst
15:20:47 =====
15:20:47 Begin custom configuration of mail db for Christopher Crumpet
15:20:47 Opening the db...
15:20:47 Got db: mail\_new_ccrumpet.nsf
15:20:47 Setting the title to Christopher Crumpet
15:20:47 Changing the ACLs...
15:20:47 Setting the mail quota size to 250000 byte(s)
15:20:47 Finished for Christopher Crumpet
15:20:59 =====
15:20:59 Begin custom configuration of mail db for Eduardo Echnidea
15:20:59 Opening the db...
15:20:59 Got db: mail\_new_eechnidea.nsf
15:20:59 Setting the title to Eduardo Echnidea
15:20:59 Changing the ACLs...
15:20:59 Setting the mail quota size to 250000 byte(s)
15:20:59 Finished for Eduardo Echnidea
15:21:11 =====
15:21:11 Begin custom configuration of mail db for Theodor Tightly
15:21:11 Opening the db...
15:21:11 Got db: mail\_new_ttightly.nsf
```

Note that Passive Connectors can be used for other purposes as well, for example to write formatted logs, send emails or pretty much anything else a Connector can do. Function components also support Passive State, adding to the wealth of custom functionality at your fingertips.

As with Notes Connector exercise, this example uses AddOnly mode for our output Connector and so will only create new users. If you want to modify existing users as well then you need to switch your DUC to *Update* mode and add Link Criteria. Don't forget *Delete* mode when you want to remove Notes users.

One thing to remember about deleting Notes user accounts: Remember that the TDI Connector is not accessing the data directly. Instead, it is putting requests into the Domino Server Administration Process queue, and you won't see results until AdminP gets around to dealing with these requests – which it does on a periodic schedule. You can bypass this schedule and get AdminP to service requests immediately by entering the following command in the Domino Server console:

```
tell adminp process all
```



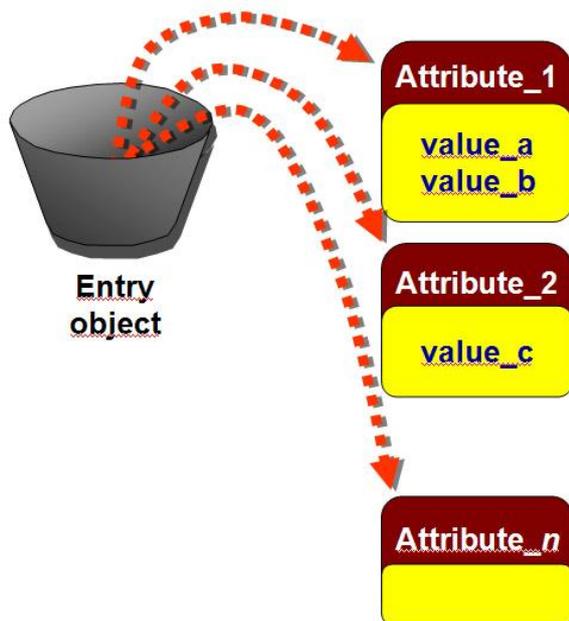
```
c:\ Edomino/IBM: Lotus Domino Server
Documents read: 0 Documents written: 0
01-04-2008 10:50:09 Opened session for wpsadmin/IBM <Release 8.0>
01-04-2008 10:50:10 Closed session for wpsadmin/IBM Databases accessed: 2
Documents read: 0 Documents written: 0
01-04-2008 10:50:11 Opened session for wpsadmin/IBM <Release 8.0>
01-04-2008 10:54:42 Closed session for wpsadmin/IBM Databases accessed: 0
Documents read: 0 Documents written: 0
01-04-2008 11:30:46 Chronos: Performing hourly full text indexing
01-04-2008 11:30:53 Chronos: Full text indexer terminating
01-04-2008 11:36:35 Admin Process: Searching Administration Requests database
01-04-2008 12:30:49 Chronos: Performing hourly full text indexing
01-04-2008 12:30:57 Chronos: Full text indexer terminating
01-04-2008 12:36:35 Admin Process: Searching Administration Requests database
01-04-2008 13:30:52 Chronos: Performing hourly full text indexing
01-04-2008 13:30:59 Chronos: Full text indexer terminating
01-04-2008 13:36:35 Admin Process: Searching Administration Requests database
01-04-2008 13:53:38 Opened session for wpsadmin/IBM <Release 8.0>
01-04-2008 13:53:41 Closed session for wpsadmin/IBM Databases accessed: 2
Documents read: 0 Documents written: 0
01-04-2008 13:53:56 Opened session for wpsadmin/IBM <Release 8.0>
01-04-2008 13:54:06 Opened session for wpsadmin/IBM <Release 8.0>
01-04-2008 13:54:06 Opened session for wpsadmin/IBM <Release 8.0>
01-04-2008 13:54:06 Closed session for wpsadmin/IBM Databases accessed: 2
Documents read: 0 Documents written: 0
01-04-2008 13:54:06 Closed session for wpsadmin/IBM Databases accessed: 1
Documents read: 0 Documents written: 0
> tell adminp process all
01-04-2008 13:54:44 Admin Process: Checking for interval requests to perform
01-04-2008 13:54:44 Admin Process: Searching Administration Requests database
> tell adminp process all
```

Note also that some operations (like deleting mail files) requires approval first, which can easily be done from the Domino Administrator application.

## 6. Detecting Changes in NSF Databases

This last section talks about detecting changes in Domino databases. However, before I dig into the specifics of the Domino Change Detection Connector I need to cover a few basic concepts.

I'll start by going back to the basic TDI Entry > Attribute > value data model:



Whenever a Connector reads data, it returns this information in an Entry object loaded with Attributes that hold the data values themselves. Furthermore, the Entry returned is considered the *current state* of the data – i.e. it is a complete representation of how the data is now and is not based on comparison with any previous value.

However, when we use TDI *delta detection* features to discover changes in data sources, the Entries returned represent the *delta* (or *difference*) between current values and those before the change occurred. This delta information is carried by the Entry object itself in the form of a *delta operation code* which will have a value of *add*, *modify* or *delete*<sup>44</sup>. As you can imagine, this delta information is vital for applying changes to one or more target systems.

Now imagine that you have an AL that is synchronizing changes from a Domino db to some other system. At some point in time, change #42 is propagated to the target system. But then TDI is stopped for some reason, for example to do scheduled maintenance or apply an upgrade. When your synchronization AssemblyLine starts up again, you want it to continue where it left off: starting with change #43. For TDI to

---

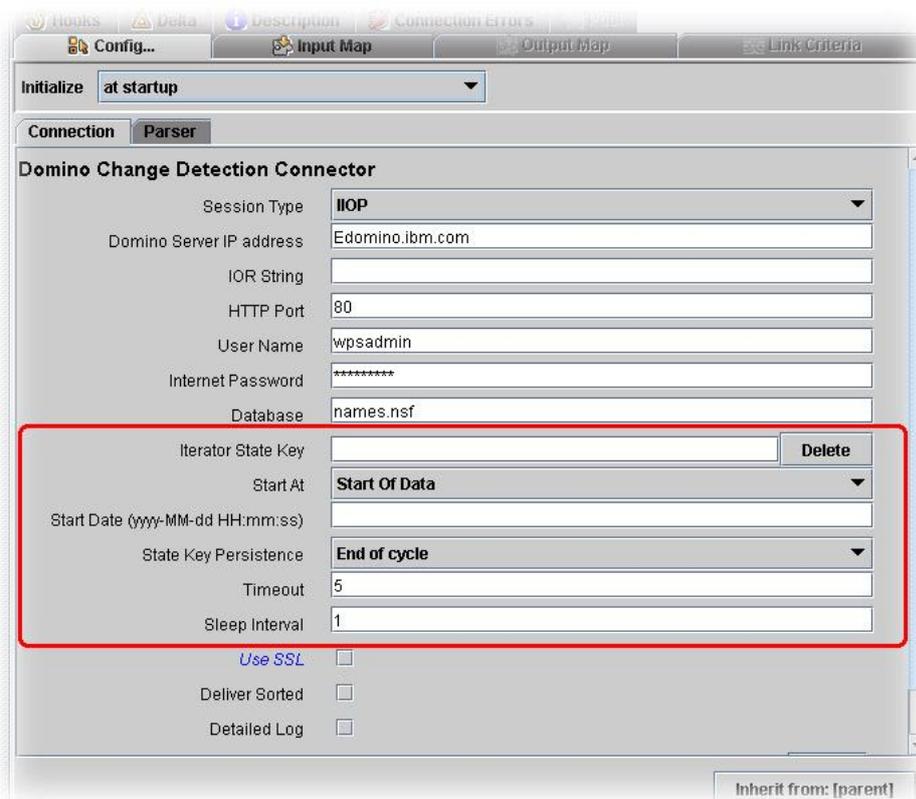
<sup>44</sup> Note that all Entry objects carry a delta operation code. However, unless you are using change detection features then this code is set to "generic", which means *no delta information*.

be able to do this, it needs to maintain *state* information about the last change processed. In TDI terms this is called *Iterator State* and it is kept in an optional database called the *TDI System Store*.

By default the System Store uses the bundled Derby database system, although you can configure TDI to use any JDBC-compliant RDBMS like DB2, Oracle or SQLServer. Regardless of which underlying data store you choose to use, the System Store features in TDI work the same. This means that you can design and implement your solutions using one option, and then switch to a different RDBMS at any time later without altering your AssemblyLines<sup>45</sup>.

That's enough theory for now. Let's get cracking with this last example.

As before, create a new AssemblyLine. Call this one "DetectingChanges" and add a Domino Change Detection Connector which you can call "DominoChanges". Configure this component as you would any other Domino-related Connector. However, since this is a Change Detection Connector (or CDC for short) there are additional parameters for dealing with Iterator State that you also need to set up.



<sup>45</sup> If you will be using the bundled Derby database as your System Store, then please refer to this page which details how to set Derby up in *networked mode*:  
<http://www.tdi-users.org/twiki/bin/view/Integrator/SystemStore>

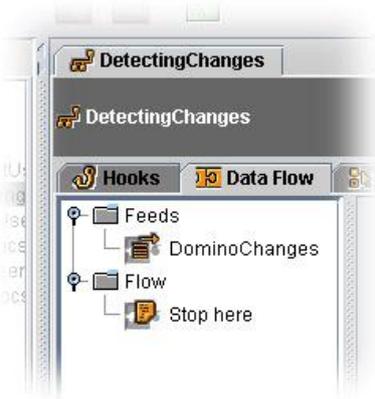
As noted above, Iterator State is persisted automatically by TDI Change Detection Connectors so that work can continue from the last change processed whenever your AL is restarted after a stop.

In addition to the Iterator State settings, there are also a couple of other important parameters for specifying how often TDI is to poll the Domino db looking for new changes, as well as which date/time to start looking for changes after. All in all this adds up to six Change Detection specific parameters:

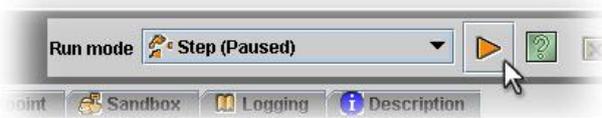
<b>Iterator State Key</b>	Here you enter a string value used as the key under which Iterator State information is stored in the System Store. Note that this value should be unique for this Connector so that no other components will be using the same key value.
<b>Start At</b>	<p>This drop-down lets you specify where you want the Connector to start looking for changes.</p> <p>NOTE: this parameter is only used if no Iterator State Key value is found in the System Store. If there is already Iterator State information for this Connector then the <b>Start At</b> parameter is ignored – so for example, you use this parameter the <i>first time</i> you use the component to detect changes.</p>
<b>Start Date</b>	The Date/Time from which the Connector should begin searching for changes. This parameter is only used if <b>Start At</b> is set to <i>Specific date</i> and when no Iterator State information is found (as noted above for <b>Start At</b> ).
<b>State Key Persistence</b>	<p>This drop-down lets you specify when during an AL cycle you want Iterator State saved to the System Store.</p> <p>The recommended setting is <i>End of cycle</i>, at which time all output Connectors have written the current change to their target systems.</p>
<b>Timeout</b>	The period of time in seconds that the Domino Change Detection Connector should wait for new changes to be detected. If this parameter is set to zero (0) then the Connector will wait indefinitely.
<b>Sleep interval</b>	<p>The time in seconds that the Connector should <i>sleep</i> between polling the Domino db for changes.</p> <p>Note that the Sleep interval should be set lower than the Timeout, except when Timeout is set to zero (0).</p>

Set the **Database** parameter to “names.nsf”, the **Iterator State Key** to “lastChangeInNamesNSF” and **Start At** to *End of Data*. Make sure the **Timeout** is set to zero and **Sleep interval** is 5. Leave the other parameters as they are. In the Input Map simply add the wildcard map.

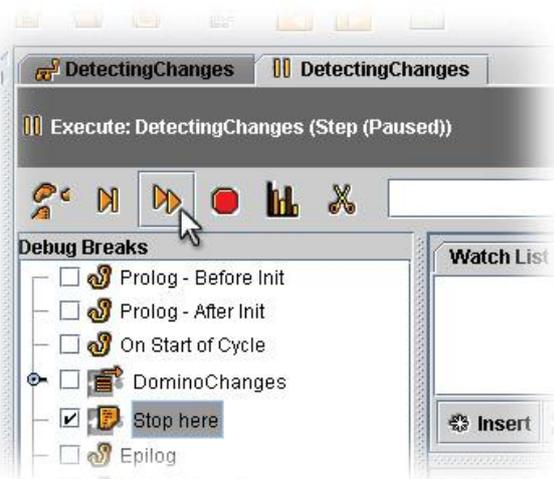
By setting the **Start At** parameter to *End of Data* we are telling the Connector to return only new changes that occur. In order to give us a handy spot for a breakpoint, add a Script component and call it “Stop here”.



Now you can start your AssemblyLine in *Step (Paused)* mode and let's watch what happens in the AL Debugger.

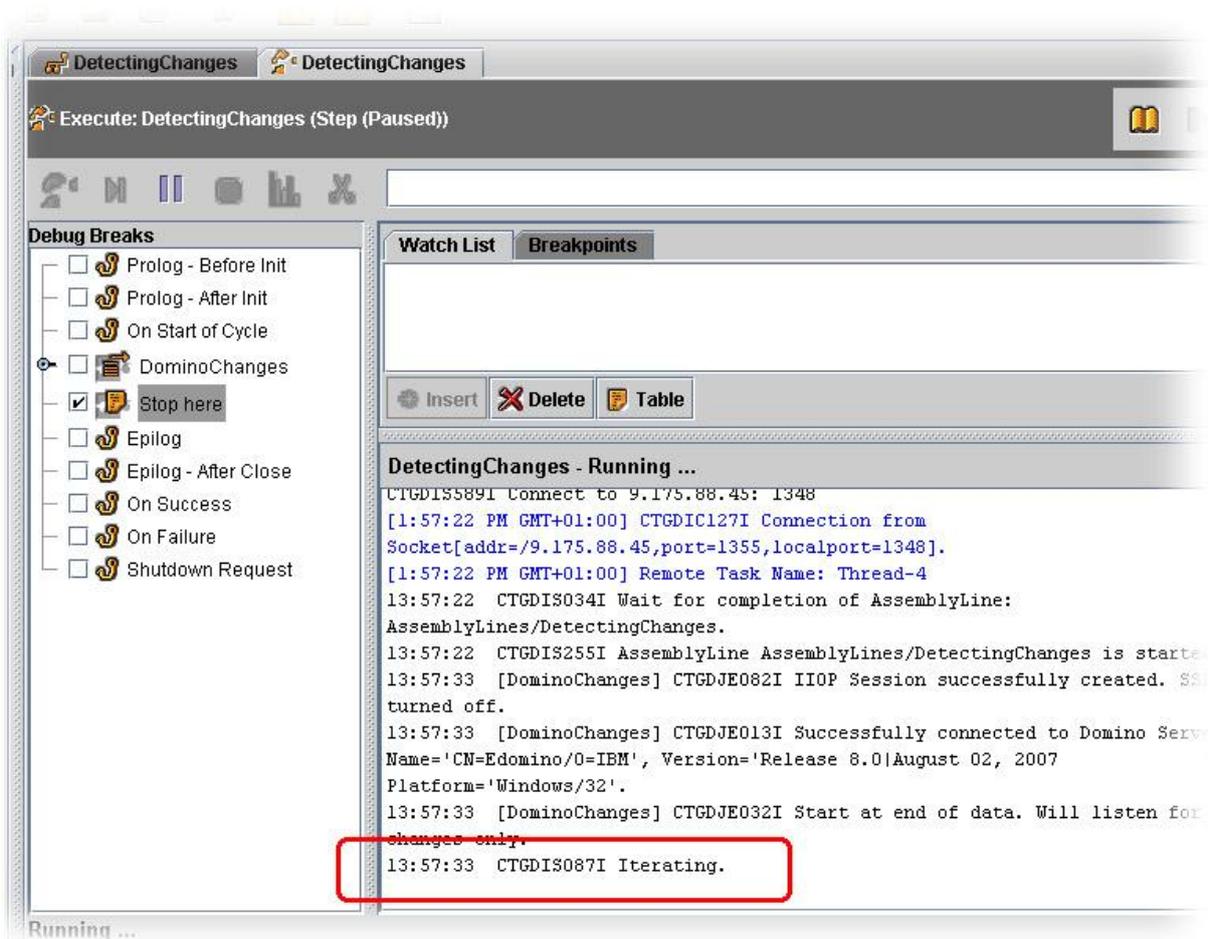


Once the Debugger loads then set a break at the “Stop here” Script component and then press the Continue button.



Your AL will initialize and then appear to hang. This is because your Domino Change Detection Connector is now waiting for modifications to occur to `names.nsf`. You

can see that your Iterator is active by the “Iterating” message in the log output window:

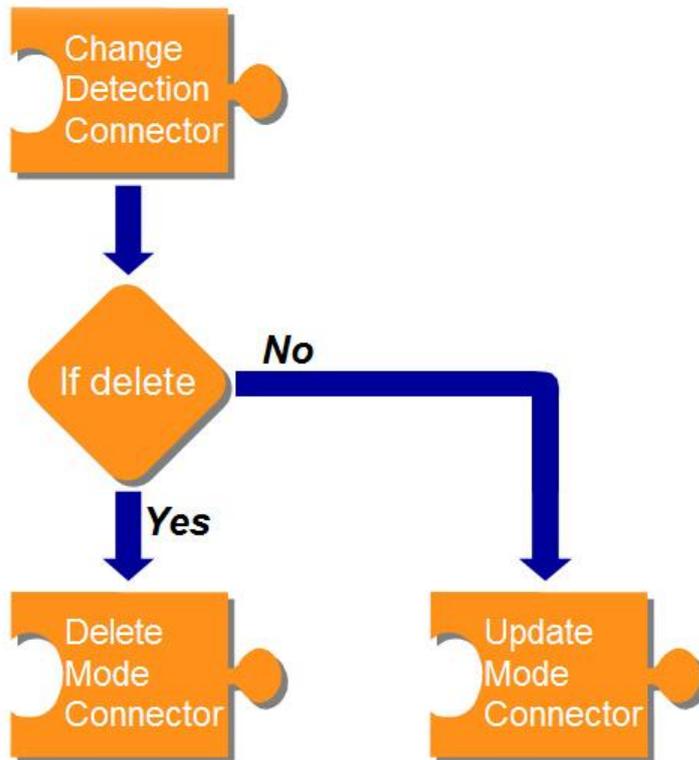


In order to get control back in the Debugger, you will have to edit an entry in the Domino database, for example via your Notes Client or Domino Administrator. Once a change is detected, the running TDI Server passed control back to the Debugger and you can enter this script snippet in the JavaScript Command Line:

```
work.getOperation()
```

This returns the Work Entry’s current delta operation code: “add”, “delete” or “modify”.

Use this delta information to propagate these changes, for example by using Branches in your AL.



Update mode handles *add* and *modify* changes, while you need a separate Connector in Delete mode to deal with *deletes*.

If your target is an LDAP or JDBC compliant system then another option is to use Delta mode. Delta mode uses the delta operation codes found in the Work Entry to perform the output operation in the most efficient manner, dealing with all types of delta codes through a single Connector.

A third option is to have a single Connector in Passive State and to drive it yourself from JavaScript, for example from a Script component:

```

if (work.getOperation() == "delete")
    MyPassiveConnector.deleteEntry(work)
else
    MyPassiveConnector.update(work);
  
```

Note that calling these AL component methods (`deleteEntry` and `update`) invokes the same Hook flow logic as if the Connectors were *Enabled* and taking part in normal AssemblyLine operation, including Input/Output mapping. If you only want to perform the write operation then use the Connector Interface methods instead.

```

if (work.getOperation() == "delete")
    MyPassiveConnector.getConnector().deleteEntry(work)
...
  
```

To learn more on how to leverage this delta information to build simple synchronization AssemblyLines, work through video tutorial #9 on this page: <http://www.tdi-users.org/twiki/bin/view/Integrator/LearningTDI>, as well as the How-To Data Synchronization found on this page: <http://www.tdi-users.org/twiki/bin/view/Integrator/HowTo>.

---

Hopefully this document has giving you a flying start at implementing your own integrated Domino solutions. I am going to ask you, Dear Reader, to use the TDI newsgroup to ask your integration questions. Note that the question may have been asked before, which means the answer is already out there waiting for you to search (Google + TDI = lots of useful information).

And be sure to use the newsgroup<sup>46</sup> actively, share your knowledge with the community when you can and keep TDI'ing!

---

<sup>46</sup> The official newsgroup is here:  
<news://news.software.ibm.com/ibm.software.network.directory-integrator>  
and requires an NNTP news reader (I use Mozilla Thunderbird – <http://www.mozilla.com/thunderbird/>).  
There is also a web-based Google Group that mirrors its contents here:  
<http://groups.google.com/group/ibm.software.network.directory-integrator?lnk=li>  
making it infinitely easier to search this community knowledge base.

## 7. Error Messages and Suggested Solutions

Following is a list of error messages I encountered while writing this document (plus some from the official docs), followed by an explanation. Note that some of the messages shown here include data from the examples: like “Edomino” for the server name, or “Eddie Hartman” as login credentials. You will need to substitute solution-specific information with your own values when trying to locate your errors in this list<sup>47</sup>.

**Unable to execute Domino Action. NotesException occurred: ID=4,005, Error Text=Notes error: Unable to find path to server. Check that your network connection is working. If you have a working connection, go to Preferences - Notes Ports and click Trace to discover where it breaks down.**

You can get this error with the Domino Users Connector if the Domino server name specified in attributes like REG\_Server (or MailServer) is incorrect.

**Unable to execute Domino Action. NotesException occurred: ID=4,005, Error Text=Notes error: You are not authorized to perform that operation**

Although the credentials used in the Domino Users Connector were sufficient to register a new user, the script code that attempts to open the mail db failed with this error (*not authorized*). To get past this problem I had to edit the mail template used for mail db creation to specify sufficient privileges for the login id used by the DUC.

I did this through the Domino Administrator by opening the template and editing its ACLs. There I added [wpsadmin/ibm] as a *Person/Manager* to the ACL list. The square brackets instructs Domino to *inherit* this ACL into new mail db's. I also discovered that since my DUC was using a Local Client connection, it was the mail template under my client installation `data` directory that needed to be modified.

---

<sup>47</sup> Some of these are from the section on Troubleshooting the Domino Change Detection Connector in the TDI Reference Guide (online documentation – Help menu select).

**NotesException: User CN=wpsadmin/O=IBM cannot open database Edomino!!<mailfile path>**

This one showed up for the Default Success Hook script code that opened the users's mail db. It turned out that although my wpsadmin account could be used by the Domino Users Connector to create new Notes users (and their mail files), this account did not have sufficient privilege to open and modify the db. To fix this I ended up in the Domino Administrator where (using super-user privileges) I opened the template used for new mail files. There I added "[wpsadmin/ibm]" as Manager (with delete rights) to the ACLs for the template. The brackets mean that this will be *inherited* into any mail db's created with this template. That gave my script code access to open the db and set the other ACLs and mail quote size.

Note also that if you are connecting via the local client, then it is the local client's mail template that is used on user mail db creation.

**javax.naming.NameNotFoundException: [LDAP: error code 32 - No Such Object]; remaining name 'cn=Alysson A. Angst,O=IBMM'**

As you can see from the message itself, this is an LDAP error and therefore will only happen when you use the LDAP Connector. The *No Such Object* error indicates that the \$dn you are creating is incorrect, and that you are trying to add a new entry under a parent that does not exist (like the "O=IBMM" above).

**Failed to execute the command. NotesException occurred: User Eddie Hartman/ibm is not a server**

You can only get this error with the Domino Users Connector because it alone supports a Local Server connection. I got it because I selected *Internet Password* for the **Authentication Mechanism** parameter (i.e. *Local Server*) but my PATH was pointing to the Notes client installation area. To correct this all I did was point my PATH to the Domino installation directory instead.

**Fatal error while executing the command. Exception occurred:  
java.lang.NoClassDefFoundError: lotus.domino.local.NotesReferenceQueue**

The *NoClassDefFoundError* indicates that you are using the wrong .jar file; for example, the ncso.jar where you should have used Notes.jar. Simply stop TDI, switch .jar files and restart TDI to fix.

**Connector was unable to initialize local Notes session to Domino Server. Exception encountered: java.lang.Exception: Native call NSFDbOpen failed with error: code 2051, 'Unable to find path to server. Check that your network connection is working. If you have a working connection, go to Preferences - Notes Ports and click Trace to discover where it breaks down.'**

This error appeared when I connected using the Domino Change Detection Connector. It turned out that I had set the **Port** parameter to 80 for this Connector, forgetting that the HTTP Port setting for my Domino Server was 88. However, even after fixing this I discovered that even though I was using DIIOP, I needed to set my PATH to point to the Domino server installation folder(?!). After I corrected this, it worked. Note that this is not necessary with the Notes Connector.

### **java.lang.reflect.InvocationTargetException**

This is an error I got when the DIIOP task was not running, or incorrectly configured; For example, when there was no Internet Document defined for DIIOP.

### **NotesException: Could not get IOR from Domino Server**

It could be the the HTTP task is not running on your Domino server, or that it is on a different port than specified in the Connector.

**NotesException: Could not open Notes session:  
org.omg.CORBA.COMM\_FAILURE: java.net.ConnectException: Connection  
refused: connect Host: <domino\_server\_ip> Port: XXXXX vmcid: 0x0 minor  
code: 1 completed**

This exception indicates that the DIIOP Server task on the Domino Server is not running, or that the IOR is incorrect. Start the DIIOP Server task on the Domino Server you are trying to access and then start your AssemblyLine again.

### **The Domino Change Detection Connector reports all database documents as deleted although they are not deleted.**

The credentials used to configure this Connector do not have sufficient rights to read the changed data. Correct this by changing the credentials, or editing the ACLs for this NSF file.

**java.lang.UnsatisfiedLinkError: <TDI\_install\_folder>\libs\ldomchdet.dll: Can't find dependent libraries**

**Note:** If you run the Integrator Server from the command prompt, then before this exception message is printed, a popup dialog box appears saying "This application has failed to start because nNOTES.dll was not found. Re-installing the application may fix this problem."

This exception message as well as the popup dialog box are displayed because the Connector is unable to locate the Lotus Notes dynamic-link libraries. Most probably the path to the Lotus Notes directory specified in `ibmditk.bat` or in `ibmdisrv.bat` is either incorrect or not specified at all. That is why you should verify that the Lotus Notes directory specified in the PATH environment variable in both `ibmditk.bat` and `ibmdisrv.bat` is correct.