Talend Open Studio for the TransLattice Enterprise Database (TED)

David Lambert
March 18, 2013

Introduction

ETL (an acronym for Extract, Transform, and Load) software is used to move data from one location/format to another location/format, often between different types of databases. Because Talend Open Studio is a major open-source ETL tool, TransLattice has chosen to help its customers use Talend with the TransLattice Enterprise Database (TED).

The purpose of this document is to describe how one might use Talend Open Studio with TED. It is assumed here that the reader has little knowledge of Talend, just in case that is true.

TransLattice's Package of Sample Talend Objects

To help TransLattice customers use Talend with TED, TransLattice has created:
1) Talend_TED.zip, a package of sample Talend objects for importing data in one sample flat file into one TED table.
2) The sample flat file, named customer.csv.
3) A slightly customized version of "tPostgresqlConnection".

Importing the Package

To use this package, it must be imported into Talend as follows:
1) Locate the Repository pane, which is in the upper left portion of the Talend window.
2) Right-click "Job Designs", and select "Import items". This brings up the "Import items" dialog.
3) At the top of this dialog, change the selection from "Select root directory" to "Select archive file".
4) To the right of the label "Select archive file" is a text box and a button labeled "Browse". Click this button, navigate to the location of Talend_TED.zip, and click the "Open" button.
5) In the area below the "Browse" button, there will appear a tree structure. Click the button "Select All" located to the right of the tree structure.
6) At the bottom of the "Import items dialog", click the "Finish" button.

Overview of the Package

Once this package has been imported, its contents will be listed in the Repository pane. The contents are summarized in Table 1 below, on this page and the next page.

1) Job Designs
   a) Import_File_Batched_Inserts_PG
   b) Import_File_Batched_Inserts_TED
   c) Import_File_Unbatched_Inserts_TED
   d) Print_Input_File
   e) Table_Clear_PG
   f) Table_Clear_TED
   g) Table_Create_PG
   h) Table_Create_TED
   i) Table_Exists_PG
   j) Table_Exists_TED
2) Contexts
   a) "All_Parameters" context group
      i) "CSV_Path" context
3) Metadata
   a) DB Connections
      i) PG
         (1) Queries
            (a) "Clear_Table_PG"
            (b) "Count_Imported_Records_PG"
            (c) "Count_Records_PG"
            (d) "Table_Cleared_PG?"
            (e) "Table_Exists_PG"
         (2) Table Schemas
            (a) "customercsv"
      ii) TED
         (1) Queries
            (a) "Clear_Table_TED"
            (b) "Count_Imported_Records_TED"
            (c) "Count_Records_TED"
            (d) "Table_Cleared_TED?"
            (e) "Table_Exists_TED"
            (f) "TED_Database?"
         (2) Table Schemas
            (a) "customercsv"
   b) File delimited
      i) "Read_From_File" metadata file
         (1) "Read_From_File" metadata file schema
   c) Generic schemas
      i) "GetBoolean" metadata file
         (1) "metadata_boolean" metadata file schema
      ii) "GetInteger" metadata file
         (1) "metadata_integer" metadata file schema
      iii) "GetString" metadata file
         (1) "metadata_string" metadata file schema

Table 1: the Sample Talend Objects.

The Data in the Flat File

The flat file customer.csv has 5000 records in it, preceded by 5 lines of comments and 1 line of field names. The field separator is a semicolon. The format of this file is described under element 3.b.i.1 in Table 1 above, and also shown in Table 2 below. To view the file format in Talend, find the element named "Read_From_File" in the Repository pane, and double-click it.

Field Name	Data Type	Field Length
idFile	Sequential Integer, 1-5000	Up to 4 digits
CustomerNameFile	Text	Up to 30 characters
CustomerAddressFile	Text	Up to 27 characters
idStateFile	Integer	Up to 2 digits
Id2File	Integer	Up to 4 digits
RegisterTimeFile	Date-Time: "yyyy-MM-dd HH:mm:ss.SSS"	23 characters
Sum1File	Integer	Up to 5 digits
Sum2File	Double-precision float	Up to 9 digits, 5 to the left of the decimal, 4 to the right.

Table 2: the Format of customer.csv.

Several of the jobs listed in Table 1 under "Job Designs" import customer.csv into the database table customercsv. This table's schema is described under elements 3.a.i.2.a and 3.a.ii.2.a (the two versions are the same, and the corresponding create table command shown in Table 3 below. To view the schema of customcsv, double-click one of these two elements.

```
CREATE TABLE customercsv (
    id integer NOT NULL,
    customername character varying(30) NOT NULL,
    customeraddress character varying(27) NOT NULL,
    idstate integer NOT NULL,
    id2 integer NOT NULL,
    registertime timestamp(6) without time zone NOT NULL,
    sum1 real NOT NULL,
    sum2 real NOT NULL,
    CONSTRAINT customercsv_pkey PRIMARY KEY (id)
)
WITH (
    OIDS=FALSE
);
```

Table 3: the Schema of Table customercsv.

A Simple Import Job: Import_File_Batched_Insert_PG

This import job is the simplest one, so we'll examine this job first. The icon for this job can be found in the Repository pane, as shown in Table 1 above, element 1.a. Double-clicking this icon opens up the representation of this job in Talend's Design pane, shown in Figure 1 below.

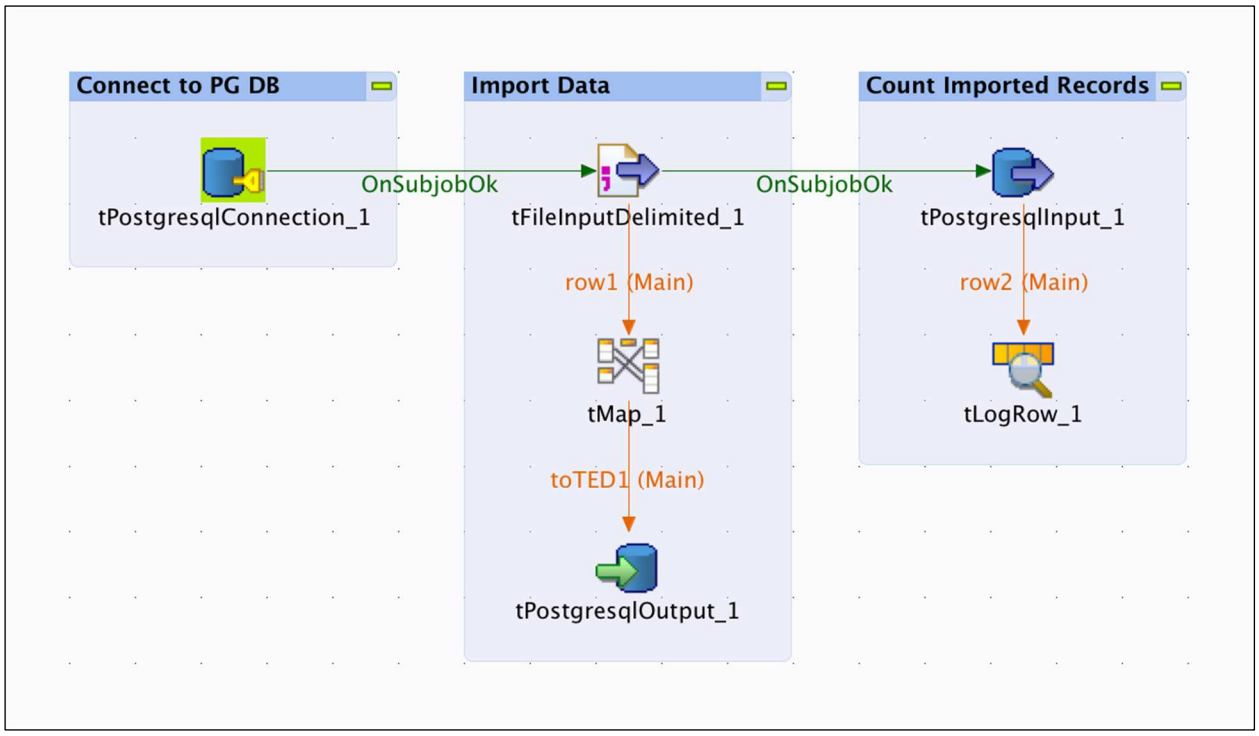

Figure 1: the Design pane view of Job "Import_File_Batched_Inserts_PG".

In Talend, a "job" is simply a sequence of actions that the Talend user wants to perform together. Each job is a collection of a number of entities. Each lavender rectangle with a blue title bar at top is called a "subjob". Each "subjob" consists of a set of "components" (each represented by an icon) connected by a "data flow", which is represented by a red arrow. Each "data flow" represents the flow of data from one component to another, in the direction of the arrow. The names of the components, in black text, must be unique within a job. The names of the data flows, in red text, must also be unique within a job.

The green arrows represent "control flows", which connect the subjobs together. A control flow simply waits for a subjob or a component to finish executing, and then starts the next subjob. The direction of the arrow shows the order in which subjobs and components execute: the component or subjob that executes first is at the tail of the arrow, and the next component or subjob to execute is at the head of the arrow. The green text next to each control flow is the type of control flow, not its name. "OnSubjobOK" indicates that the control flow waits for a subjob to finish executing before the next subjob starts executing. "OnComponentOK", which is not present in this job, indicates that the control flow waits for a component to finish executing before the next component starts executing.

When we've said "component" above, we really mean an *instance* of a component. When looking at a job in the Design pane, the Component pane will be visible to the right, by default. The Component pane lists all the components that can be used, and a component is used to create an instance of it in the Design pane. There can be more than one instance of a component in the Design pane. In Talend literature, components and component instances are both called components.

A component instance's name is the component's name followed by an underscore ("_") and then a sequential instance number. Thus, each component instance in a job has a unique name.

In this job, the first component instance to execute is named “tPostgresqlConnection_1”. This component instance creates a connection to a Postgresql database. This database may or may not be a TransLattice Elastic Database. This connection is used by the other components in the job. When tPostgresqlConnection_1 finishes executing, the subjob named “Import Data” starts executing. “tInputFileDelimited_1” opens a flat file and starts reading records from it. The records are processed through “tMap_1”, and “tPostgresqlOutput_1” inserts them into the database opened by “tPostgresqlConnection_1”. In this job, the flat file is customer.csv, and the records are inserted into table customercsv (customer.csv and customercsv were both described above) in batches of many records per insert, rather than one record per insert.

Once the entire file has been read and its records inserted into table customercsv, the subjob titled “Count Imported Records” starts executing. “tPostgresqlInput_1” executes the query "select count(*)||' imported records' from customercsv;" and the output displayed by “tLogRow_1” in the Run window below the Design pane, as shown in Figure 2 below.

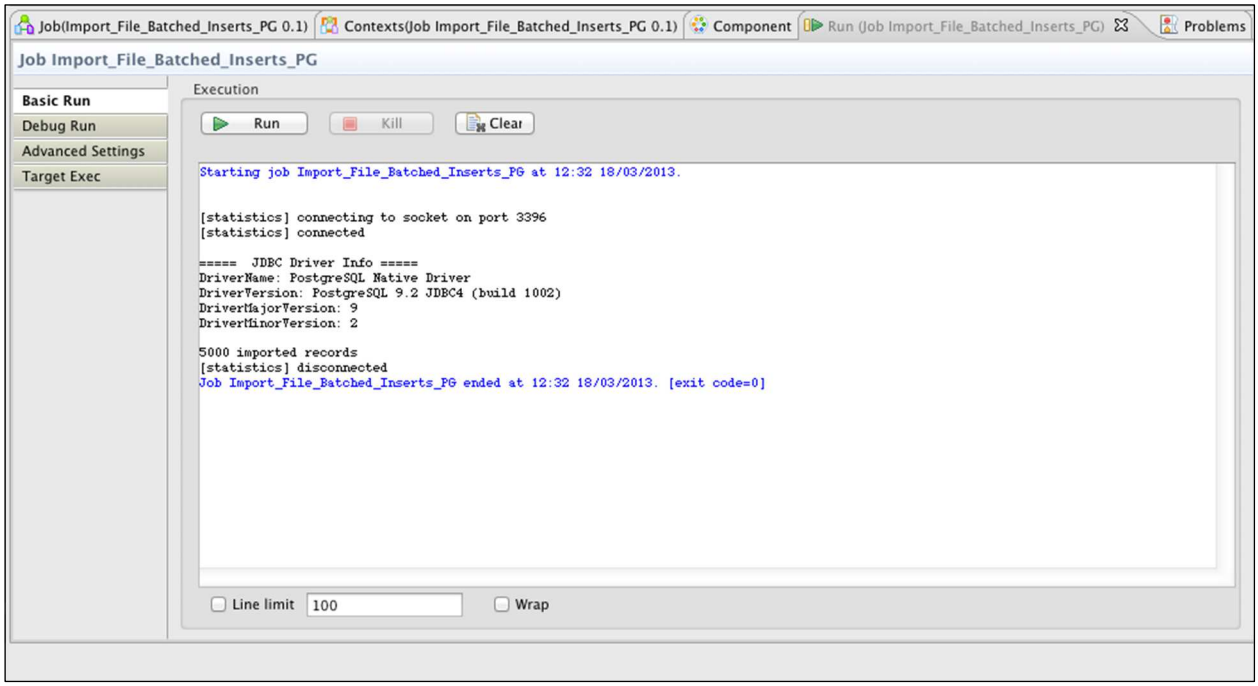

Figure 2: the Output of the Job “Import_File_Batched_Inserts_PG”.

At the top of Figure 2, the “Run” tab is greyed out, meaning that tab is selected but Talend does not have focus. When Talend does have focus, the tab turns blue and the text “Run” becomes bold. The output of a job always goes to the “Run” tab. If another tab is selected and a job is executed, the Run tab will be selected. To execute a job, first open it up in the Design view by double-clicking the name of the job in the Repository pane at upper left. Just below the job’s Design view will appear the output pane, select the Run tab, and click the button with the green triangle and the label “Run” at upper left.

The output shown in Figure 2 contains two sections worth describing. The first section is the set of five lines starting with “===== JDBC Driver Info =====”. They are printed by TransLattice-customized code in component “tPostgresConnection”, which is inherited by the component instance

"tPostgresConnection_1". The output in this first section shows the version of the Postgresql JDBC version being used by this installation of Talend. This version of Talend, 5.2.1, is shipped with Postgresql JDBC version 8.3.[1]

The second section worth describing in the output in Figure is the line "5000 imported records", which is printed out by the component instance "tPostgresqlInput_1". This output is expected, since the file customer.csv contains 5,000 records.

There are two assumptions made by the jobs Import_File_Batched_Insert_PG (and Import_File_Batched_Insert_TED which will be described later):

1) The table customercsv already exists.
2) The table customercsv contains zero records.

If table customercsv does not exist, it can be created in a non-TED database by the job Table_Create_PG, and in a TED database by Table_Create_TED.

If table customercsv exists and contains records, they can be deleted (not truncated) from a non-TED database by the job Table_Clear_PG, and in a TED database by Table_Clear_TED.

If it's not known if the table customercsv exists, its existence can be tested in a non-TED database by the job Table_Exists_PG, and in a TED database by Table_Exists_TED.

The jobs Table_Create_PG, Table_Create_TED, Table_Clear_PG, Table_Clear_TED, Table_Exists_PG, and Table_Exists_TED will be discussed below, in that order.

The Job Table_Create_PG

The job Table_Create_PG is also in the Repository pane. Its purpose is to create the table customercsv. In Table 1 above, this job is element 1.g. Double-clicking its icon opens it up a Design pane view of this job, along with typical output shown in the Run tab, which is below the Design pane.

[1] Currently, upgrading JDBC has no benefit, but for those wishing to upgrade:

1) Download your desired jar file from http://jdbc.postgresql.org/download.html, and rename it postgresql-8.3-603.jdbc3.jar.
2) Search for all copies of postgresql-8.3-603.jdbc3.jar in the folder tree created by installing Talend and rename them postgresql-8.3-603.jdbc3.jar.backup.
3) In each folder found in step 2 that contains postgresql-8.3-603.jdbc3.jar.backup, put a copy of the file postgresql-8.3-603.jdbc3.jar created in step 1 above.

According to Talend Forum entry www.talendforge.org/forum/viewtopic.php?id=27376, Talend uses out-of-date JDBC drivers in many places.

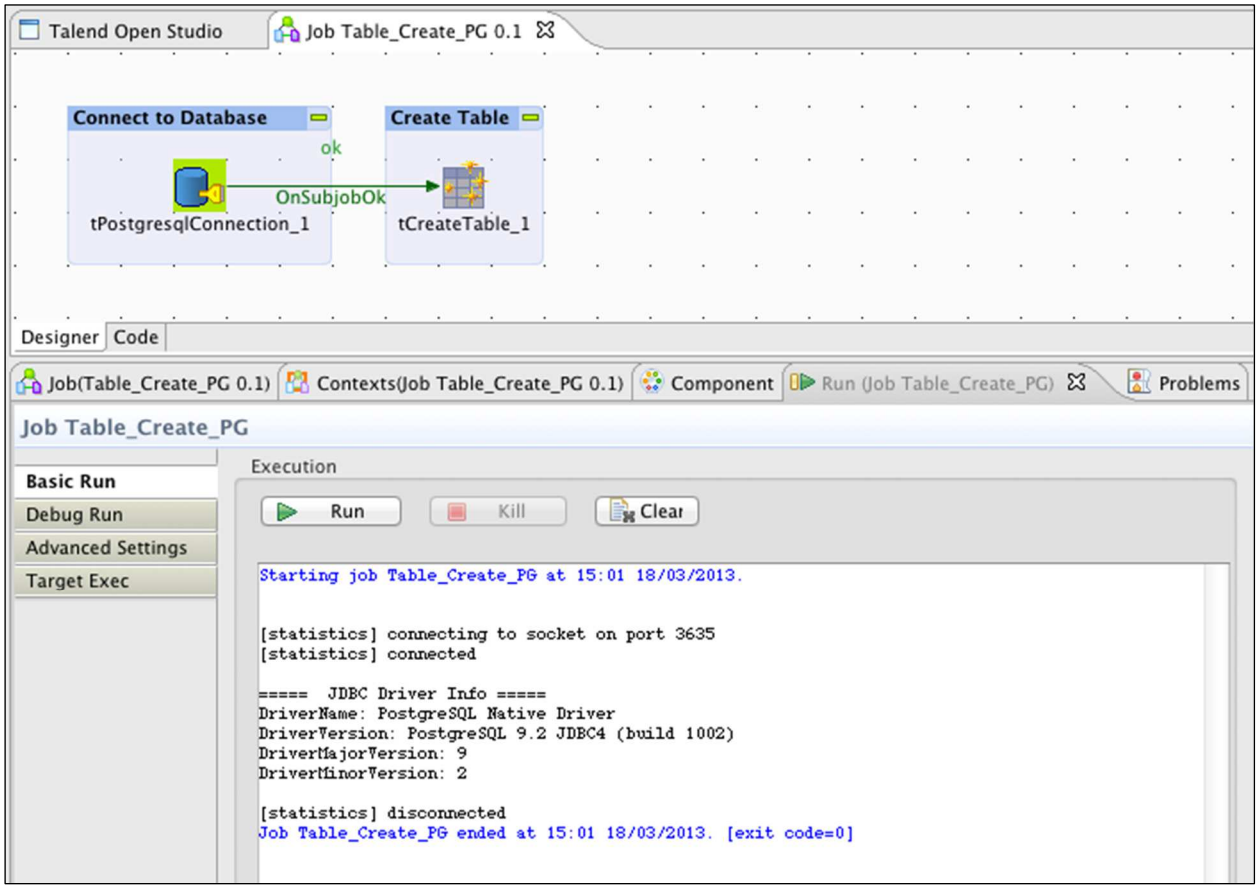

Figure 3: the Design pane view of the Job "Table_Create_PG".

This job has two subjobs, each with one component instance. To see how tPostgresqlConnection_1 is configured, select the database icon just above the text "tPostgresqlConnection_1". The result of selecting the icon is shown in Figure 4 below.

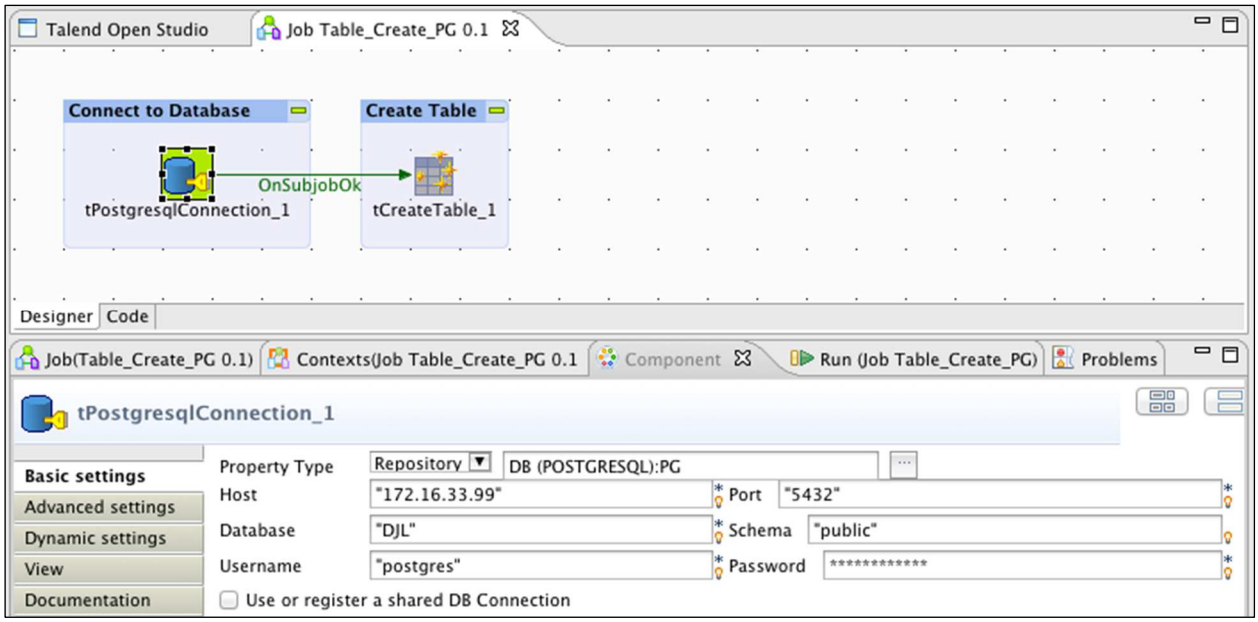

Figure 4: in the Job Table_Create_PG,
the Basic Settings of tPostgresqlConnection_1

The component instance tPostgresqlConnection_1 now has a black square around it, and the "Component" tab is now selected. Notice that on the left end of this tab, there are 5 buttons, with the top-most one labeled "Basic settings" selected. The basic settings tPostgresqlConnection_1 uses to create a database connection are displayed. Most component instances have Basic Settings.

The top line displayed starts with the label "Property Type". To the immediate right of this label is a drop-down selection that is currently set to "Repository", but could also be set to "Built-In". When Built-In is selected, all of the text boxes below it become editable and their contents can be changed.

On the other hand, when Repository is selected, all of the text boxes are filled in for you. The values placed in the text boxes are stored in the Repository pane shown in Table 1, in element 3.a.i, which is labeled "PG", short for Postgresql. In the Repository pane, select the label "PG", right-click it, and select "Edit Connection". This will bring up a window labeled "Update Database Connect – Step 1/2". At the bottom of that window, click the button labeled "Next >". This will dismiss the "…Step 1/2" window and bring up a window titled "Update Database Connection - Step 2/2", which holds all the values displayed at the bottom of Figure 4. Note that the top-most control on this window allows you to select the type of database. If you press the button in the center of this window labeled "Check", Talend will attempt to connect to the database specified in that window.

Data of many types is kept in the Repository: database connection settings, table schemas, SQL statements, file format specifications (shown in Table 1 in element 3.b, "File Delimited"), custom data types (element 3.c, "Generic schemas"), and parameters (element 2, "Contexts"). Data in a single Repository element can be used by any number of jobs, so that data duplication is eliminated. Data can be updated in one element in the Repository, and updates will be instantly visible in the jobs that use that element (you may need to reopen affected jobs, and you may get dialogs asking

whether to continue to use the updated element in the Repository, or to save the previous element's data in that job as Built-In data (data stored in the job, not in the Repository).

In addition to Basic Settings, all component instances also have "Advanced Settings", as shown in Figure 5 below.

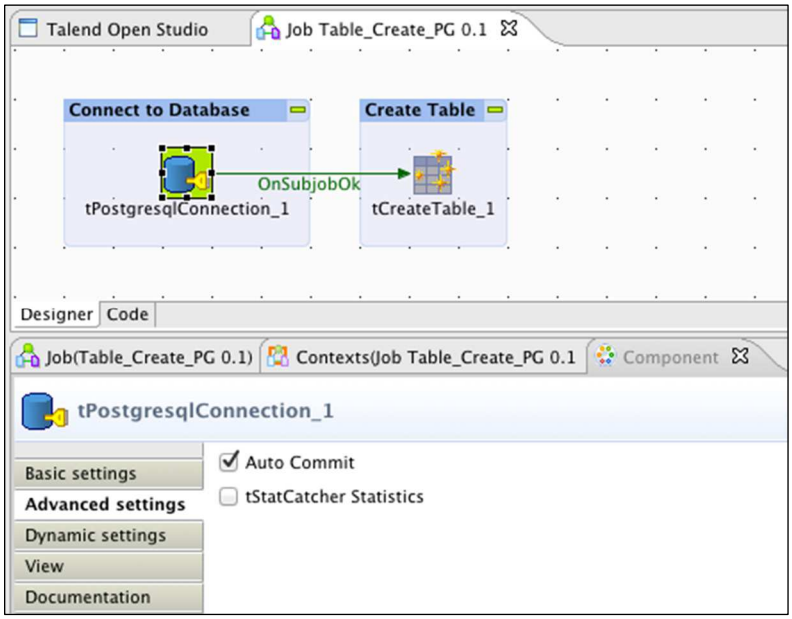

Figure 5: in the Job Table_Create_PG,
the Advanced Settings of tPostgresqlConnection_1

For each instance of the component tPostgresqlConnection, the check box "Auto Commit" determines whether the database connection made by that instance has auto-commit set to ON or OFF. When auto-commit is ON, there are no SQL transactions containing multiple SQL statements, and each SQL insert, update, or delete is committed after it is executed. TED databases should always have auto-commit set to ON. To make apples-to-apples comparisons between TED and non-TED ("PG") connections, in every job in this sample, the Auto Commit check box is checked.

The Job Table_Create_TED

The job Table_Create_TED is basically a clone of job Table_Create_PG, except that it creates the table customercsv in a TED database. This job is element 1.h in Table 1. When you click on this job's version of tPostgresqlConnection_1, as shown in Table_Create_PG's Figure 4, you will see the database connections for the TED database located in element 3.a.ii of Table 1. This ability to clone a job and easily change the clone's database connection settings shows the wisdom of storing as much information as possible in the Repository.

Two jobs Table_Create_TED and Table_Create_PG were created, each using a different database connection, so that it is possible to make comparisons between a TED database and a non-TED database in parallel. If only one job were created, every time a comparison between TED and non-TED is made, the database connection would have to be changed in this one job.

The Job Table_Clear_PG

As described above on page 6, the job Table_Clear_PG deletes (not truncates) all records from customercsv in a non-TED database. In Table 1 above, this job is element 1.e. As usual, double-clicking its icon opens it up a Design pane view of this job. When the component instance tPostgresqlRow_2 is clicked, the result is Figure 6.

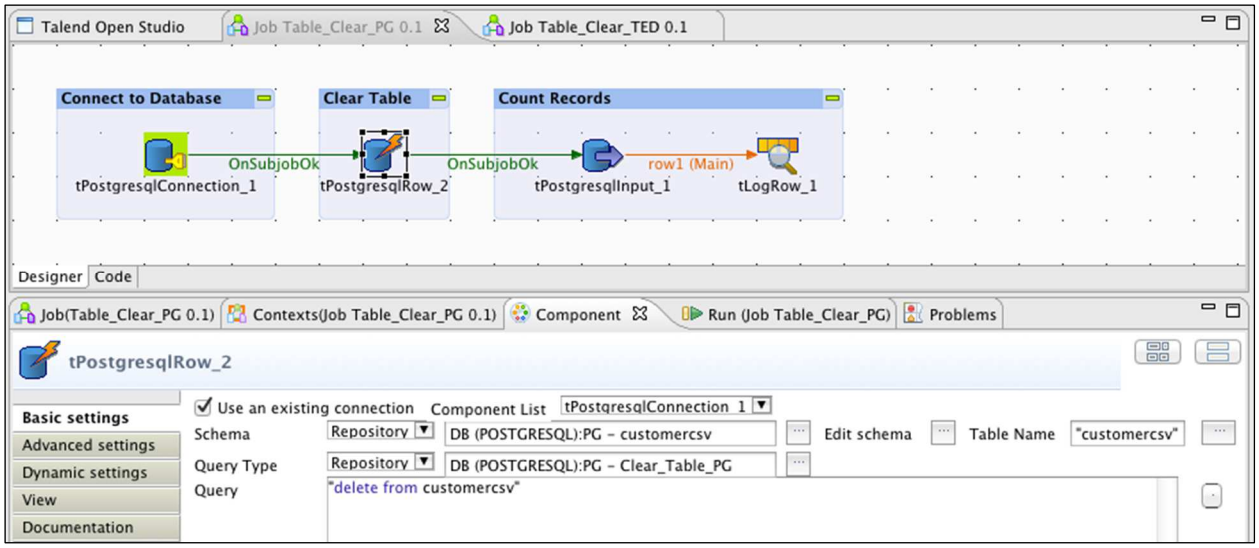

Figure 6: in the Job Table_Clear_PG,
the Basic Settings of tPostgresqlRow_2

Note that to the right of the "Query Type" label, note that the first drop-down is "Repository", meaning that the DELETE statement shown in the "Query" text box was not typed in there, it is stored in the Repository. In this case, the Repository element is named "Clear_Table_PG", which is element 3.a.i.1.a in Table 1 above.

In general, the component tPostgresqlRow is used to execute SQL commands that only return success codes, not records like SELECT does, and not parameters like SHOW does.

Below in Figure 7 is the component pane for the component element tPostgresqlInput_1.

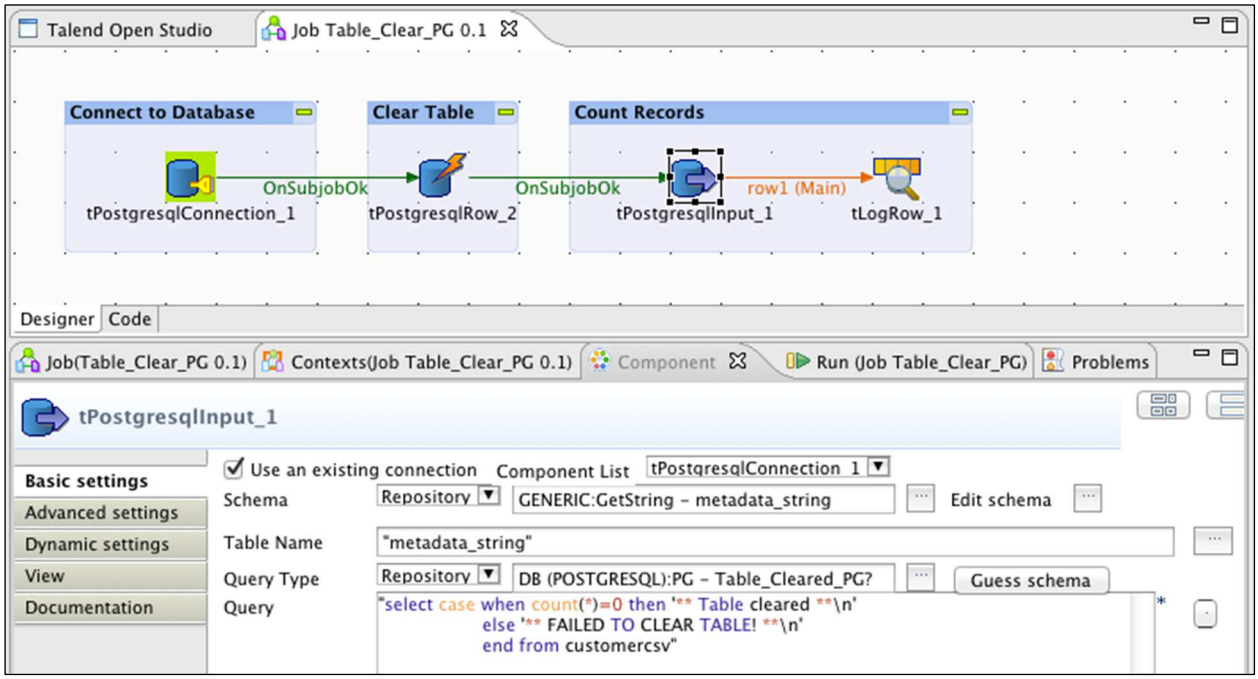

Figure 7: in the Job Table_Clear_PG,
the Basic Settings of tPostgresqlInput_1

To the right of the "Query Type" label, the first drop-down is set to Repository, and the second one is set to "Table_Cleared_PG?", element 3.a.i.2.c , which contains the rather long SQL statement in the Query field. In the upper left corner is a check box labeled "Use an existing connection", meaning this component instance uses the database connection made by tPostgresqlConnection_1.

In general, the component tPostgresqlInput is used to execute SQL commands that return more than success codes, such as records from SELECT or parameters from SHOW.

The Job Table_Clear_TED

The job Table_Clear_TED, is element 1.f in Table 1 above. It is the same as job Table_Clear_PG above, except that a TED database connection is used, rather than a non-TED database connection.

The Job Table_Exists_PG

As described above on page 6, the job Table_Exists_PG tests whether or not the table customercsv exists in a non-TED database. This job is element 1.i in Table 1 above. Figure 8 below shows what this job looks like in the Design pane when its icon in the Repository is double-clicked, and then component instance tPostgresqlInput_1 selected.

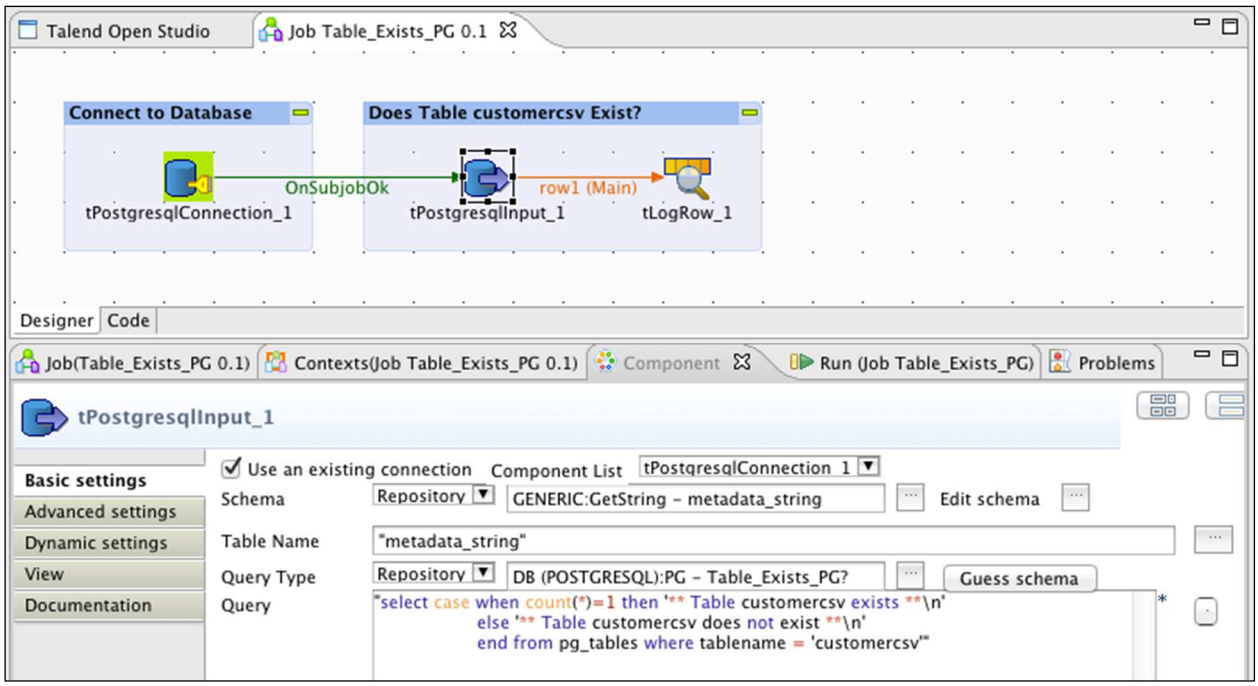

Figure 8: in the Job Table_Exists_PG,
the Basic Settings of tPostgresqlInput_1

In the Component tab is the query used by Table_Exists_PG to show whether or not table customercsv exists.

The Job Table_Exists_TED

The job Table_Exists_TED is the same as Table_Exists_PG, except that the connection is made to a TED database.

The Job Print_Input_File

The job Print_Input_File does not involve any database at all. It's function is to print the contents of customer.csv. The Design pane view is below in Figure 9.

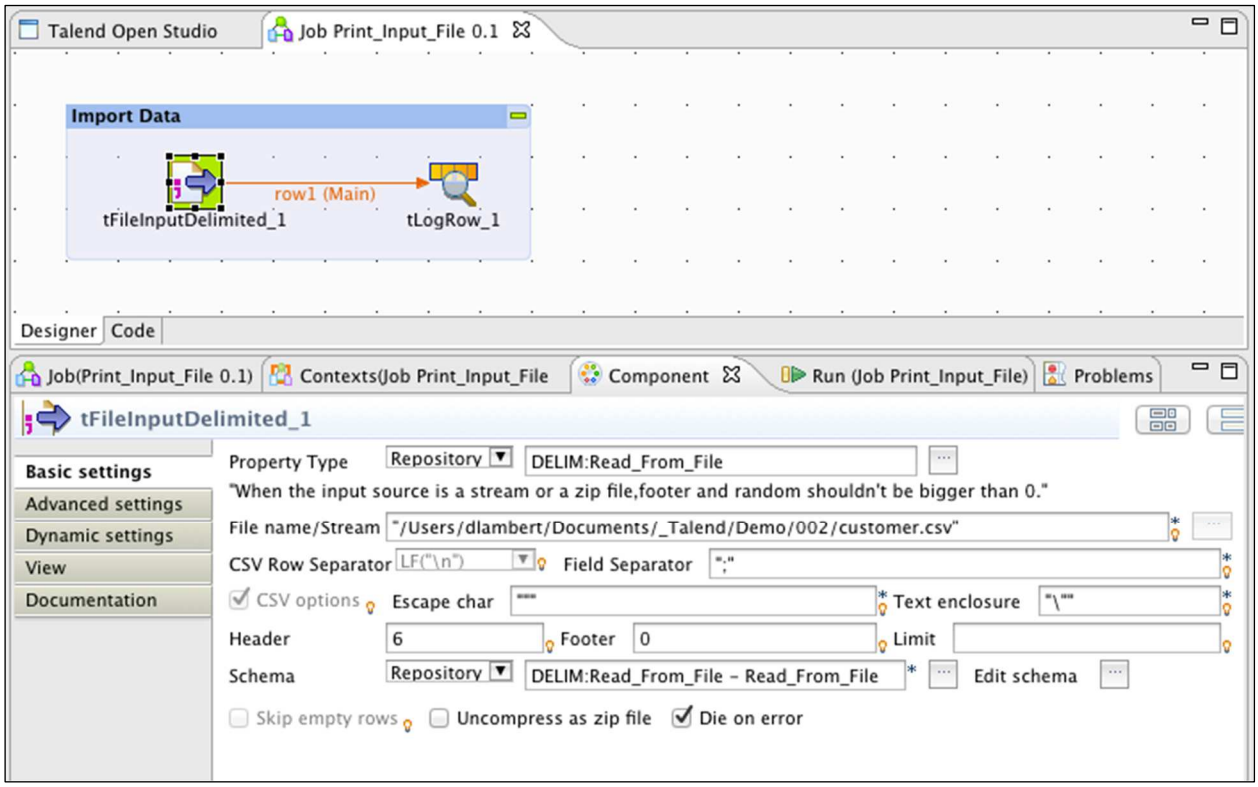

Figure 9: in the Job Print_Input_File,
the Basic Settings of tFileInputDelimited_1

The "Property Type" on the top line of the Component tab is, as usual, stored in the Repository. This time, it's stored in element 3.b.i, "Read_From_File", which contains information about where the file customer.csv is located, as well as the other parameters below. The format ("schema") of the file is stored in element 3.b.i.1.

The Settings in Job Import_File_Batched_Insert_PG

Previously, the component instances and their functions were described. Here is a look at how the component instances are configured. In reviewing the other jobs, we've seen how instances of tPostgresqlConnect, tFileInputDelimited, tPostgresqlInput are configured. Let's now look at the configuration of tPostgresqlOutput_1 in Figure 1. The configuration is shown below in Figure 10.

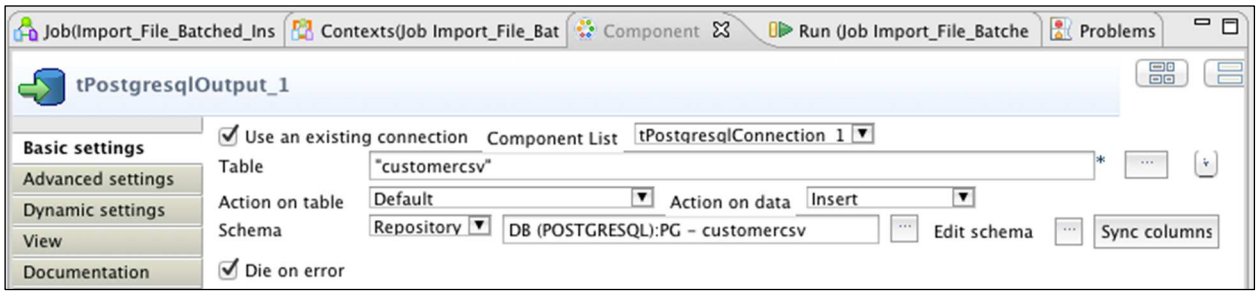

Figure 10: in the Job Import_File_Batched_Insert_PG,
the Basic Settings of tPostgresqlOutput_1

If you recall, this component is used to insert records into a table. In this case, records are read from customer.csv, the fields in the records mapped to the fields of table customercsv by tMap_1, and the records inserted into table customercsv (specified in the "Table" field of Figure 10 above). The "Action on table" drop-down can take on a variety of values, such as "Default", "Drop and create table", and "Clear table". The value "Default" means that the table is assumed to exist (and is neither created nor dropped), and its contents will be left alone.

The "Action on data" drop-down can take on the values listed in Table 4 below. Note that records are identified by the values of their unique keys, which must be configured in the schema selected by the selection for "Schema" in the Component tab above.

Value	Action
Insert	Insert new records into table. If a record already exists, abort job.
Update	Update existing records. Do not insert new records.
Insert or update	Insert new records or update existing records. Same as Update or insert.
Update or insert	Update existing records or insert new records. Same as Insert or update.
Delete	Delete existing records.

Table 4: the values of the "Action on data" drop-down in tPostgresqlOutput

How Talend Communicates with Databases

Talend communicates with databases through JDBC. Because each database is, in some way, different from other databases, each database vendor creates a JDBC jar file for its own database. Thus, the PostgreSQL project releases its own JDBC drivers for use with each major version of PostgreSQL.

TED is very similar to PostgreSQL:
1) They use the same data types and formats.
2) With a few exceptions, the SQL is identical.
3) With a handful of exceptions, the metadata are identical.
4) TED is constructed to hide the architectural differences between it and PostgreSQL.

Talend uses the JDBC Driver for PostgreSQL 8.3,

Getting Talend to Use the JDBC Driver for PostgreSQL 9.2

We discuss this further in the next section.

The Limitations of Talend

The PostreSQL command COPY FROM is the most efficient way to load data into PostgreSQL or TED. COPY FROM comes in two versions:
1) COPY <table-name> FROM <file-path> …
   <file-path> is the path, on the database host, of the file to be imported.
2) COPY <table-name> FROM STDIN …

The first version of COPY FROM gives customers OS-level direct access to TAP, which is undesirable. We tried to use it with a component in Talend for named pipes, but the only example of how to use the named pipes component requires another component called tParallelize, which is not available in the open source version of Talend. We failed to figure out how to used the named pipes component without tParallelize.

This leaves us with the second version of COPY FROM, which is not supported by the PostgreSQL 8.3 JDBC Driver. JDBC Drivers are embedded deep in the architecture of Talend, and our attempts to replace the PostgreSQL 8.3 JDBC Driver with a more recent version failed (see Talend Forum entry www.talendforge.org/forum/viewtopic.php?id=27750).