



Eddie Hartman
Axel Buecker

IBM Lotus Domino Integration Using IBM Tivoli Directory Integrator

Introduction

In this IBM® Redpaper™ publication we take you step-by-step through a series of integration scenarios to help you start building your own solutions with IBM Tivoli® Directory Integrator¹. Once you have mastered basic skills you will quickly be wiring your IBM Lotus® Domino® applications (both the shrink-wrapped and home-grown varieties) more tightly into the infrastructure, and you will also be sharing data between Lotus Domino and other systems more simply and easily than you probably thought was possible.

Note: Note that although this paper highlights Tivoli Directory Integrator Version 7, the same techniques apply to previous versions.

Introducing Tivoli Directory Integrator

Tivoli Directory Integrator (TDI) is a graphical integration toolkit for accessing and detecting changes in practically any type of system, data store, protocol, or API. It also lets you transform, filter, and validate this data before driving it to the output targets of your choosing. Tivoli Directory Integrator is light-footed, installs in minutes, and can do serious work right from your mobile computer, just as it can on an enterprise server or mainframe platform.

To begin your journey in rapid integration you must understand a few basic Tivoli Directory Integrator concepts.

Components

Connectivity to a wide range of systems is provided through Tivoli Directory Integrator's rich library of components. If there is not a suitable component available then you can download

¹ There are already several self-training resources available online, including a number of YouTube videos and How-To documents, in addition to these IBM Education resources:
http://publib.boulder.ibm.com/infocenter/ieduasst/tivv1r0/index.jsp?topic=/com.ibm.iea.tdi/tdi/7.0/configuration_editor.html

new ones or create your own. You can even build and test new components using JavaScript directly in the Tivoli Directory Integrator development environment.

The component type that you use the most is called a *Connector*. Connectors abstract away the technical complexities of working with a technology or platform, making all connected systems behave in a similar and predictable fashion. This *levels the playing field* and lets you focus on solving the business problem at hand. As you have probably guessed already, Tivoli Directory Integrator provides Connectors for working with Lotus Domino. Because Lotus Domino is such a rich and versatile application framework, Tivoli Directory Integrator offers four different Connectors, each suited to a particular aspect of Lotus Domino integration work:

- ▶ If you want to access Lotus Domino Directory then use the LDAP Connector.
- ▶ To read or write Documents in an NSF database, use the Lotus Notes® Connector.
- ▶ A Lotus Domino Users Connector is provided for managing User Accounts, including mail and ID file creation.
- ▶ There is a Lotus Domino Change Detection Connector for catching changes that occur in an NSF database, allowing you to trigger follow-up logic such as passing these changes to other systems and sending alerts to audit systems (for example, when certain group memberships change)².

AssemblyLines

You create a solution by selecting appropriate components and stringing them together to form an *AssemblyLine*. Each AssemblyLine (AL) represents a single flow of data, as shown in Figure 1. The Tivoli Directory Integrator approach is to simplify a problem into the individual data flows and then implement these as individual AssemblyLines step-by-step with lots of visual feedback.

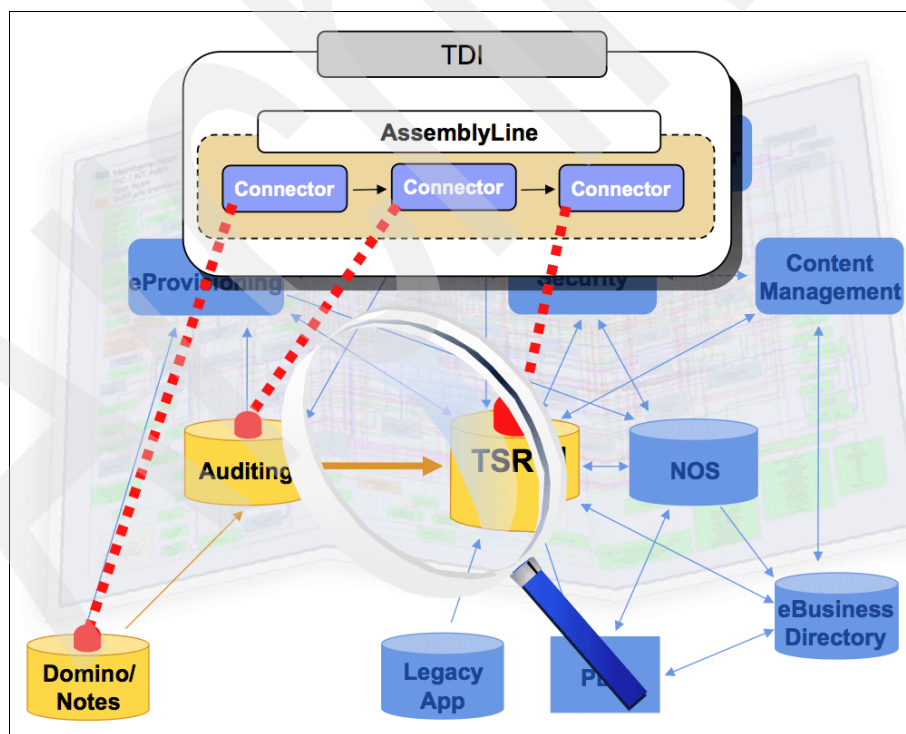


Figure 1 An AssemblyLine is a series of components

² Tivoli Directory Integrator provides similar Change Detection features for other systems, including the ability to compute changes occurring in data sources that do not provide any time stamp or notification services (like files).

The built-in automation logic of the AssemblyLine drives your Connectors one-by-one according to their *mode* setting. So if a Connector is in *Delete mode* then it will delete data in the connected system, whereas a Connector in *Lookup mode* will do a search.

Most of your data flows will be pulling information from one system in order to transform and transfer it to one or more targets. To facilitate this, there is a Connector mode specifically designed for cycling through an input data set. This mode is called *Iterator mode* and a Connector set to this mode is called an *Iterator*. In other words, most of your AssemblyLines will start with an Iterator for reading information—and possibly also detecting or computing changes—and passing this data down the flow for processing by the other components that you have added to your AL.

The Entry-Attribute-value data model

In order to pass data between components in an AL, Tivoli Directory Integrator provides a Java™ *bucket* that carries information between your components. This means that all Connectors in an input mode (like Iterator or Lookup) put data into this bucket, while Connectors in output modes (for example, AddOnly or Update) use its contents to perform write operations. This bucket is called an *Entry* in Tivoli Directory Integrator terms, and it can hold any number of *Attributes*. Attributes are named containers holding the actual data. If you are reading directly from `names.nsf` with the Lotus Notes Connector then you will see Attributes with names like `FullName` and `FirstName`, whereas the LDAP Connector works with the corresponding Lotus Domino Directory attributes `cn` and `givenname`.

The shared Entry object holding the Attributes that your AL is working on is called the AssemblyLine's *Work Entry*.

In addition to the Work Entry, each Connector has its own local bucket, called its *Conn Entry*, which it uses to cache reads and writes. All data movement between the Work Entry and the Connector's cache (Conn Entry) is handled by an *Attribute Map*, allowing you to select what information is read in or written out by a given Connector. This concept is shown in Figure 2.

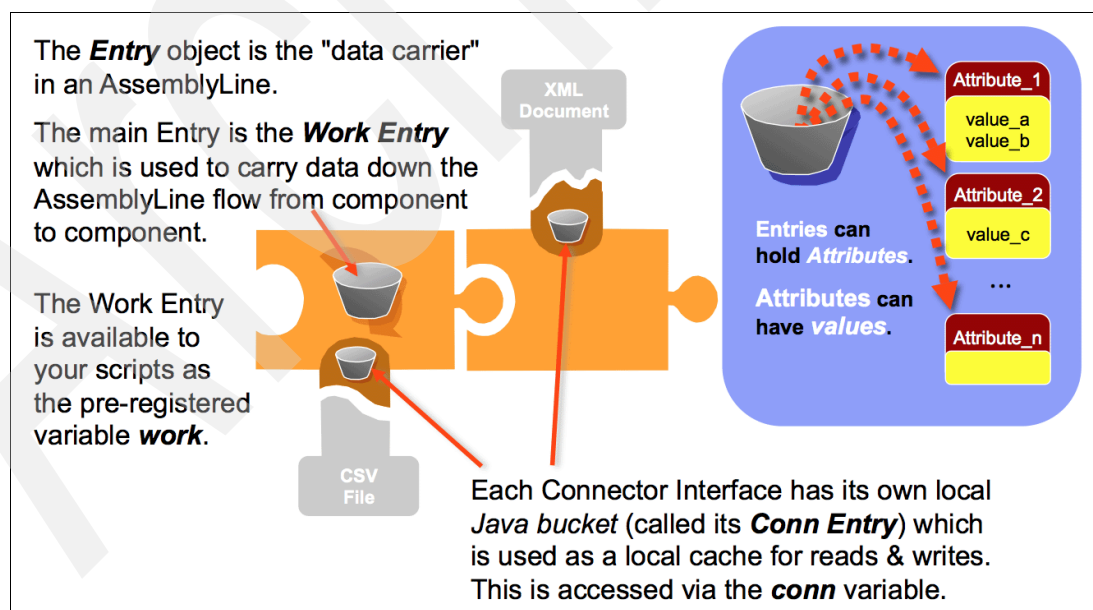


Figure 2 Tivoli Directory Integrator Entry-Attribute-value data model

A Connector in an input mode offers an Input Map for bringing data from the Connector's cache (Conn Entry) into shared data space (the Work Entry). Conversely, an output mode Connector provides an Output Map for selecting which values in your Work Entry Attributes

are to be put into the Conn Entry and thereby be available for the Connector's add or modify operation.

Just like the AssemblyLine, each component has an automation workflow that controls how the AL uses it through each step in the AssemblyLine life cycle:

- ▶ Initialization (and error handling)
- ▶ Data flow (including Attribute Maps, data access, and error handling)
- ▶ Connection close and AL termination (and error handling)

Flow Diagrams detailing the AssemblyLine's automation logic, as well as the workflow for components (including one for each Connector mode), can be found in the *IBM Tivoli Directory Integrator Reference Guide Version 7.0*, SC23-6562, available at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/TDI_7.0_FlowDiagrams.pdf

JavaScript

As previously mentioned, Tivoli Directory Integrator provides a set of automated workflows and features to speed up your integration work. However, you will also want to perform custom operations on your data. You do this by adding snippets of JavaScript to your AssemblyLine.

In addition to dropping Scripts directly as components into the flow of your AL, Attribute Maps can be scripted, as can the search criteria used for Lookup mode Connectors, as well as the conditions of Branches and Loops. Furthermore, there are scripting points called *Hooks* throughout the automation workflows of Tivoli Directory Integrator where you insert your own logic to extend or even override built-in behaviors.

See more information about scripting in Tivoli Directory Integrator here:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/scriptingintdi.htm

There are tips and gotchas available here:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/referenceguide114.htm#wq1070

Setting up your system

Setting up your system is divided into two parts:

1. Installing and configuring Tivoli Directory Integrator
2. Preparing your Lotus Domino server

The steps taken depend on how you intend to access your Lotus Domino data.

Setting up Tivoli Directory Integrator

The first thing that you must do is install Tivoli Directory Integrator. If you do not have access to IBM XtremeLeverage or the Passport Advantage® or PartnerWorld® portals, then you can grab a free copy for a 90-day evaluation here:

https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en_US&source=swg-itdid

During installation you will be asked to specify two locations:

- ▶ The Installation folder, where the binaries and application support files go
- ▶ The Solution Directory, where your own work is kept

This second location, the Solution Directory, is where all relative paths in your solution are resolved from and where Tivoli Directory Integrator stores your project work by default. It is also the default location of your Tivoli Directory Integrator workspace. Since Tivoli Directory Integrator 7 uses the Eclipse project model, this workspace folder is where all projects, AssemblyLines, components, and scripts that you create are saved as individual files.

If you are completely new to Tivoli Directory Integrator then we recommend that you work through the Getting Started tutorials found here:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/introducingtdi.htm

At this point you must decide how you will access Lotus Domino.

In order to work with Lotus Domino Directory

This is the simplest access method from a Tivoli Directory Integrator standpoint since you do not have to do anything with your Tivoli Directory Integrator installation. You will use the LDAP Connector, just as you would to access Active Directory, Tivoli Directory Server, Sun One, or any other LDAP V3 compliant directory.

Note that Lotus Domino Directory does not provide a changelog feature, and as a result you cannot use the Changelog Connectors in Tivoli Directory Integrator to detect changes. You can, however, either use the Lotus Domino Change Connector or enable the *Delta Engine*. The Delta Engine is available for any Connector in Iterator mode and provides automatic computation of data changes occurring between one AL run and the next. "Detecting and syncing changes" on page 28 includes an overview and exercises for the Delta Engine. More information about this powerful tool, which can be applied to all kinds of input, like files and ODBC data sources, as well as Delta handling in general, can be found here:

[http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/deltadetection.htm?resultof="delta""engine""engin](http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/deltadetection.htm?resultof=)

To read/write documents, detect changes, or manage accounts

If you need more than LDAP access to your Lotus Domino data then you will be using one of the Lotus-specific Connectors. But first you must make the Lotus Domino/Notes API available to Tivoli Directory Integrator. This can be done in one of two ways:

- ▶ Copy the required jar-file file (we will get to which one that is in just a moment) into the jars sub-folder of your Tivoli Directory Integrator installation directory, preferably to a new folder (for example, that you name Domino).
- ▶ Set the Tivoli Directory Integrator property `com.ibm.di.loader.userjars` to point to a folder where the jar-file is located. For example:

```
com.ibm.di.loader.userjars=C:\MyJarFiles
```

If you chose to have a Solution Directory during installation then you can edit the `solution.properties` file in this folder to specify the property. If instead you are using the installation folder as your Solution Directory, you must make this change to the `global.properties` file found in the `etc` sub-folder of your Tivoli Directory Integrator installation.

Note that if you direct this property at your Lotus Domino install area then you do not have to modify the PATH as described below.

For Lotus Domino Version 8 and later, the jar-file required is called `Notes.jar` and is located under the `jvm\lib\ext` folder of your Lotus Domino Server or Lotus Notes Client installation³.

Once the Lotus Domino API jar-file is in place (and if you did not point the `userjars` property to your Lotus Domino/Notes installation folder), the last step is to ensure that your `PATH` points to the remaining Lotus Domino libraries. For example:

```
PATH=C:\Program Files\IBM\Lotus\Domino;%PATH%
```

Point your `PATH` to the same installation where you found the jar-file and be sure to add the Lotus Domino location to the beginning of your `PATH`.

Note that whenever you make changes to where Tivoli Directory Integrator loads its libraries you must stop and restart Tivoli Directory Integrator.

Setting up Lotus Domino

As with the previous section, configuration requirements depend on how you intend to access Lotus Domino.

Lotus Domino Directory

As noted above, if you plan to access Lotus Domino Directory, then Tivoli Directory Integrator is ready to go without any additional configuration. On the Lotus Domino side you must make sure that the LDAP task is running.

³ For earlier versions of Lotus Domino, you need to use the `Notes.jar` file from your Lotus Notes Client installation if you plan to use the LocalClient connection option in your Tivoli Directory Integrator Connectors. This option means that you must have a Lotus Notes Client installed on the machine running your Tivoli Directory Integrator solution, but does not require additional tasks running on the Lotus Domino server.

The other approach is to choose the DIIOP connection option in Tivoli Directory Integrator, in which case you must retrieve the `ncso.jar` file from your Lotus Domino server installation. DIIOP also requires you to have the DIIOP task running on Lotus Domino, but does not require the Lotus Notes Client installed, which means that your solution can be run on a wider range of platforms.

Also note that with versions of Lotus Domino earlier than 8, where you have to differentiate between the two jar-files, make sure only one of these is present at a time. Otherwise you will get class loading errors in Tivoli Directory Integrator.

You also need the credentials of a user who has the necessary access rights (that is, via ACLs) to `names.nsf`. In addition, if your solution is to update Lotus Domino Directory then you must give LDAP users permission to perform write operations. This is done through the Lotus Domino Administrator under **Configuration** → **Directory** → **LDAP** → **Settings** (Figure 3).

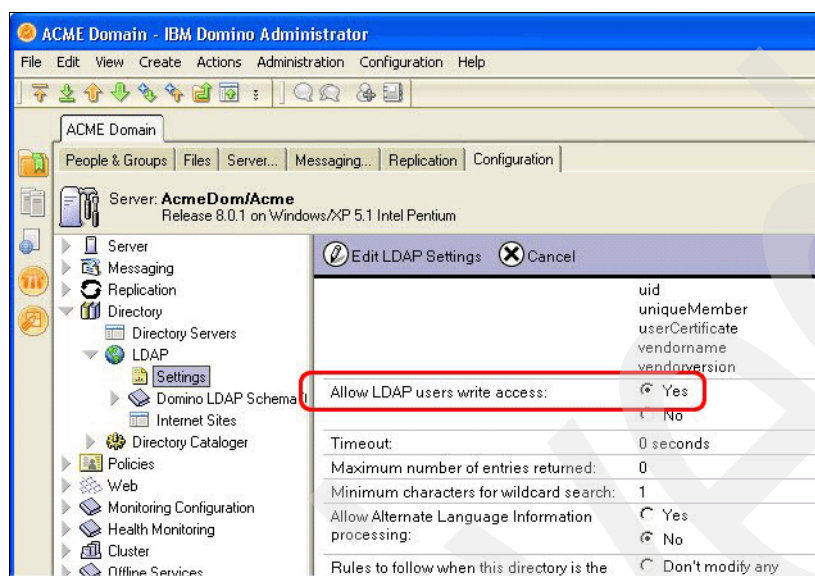


Figure 3 Allow LDAP users write access

The Domino LDAP Schema node shown in the figure above presents you with the list of LDAP attributes available and how they correspond to the various fields in a Person Document. So even though you have the standard `cn` attribute for the user's common name, you can still access fields like `fullname` and `shortname`, as long as the Person Document is for a registered user and these fields are in place. It is important to note that user entries created through LDAP will not be full Lotus Notes users. You must use the Lotus Domino Users Connector in order to provision new accounts.

DIIOp (also known as Corba)

This option does not require any local Lotus Domino installation (Server or Client). However, it does not allow access to local NSFs, only those found on the Lotus Domino server⁴.

Your first step is to enable the DIIOp task in your Lotus Domino server. Configuring and starting this service creates a file named `diio_p_ior.txt` in the `html` folder of your Lotus Domino server. This file contains a specially encoded string that starts with "IOR:" and describes how to communicate with the DIIOp service.

The Tivoli Directory Integrator Connectors that support DIIOp connections will do an HTTP GET to retrieve `diio_p_ior.txt`, in which case you must also have the HTTP task running on Lotus Domino. Another option is to open this file in a text editor, copy the entire string to your copy buffer, then paste it into the Hostname parameter of the Connector, in which case it skips the HTTP GET and goes straight to establishing the DIIOp connection using the encoded instructions.

As with Lotus Domino Directory access, you must use credentials in the Connector that have the necessary authorization to access the NSF database that you are working with.

⁴ A common practice is to use Local Client during development, allowing work to be done on local copies of NSF databases, and then switching to DIIOp for deployment.

Finally, to avoid getting a Login pop-up when the Connector initializes, you can tell Lotus Domino to disable this extra security step. This is done in either the Lotus Domino Administrator or the Lotus Notes Client from the **File** → **Security** → **User Security** menu selection, under Security Basics, and must be for the user ID that you will use in Tivoli Directory Integrator (Figure 4).

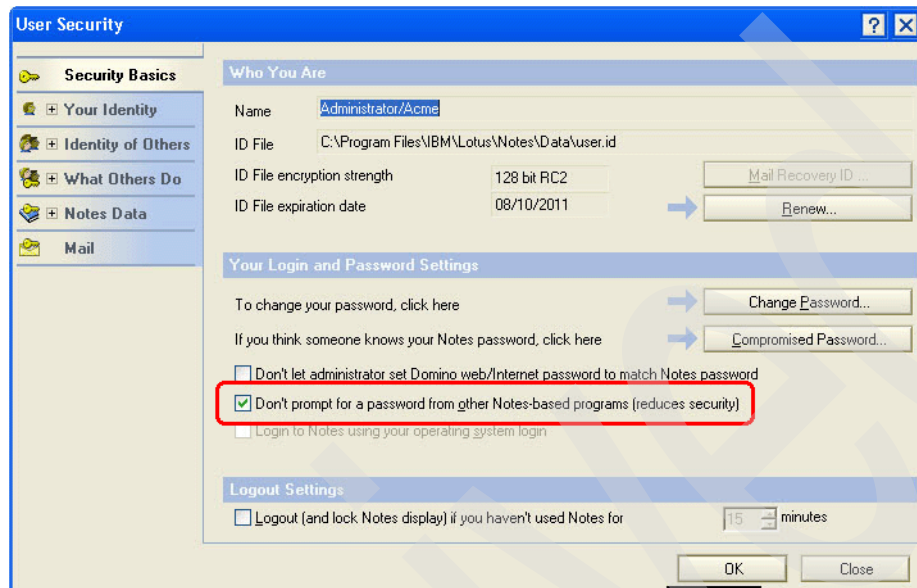


Figure 4 Disabling the extra security pop-up

This setting is also applicable if you plan to use a LocalClient connection option in Tivoli Directory Integrator Connectors.

LocalClient

Apart from disabling the extra security pop-up as detailed in the previous section, all you need is a Lotus Notes Client installed and an ID file for Tivoli Directory Integrator to authenticate against. Of course, the account tied to the ID file must have sufficient access rights to the NSF databases that you will be accessing.

You are now ready to start integrating.

Integration scenarios

Just as before, this material will be split into sections focused on either Lotus Domino Directory or working directly with Documents in NSF databases. But before you get this far, there are a few steps that are common for any Tivoli Directory Integrator work, Lotus Domino related or not.

Preparation: creating a Tivoli Directory Integrator project

When you start Tivoli Directory Integrator you first are asked where you want your workspace to be. The default location is in your *solution directory*, but you can place this folder wherever you want (preferably someplace already included in your backups).

Once the system starts you are presented with the Welcome page shown in Figure 5.

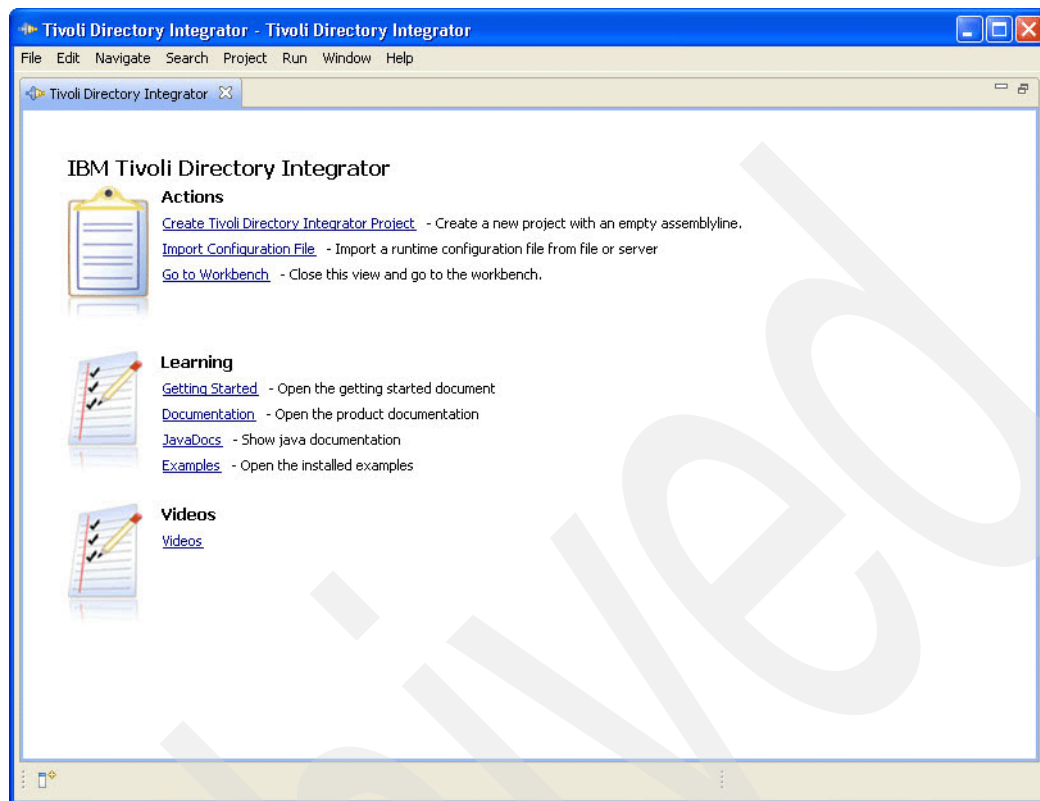


Figure 5 Tivoli Directory Integrator Welcome panel

This panel gives you a lot of shortcut links, including the topmost one for creating your Tivoli Directory Integrator project. Click this now and name your project something descriptive, such as Domino Integration. The panel switches to the *workbench*, which is the main development environment area. Your new project is highlighted in the Navigator view in the upper left of the panel, as shown in Figure 6.

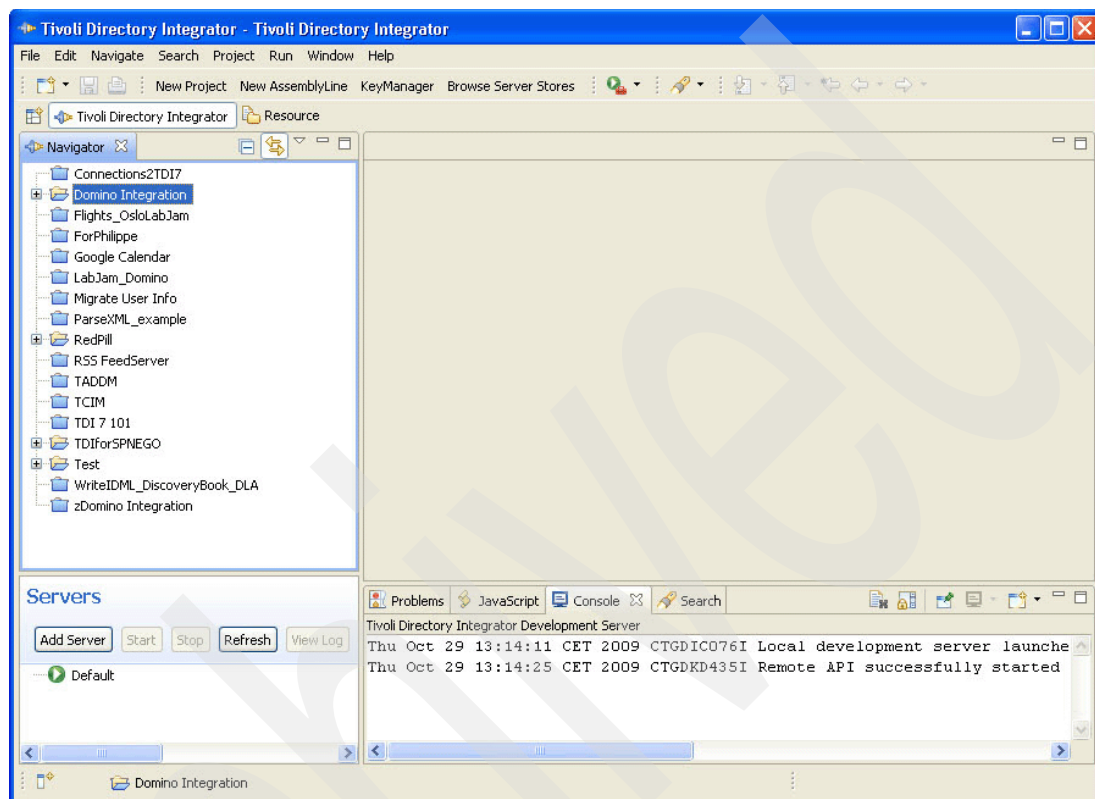


Figure 6 Tivoli Directory Integrator workbench

The Servers view just below the Navigator shows you Tivoli Directory Integrator servers that are available to test and run your solutions on. After the base install Tivoli Directory Integrator gives you a Default Server running locally on your machine.

Just to the right of Servers is a set of tabs that include the *Console* output from your server. Figure 6 shows you that your server has been launched automatically and is now running⁵.

Integrating Lotus Domino Directory

Lotus Domino Directory is often used for authentication into systems like Lotus Sametime® and Lotus Connections. It can also put an LDAP face on Lotus Notes User data so that directory-enabled applications can easily reach this information. As a result, integrating Lotus Domino Directory with other user repositories is a common requirement.

As long as you do not need to provision Lotus Notes User accounts, adding user entries to the Lotus Domino Directory—as well as group management through LDAP—may be all you need.

⁵ If for some reason your Tivoli Directory Integrator server is not starting, double-click it to open the Server Document. Here you can ensure that the installation and solution paths are correct. You can also change the port being used (in case there is a conflict) and use the Create Solution Directory button to make sure that your Solution Directory is set up correctly. Once this is done, close the Server Document and click **Start** to relaunch your test server. If that still fails, check the forum or contact Tivoli Directory Integrator support.

In this example we take the contents of a Lotus Domino Directory and export a subset of the data to a custom CSV file.

To begin configure an LDAP Connector to read from the Lotus Domino Directory. You must create a reusable library component by opening the **Resources** folder under your new Tivoli Directory Integrator project (in the Navigator) and then right-clicking **Connectors** and selecting the **New Connector** option. This opens the New Connector wizard, shown in Figure 7.

Here you type LDAP in the Search field to limit the choices presented. Select the LDAP Connector, name it DominoDirectory, then choose Iterator mode. Then click **Next** to configure it.

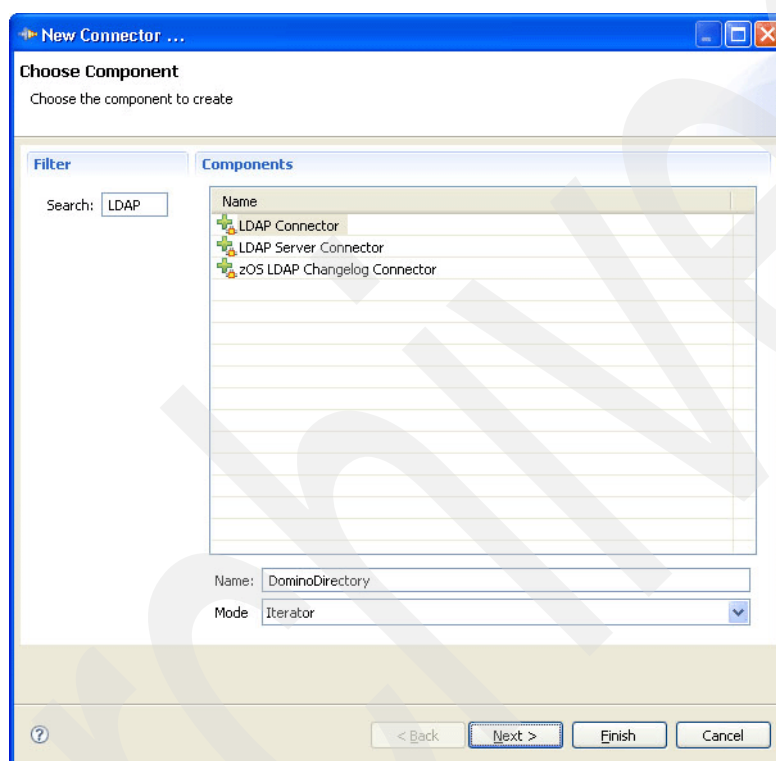


Figure 7 Make DominoDirectory Connector

Point the LDAP URL at the host and port serviced by your Lotus Domino Directory and enter a valid user name and password. Unfortunately, you cannot use the Context button to discover available suffixes, so just type in the organization (that is, O=ACME in our example), as shown in Figure 8. When you click **Finish**, your new Connector will now be a Resource that can be reused in your project. The advantage of using Resources is that if a configuration parameter changes (for example, the port number or credentials used), you can simply update this library component and all your AssemblyLines will inherit these changes.

The screenshot shows the 'New Connector ...' dialog box with the 'LDAP Connector' tab selected. The 'Connector Configuration' section contains the following fields and values:

- LDAP URL *: ldap://domino.acme.com:389
- Login username: cn=Administrator,o=ACME
- Login password: *****
- Search Base: O=ACME
- Search Filter: cn=*
- Search Scope: subtree
- Comment: (empty text area)
- Detailed Log: (unchecked checkbox)

Navigation buttons at the bottom include '< Back', 'Next >', 'Finish', and 'Cancel'. A 'Help' button is located near the top left of the configuration area.

Figure 8 Configure DominoDirectory Connector

The next step is to discover the schema and choose which attributes to read. Do this by right-clicking your DominoDirectory Connector and selecting **Browse Data** from the object menu. This opens the Data Browser (Figure 9).

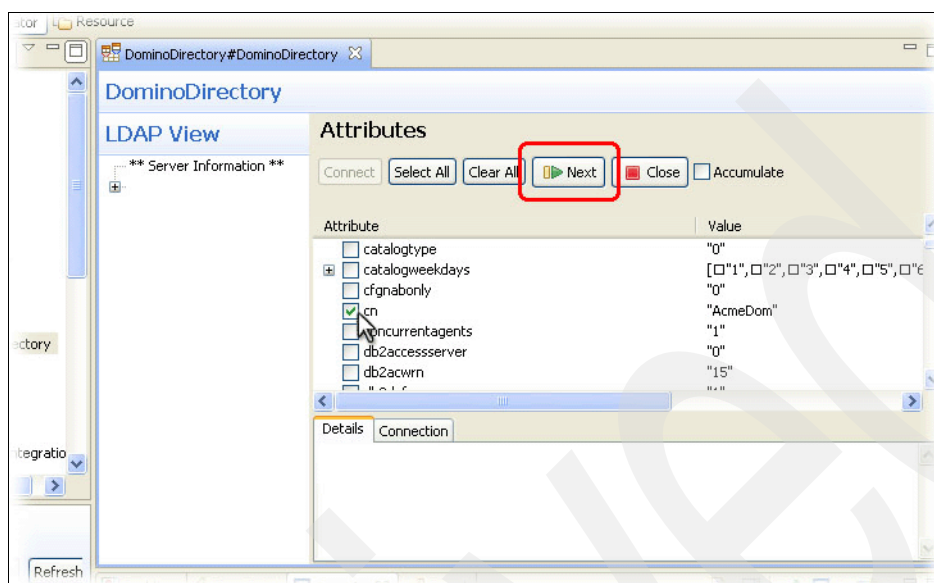


Figure 9 Browse Lotus Domino Directory

The Data Browser is a powerful and versatile feature that lets you step through information in any type of connected system by clicking **Next**. You can then scroll through the list of attributes and select those that you want to bring into your AssemblyLine. Choose **cn**, **title**, and **telephoneNumber** to make these available for processing in your AL.

When you close the Data Browser and then select the **Input Map** tab of the Connector, you can see that the selected Attributes are already configured in the map, as shown in Figure 10.

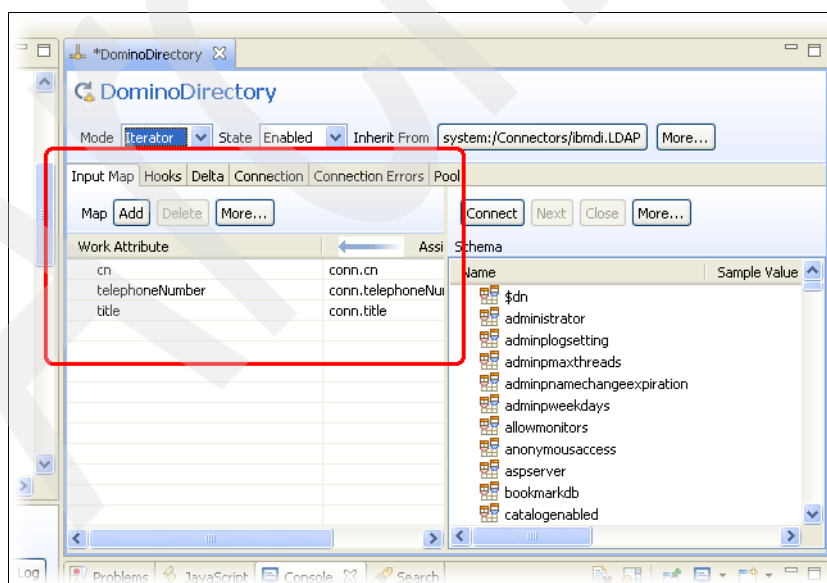


Figure 10 DominoDirectory Input Map

It is time to create your first AssemblyLine by clicking **New AssemblyLine** at the top of the Tivoli Directory Integrator window. Call this AssemblyLine Export Domino Directory.

Once the AssemblyLine editor appears, click **Add component**. Choose the File System Connector, call it WriteCSV, and set it to *AddOnly* mode. Then in the Configuration dialog, instruct it to write to a file called Domino.csv⁶. To keep things simple, this Connector will write out any Attributes available. Configure it to do so by selecting the Connector's Output Map and then clicking **Add**. In the Add attribute dialog that appears, select * (Map all attributes).

Now your AL has an Output Connector ready to write anything passed to it, but no input. Fix this by dragging your DominoDirectory Connector from **Resources** → **Connectors** into the Feed section of your AssemblyLine. The Feed section automatically cycles your Iterator, retrieving one entry at a time and passing it to components under the Flow section for processing.

When you click **Show mapping** your AssemblyLine should look like Figure 11.

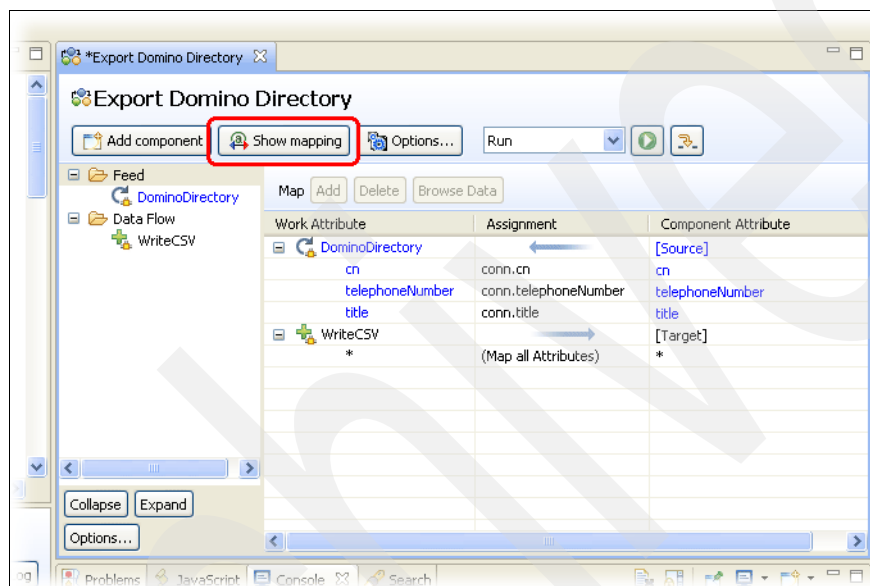


Figure 11 Export Lotus Domino Directory AssemblyLine completed

The Show mapping button presents you with a list of all data movement in and out of your AssemblyLine. You can now click the Run button (the green circle with the white play symbol) to test your work.

Clicking Run opens up a log output window where all messages written to log files are displayed on panel. This includes any errors that occur during AL processing.

When the AssemblyLine completes, return to your AssemblyLine editor tab, right-click your output Connector (**WriteCSV**), and choose **Browse Data** to examine and verify your output. Of course, you could open the output file in a text editor to view the results, but the Data Browser is a handy tool for this as well, and it works for all kinds of output types.

At this point the solution is writing out all entries found in the Lotus Domino Directory. To limit the list for export you can either change the *Search Filter* parameter of the DominoDirectory Connector or add an IF Branch to the AssemblyLine with conditions that define your filtering needs.

⁶ As mentioned at the start of this paper, if you use a relative path then it will be resolved from your Solution Directory. Otherwise, you can enter an explicit path to control where your file is located.

The first option requires you to know how to format an LDAP search filter, for example:

```
cn=E*  
or objectclass=dominoGroup
```

The first example filter above returns all entries where the cn attribute starts with E, while the second one gets you all of type dominoGroup.

Using IF Branches (and Loops) is part of the tutorial exercises found in the *IBM Tivoli Directory Integrator Getting Started Guide Version 7.0*, GI11-8185, so you are encouraged to work through the exercises found there to gain skills with this AssemblyLine feature (as well as many others).

NSF integration (optionally with an RDBMS)

In this example, information about a book collection will be read from an RDBMS database and loaded into Books.nsf through a DIIOP connection to Lotus Domino. As previously mentioned, since DIIOP cannot reach local NSF files, Books.nsf must reside in the data folder of the Lotus Domino server.

To begin you need a database table loaded with input data and the Books.nsf file. At the following URL you can get an XML file containing this information that you should copy locally to your machine:

<http://dl.getdropbox.com/u/375185/TUTORIALS/DominoIntegration/Books.xml>

You could forego building your own database table and instead use the XML file linked above as your input. If you choose to do this, then skip to “Loading data from an RDBMS into Lotus Domino Notes” on page 22. However, you will miss out on some powerful features being highlighted in this example.

Building and populating a DB table

First you must choose which RDBMS to work with. You can use one that you already have installed or you can use the database system included with your Tivoli Directory Integrator installation, the open source Derby RDBMS.

Preparing Tivoli Directory Integrator for your RDBMS

Just like when you configured Tivoli Directory Integrator for Lotus Domino/Notes communication, you must also prepare Tivoli Directory Integrator for working with your RDBMS. The first step is to find the driver, which will be either a JAR or a ZIP file. Tivoli Directory Integrator uses the JDBC protocol to access database systems, so you will be looking for JDBC driver files that you make available to Tivoli Directory Integrator in the same way that you did the Notes.jar file in “Setting up Tivoli Directory Integrator” on page 4⁷.

Your next step is to secure a license file, making this accessible to Tivoli Directory Integrator as well⁸. For example, the license file required for IBM DB2 is called db2jcc_license_cu.jar.

⁷ The JDBC driver for your database system is usually found somewhere in the installation folders of the RDBMS and you can download updated versions from the vendor's Web site. For example, a JDBC driver file available for Oracle 11g is called ora118.jar, whereas one version of the JDBC Driver 2.0 file for Microsoft® SQL Server is named sqljdbc.jar. Note that if you plan to use Derby or DB2®, then the driver, which is the same one used for both database systems, is already in place.

⁸ As mentioned in “Setting up Tivoli Directory Integrator” on page 4, as an alternative to copying the JAR/ZIP files you can instead configure the Tivoli Directory Integrator property com.ibm.di.loader.userjars to point to the area on disk where both the driver and license files are located (for example, the RDBMS installation directory).

Once you have made these preparations and restarted Tivoli Directory Integrator you are ready to begin building your AssemblyLine.

Working with JDBC

Here you will make two AssemblyLines: one to create the table called Books and the other to load it with data from the downloaded Books.xml file.

Start by creating a new Connector under **Resources** → **Connectors**. Choose the JDBC Connector, call it BooksDB and set it to *AddOnly* mode. Then fill out the parameter settings shown in Table 1.

Table 1 JDBC parameters

JDBC URL	<p>This is the URL to the database that you will work with. For example, if you are using DB2 running locally on port 50000 and the database is called ACME then your URL will look like this:</p> <pre>jdbc:db2://localhost:50000/ACME</pre> <p>For Derby running locally on port 1527 and with a database named ACME (which will be a folder in your Solution Directory with the same name) it would be this:</p> <pre>jdbc:derby://localhost:1527/ACME;create=true</pre>
JDBC Driver	<p>This is the class name of the driver that Tivoli Directory Integrator will use. For DB2 this will be:</p> <pre>com.ibm.db2.jcc.DB2Driver</pre> <p>For Derby it will look like this:</p> <pre>org.apache.derby.jdbc.ClientDriver</pre>
Username	ID that Tivoli Directory Integrator will authenticate with. The default user name for Derby is APP.
Password	The password associated with the ID used in the previous parameter. The default password for Derby is APP.
Schema	Required if your database system uses schema to segment databases. You do not have to set this parameter if you are using Derby.
Table Name	The name of the table to which you will be writing. For this example you will call the table Books.

Now create an AssemblyLine called CreateTable and drag in your **BooksDB** Connector, dropping it on the folder entitled Flow. Select the Connector in the AL component list and set it to **Passive** with the State drop-down, as shown in Figure 12.

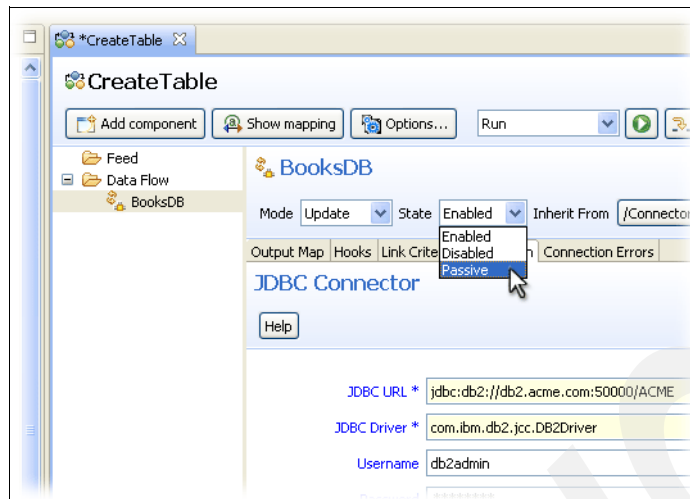


Figure 12 Set your BooksDB Connector to Passive state

Passive state means that your Connector will be initialized when the AL starts up and closed when execution is completed, but it will not take part in standard AssemblyLine processing. Instead, you will use the connection that it has to your database to fire off the commands to create the database table.

Creating the table

Now you will define the Data Definition Language (DDL) calls that are needed to create the table by adding an *Attribute Map* component to your AssemblyLine. Call it Set up DDL and click **Finish**.

Select this new component and click **Add** at the top of the Attribute Map list to create a mapping rule, as shown in Figure 13. Name this Attribute Map item DropTable and then click **OK**.

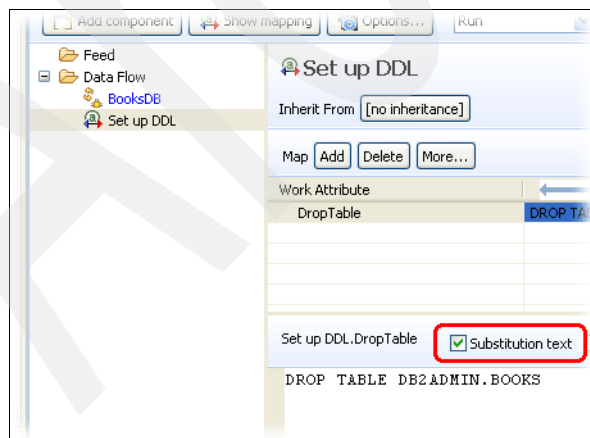


Figure 13 Set up the DropTable Attribute

Now double-click the new mapping rule in order to open the assignment editor. Once the editor panel is open, select the Substitution text check box and enter the following SQL for deleting the Books table:

```
DROP TABLE DB2ADMIN.BOOKS
```

This example is for DB2 and uses a Schema called "DB2ADMIN". You may have to modify this and the other DDL examples to fit the needs of your RDBMS⁹.

The Substitution text option for a mapping rule allows you to enter any literal value for this Attribute, even one with multiple lines of text.

Now create two more mapping rules, "CreateTable" and "CreateIndex", using the same steps that you did for "DropTable". However, for "CreateTable" the DDL is shown in Example 1 (again, possibly modified to fit your RDBMS syntax).

Example 1 Example DDL for creating the BOOKS table

```
CREATE TABLE "DB2ADMIN"."BOOKS" (  
    "ACQUIRED" DATE ,  
    "AUTHOR" VARCHAR(150) NOT NULL ,  
    "BINDING" VARCHAR(20) NOT NULL ,  
    "CALLNO" VARCHAR(25) NOT NULL ,  
    "CATEGORY" VARCHAR(30) NOT NULL ,  
    "COST" DECIMAL(5,0) ,  
    "EDITION" VARCHAR(15) ,  
    "EXLIB_DATE" DATE ,  
    "LOSS_DATE" DATE ,  
    "NEW_USED" VARCHAR(10) ,  
    "CALLNO_OLD" VARCHAR(25) ,  
    "PUBLISHER" VARCHAR(150) NOT NULL ,  
    "TYPE" VARCHAR(20) ,  
    "SOURCE" VARCHAR(60) ,  
    "TITLE" VARCHAR(500) NOT NULL )
```

Then use the code in Example 2 for the "CreateIndex" mapping rule.

Example 2 Example DDL for creating the index for the BOOKS table

```
ALTER TABLE "DB2ADMIN"."BOOKS"  
    ADD CONSTRAINT "CC1257156491459" PRIMARY KEY  
    ("CALLNO")
```

You are now ready to add an Empty Script to your AssemblyLine (just type Empty in the Search field of the New Component dialog to find it). Call this Script Make DDL calls and then enter the JavaScript snippet as shown in Example 3.

Example 3 JavaScript for sending DDL calls to the RDBMS

```
// Get the JDBC Connector Interface in order  
// to push the SQL/DDl calls through to the RDBMS  
jdbc = BooksDB.getConnector();  
  
// First issue the call to drop the table (in case  
// it already exists). Note that it is not a problem  
// if the DROP TABLE fails, since the first time you
```

⁹ For Derby you would drop "DB2ADMIN." from the table name: DROP TABLE BOOKS

```
// run this it does not exist yet.
result = jdbc.execSQL(work.DropTable);

// Now to create the table
result = jdbc.execSQL(work.CreateTable);

if (result) // if you got an error message back
    task.logmsg("**ERROR Creating table: " +
        result)
else {
    // If the table is in place, create the index
    result = jdbc.execSQL(work.CreateIndex);
    if (result) // if you got an error message back
        task.logmsg("**ERROR Creating index: " +
            result)
    else
        task.logmsg("-> Table created successfully!")
}
```

At this point your AssemblyLine should look like Figure 14.

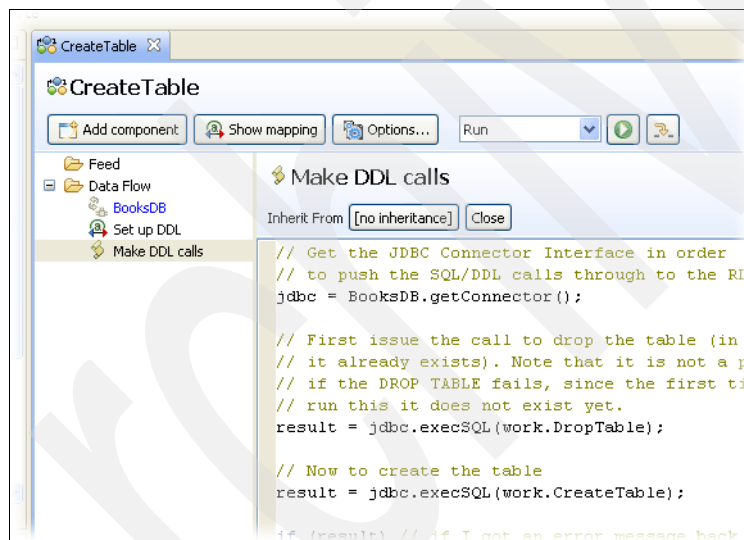


Figure 14 CreateTable AssemblyLine

You are now ready to run your AL¹⁰, which you do by clicking Run at the top of the AssemblyLine editor window (the white *play* symbol in the green circle). The AssemblyLine is then started on your Default test and the log output is displayed. If all goes well then you will see the message Table created successfully!, as shown in Figure 15.

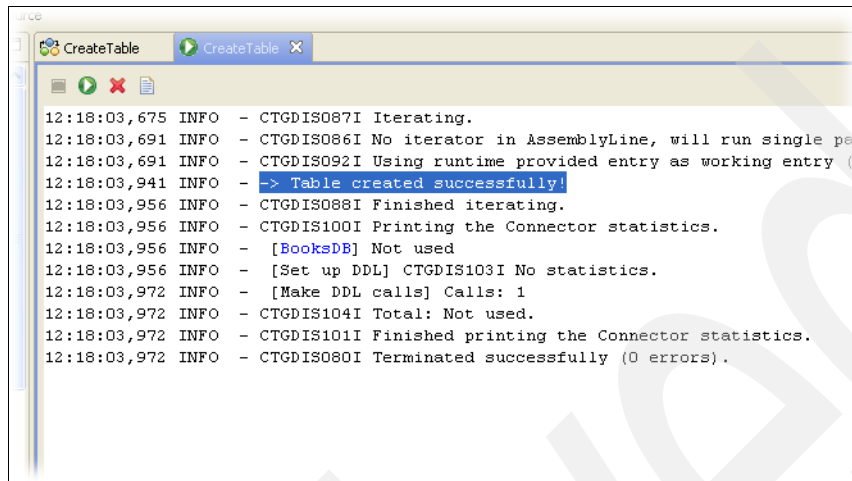


Figure 15 Log output from successful CreateTable AL run

Just in case: If you encountered errors then there will be some guidance for correcting them in the log output. However, you can get more details by selecting your Default server in the Server view (under the Navigator) and clicking **View Log**. Scroll to the end of the log file displayed to get the full error message and context. If you are unable to solve this problem then drop a note in the Tivoli Directory Integrator User Forum and leverage a little community know-how:

<http://groups.google.com/group/ibm.software.network.directory-integrator/topics?gvc=2>

Once the table is created, your next step is to load it with data.

Reading XML and writing to the database table

Create a new AssemblyLine and name it "LoadTable". Then add a File System Connector, name it "ReadXML", and select Iterator mode. Click **Next** and set the File Path to where you downloaded the Books.xml file. Click **Next** again and select the XML Parser. Instead of configuring the Parser now, click **Finish** and then right-click your new Connector and select **Browse Data**.

¹⁰ Before you do, stop to appreciate what is happening here: your BooksDB Connector is providing an *open pipe* to the connected system so that you can make calls directly to it. Other Connectors, such as LDAP and Domino Notes, provide the same functionality, enabling you to drive underlying protocols, transports, and APIs as much as you would like. For example, you would use a similar technique to modify the ACLs of an open NSF.

This time when the Data Browser opens it detects that you are reading from a byte stream and formats its display to enable you to see what the stream looks like on the Details tab (Figure 16). Here you can see that the node that you want to return as an Entry is called "Book".

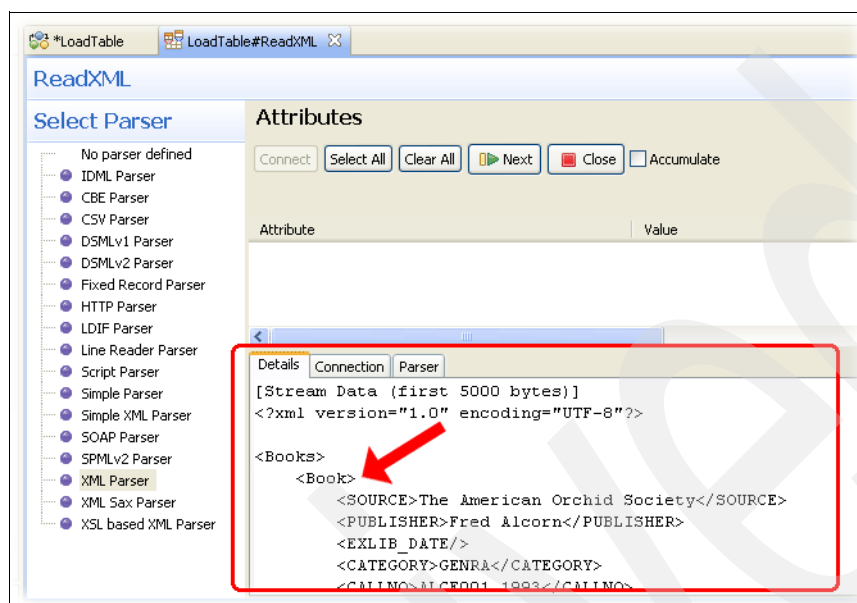


Figure 16 Use the Data Browser to examine the XML stream

Since you now know what the Entry-level node is, click the **Parser** tab of the Data Browser (to the right of "Details") and set the Entry Tag parameter to "Book". You can now click **Close** and **Connect** to reinitialize the Parser and then click **Next** to read an Entry. Finally, click **Select All** to choose all Attributes for mapping. When you close the Data Browser your Input Map (and Input Schema) is filled out (Figure 17).

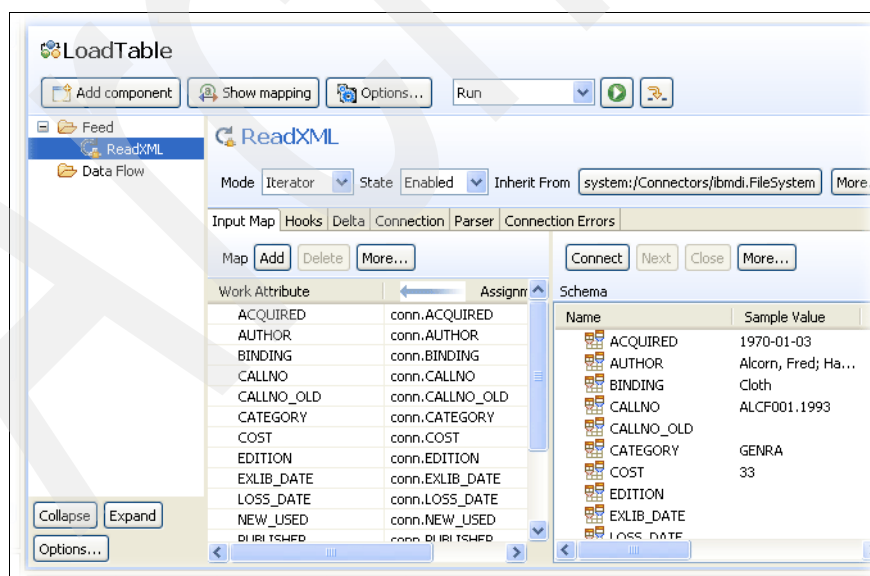


Figure 17 Input Map and Schema complete for ReadXML

Now you can drag in your "BooksDB" Connector, ensuring that the Mode drop-down list is set to AddOnly. Select the Input Map, click **Add**, then select the option to Map all attributes (*). Since the Attributes in the XML file have the same names as the columns of the Books table,

no special mapping or transformation is required. Your LoadTable AssemblyLine is ready to run. Do so by clicking **Run**. Finally, you can confirm the statistics in the log output, shown in Figure 18.

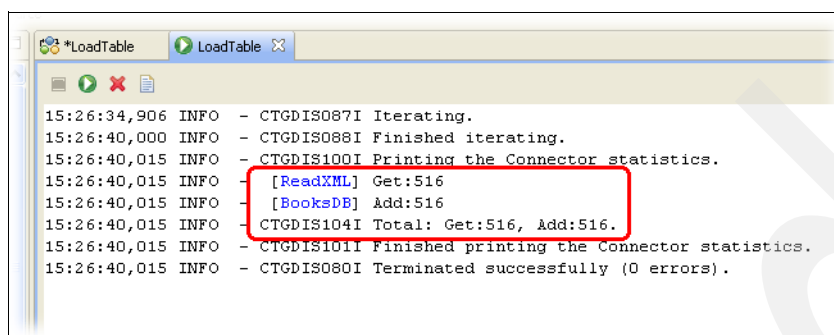


Figure 18 Successful run of the LoadTable AL

As one last step, you can combine your two Table-building ALs into a single one by having CreateTable call LoadTable. Do this by opening the CreateTable AL again and then adding a new Component to the end of the Flow section. Choose the AssemblyLine Function Component (AL FC), name it "Load the table", and in the Configuration panel select your LoadTable AL for the AssemblyLine parameter. Now open the Advanced section at the bottom and enable Share Logging so that any log messages from LoadTable appear together with those from the calling AL.

Now run it again and look at the combined log information in Figure 19.

You have created a single AssemblyLine that will both create and populate your BooksDB table.

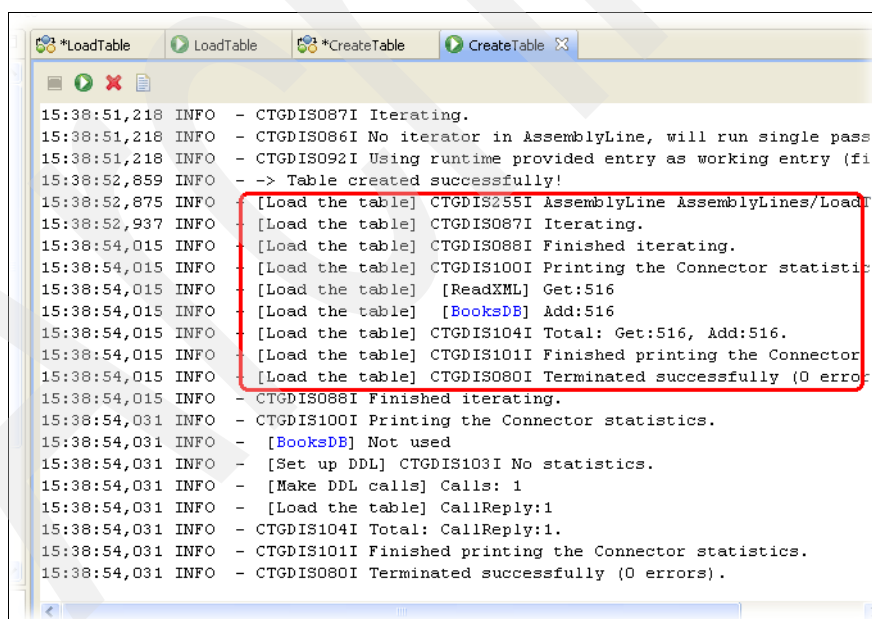


Figure 19 CreateTable calling the LoadTable AL

Loading data from an RDBMS into Lotus Domino Notes

For this exercise you must download the Books.nsf file and add that to your Domino Server:

<http://dl.dropbox.com/u/375185/TUTORIALS/DominoIntegration/Books.nsf>

Now create a Connector under **Resources** → **Connectors** for working with this Lotus Notes application. Choose the Lotus Notes Connector and call it "BooksNSF". Set the mode to *AddOnly* and click **Next** to configure it. Here you will need to set the parameters shown in Table 2.

Table 2 Lotus Notes Connector Parameters

Lotus Domino Server IP Address	This is the IP address to or host name of your Lotus Domino Server. Alternatively, you can copy the contents of your Lotus Domino Server's DIIOP_IOR.txt file and paste it here.
Lotus Domino Server Name	Optionally, you can put the Server name here to aid legibility.
HTTP Port	If you are requesting DIIOP_IOR.txt from your Lotus Domino Server then you must put the HTTP port number serviced here. Otherwise, if you have copied the contents of this file to the topmost parameter above then this value can be left blank.
Session Type	Choose DIIOP for this exercise ^a .
Username	Here you enter the user ID that Tivoli Directory Integrator will use to connect to Lotus Domino.
Password	The password for the user ID entered above.
Database	This parameter is set to Books.nsf.

a. You can of course use LocalClient if you have a Lotus Notes client installed and you add Books.NSF to your client's data area.

Once configured, click **Finish** to close the New Connector wizard. You can now right-click your new Connector and choose **Browse Data** to check your configuration and examine this data source.

Since the Tivoli Directory Integrator cannot discover the schema of a Lotus Notes Document, there is a single entry available in this example. This allows you to click **Next** to view this Document's Attributes and then select them all, as shown in Figure 20.

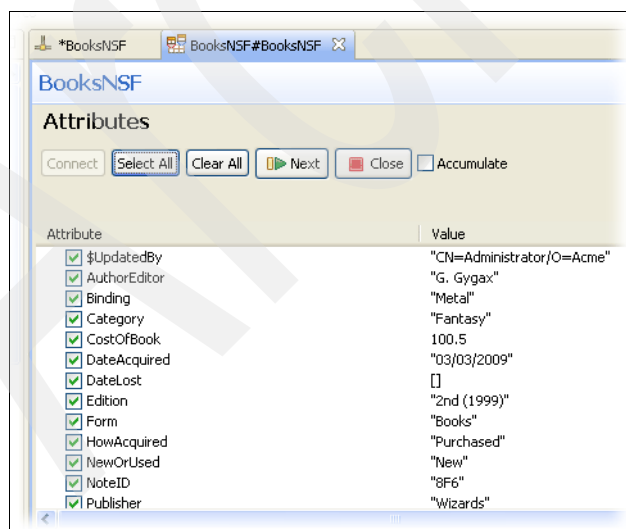


Figure 20 Browse BooksNSF and select all attributes

When you close the Data Browser the Output Map and Schema should be filled with the list of available Attributes. You will want to delete the special, computed Attributes: "\$UpdatedBy", "NotelD", and "UNID". This leaves you with only data-content Attributes in your map.

It is time to create your AssemblyLine. Call it "DB2NSF". If you worked through the previous section then you have a BooksDB Connector that you can drag into the Feed section. Remember that you must change the mode to *Iterator*. If you did not build the database table in the previous section then you must instead set up a File System Connector to iterate through the XML file as described in "Reading XML and writing to the database table" on page 20. Set the Input Map to read in all Attributes.

Now drag your BooksNSF Connector into the Data Flow section of the AssemblyLine, making sure that it is in *AddOnly* mode. The last step is to set up the Output Map of this Connector to deal with the differences between the input source field names and those of the Lotus Domino target.

If you click **Show Mapping**, as depicted in Figure 21, you get an overview of both lists of Attributes: incoming from the database and outgoing to Lotus Domino/Notes.

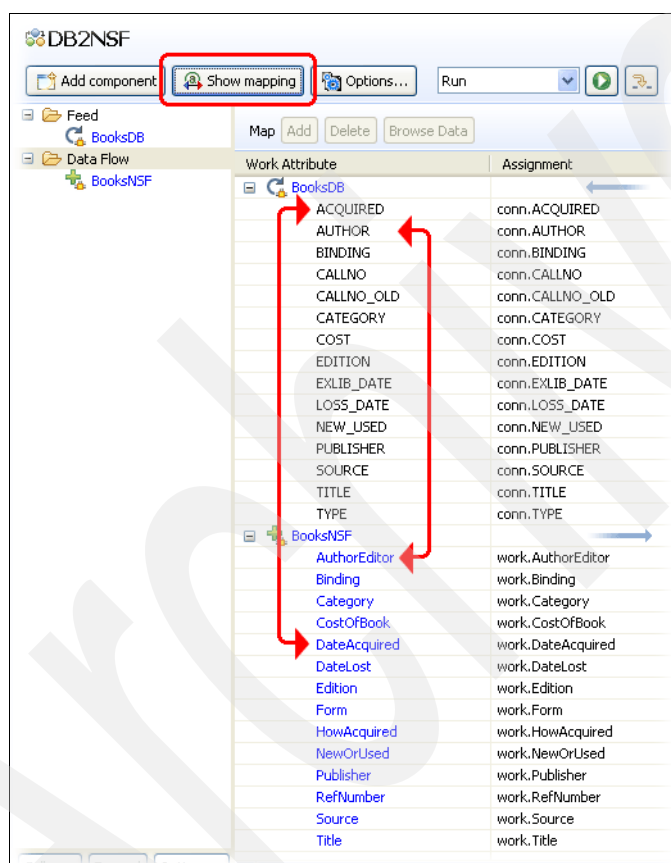


Figure 21 Mapping between input and output schema

As you can see from the mapping overview, there are fewer Attributes in the target output schema. Furthermore, many of the names do not match. Since the default mapping assignments for our Output Map expect similarly named Attributes in our working set (for example, work.AuthorEditor), you must double-click the Assignment of some of these and correct them to use those fields being read.

Start by double-clicking the Assignment of the "AuthorEditor" output Attribute to open the editor. Now remove the current assignment script and type "work.", then wait for the auto-completion list to appear (Figure 22).

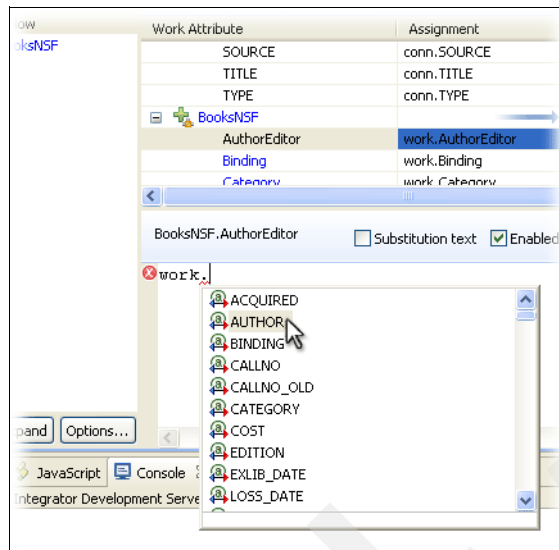


Figure 22 Editing Attribute assignments

Using the procedure outlined above, correct the assignments for the Attributes listed in Table 3.

Table 3 Correct these Attributes in the Lotus NotesNSF Output Map

AuthorEditor	work.AUTHOR
CostOfBook	work.COST
DateAcquired	work.ACQUIRED
DateLost	work.LOSS_DATE
Form	"Book" // JavaScript literal string value
HowAcquired	// Since there is no input value directly for // this, you will calculate one. if (work.NEW_USED == "New") return "Purchased" else return "Donated"
NewUsed	work.NEW_USED
RefNumber	work.CALLNO

Now click **Run** and test your work. This time you should see an error displayed in the Log output (Figure 23).

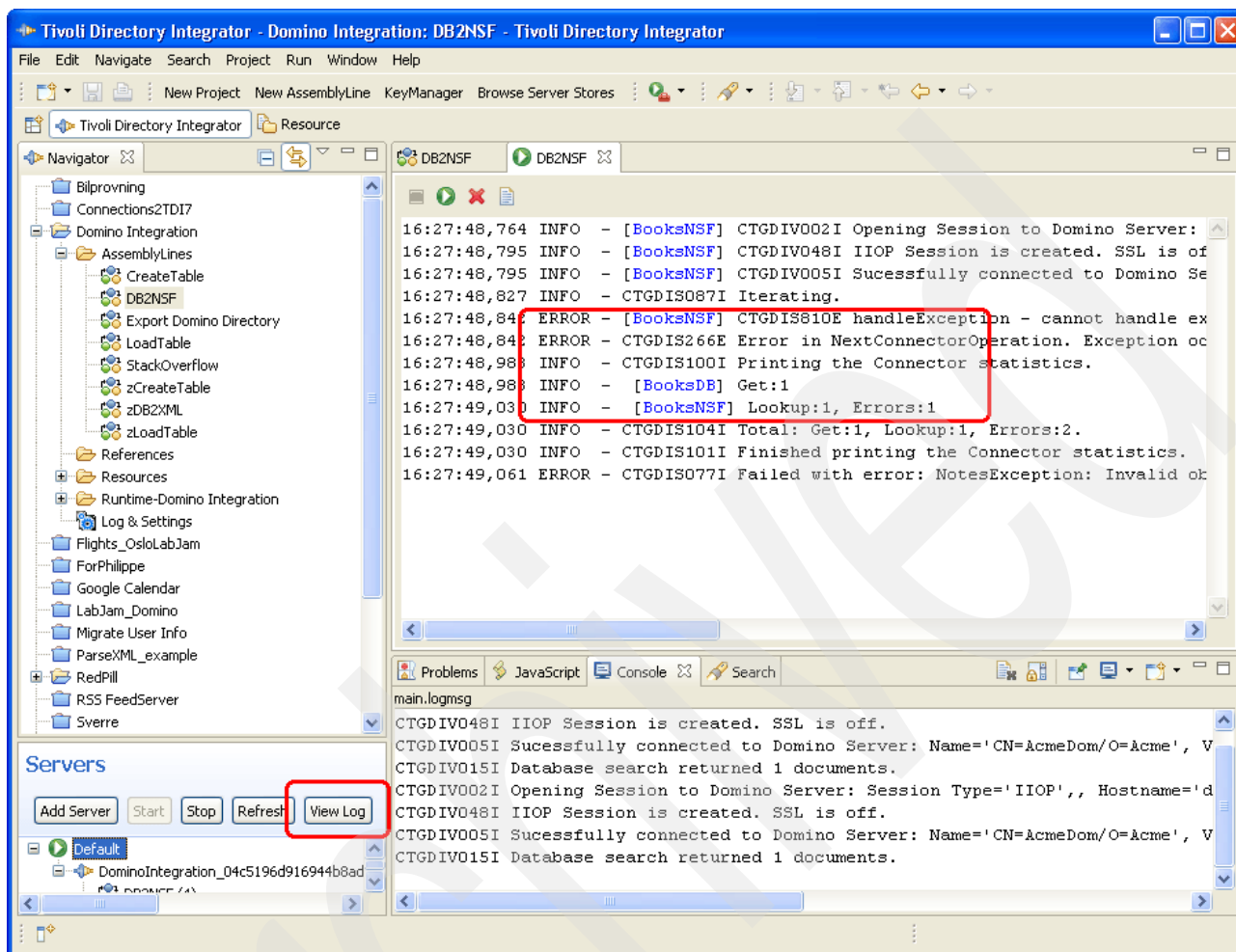


Figure 23 Error when running your AssemblyLine

To get a better idea of the problem, select your test server, "Default", and click **View Log** to get the entire log output. Example 4 displays a snippet from near the end of the log, which is where the results of the most recent run are found.

Example 4 Error log output in the Tivoli Directory Integrator Server log

```
2009-11-17 16:27:48,827 INFO [AssemblyLine.AssemblyLines/DB2NSF.4] - CTGDIS087I Iterating.
2009-11-17 16:27:48,842 ERROR [AssemblyLine.AssemblyLines/DB2NSF.4] - [BooksNSF] CTGDIS810E
handleException - cannot handle exception , add
Lotus NotesException: Invalid object type for method argument
    at lotus.domino.cso.Utils.buildItemValue(Unknown Source)
    at lotus.domino.cso.Document.replaceItemValue(Unknown Source)
    at com.ibm.di.connector.DominoConnector.inner_modDocument(DominoConnector.java:1377)
    at com.ibm.di.connector.DominoConnector.inner_putEntry(DominoConnector.java:866)
    at com.ibm.di.connector.DominoConnector.run(DominoConnector.java:1657)
    at java.lang.Thread.run(Thread.java:810)
```

Just before the stack dump, you will see the name of the component that failed displayed in brackets: [BooksNSF]. This is followed by the error message from Tivoli Directory Integrator

and the operation (add). The next line is the underlying error from the connected system, and here you see that one of the Attributes being written was of the wrong object type.

One way to find which Attribute is in error is to disable them one at a time until the problem goes away. You can also look at the Schema for the Input Map of the "NotesDB" Connector to see what types of Java objects are being used to return the values read. Then compare this with those that you see when you use the Connect and Next buttons to discover data in the Schema part of the Output Map for "NotesNSF". Here you can see that the Connectors are working with different Java classes to hold the same date values (Figure 24).

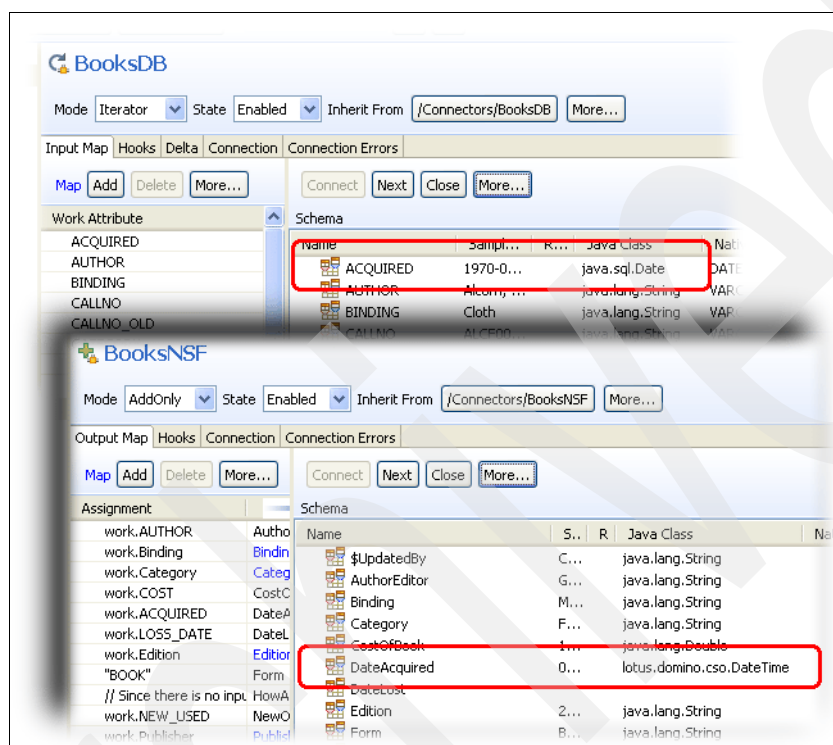


Figure 24 Comparing the Schema of your Connectors

In order to write the value for "ACQUIRED" to Lotus Domino/Notes, you must first create a Java object of type `lotus.domino.cso.DateTime`¹¹. A quick glance at a Java Reference shows that `java.sql.Date` is derived from `java.util.Date`, and the latter provides a `.getTime()` method to retrieve the date value as milliseconds. You can use this to create a new `java.util.Date`, which you then use to make your `lotus.domino.cso.DateTime` object.

¹¹ JavaScript and Java let you pick up where automatic conversion features of many Tivoli Directory Integrator Connectors fall short. For example, you can map out JDBC dates and datetimes as strings and the Connector converts for you to the proper `java.sql.*` types as long as you format the string correctly. Ask Java what this format is using the "JavaScript" view, which you will find in the same tab row as "Console". For example, enter the following snippet into the JavaScript view editor area:

```
d = new java.sql.Date(java.util.Date(103,1,1).getTime())
```

Then run it by selecting it, right-clicking, and choosing **Execute script**, or using the **Run** button at the top-right corner of this tabbed window. To the right a new variable shows up in the "Script Engine Variables" list - d - referencing your new `java.sql.Date` object. Its text representation shows you how default date parsing looks. From there you should be able to decipher the appropriate mask to use with a function like `system.formatDate()`:

```
datestr = system.formatDate(work.dbDate, "YYYY-MM-DD");
```

Do this by replacing the Assignment for the date Attributes, "DateAcquired" and "Date-Lost", using code shown in Example 5.

Example 5 Example script to convert between a DB Date column and the Lotus Notes Date field

```
// return the object used to hold the Attribute value
sqlDate = work.getObject("ACQUIRED");
// if the value returned is null, just
// use this for the Assignment
if (sqlDate == null)
    return sqlDate;
// now create the java.util.Date
javaDate = new java.util.Date(sqlDate.getTime());
// To make the required Domino/Notes Date object you will
// use a similar technique to that applied earlier with the JDBC
// Connector, this time grabbing the Domino Session from the
// Lotus Notes Connector and using this to create our object -
// and you use the javaDate created above to do so.
sess = BooksNSF.getConnector().getDominoSession();
domDate = sess.createDateTime(javaDate);
return domDate;
```

You could have scripted a function to do this and then reused the code instead of typing it out twice for both date-related Attributes (DateAcquired and DateLost). Your scripted function could be in a Script component anywhere in the AL before the "BooksNSF" Connector or it could be in a Prolog Hook.

Regardless, once your conversion logic is in place you can run the AssemblyLine and import all 516 books.

Detecting and syncing changes

What follows is really a continuation of the exercise in the previous section, but we wanted its title to stand out in the table of contents because no job is complete before the data is not only in place, but will continue to be so. Tivoli Directory Integrator has features to let you catch changes in practically any input source, such as files, databases of all shapes and sizes, directories, and so forth. As you have seen in the previous section, access to JavaScript and Java let you reshape the data as necessary. Tivoli Directory Integrator also makes applying changes a snap (once you have squared away connectivity, politics, and security).

Think back to the Entry → Attribute → Value illustration in Figure 2 on page 3. The Entry object is depicted as a bucket that can hold Attributes, which in turn carry the data values themselves. Each level in this data model can also carry information about changes. In fact, the data that you read is always tagged with a delta operation code to indicate its state-of-change. However, as long as your Iterator is not configured to detect changes, your data is tagged with default codes. For an Entry this default delta tag is "generic", while for Attributes it is "replace".

You have three ways of detecting changes (in other words, reading data that is tagged with delta operation codes):

- ▶ Change Detection Connectors, which are components designed for catching changes in a specific system or protocol. Examples are the Lotus Domino Change Detection and the IBM Directory Server Changelog Connectors;
- ▶ Reading and parsing incremental LDIF streams (for example, files) with the LDIF Parser;
- ▶ Turning on the Delta Engine and computing deltas for any data read by a Connector in Iterator mode.

We use this third method to pick up changes in the BooksDB (or XML file) and drive these into the NSF database.

As you would suspect, you can get at these operation codes through JavaScript, as shown in Example 6.

Example 6 Example script to display delta operation codes

```
// return a multi-lined display of data with delta tags
allcodes = work.toDeltaString();
// get just the Entry-level tag
entryTag = work.getOperation();
// get the operation code from an Attribute
if (work.telephoneNumber != null)
    attTag = work.telephoneNumber.getOperation();
// print the operation tags for all values of an Attribute
if (work.telephoneNumber.getOperation() == "modify")
    for (i = 0; i < work.telephoneNumber.size(); i++)
        task.logmsg("#" + i + ": " + work.telephoneNumber.getValueOperation(0));
```

As a quick way to get your *hands dirty* with a little delta detection, create an AssemblyLine named "DeltaTest". Add a Form Entry Connector in Iterator mode to look at its Configuration form (Figure 25).

Figure 25 Configuring the Form Entry Connector

This handy component lets you enter any text that you want as the "Raw Data Text" and then configure a Parser to read it and return Entries. You can even create a perpetual Iterator by selecting the "Infinite Loop" option.

Leave the default "Raw Data Text" in place and click **Next** to choose the Parser. The format of this default raw data is suitable for the "Simple Parser", so select this. Once you click **Finish** you can go to the Input Map of your Connector and use the **Connect** and **Next** buttons over the "Schema" area to read this test data stream. Drag all three Attributes over to the Input Map so that they will be used for delta computation.

To turn the Delta Engine on for this Connector, select its **Delta** tab (Figure 26).

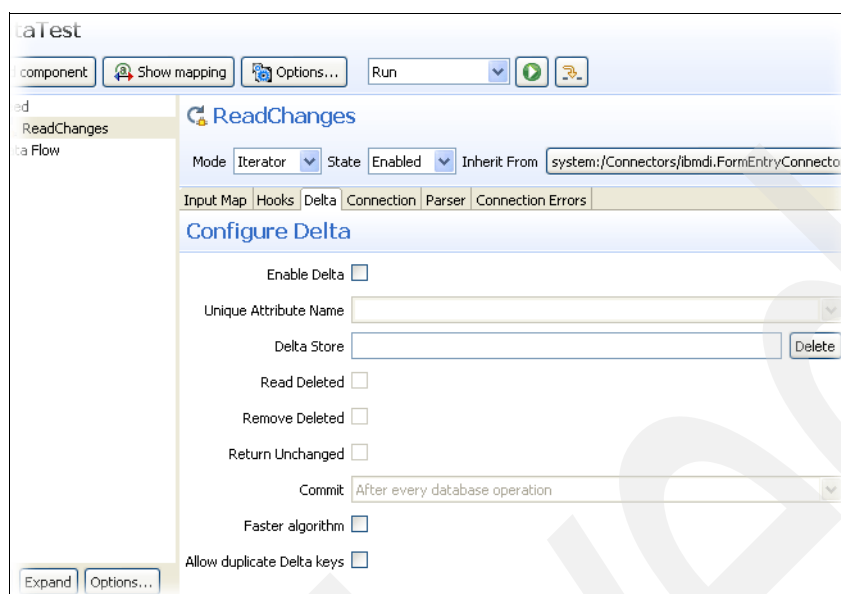


Figure 26 Configuring the Delta Engine

Let us take a closer look at the parameters in Table 4.

Table 4 Delta Engine parameters

Enable Delta	Select this check box to enable delta computation.
Unique Attribute Name	The name of an Attribute in the Input Map that uniquely identifies each Entry. Note that this can be a custom Attribute with a scripted Assignment to compute its value.
Delta Store	Here you enter the name of a table that the Delta Engine creates in the System Store. This is where it keeps snapshots of incoming data.
Read Deleted	Once End-of-Data is reached in the input source, the Delta Engine can check to see whether any entries seen on previous runs were missing from this one. These will be returned with "delete" tags if you select this check box.
Remove Deleted	Select whether you want deleted entries removed from the Snapshot™ table. If you are reading deleted entries using the previous check box then it is common to also remove them from the snapshot db.
Return Unchanged	If you select this option then all entries (even unchanged ones) are returned by the Iterator. Otherwise, unchanged entries are skipped during reading.
Commit	<p>Since the System Store is a database, this parameter lets you control when updates to the Snapshot table are committed by the underlying RDBMS.</p> <p>A recommended setting is "At end of cycle", which means that all your Flow components have finished their work with the changed data.</p>

Faster algorithm	This memory-based feature greatly improves Delta Engine performance and is recommended.
Allow duplicate Delta Keys	This option is used when you are doing configuration delta computation for a "never-ending" input source, like a Change Detection Connector or a transport like MQ/JMS or HTTP. You can skip this option for the exercise.

Set up the Delta Engine for the unique Attribute "id", have it read, then remove deleted entries and use the faster algorithm. You can call this Delta Store "TestSnaps". Now click **Run** and watch as the Delta Engine builds your baseline delta store by adding two new snapshots.

Change the "Commit" parameter to "no autocommit". This way any changes that you make to the input data (which you will do in a moment) will continue to be detected as changes as you play with your test AssemblyLine.

In order to see what information is returned by the Delta Engine, add the "Dump Work Entry" script to the Flow section of your AssemblyLine and then click **Run**. This time two things should happen. First, your Iterator reports that it got two entries and that both were unchanged (Nochange:2). Secondly, the Dump Work Entry script is never used, which means that no Entries were pass on to the Flow section for processing. Nothing more will happen before the input data changes, so do this now.

Bring up the Connection tab of the Form Entry Iterator again and modify the Raw Data Text value. Note that if you change the "id" value for an entry then the Delta Engine will respond by first detecting an "add" of the new "id", then reporting the old "id" as deleted. Do this now by changing the "id" of the first entry from "1" to "3". Then move down to the next entry and modify the value of "first" for "Jane" to "Janey", as shown in Example 7.

Example 7 Changes made to the Raw Data Text of the Form Entry Iterator

```
id:3
first:John
last:Smith
.
id:2
first:Janey
last:Doe
```

When you run your AssemblyLine again you should see that your Iterator returned one "add", one "modify", and a single "delete". Looking further down in the log output you will see the result of the Dump Work Entry script calls. These include the "operation" code of the Entry, along with all the Attributes and their values. The second of these dumps is shown in Example 8.

Example 8 Result of dumpEntry() calls in your AssemblyLine

```
11:11:20,953 INFO - CTGDIS003I *** Start dumping Entry
11:11:20,968 INFO - Operation: modify
11:11:21,031 INFO - Entry attributes:
11:11:21,031 INFO - last:'Doe'
11:11:21,031 INFO - first:'Janey''Jane'
11:11:21,078 INFO - id:'2'
11:11:21,078 INFO - Entry properties:
11:11:21,078 INFO - delta.old:'{
    "last": "Doe",
    "first": "Jane",
    "id": "2"
}'
```

Notice how the "first" Attribute has two values: the old one and the new one¹². However, the dumpEntry() method does not show you the operation codes of the Attribute or its values. To get this you must modify your Dump Work Entry script, changing it to this snippet:

```
task.logmsg(work.toDeltaString());
```

Run your AL again and then scroll down to the second entry displayed in the log. Here the tagging of the "first" Attribute, as well as the new and deleted values, is visible, as shown in Example 9.

Example 9 Delta information shown by the toDeltaString() method

```
...
],
"first": [
    "#type": "modify",
    "#count": 2,
    "#add": "Janey",
    "#delete": "Jane"
],
...
```

If you were to write this delta-tagged Work Entry to an output target using AddOnly mode, the result would be a multi-valued Attribute for "first". Fortunately, Update mode automatically detects value operation codes and adjusts the contents of the write accordingly.

Let us take what you have learned about delta detection and apply it to the DB2NSF AssemblyLine that you built in the previous exercise. Start by right-clicking your DB2NSF AL, copying it, then pasting it back to make a copy. Rename this copy "DB2NSF_Sync" and open it for editing.

¹² You may have noticed that the Work Entry carries a "delta.old" property as well. This is an Entry object representing the old snapshot, and you retrieve it like this:

```
oldEntry = work.getProperty("delta.old");
```

Select your Iterator (**BooksDB**) and enable Delta as you did for the test AL, configuring it to both return and remove deletes. Be sure to give it a new Delta Store name so that it creates a new table for this input data. Set the Commit drop-down to **On end of cycle**. The Input Map Attribute named "CALLNO" is the unique identifier for each entry, so use this in the Unique Attribute Name parameter, shown in Figure 27.

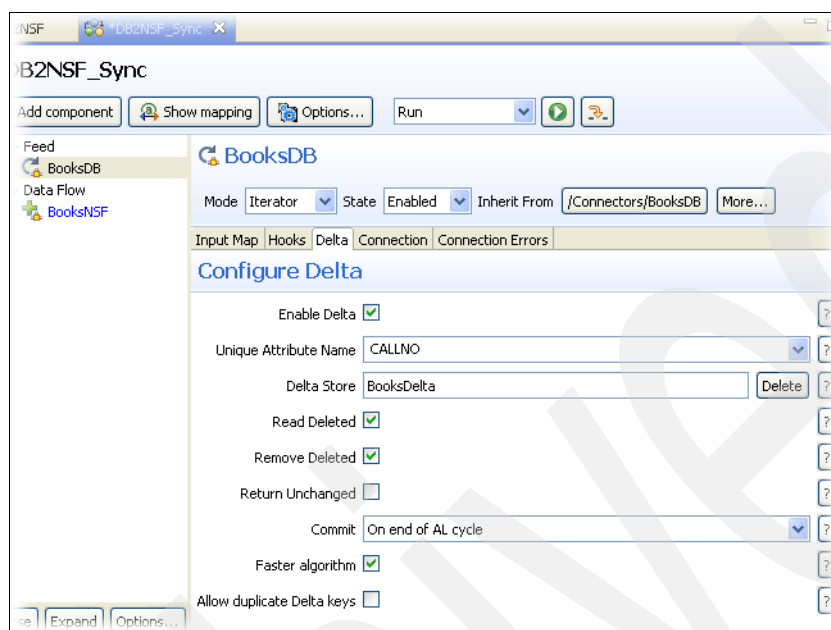


Figure 27 Delta Engine settings for BooksDB

Since your target NSF already contains these entries, instruct your AL to skip writing them again. Do this by right-clicking the BooksNSF Connector and disabling it. Then run this AssemblyLine to build the baseline delta that will contain 516 snapshots. You can now reenable BooksNSF.

At this point your Iterator is ready to start detecting changes. However, your output Connector is not ready to receive them yet. To fix this, change the mode setting of the BooksNSF Connector from AddOnly to Update. A new tab appears for the Connector (Link Criteria) where you define how data in the connected system is matched with information in your AssemblyLine. Add a Link Criteria and use the drop-downs to configure:

RefNumber equals \$CALLNO

This instructs the Connector to search for a RefNumber value in the Lotus Notes NSF that matches the value of the Work Entry Attribute named "CALLNO", as shown in Figure 28.

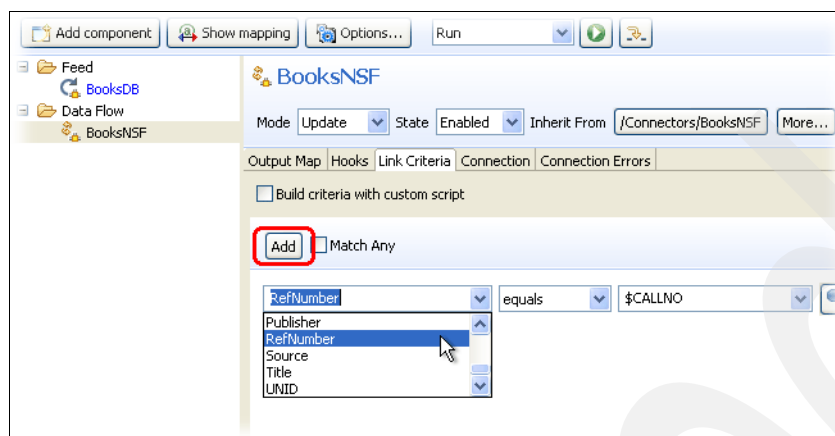


Figure 28 Configuring Link Criteria

Update mode first searches for an entry by using the Link Criteria specified. If a single match is found then a modify operation is performed. If no match is found, then the Output Map data is used to write a new record in the connected system¹³. In this way, Update mode is all that you need to sync "add" and "modify" changes to your target.

You may have noticed that when you changed the Connector mode to Update that your Output Map grew a couple of additional columns. These are for enabling and disabling Attributes for either add or modify operations. For example, it does not make sense to modify the unique identifier RefNumber, so you should set this to false by clicking that column (Figure 29).

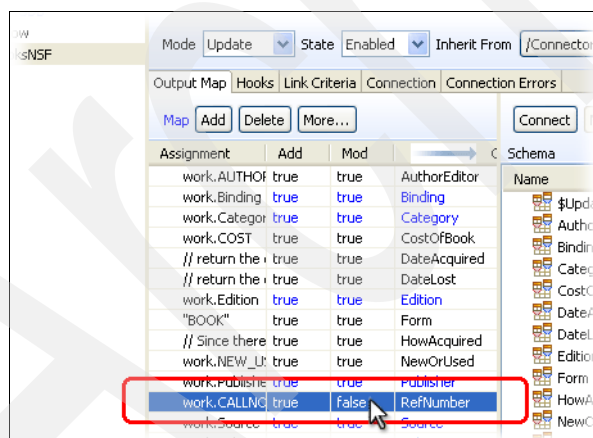


Figure 29 Disable RefNumber in Output Map for modify

Do the same for the "Form" Attribute.

Your AssemblyLine is now ready to synchronize new and modified books. To also deal with deletes you will need a second copy of your BooksNSF Connector. This one you can name "DeleteFromNSF" and set to Delete mode. Set up the same Link Criteria as for your Update Connector.

¹³ Note that if multiple items are found that match the Link Criteria then this is considered an exceptional situation and you must deal with this by scripting the On No Match Hook of the Connector. You can find an example script in the sidebar of the Flow Diagrams for Lookup, Update, and Delete modes:
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/TDI_7.0_FlowDiagrams.pdf

To keep every entry from first going to your Update Connector and then on to Delete, you must add IF branches to control how data flows in your AL. Call the IF Branch "not deleted" and then in the editor click **Script** to allow you to enter a scripted condition, as shown in Figure 30.

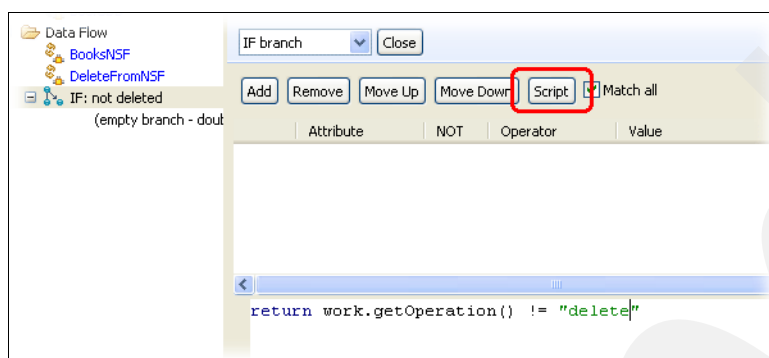


Figure 30 Scripting a Branch Condition

The script to use is this:

```
return work.getOperation() != "delete"
```

Drag your Update Connector on top of the IF Branch so that it appears beneath it in the component list. Now this component will only be used if the Work Entry is tagged as "add" or "modify" (that is, not tagged as "delete").

To complete the branching logic, add an ELSE Branch to the AssemblyLine and call it "delete", making sure that it comes after the IF. Drag the Delete mode Connector onto it. Your AL should now look like Figure 31.

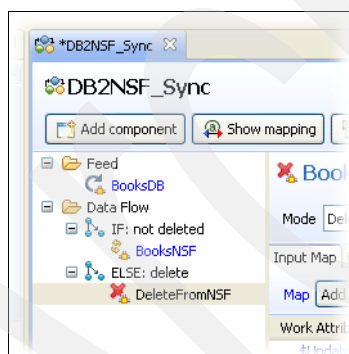


Figure 31 Completed DB2NSF_Sync AL

You have created a synchronization AssemblyLine. All you must now is change your input data, run the AL, and see how data is updated, which is an exercise left to you.

You can use the same approach with the Lotus Domino Change Detection Connector, directory changelog Connectors, and any of the other delta detection features in Tivoli Directory Integrator.

Troubleshooting

In this section we include some typical errors that you may experience along with suggested steps to alleviate these. The last section of the *IBM Tivoli Directory Integrator Getting Started*

Guide Version 7.0, GI11-8185, also provides troubleshooting tips and techniques for hardening your solution so that it can better deal with errors.

In general, the error messages displayed in the Tivoli Directory Integrator log output are over two lines. First you get the coded Tivoli Directory Integrator error message along with the component (shown in brackets) and the operation that failed. Next is the underlying error message, often originating from the API, driver, or service to which you are connecting. Sometimes you can get more information by selecting the Tivoli Directory Integrator Server (for example, "Default") and choosing **View log**. You must scroll to the end of the logfile to see the most recent messages.

Note also that the errors listed in this chapter are not the complete texts displayed by Tivoli Directory Integrator, so you must examine your log output (either in the CE or in the Server or AssemblyLine logfile) to find a match with those shown here. Some of the errors that you will see will be more-or-less self-explanatory, such as `invalid username/password` or `license not found`. But if you cannot figure out what the message text signifies and cannot find your specific message here, then Google it or post a question to the Tivoli Directory Integrator forum, or do both.

Some common error messages

In this section we look at some common error messages.

java.lang.NoClassDefFoundError: lotus.domino.NotesException

This is because you have not made the Lotus Notes API jar-file available to Tivoli Directory Integrator.

Note that for Lotus Domino versions prior to 8 this error can indicate that you have made the wrong jar-file available. Use the `ncso.jar` from the Lotus Domino server install for DIIOP connections and the `Notes.jar` file from the Lotus Notes Client install for LocalClient access.

See "Setting up Lotus Domino" on page 6 for instructions on how to do this.

NotesException: Database %1 has not been opened yet

The %1 in the above message may contain the name of the NSF that you have configured in your Lotus Notes Connector. Regardless, it means that the connection has not been successfully opened. Make sure that the Lotus Domino server is up and that the necessary tasks are running. This can also be due to a problem with the credentials that you are using in the Connector, for example, that the ACLs of the NSF do not permit the configured user ID to read the database. It could also mean that either you misspelled the name of the NSF or you are looking for it on the Lotus Domino server while the file is local, or vice versa.

Could not get IOR from Lotus Domino Server:<host:port>diop_ior.txt

Here you are trying to connect to DIIOP and either the HTTP task is not running on the Lotus Domino server or the `diop_ior.txt` file is missing. Perhaps DIIOP has not been configured.

java.lang.Exception: Could not open Notes session

Most likely the DIIOP task is not running on your Lotus Notes server. If it is, then this could indicate a problem with the credentials that you are using to connect or a firewall blocking the DIIOP port.

No license was found

This message is returned by DB2 if you have not made the license JAR file available to Tivoli Directory Integrator as described in “Preparing Tivoli Directory Integrator for your RDBMS” on page 15. The error text from other RDBMSs will be a variation on this theme.

com.ibm.db2.jcc.b.SqlException: DB2 SQL error: SQLCODE: -803,...

SQLCODEs are errors coming from the RDBMS. There are plenty of sites on the Internet where you can decipher the meaning of these codes, such as this one:

<http://www.theamericanprogrammer.com/programming/sqlcodes.shtml>

Here you can see that the -803 means that the write operation failed because you are writing a duplicate value. You can either choose to ignore this error, for example, by scripting the Default On Error Hook of your output Connector, or switch it to Update mode and modify existing records instead.

In the case of the LoadTable exercise, it means that your table already has the Book information loaded, so you can either recreate it again or use Update mode to write changes.

The team who wrote this IBM Redpaper

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

Axel Buecker is a Certified Consulting Software IT Specialist at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on areas of Software Security Architecture and Network Computing Technologies. He holds a degree in Computer Science from the University of Bremen, Germany. He has 23 years of experience in a variety of areas related to Workstation and Systems Management, Network Computing, and e-business Solutions. Before joining the ITSO in March 2000, Axel worked for IBM in Germany as a Senior IT Specialist in Software Security Architecture.

Eddie Hartman is the Tivoli Directory Integrator Storyteller and he is responsible for community outreach and enablement. An ex-pat Texan who now lives in Norway and works in the Oslo lab, Eddie has been involved with Tivoli Directory Integrator for over a decade and has the pleasure and honor of working with the unparalleled development, release, and support teams in their efforts to improve time-to-value for Tivoli Directory Integrator users.

Thanks to the following person for his contributions to this project:

Paul Bemelmans, Johan Varno
IBM

Notices

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

This document REDP-4629-00 was created or updated on December 31, 2009.



Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400 U.S.A.




Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

DB2®
Domino®
IBM®
Lotus Notes®

Lotus®
Notes®
PartnerWorld®
Passport Advantage®

Redpaper™
Redbooks (logo) ®
Sametime®
Tivoli®

The following terms are trademarks of other companies:

Snapshot, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

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

Microsoft, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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