Class: BCA Semester -VI

Subject: Data Warehousing with SQL Server 2012

Academic Year – 2018 – 2019

UNIT-4

Enforcing Data Quality, Extending SQL Server Integration Services

Introduction to Data Quality

Data quality is a perception or an assessment of data's fitness to serve its purpose in a given context. The quality of data is determined by factors such as accuracy, completeness, reliability, relevance and how up to date it is.

Data Quality Services

Data Quality Services (DQS) in SQL Server 2012 is new feature used for improving and maintaining the quality of your data across enterprise. Following are some of the Data Quality Challenges:

- Incompleteness - Is the data value complete to have the proper meaning?
- Inconsistency - Is data value consistent throughout your organization?
- Invalid - Do the data values fall within the defined domain?
- Inaccuracy - Is your data value accurate?
- Non-conformity - Does the data value conform to the specific standard/representation or format?
- Duplicity - Is your data value is duplicated?

What is Data Quality Services?

DQS is a new feature that came out with SQL Server 2012 that can be used to cleanse, match and transform incoming data. It consists of two components: Data Quality Server and Data Quality Client. Data Quality Server is installed on top of a SQL Server 2012 database engine, whereas the Data Quality Client can be installed on a SQL Server machine or a separate client machine. When you install Data Quality Server three databases will be created on the database instance where it is installed: DQS_MAIN, DQS_PROJECTS, and DQS_STAGING_DATA.

DQS_MAIN, as the name suggests, is the main database for DQS. This database contains all the DQS stored procedures for the DQS engine, and the published data quality knowledge base information that comes with DQS. The DQS_PROJECTS database will be used to store DQS Project information. DQS Projects are created to allow you to cleanse, and match and then export the clean data to a SQL Server database or file. Additional DQS Projects can be used to profile data to help you better understand the quality of your data. The DQS_STAGING_DATA database is where you can copy your incoming data so it can be processed and cleaned up by DQS operations.

Additionally there is a DQS cleansing task that has been incorporated into SQL Server Integration Services (SSIS) packages. This SSIS task allows you to cleanse your data within an SSIS package. This DQS cleansing component is automatically installed when you install Integration Services.

Out of the box DQS has a built-in knowledge base that you can use to validate your data. Here is a list of the built-in knowledge base domains that come with DQS as documented in Books Online:

- **Country/Region:** Contains the conventional long (official name as designated by the country/region) and short names (common name used in lists, on maps, etc.), two-letter abbreviation, three-letter abbreviation and three-digit code for each location. Leading value is set to the long country name.
- **Country/Region (three-letter leading):** Contains the conventional long (official name as designated by the country/region) and short names (common name used in lists, on maps, and so on), two-letter abbreviation, three-letter abbreviation and three-digit code for each location. Leading values is set to County three-letter abbreviation.
- **Country/Region (two-letter leading):** Contains the conventional long (official name as designated by the country/region) and short names (common name used in lists, on maps, etc.), two-letter abbreviation, three-letter abbreviation and three-digit code for each location. Leading value is set to the Country two-letter abbreviation.
- **US - Counties:** Contains a list of US counties.
- **US - Last Name:** Contains a list of last names (surnames) occurring 100 or more times in the Census 2000.
- **US - Places:** Contains a list of places for the 50 states, the District of Columbia, and Puerto Rico extracted from the Census 2010.
- **US - States:** Contains the conventional long (official) name and two-letter abbreviation for each state in the US. Leading value is set to the conventional state name.

With DQS you cannot only use the built-in domains to cleanse, match and transform your data, but you can expand the knowledge base with your own domains. This allows you to import and define new domains that are appropriate for cleansing, matching, and transforming data for your environment.

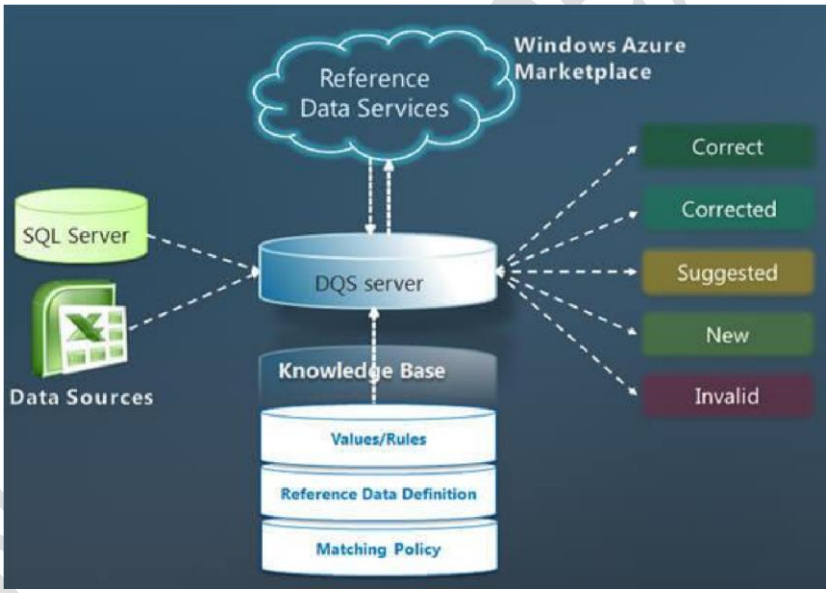
Answering that Need with DQS

Data quality is not defined in absolute terms. It depends upon whether data is appropriate for the purpose for which it is intended. DQS identifies potentially incorrect data, and provides you with an assessment of the likelihood that the data is in fact incorrect. DQS provides you with a semantic understanding of the data so you can decide its appropriateness. DQS enables you to resolve issues involving incompleteness, lack of conformity, inconsistency, inaccuracy, invalidity, and data duplication.

Here are some of the operations that DQS performs or allows us to perform:

- **Monitoring** - It allows to track and monitor the state of data quality activities and quality of data; this way you can monitor DQS activities and its progress.

- **Profiling** - It does the analysis of the data source to provide insight into the quality of the data and helps to identify data quality issues at every stage in knowledge discovery, domain management, matching and data cleansing processes.
- **Cleansing** - It can be performed interactively via the DQS Client tool or in batch mode via use of an SSIS component. It updates, removes or enriches data that is incorrect or incomplete. This includes correction, standardization and enrichment. The cleansing process classifies the data to different categories as follows: ○ Correct - Terms/data values that were found correct; for instance, matched a domain value or returned from the RDS (Reference Data Service) as _correct'.
    - Corrected - Terms/data values that were automatically corrected by the DQS engine by using the knowledge base that it uses or the RDS (Reference Data Service) provider.
    - Not Corrected - Terms/data values that were not recognized by the system as _correct' or terms/ data values that violate a domain rule.
    - Auto suggested - Terms/data values that the DQS engine has found a correction for, but with medium or low confidence and require user intervention to move ahead (Approve/Reject). ○ New - Terms/data values that the DQS engine has identified as new values at the data source (Approve/Reject).



- **Matching** - Identifying, linking or merging related entries within or across sets of data. The goal of matching is to remove duplicates from the source. It can be done intra-source (matching a source against itself) or inter-source (matching a source against a lookup table). A matching process includes four steps:
    - Matching Policy Training - To define and test the set of policy rules that govern the matching process. The matching policy rules help identifying the relevant duplicates.
    - Matching - To run a matching project by matching the data at source against itself or against a lookup table. ○ Auto-Approve - To define policy rules for auto-approving matching result clusters based on the match results obtained.

- Merge/Survivorship - To define the policy rules for selecting one approved record from each cluster of matched records.

Reference Data Services: verification of the quality of your data using the services of a reference data provider.
Knowledge Base: Data Quality Services is a knowledge-driven solution that analyzes data based upon knowledge that you build with DQS. This enables you to create data quality processes that continually enhances the knowledge about your data and in so doing, continually improves the quality of your data.

DQS Architecture
With DQS, data quality is now available to a broader audience than was the case with previous SQL Server tools. DQS was designed for ease of use. Through a simple and intuitive interface, DQS empowers business users and DBAs to engage more directly in data quality activities. With this functionality, you can realize data quality improvements in a very short time. As mentioned earlier, DQS includes server and client components. Figure 14-2 shows a quick overview of DQS architecture.

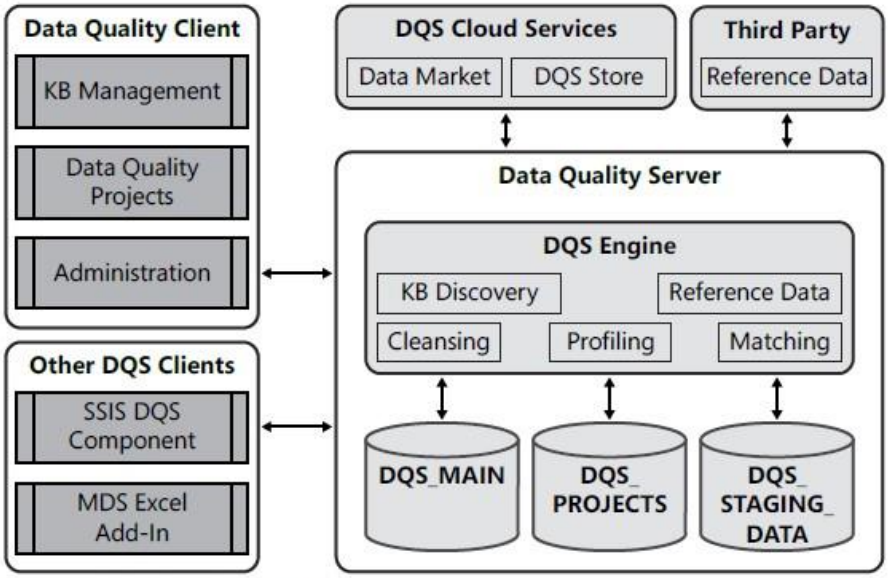


FIGURE 14-2 DQS architecture.

The Data Quality Server component includes three databases:

■■ DQS_MAIN, which includes DQS stored procedures. The DQS stored procedures make up the actual DQS engine. In addition, DQS_MAIN includes published knowledge bases. A published KB is a KB that has been prepared to be used in cleansing projects.

■■ DQS_PROJECTS, which includes data for knowledge base management and data needed during cleansing and matching projects.

■■ DQS_STAGING_DATA, which provides an intermediate storage area where you can copy source data for cleansing and where you can export cleansing results. You can prepare your own knowledge bases locally, including reference data. However, you can also use reference data from the cloud. You can use Windows Azure MarketPlace DataMarket to connect to reference data providers. Of course, you can also use a direct connection to a third-party reference data provider through a predefined interface.

With the Data Quality Client application, you can manage knowledge bases; execute cleansing, profiling, and matching projects; and administer Data Quality Services. SQL Server 2012 includes two new tools to assist with these tasks. You can use the SSIS DQS Cleansing transformation to perform cleansing inside a data flow of your SSIS package. This allows you to perform batch cleansing without the need for interactivity required by Data Quality Client. With the free Master Data Services (MDS) Microsoft Excel add-in, you can perform matching of master data in an Excel worksheet. The DQS components must be installed together with MDS in order to enable DQS/MDS integration. Additional clients, including third-party applications, will very likely be forthcoming in the future.

DQS Installation

You install the DQS components by using SQL Server 2012 Setup. You will need to select the following features:

■■ Data Quality Services under Database Engine Services to install Data Quality Server.

■■ Data Quality Client to install Data Quality Client.

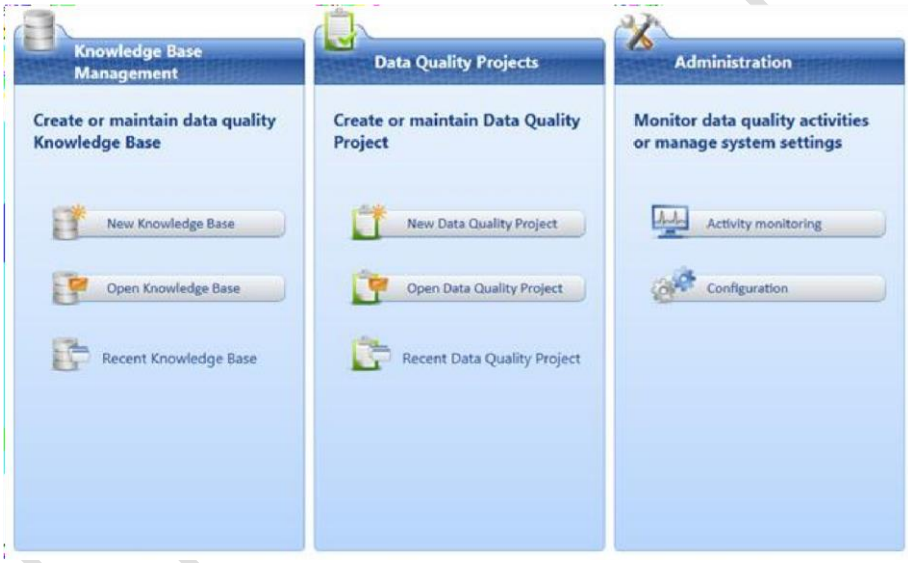
Using Data Quality Service to Cleanse data

Creating a Knowledge Base

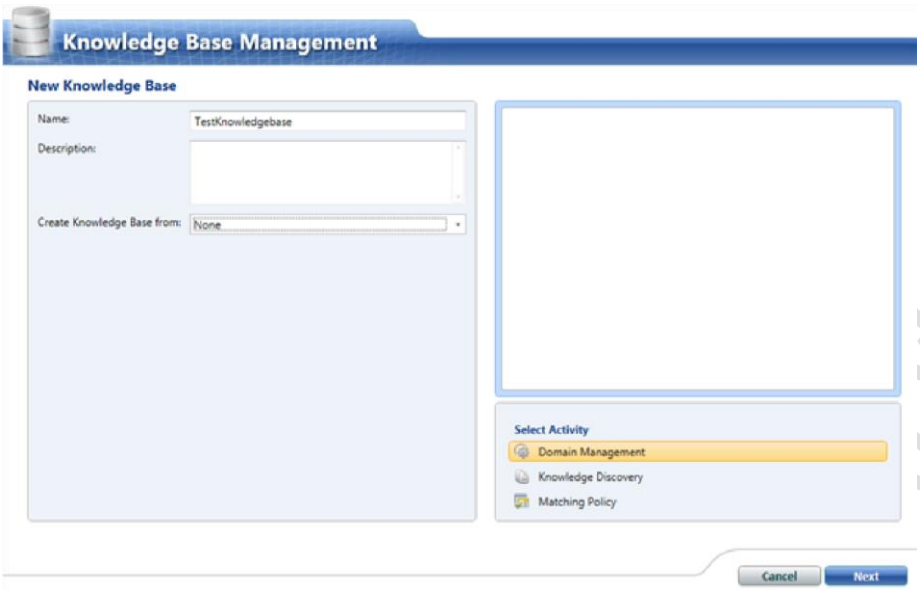
Start Data Quality client. The next step is to create a DQS Knowledge Base. This knowledge base is used to contain information about our data and how it can be used for cleaning purposes later on. The knowledge base is actually an object which can be inherited by knowledge bases created later on.

Here is a short description of this screen:

- Knowledge Base Management, mentioned earlier, is where we can create a new knowledge base or open a knowledge base for further development
- Data Quality Projects are used for data cleansing or data matching by mapping the source columns to the domains in the knowledge base and providing a set of rules for the further modification
- The Administration tab gives us an opportunity to monitor the data cleaning activities and to configure the DataMarket account we would like to use (if we have one)

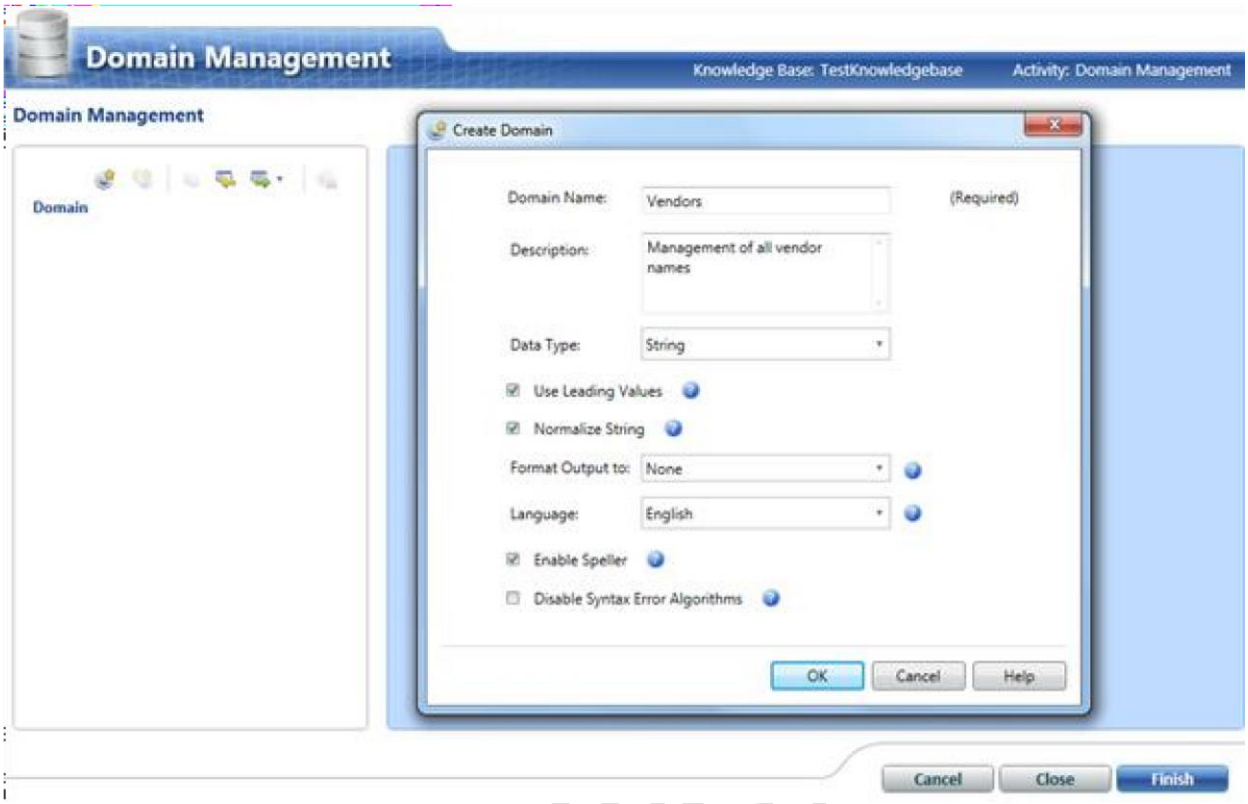


To create a new knowledge base, we need to click on the _New knowledge Base‘ button and go through the wizard. This is how the first screen looks like:

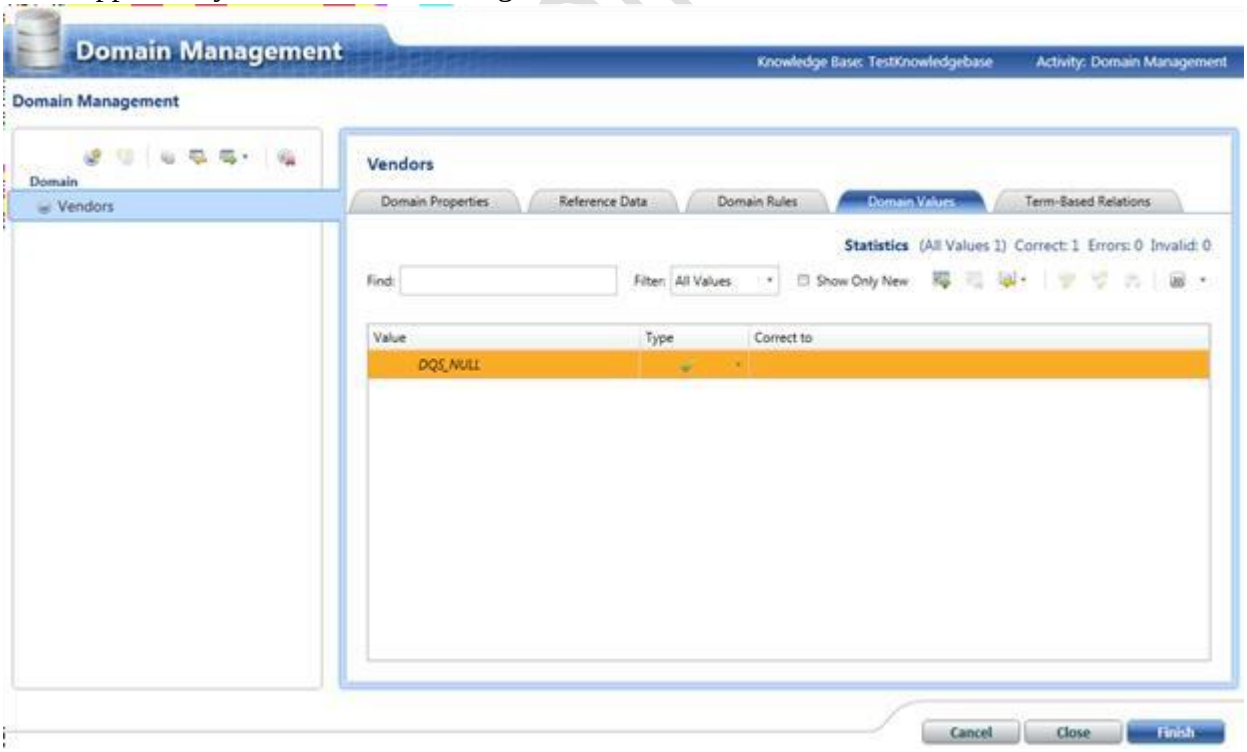


In this case I have named it TestKnowledge base and it will be brand new knowledge base. There are two other options under the _Create knowledge base from‘ tab:

- **Existing knowledge base** – there is a built-in DQS Data knowledge base which comes with the installation and it contains some data about countries / regions and US states. If we had some other knowledge bases created before, they would also be available in this list
- **Import from DQS file** – we can use this functionality to import a previously exported knowledge base. This way we can share and reuse knowledge bases between servers. In the next screen of the wizard, we are asked to create a domain. In our case we are going to be maintaining data cleanliness of our Vendors, so we create an appropriate domain.

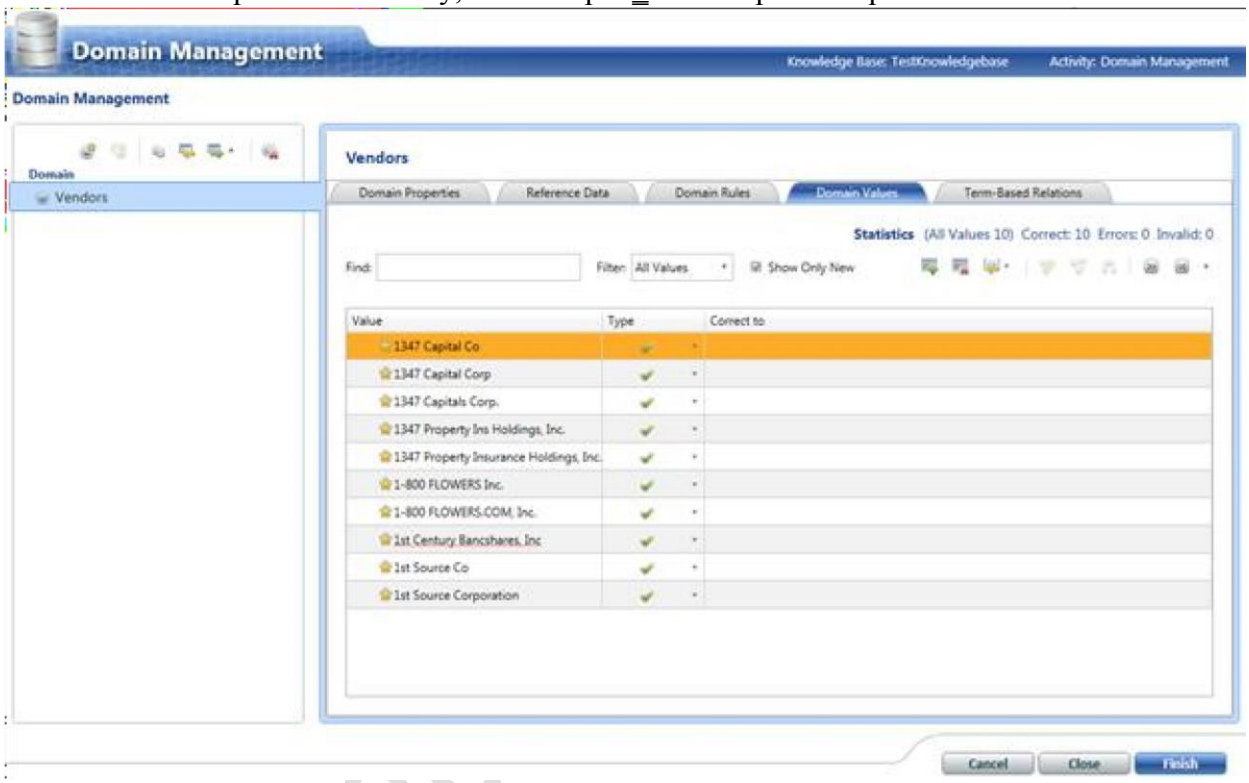


After we accept the above screen, we are presented with a new screen with five tabs, which gives us the opportunity to train our knowledge base.

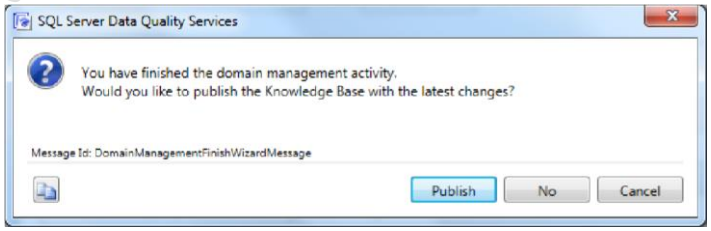


Training the knowledge base

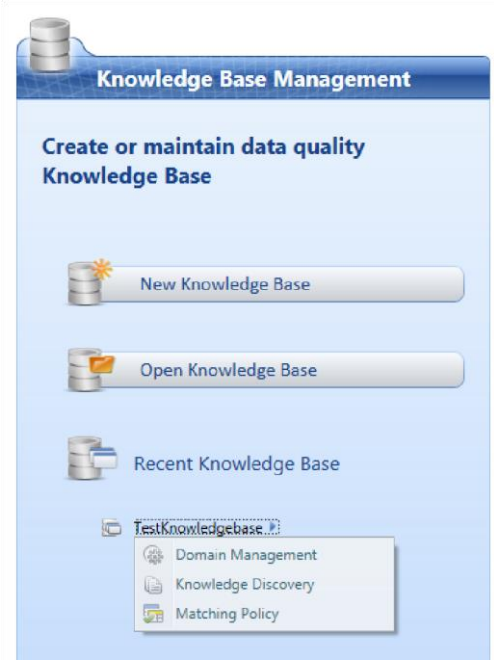
To train our knowledge base, we first need to import a list of our vendors. In reality this list may take a while to compose. It is a very bizarre fact that the only way we can import the list is if we have it in Excel format. This means that there is no way for us to use a SQL Server database table or any other data source – as you can imagine, this problem has been raised with Microsoft. For the purpose of this article I am using a very small list of Vendors (just some random company names I put in an excel sheet).
After importing the excel list into the Domain Values, we can see that there are a few vendors which have been spelled differently, for example _1347 Capital Corp.‘.



In this case I want _1347 Capital Corp.‘ to be the correct display of the data and the other two versions to be marked as synonyms. I mark the first three rows, right-click and select *_Set as synonyms"*. Then I click on the row containing _1347 Capital Corp.‘, right-click and select *_Set as leading"*.
After we have gone through all values and identified the synonyms and the correct spelling, we click on the _Finish‘ button and we are asked whether we want to Publish the knowledge base.

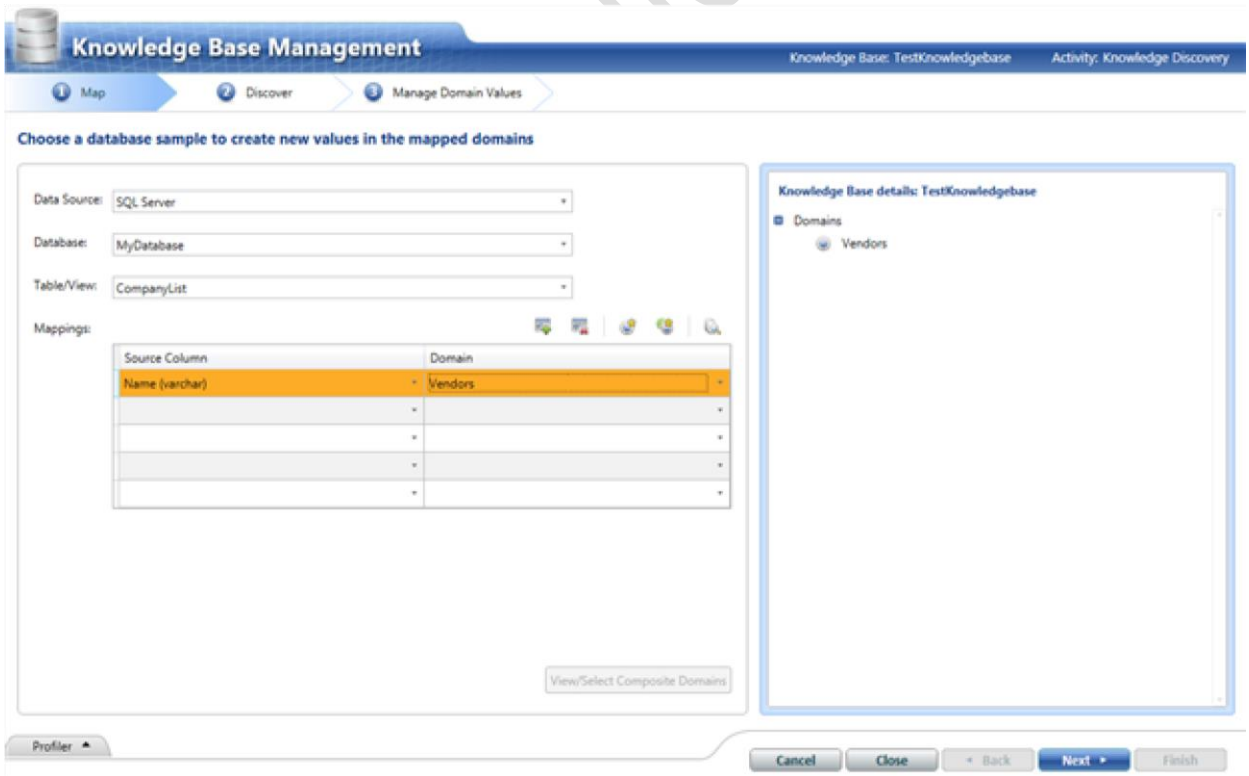


Knowledge discovery
Now it is time to use the knowledge base we just created and see how it performs on larger datasets of incoming data.

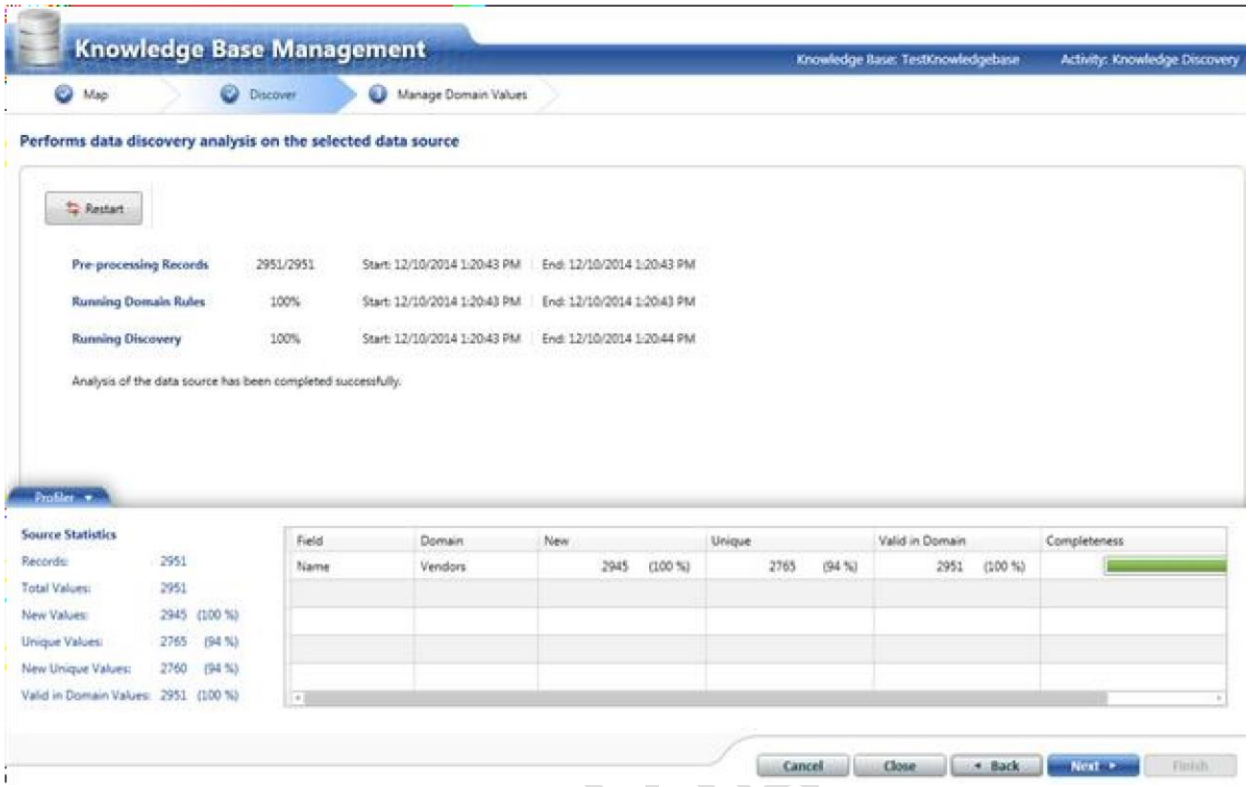


In the main screen we click on the _*Knowledge discovery*_ button under our newly created knowledge base and we are prompted with options to connect to a data source.

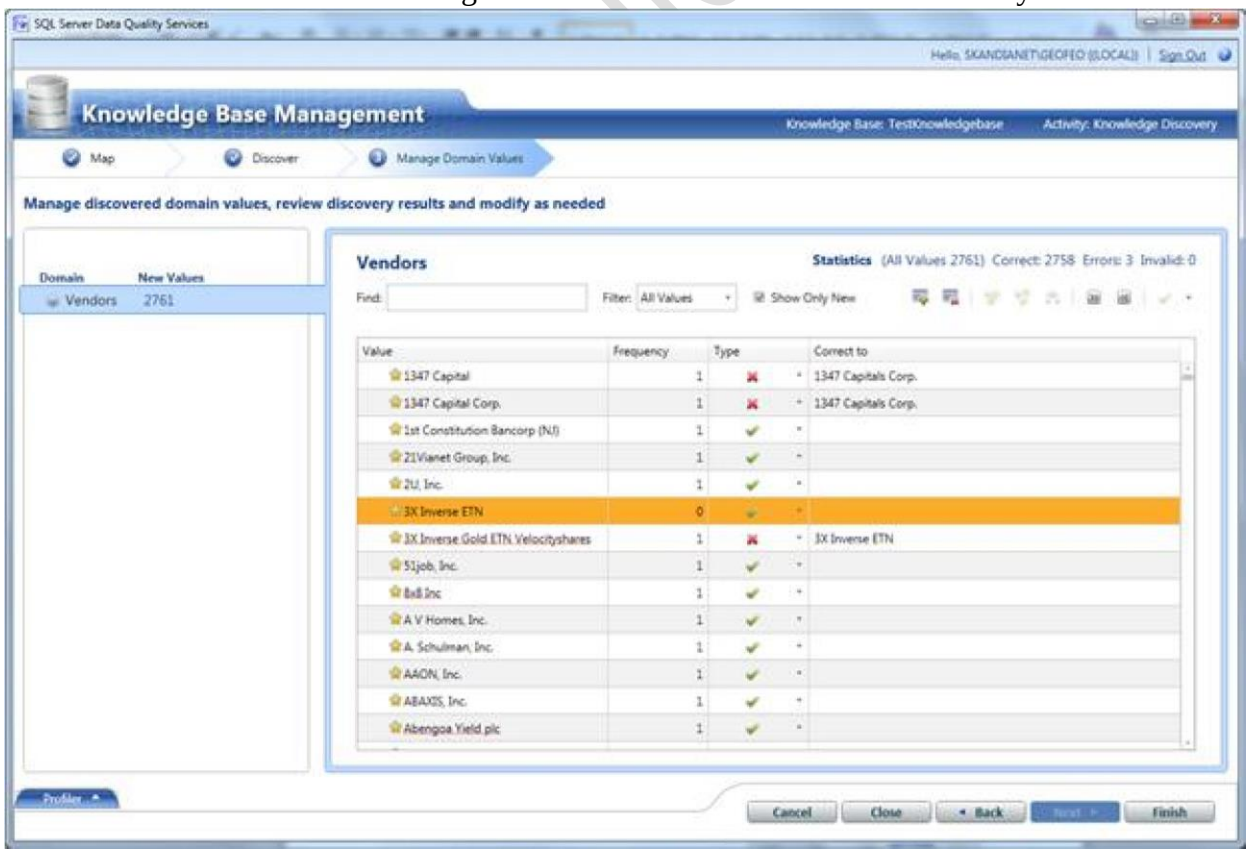
In this case we can use SQL Server table as a data source. We point to our database and table which contains the vendors and we map the column, containing the vendor name to the knowledge base domain.



We click _*Next*_, and in the next screen we get to analyze the data mapping we did.



We see by these statistics that our data if fairly clean. We click _*Next*_ and in the next screen we see what was corrected and we also get a chance to correct data further manually.

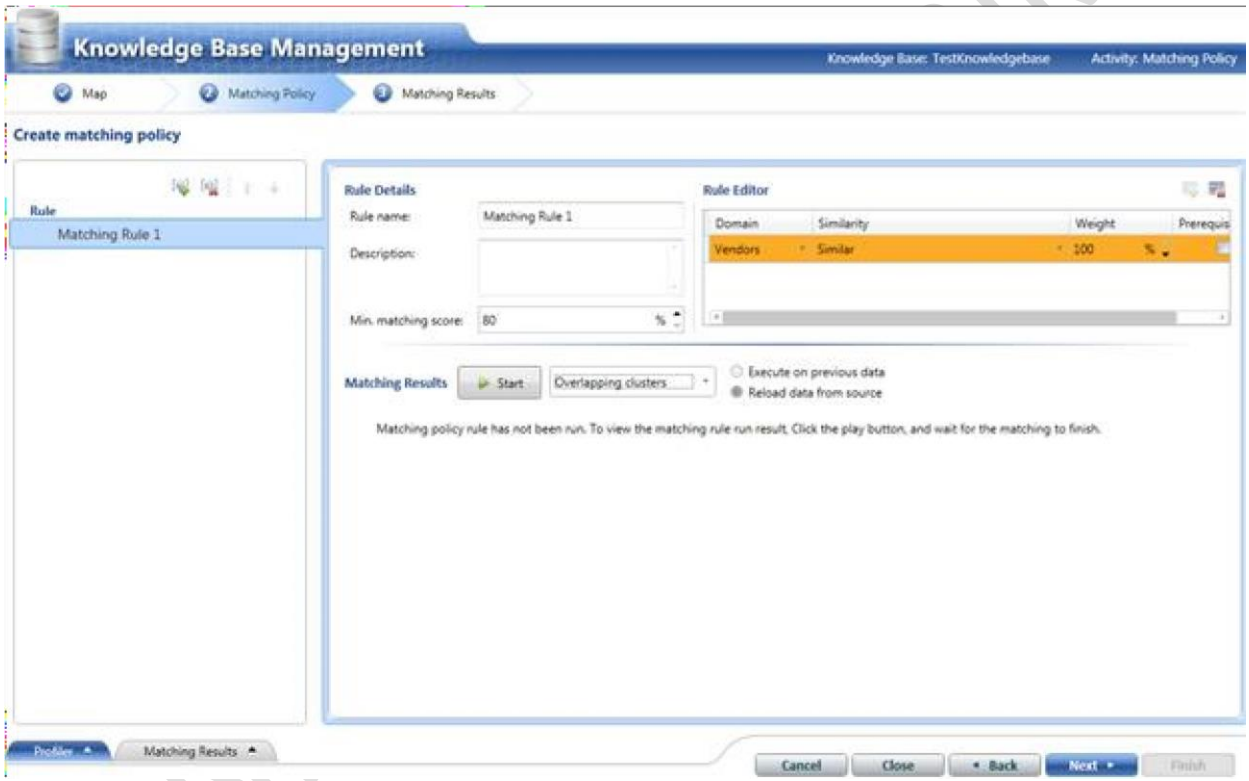


Develop by: Ravi joshi

In this case, _1347 Capitals Corp.‘ was automatically corrected and _3X Inverse ETN‘ was corrected manually.
As we click *_Finish_*, we are asked whether we want to publish the knowledge base.

Matching policy
The third part of our data cleaning process is to define the percentage of certainty which defines a correct record vs. incorrect record. This process will look at DQS statistics and tell us what DQS considers to be a valid record.

We start again from the home screen, click on our knowledge base and this time we select *_Matching policy_*. We connect to our data source in the same way as in the previous section, we click *_Next_* and in the *_Create matching policy_* screen we are asked to create at least one domain element in the rule editor.

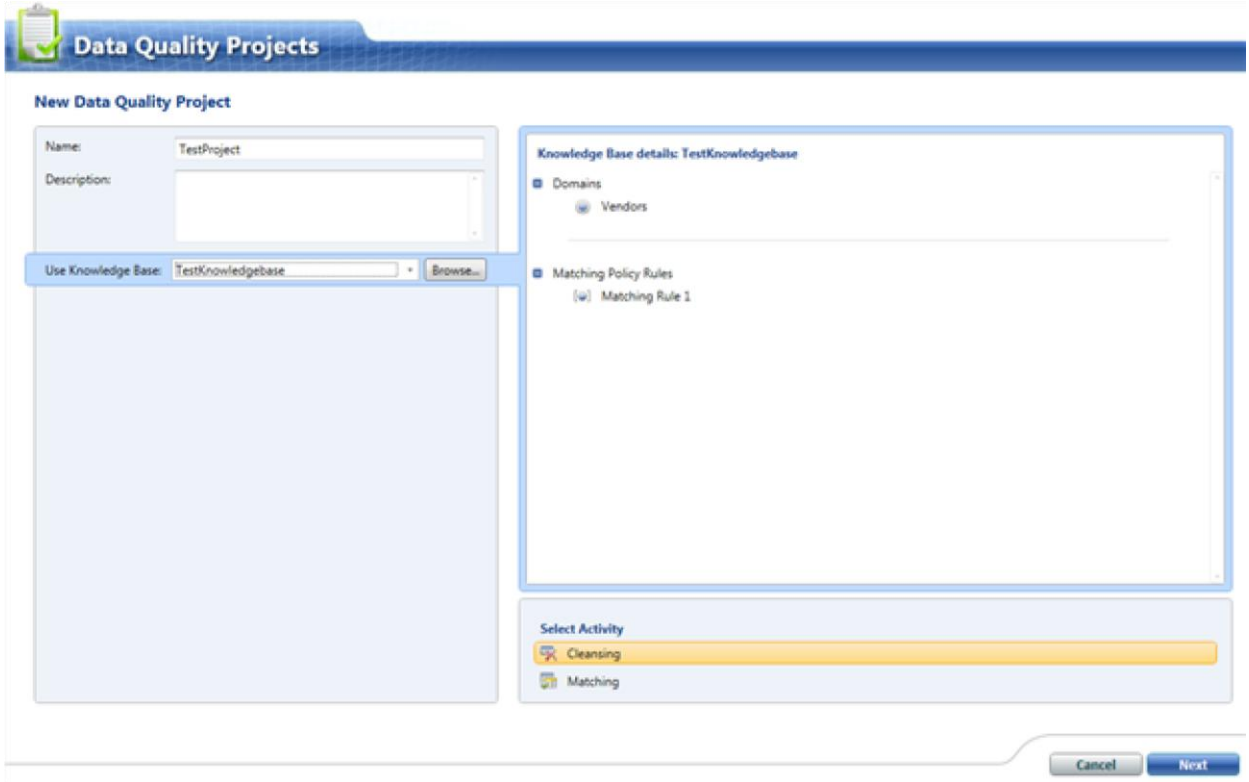


In real world cases, we would be considering composite evaluations, let‘s say of Vendor, Product, etc. And in such cases the attributes can have any percentage weight, but they have to amount to 100%.

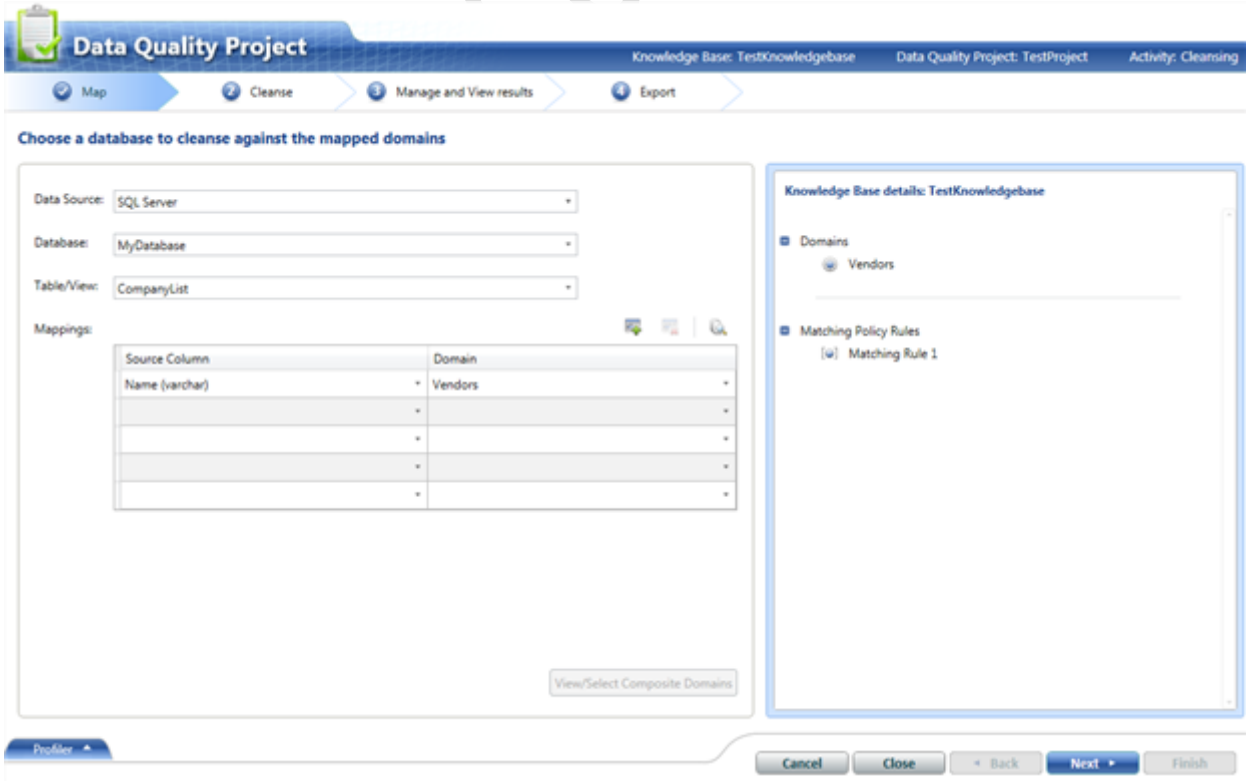
In this case I am looking only at the vendor name, which means that it must have 100% weight in order to be correct, otherwise it is not correct.
We click the *_Start_* button and we evaluate the results. We
click *_Finish_* and we publish the results.

Data Quality project
So far we have created, managed and trained our knowledge base and now it is time to see how we can apply it towards our production data or towards our daily data flows.

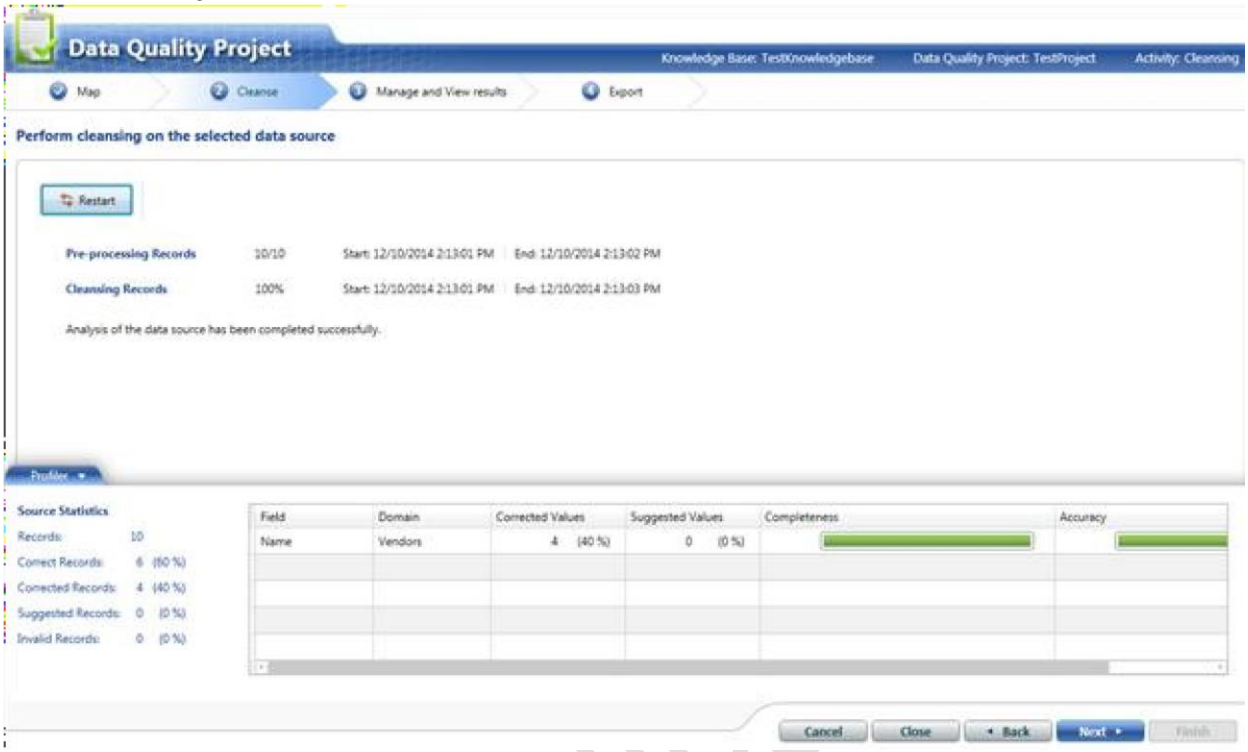
For this we need to create a Data Quality Project from the home screen.
We give the project a name and we select our newly created knowledge base and we select data cleansing, like in the screen below:



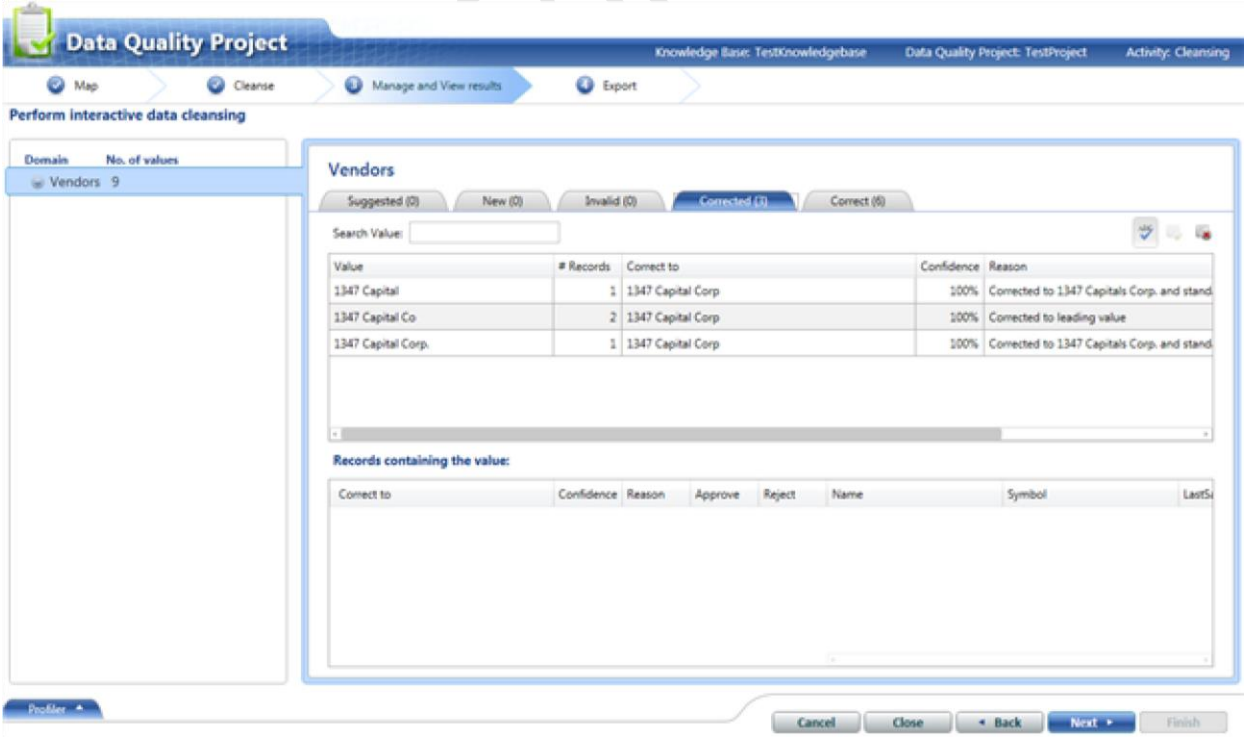
The same screen as before appears, where we need to connect to the database:



We click _*Next*_, and in the next screen we click _*Start*_ to begin the cleansing process.
In this case, DQS indicates that there were 60% correct records and 40% corrected.



In the next screen we see more details about what was correct and what was corrected, and even at that point we have the opportunity to approve or disapprove changes.

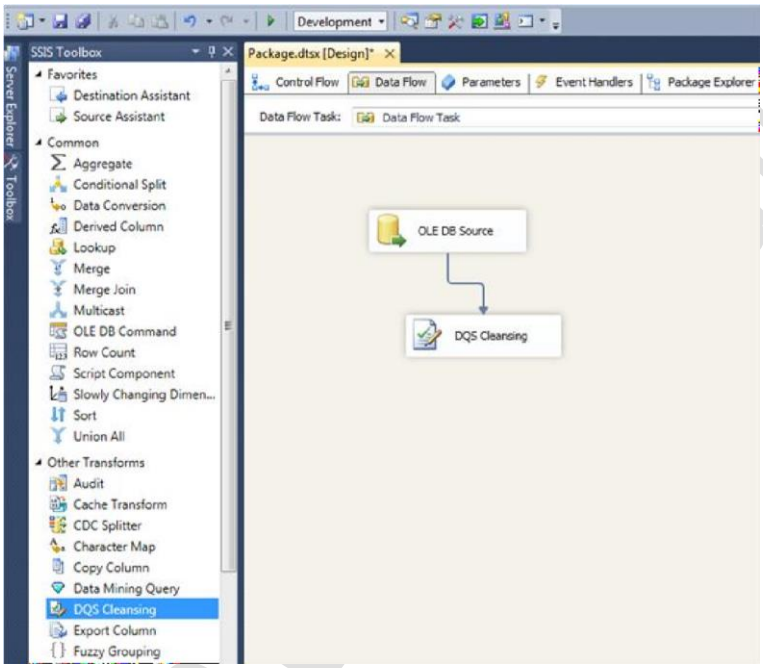


We click _*Next*_ and in the following screen we have the opportunity to save the cleansed data to a database table and to save our project.

In this case we performed interactive cleaning of our data and saved the results for further use. The saved results can be used to be loaded in other systems, or to be looked into further. Let‘s see how we can automate the daily data loads.

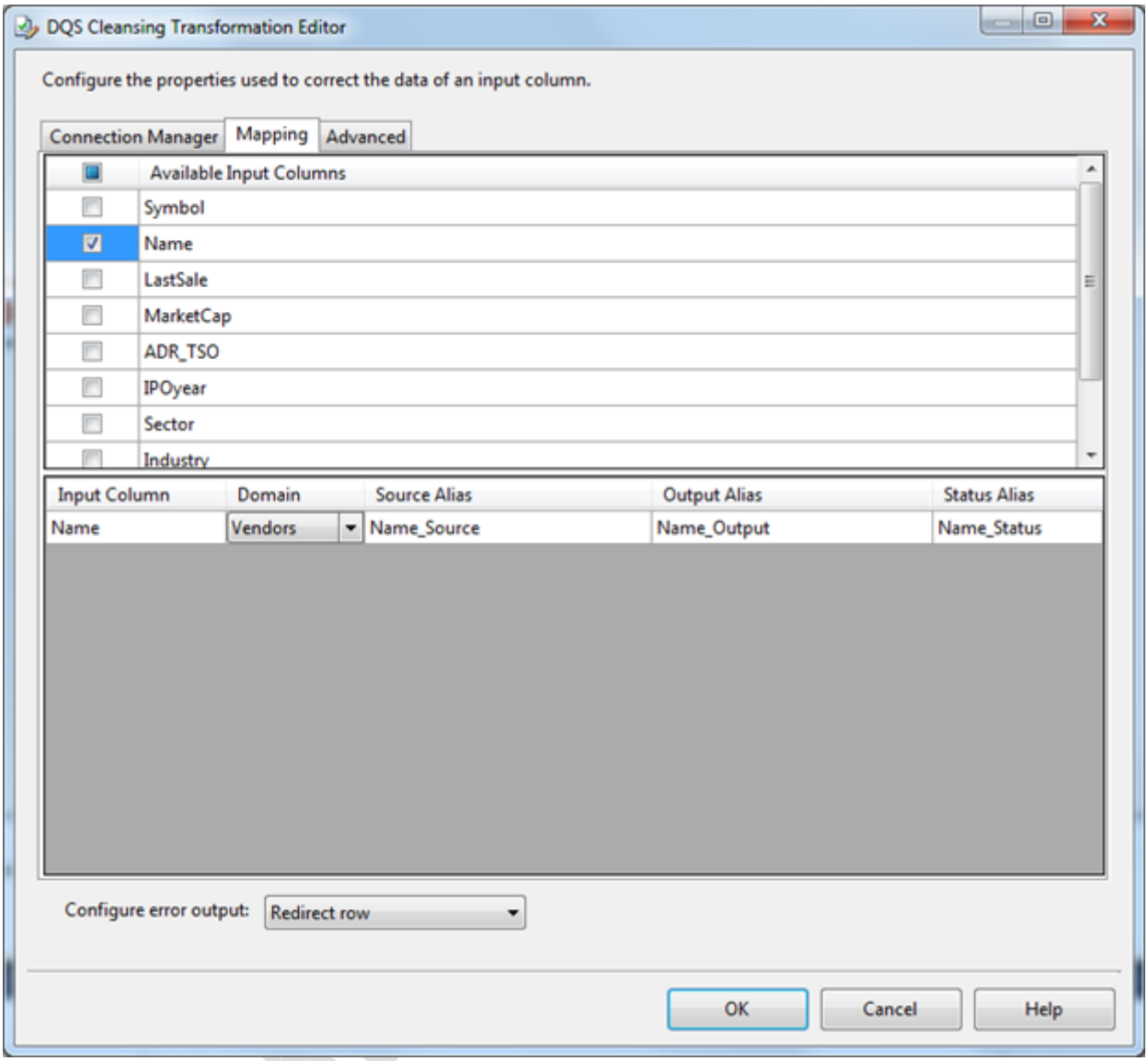
Automating the daily load of data (batch mode)

For this example we need to create a new SSIS project, and configure a datasource connection. The pipeline, in our case, will consist of a single data flow component, in which we will have a datasource, followed by a DQS cleansing component.

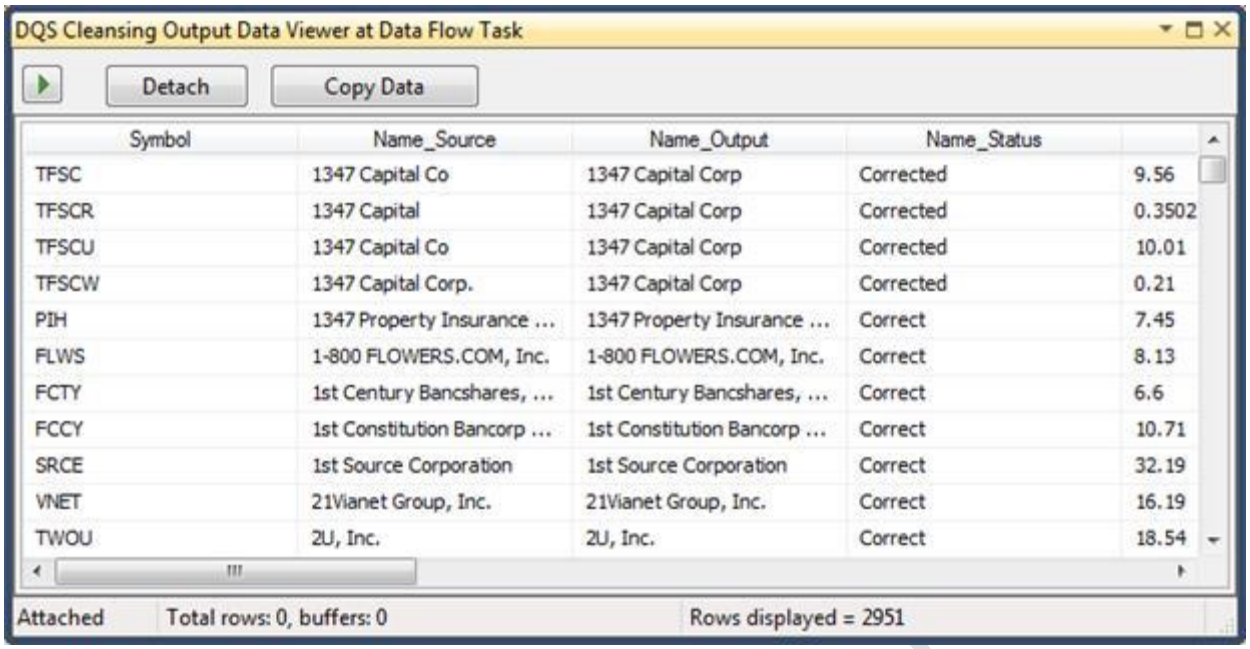


When we configure the DQS component, we have to indicate which DQS server we want to use, as well as which Data Quality knowledge base.

After that, in the Mapping tab, we need to specify which input column we want map to which Domain of our knowledge base.

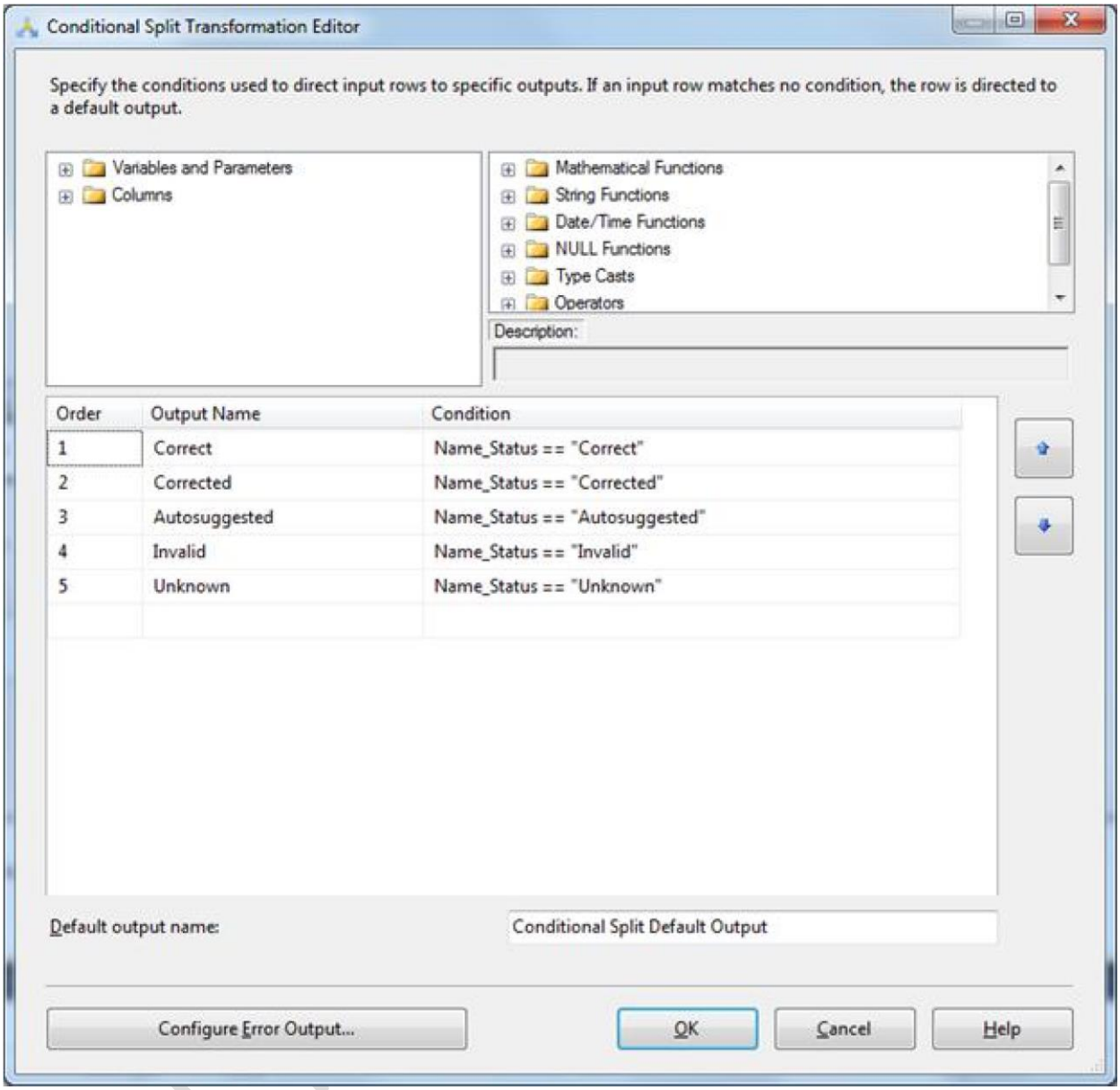


If we add a data viewer after the DQS Cleansing task, we will see that it adds a column called _Name_Status‘.

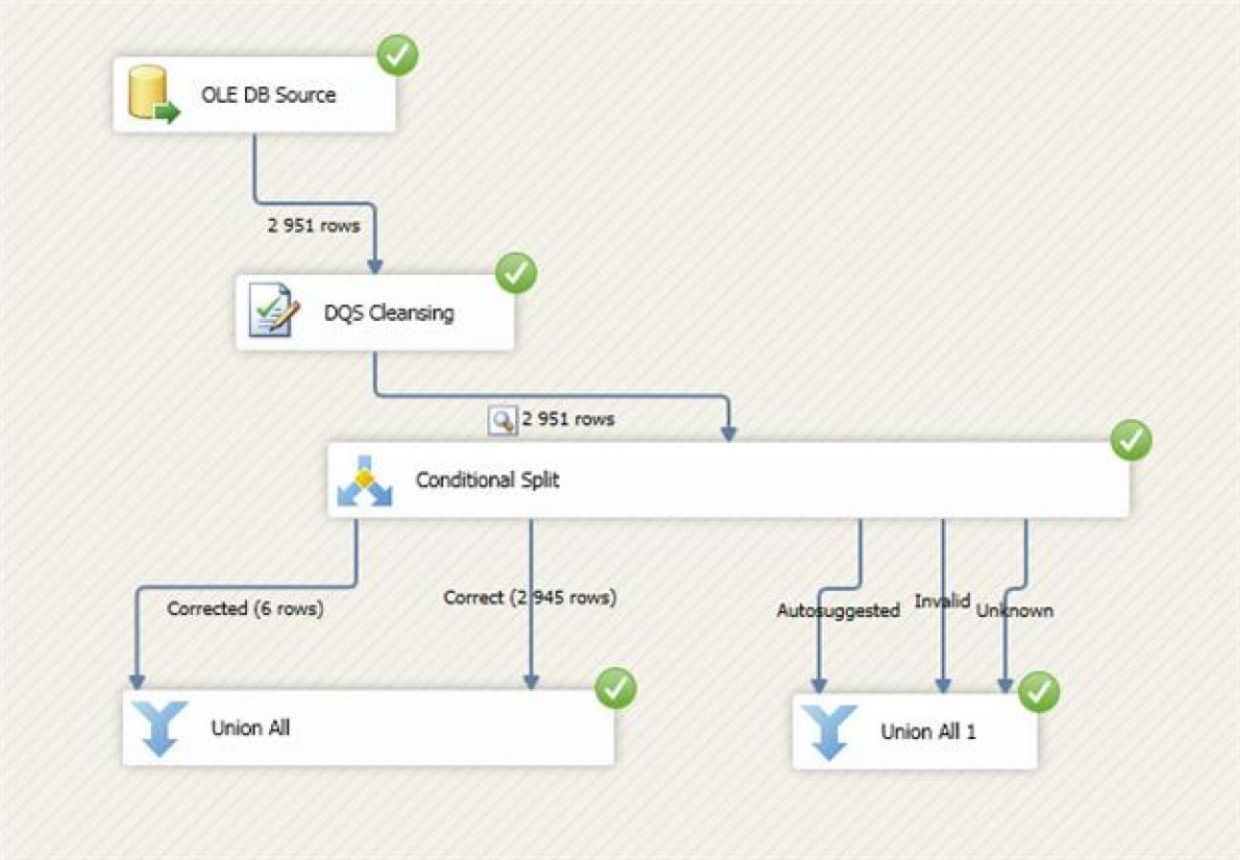


We will use this column in our pipeline to provide Conditional split and to redirect the rows after they are cleansed.

The conditional split component will look similar to this:



And our SSIS package will look like this:



In this case I am using the Union All operator, but in reality we would use the proper destinations for the different types of rows.

OR

Using Data Quality Service to Cleanse data

step1: Creating and Maintaining a Knowledge Base

In order to successfully clean your data with DQS, you need to have a good knowledge base. You can create a KB interactively or with automatic knowledge discovery from existing trustworthy data. Then you can use this knowledge to cleanse your data by using a DQS project. During cleansing, you gain new knowledge; you can integrate this new knowledge into your knowledge base. A knowledge base consists of *domains* that are mapped to data source fields. You can create composite domains for advanced rules that span multiple fields.

Building a DQS Knowledge Base

Typically, you start building a knowledge base with a computer-guided process called knowledge discovery. The DQS knowledge discovery process has a lot of built-in heuristics. You feed the process with sample data that you trust. The process analyzes the data for data inconsistencies and syntax errors, and it proposes corrections.

To initiate knowledge discovery, you as a data steward link a DQS KB domain to a field of

an existing table that has data similar to the data you need to cleanse. You do not need to use the data you need to cleanse for the knowledge discovery activity. A DQS KB is extensible; after initial knowledge discovery, you can edit the KB manually. You can also re-run the knowledge discovery process.

Building a DQS KB involves the following processes:
■■ ***Knowledge discovery*** A computer-guided process that uses a data sample ■■ ***Domain management*** An interactive process in which you manually verify and extend domains in a KB
■■ ***Reference data services*** A process in which you validate domain data against external data maintained and guaranteed by an external provider
■■ ***Matching policy*** A process in which you define rules to identify duplicates
DQS is case-insensitive. It does not distinguish values by case when you perform knowledge discovery, prepare a matching policy, or manage a domain. However, you can control the case when you export cleansing results.

A matching policy includes matching rules that you create to help DQS identify duplicate rows for a single entity. To do so, you must define which columns DQS should use to compare the rows and calculate the probability of a match. You will learn more about identity mapping and de-duplicating with DQS and other SQL Server tools in Chapter 20, ―Identity Mapping and DeDuplicating.‖

DQS includes some default knowledge out of the box. This knowledge is stored in the DQS Data knowledge base. You can use this KB to start a cleansing project quickly. You should treat this KB as read-only. It is not attached to a reference data provider. You can use this default KB to create your own read-write KB, which you can then edit. The DQS Data KB includes three Country/Region domains (one with full-name leading values, one with three-letter abbreviation leading values, and one with two-letter abbreviation leading values), two US State domains (one with full-name leading values and one with two-letter abbreviation leading values), US Counties, US Last Name, and US Places domains. A *leading value* in a domain is the value to which you want DQS to correct the data.

You can export or import a domain, an entire knowledge base, or all knowledge bases from a DQS instance. You have the following options:

■■ Import and export all knowledge bases from an instance with the DQSInstaller.exe command prompt utility. When you export all KBs, you create a DQS backup (.dqsb) file.
■■ Import or export (to a .dqs file) an entire knowledge base from Data Quality Client.
■■ Import or export (to a .dqs file) a domain with Data Quality Client.
■■ Import values from a Microsoft Excel file to a domain with Data Quality Client. ■■ Import domains from an Excel file with the knowledge discovery activity in Data Quality Client.
■■ Import new values from a finished DQS cleansing project with Data Quality Client.

Domain Management
A domain contains a semantic representation of a specific field (column of a table) in your

data source. You can define the following properties for a domain:
■■ **Data type** The data type of a domain can be string, date, integer, or decimal. ■■ **Use leading values** This property defines whether to replace all synonyms of a value with the leading value you define.
■■ **Normalize** With this property, you can normalize strings to remove special characters and thus improve the likelihood of matches.
■■ **Format output** You can use this property to format strings to uppercase, lowercase, or capitalized for each word.
■■ **Speller** With this property, you can enable the spelling checker on a string domain.
■■ **Syntax algorithms** You can use this property to disable checking strings for syntax errors. This is useful for names and addresses, for which you probably don‘t want DQS to automatically correct the syntax of strings for international values.

Besides creating a domain manually, creating one with knowledge discovery, or importing one, you can also create a domain as a copy of an existing domain. In addition, you can also create a *linked domain*. Linked domains are useful for mapping two data source fields to the same domain. You cannot map two fields to a single domain directly; instead, you can create a linked domain and link one field to the original domain and one field to the linked domain.
You can change the domain values after you populate the domain values through the knowledge discovery process, manually, or by importing the values. You can change the type of the value, designate synonyms, and define the value to correct to a selected value. The type of a value can be correct, invalid (meaning that the value does not conform to the rules for the domain and is thus out of scope for the domain), or error (meaning that the value is in the scope of the domain but is erroneous). If you change the type to invalid or error, you should provide the corrected value. When you select a set of values as synonyms, you should also select the leading value. DQS uses the leading value as a replacement for its synonyms. If you use the spelling checker and the value is underscored with a red squiggly line, you can select a correction if the spelling checker proposes any. The value will remain erroneous, and the correction is added as the corrected value.

Besides defining correct, invalid, erroneous, and corrected values for a domain, you can also define domain rules. A *domain rule* is a condition that DQS uses to validate, standardize, and/or correct domain values. You can define rules such that the value must be greater than a selected value, must begin with a value, must comply with a pattern or a regular expression, must contain a value, must be in a list of values you specify, and more. A single rule can have multiple conditions connected with logical AND or logical OR operators.

In addition to synonyms, you can create term-based relations. You use a *term-based relation* to correct a term that is part of a domain value and not the complete domain value. You define a term-based relation once, and DQS uses it for all values in a domain. For example, you could define that the string Inc. should always be expanded to Incorporated.

Sometimes a single domain does not represent real data satisfactorily. For example, a complete address can consist of street address, postal code, city, and country, and you need the complete address for de-duplicating. For cases like this, you can define a composite domain that consists of two or more single domains. A composite domain can have a *cross-domain rule*, a rule that tests the relationships between two or more single domains. For example, you can check that a specific

city is always in a specific country. In Data Quality Client, you can check the number of occurrences of value combinations of a composite domain in order to mitigate the process of creating the cross-domain rules. If you need to change the values of a composite domain, you change them in each of the single domains that the composite domain consists of.

After you finish with knowledge base editing, you have to publish the KB. If it is not published, it is locked. You can only unlock your own knowledge bases.

You want to use a knowledge base that exists in one DQS instance in another DQS instance. Should you re-create the knowledge base on the second DQS instance manually?

No, you should export the knowledge base from the first DQS instance and import it into the other.

Step 2. Creating a Data Quality Project

After you have a knowledge base, you can use DQS projects to validate and cleanse your data. You do not modify the data directly on the source; instead, you export the results of a DQS project to SQL Server tables or Excel files and then use queries to cleanse the source data.

DQS Projects

You can create a *cleansing* or a *matching* DQS project. Creating a DQS project is a quite simple task; a wizard guides you through individual steps. During the cleansing and matching, you can also see the profiling results for your data. DQS profiling provides you with two data quality dimensions: *completeness* and *accuracy*. Based on data profiling information and defined notifications, you can also get warnings when a threshold is met. You can use the SSIS DQS Cleansing component to cleanse batches of data. It is a best practice to cleanse data before you perform de-duplication.

Use the DQS Cleansing transformation for batch cleaning during your extract transformload (ETL) process.

Sometimes you need to perform some management activities on an existing project. These management activities can include:

■■ Opening an existing data quality project. You use the Open Data Quality Project button of the Data Quality Client tool to display a grid of existing DQS projects.

■■ Unlocking a DQS project. You use the Open Data Quality Project button of the Data Quality Client tool to display a grid of existing DQS projects. Then you right-click a project in the grid and select the Unlock option. A project is locked when someone edited it without finishing the edit.

■■ Renaming a DQS project. You can right-click a project and rename it.

■■ Deleting a DQS project. Again in the displayed list of existing projects, you select a project and right-click it to display the Delete option.

Note that you cannot open, unlock, rename, or delete a data quality project that was created by another user. You can unlock only those projects created by you. Also, you cannot delete a locked project; you first have to unlock it.

Data Cleansing

DQS uses knowledge bases for automatic, computer-assisted data cleansing. After the automatic process is done, you can manually review and additionally edit the processed data.

When you finish with editing, you can export the cleansed data. In addition, you can use the results of a DQS project to improve a knowledge base—for example, by adding new correct or corrected values to existing domain values in the knowledge base. Figure 17-2 shows the complete life cycle of DQS activities.

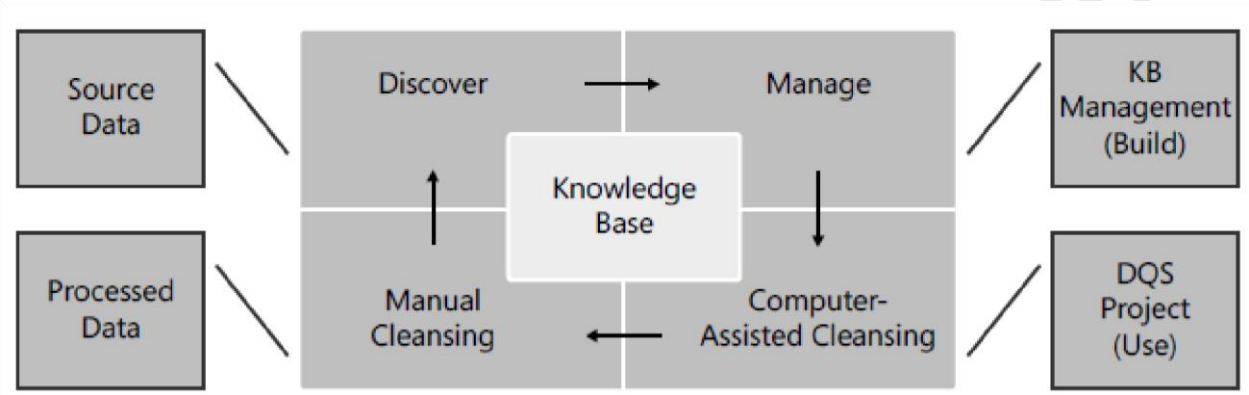


FIGURE 17-2 The life cycle of DQS activities.

You can use SQL Server or Excel data sources. If the source data is in an Excel file, then Excel must be installed on the same computer as Data Quality Client. The Excel files can have the extension .xlsx, .xls, or .csv. However, if you are using a 64-bit version of Excel, .xlsx files (from Excel 2007 and Excel 2010) are not supported. Save the Excel files you want to use as source data in .xls or .csv format.

A DQS knowledge base must exist before you can start a DQS cleansing or matching project. Of course, the knowledge base must contain the knowledge about the type of data you are going to cleanse. For example, if you are cleansing company names, the knowledge base you use should have high-quality data about company names. In addition, a KB used for cleansing company names could have synonyms and term-based relations defined. A DQS project uses a single KB; multiple projects can use the same KB.

A cleansing project has the following stages:

1. **Mapping** In this stage, you map source columns to KB domains.
2. **Computer-assisted cleansing** In this stage, DQS uses the KB with built-in algorithms and heuristics to find the best match of an instance of data you are cleansing to known data domain values.
3. **Interactive cleansing** In this stage, you review the results of the computer-assisted cleansing and additionally correct data. You see the changes that DQS proposes and decide whether to approve or reject the changes.

4. **Export** In this stage, you export the cleansed data. You can export the data to SQL Server tables or Excel files (.xlsx, .xls, and .csv). You can also standardize output if you defined standardized output in the appropriate domain of the knowledge base used. You can decide to export data only, or data and cleansing information. The data cleansing information includes source value, output value, reason for correction, confidence for correction, and the status of the operation performed on the data.

You can see and modify the status of the operation performed on the data during the interactive cleansing stage. Based on the confidence level and thresholds defined, the status can be one of the following:

■■ **Invalid** This status denotes values that DQS found as invalid because they do not comply with domain rules.

■■ **Corrected** This status denotes values that DQS corrected during the computerassisted cleansing because the confidence for the correction was above the minimal score for the autocorrections threshold.

■■ **Suggested** The values have a confidence level higher than the auto-suggestions threshold and lower level than the threshold for auto-corrections.

■■ **Correct** The values were found as correct. For example, a value matched a domain value in the knowledge base used for the project.

■■ **New** The value cannot be mapped to another status. However, the value complies with the domain rules. This means that this value is either a correct new value or a potentially incorrect value for which DQS does not have enough information to map it to another status, or that the confidence level is too low for mapping the value to corrected or suggested values. If you approve a new value, DQS moves it to the corrected values; if you reject it, DQS moves it to the invalid values.

Which are the two cleansing phases of a DQS project?
The two cleansing phases of a DQS project are computer-assisted cleansing and interactive cleansing.

Using DQS and the DQS Cleansing Transformation

The *DQS Cleansing transformation* is useful for cleansing batches of data inside the SSIS data flow. It should be a starting point for identity mapping and de-duplication, in order to minimize the number of rows that need approximate matching. You can use DQS to do the matching by creating a matching policy knowledge base.

DQS Cleansing Transformation

The DQS Cleansing transformation uses Data Quality Services (DQS) to correct data. You need to have a DQS knowledge base created in advance. You connect to a data source, select the columns to correct, and map the columns to the appropriate DQS KB domains.

In order to map the source columns to a KB domain, you need to create a connection to your DQS service. Then you select the knowledge base. Remember that you can have a single domain or composite domains. If you map multiple columns to each column of a composite domain, then the

composite domain rules are applied. If you don't map to all domains from a composite domain, then single domain rules are applied, and each column is processed and cleansed individually.

Some advanced configuration options for the DQS Cleansing transformations are also available:
■■ **Standardize output** Use this option to standardize the output as you define in domain settings. You can standardize strings to uppercase or lowercase, or capitalize each word. In addition, the values are standardized to the leading value.
■■ **Confidence** If you select this option, you also get the confidence level for a correction or a suggestion.
■■ **Reason** You can include the reason for a correction or a suggestion.
■■ **Appended data** This setting is valid only if you are using a reference data provider. Some reference data providers can include additional information, such as geographic coordinates when you check an address. You get this additional information in the field called Appended Data.
■■ **Appended data schema** If you append a reference provider's data, you get information about the schema for this data in this field.

The DQS Cleansing transformation, like other transformations, includes an error output allowing you to handle potential row-level errors.

Remember that you should cleanse your data before matching.

Using Data Quality Service to match data (DQS Matching)

The Data Quality Services (DQS) data matching process enables you to reduce data duplication and improve data accuracy in a data source. Matching analyzes the degree of duplication in all records of a single data source, returning weighted probabilities of a match between each set of records compared. You can then decide which records are matches and take the appropriate action on the source data.

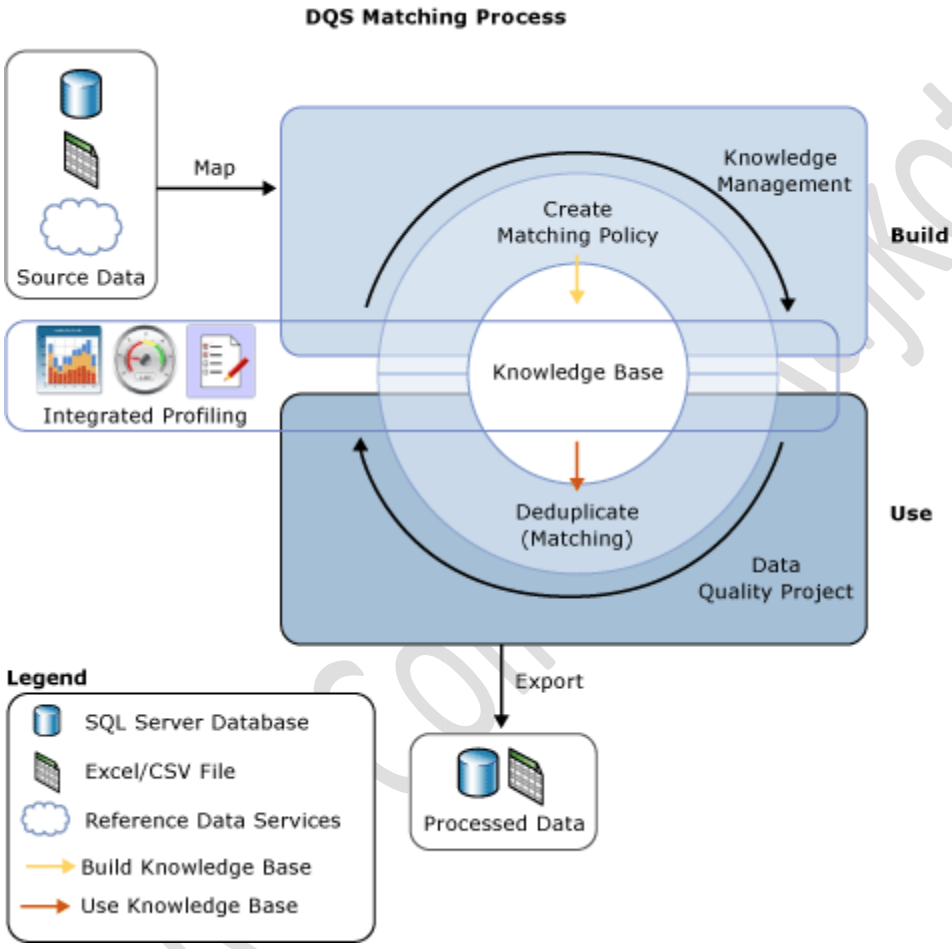
The DQS matching process has the following benefits:

- Matching enables you to eliminate differences between data values that should be equal, determining the correct value and reducing the errors that data differences can cause. For example, names and addresses are often the identifying data for a data source, particularly customer data, but the data can become dirty and deteriorate over time. Performing matching to identify and correct these errors can make data use and maintenance much easier.
- Matching enables you to ensure that values that are equivalent, but were entered in a different format or style, are rendered uniform.
- Matching identifies exact and approximate matches, enabling you to remove duplicate data as you define it. You define the point at which an approximate match is in fact a match. You define which fields are assessed for matching, and which are not.
- DQS enables you to create a matching policy using a computer-assisted process, modify it interactively based upon matching results, and add it to a knowledge base that is reusable.

- You can re-index data copied from the source to the staging table, or not re-index, depending on the state of the matching policy and the source data. Not re-indexing can improve performance.

You can perform the matching process in conjunction with other data cleansing processes to improve overall data quality. You can also perform data de-duplication using DQS functionality built into Master Data Services. For more information, see Master Data Services Overview (MDS).

The following illustration displays how data matching is done in DQS:



How to Perform Data Matching

As with other data quality processes in DQS, you perform matching by building a knowledge base and executing a matching activity in a data quality project in the following steps:

1. Create a matching policy in the knowledge base
2. Perform a de-duplication process in a matching activity that is part of a data quality project.

Building a Matching Policy

You prepare the knowledge base for performing matching by creating a matching policy in the knowledge base to define how DQS assigns matching probability. A matching policy consists of one or more matching rules that identify which domains will be used when DQS assesses how well one record matches to another, and specify the weight that each domain value carries in the matching assessment. You specify in the rule whether domain values have to be an exact match or

can just be similar, and to what degree of similarity. You also specify whether a domain match is a prerequisite.

The matching policy activity in the Knowledge Base Management wizard analyzes sample data by applying each matching rule to compare two records at a time throughout the range of records. Records whose matching scores are greater than a specified minimum are grouped in clusters in the matching results. These matching results are not added to the knowledge base; you use them to tune the matching rules. Creating a matching policy can be an iterative process in which you modify matching rules based on the matching results or profiling statistics.

You can specify for a domain that data strings will be normalized when you load data from the data source into the domain. This process consists of replacing special characters with a null or a space, which often removes the difference between two strings. This can increase matching accuracy, and can often enable a matching result to surpass the minimum matching threshold, when without normalization it would not pass. **Note**

Null values in the corresponding fields of two records will be considered a match.

The matching policy is run on domains mapped to the sample data. You can specify whether data is copied from the data source into the staging table and re-indexed when you run the matching policy, or not. You can do so both when building the knowledge base and when running the matching project. Not re-indexing could result in improved performance. Re-indexing is not necessary if the following is true: the matching policy has not changed, and you have not updated the data source, remapped the policy, selected a new data source, or mapped one or more new domains.

Each matching rule is saved in the knowledge base when it is created. However, a knowledge base is available for use in a data quality project only when it is published. In addition, until the knowledge base is published, the matching rules in it cannot be changed by a user other than the person who created it.

Running a Matching Project

DQS performs data de-duplication by comparing each row in the source data to every other row, using the matching policy defined in the knowledge base, and producing a probability that the rows are a match. This is done in a data quality project with a type of Matching. Matching is one of the major steps in a data quality project. It is best performed after data cleansing, so that the data to be matched is free from error. Before running a matching process, you can export the results of the cleansing project into a data table or .csv file, and then create a matching project in which you map the cleansing results to domains in the matching project.

A data matching project consists of a computer-assisted process and an interactive process. The matching project applies the matching rules in the matching policy to the data source to be assessed. This process assesses the likelihood that any two rows are matches in a matching score. Only those records with a probability of a match greater than a value set by the data steward in the matching policy will be considered a match.

When DQS performs the matching analysis, it creates clusters of records that DQS considers matches. DQS randomly identifies one of the records in each cluster as the pivot, or leading, record. The data steward verifies the matching results, and rejects any record that is not an appropriate match for a cluster. The data steward then selects a survivorship rule that DQS will use to determine the record that will survive the matching process and replace the matching records. The survivorship rule can be "Pivot record" (the default), "most complete and longest record", "most complete record", or "longest record". DQS determines the survivor (leading) record in each cluster based upon which record most closely matches the criteria or criterion in the survivorship rule. If multiple records in a given cluster comply with the survivorship rule, DQS selects one of those records randomly. DQS gives you the choice of displaying clusters that have records in common as a single cluster by selecting "show non-overlapping clusters". You must execute the matching process in order to display the results according to this setting.

You can export the results of the matching process either to a SQL Server table or a .csv file. You can export matching results in two forms: first, the matched records and the unmatched records, or second, survivorship records that include only the survivor record for a cluster and the unmatched results. In the survivorship records, if the same record is identified as the survivor for multiple clusters, which record will only be exported once.

Following are steps for matching:

To use DQS to match data, you must first add a *matching policy* to a knowledge base. You can use an existing knowledge base that is also used for data cleansing, or you can create a knowledge base specifically for data matching. In this example, I'm opening an existing knowledge base that contains domains for customer records for the **Matching Policy** activity.

Just as when performing knowledge discovery, I need to map some sample data to the domains defined in the knowledge base. This enables me to test the matching policy against a known data set as I build it, and therefore verify that it successfully identifies known duplicate records. In this case, I'm using data in an Excel workbook as the source for my sample data, but you can also use a table in a SQL Server database.

Having mapped sample data to the domains, I can now define the matching rules for my matching policy. You can include multiple rules, and each one uses a set of weighted comparisons of domain values to identify clusters of records that are potential duplicates of one another.

Potential matches are determines based on a score that is calculated from the weighted comparisons you define in the rule. Here are the comparisons I've used in my **Match Customer** rule:

Domain	Similarity	Weight	Prerequisite
Birth Date	Exact		X
Email Address	Exact	20	
Postal Code	Exact	10	

Country/Region	Exact	10
First Name	Similar	10
Last Name	Similar	10
Street Address	Similar	20
City	Similar	10
State	Similar	10

Note that an exact match of the **Birth Date** domain is specified as a prerequisite. In other words, only records where the birth date is an exact match will be considered as candidates for a potential duplicate. Prerequisite domains in a matching rule must use the **Exact** similarity and have no weighting value. All of the other domains are calculated based on an exact or similar match, and have weightings, which add up to a total of 100.

Assuming the birth date for the records being compared is a match, DQS then makes the other comparisons defined in the matching rule and adds the specified weighting value for each comparison that is true to produce an overall score. For example, consider two records with identical **Birth Date** values being compared using the **Match Customer** rule defined above. If the **Email Address** domains for both records is an exact match, 20 is added to the score. If the **First Name** domains are similar (for example, ―Rob‖ and ―Robert‖), another 10 is added to the score, and so on until all of the comparisons in the rule have been made. The resulting score is then compared to the minimum matching score defined for the matching rule (in this case 80). If the score exceeds the minimum matching score, then the records are considered a match. Multiple records that are considered matches for one another are grouped into a cluster.

After you have defined the matching rules, you can use them to find matches in the sample data you mapped earlier. This gives you the opportunity to verify that the rules behave as expected against a known dataset. In this case, the dataset results in a single cluster of matches that includes two records – one for Daniel Garcia and another for Dan Garcia.

Now that I‘ve defined my matching policy, I can publish the knowledge base and allow the data stewards in my organization to use it for data matching.

To use a knowledge base to perform data matching, create a new data quality project, specify the knowledge base, and specify the **Matching** activity as shown here.

The first step, as it is in any DQS project, is to map the fields in your data source to the domains in the knowledge base. Just as before, the data source can be a table in a SQL Server database or an Excel file. This time, I‘m using the **Customers** table in the **Staging** SQL Server database.

After you‘ve mapped the domains, you can start the matching process. When the process is complete, the clusters of matched records is displayed. In this case, there are two clusters, each containing two matches. At tis stage, you can choose to reject any matches that you know aren‘t duplicates.
When the matches have all been identified, you can export the results to a SQL Server table or an Excel file. You can also export survivors (one record from each cluster that is chosen as the correct one) based on one of the following survivorship rules:

- **Pivot record** – A record in the cluster that is chosen arbitrarily by DQS.
- **Most complete and longest record** – The record that has the fewest null field values and the longest overall data length.
- **Most complete record** – The record that has the fewest null fields.
- **Longest record** – The record that has the longest overall data length.

The exported results include all of the source data with additional columns for the clusters of matching records to indicate the matching rule used and score calculated for each match, and the pivot record for each match cluster.

The exported survivors contain all of the non-matching records from the original data source and one version of each matched record based on the survivorship rule you selected. In the following example, I've highlighted the surviving records from my matching process.

In some case, you can simply replace the original data set with the survivor records to create a de-duplicated set of records. However, in most business scenarios you'll need to apply some logic (manual or automated) to handle relationships between duplicate records and other tables. For example, before I eliminate the duplicate customer records identified by the matching process in the above example, I would need to reassign any sales orders that are currently related to those customer records to the surviving records.

How can you influence which column in a table should be more important for matching than other columns?
When creating a matching KB, you can define weight for each domain. Give higher weight to the domain mapped to the column you want to be more important for matching.

In this practice, you will use the DQS Cleansing transformation. Then you will create a DQS matching policy KB and use it to perform de-duplication.

Using Scripts in SSIS

Script tasks are a great way of extending SSIS functionality, when the buit-in functionality isn't quite Script tasks are a great way of extending SSIS functionality, when none of the buit-in components are quite right for the task you need to perform. But how to go about creating a script task? No worries, once again Robert Sheldon is on hand to provide easy instructions on how to do it.

One of the most effective ways to extend your SQL Server Integration Services (SSIS) control flow is to use a **Script** task to write custom code that perform tasks you cannot perform with the built-in components. For example, you can use the **Script** task to access Active Directory information or to create package-specific performance counters. You can also use the **Script** task to combine functions that might normally require multiple tasks and data flow components.

The Script Task and Script Component have greatly increased your possibilities when it comes to script-based ETL development in SSIS. However, it is important to know when to use which

component and what things can be done in each. The following matrix explains when to use each component:

COMPONENT	WHEN TO USE
Script Task	This task is used in the Control Flow. Use this task when you need to program logic that either controls package execution or performs a task of retrieving or setting variables within a package during runtime.
Script Component	This component is used in the Data Flow. Use this component when moving data using the Data Flow Task. Here you can apply programmatic logic to massage, create, or consume data in the pipeline.

Script Task

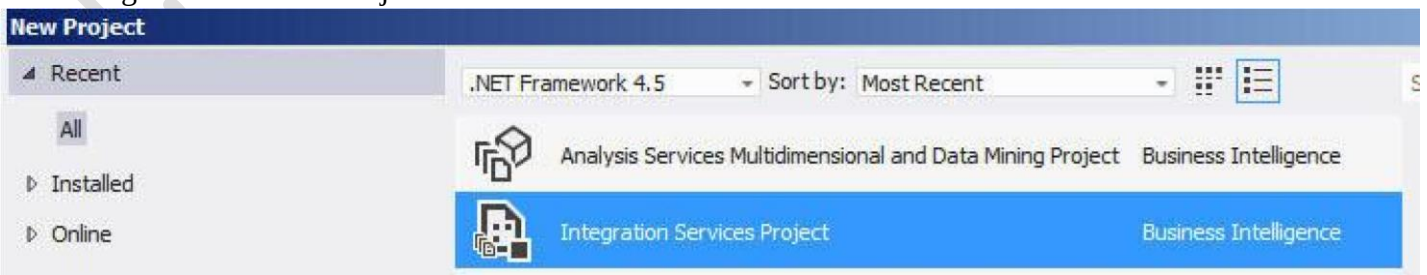
Example 1 - Hello World

Let's start with the Hello World example using a simple Script Task.

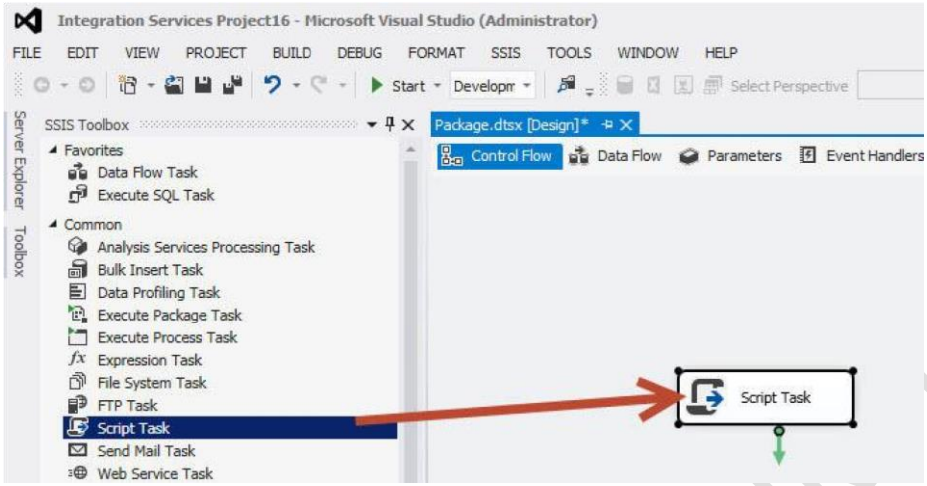
In order to start, open the SQL Server Data Tools for Visual Studio.

Go to File > New > Project

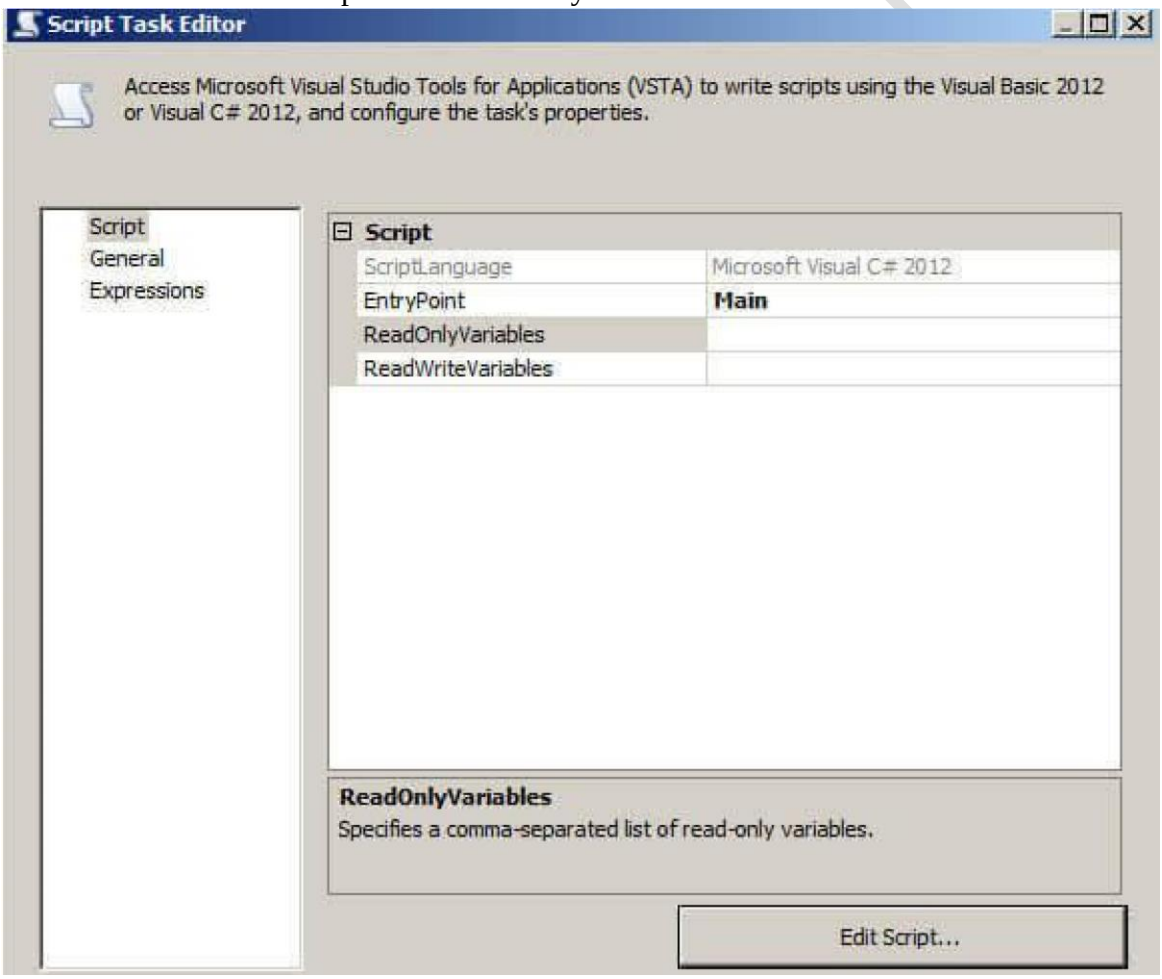
Select Integration Services Project.



Drag and drop the Script Task to the design pane and double click on it.



The following window will open. The ScriptLanguage is used to select which language to use, either C# or Visual Basic. EntryPoint is used to select where to start in the code, by default it starts in Main. Press the Edit Script button to write your code.

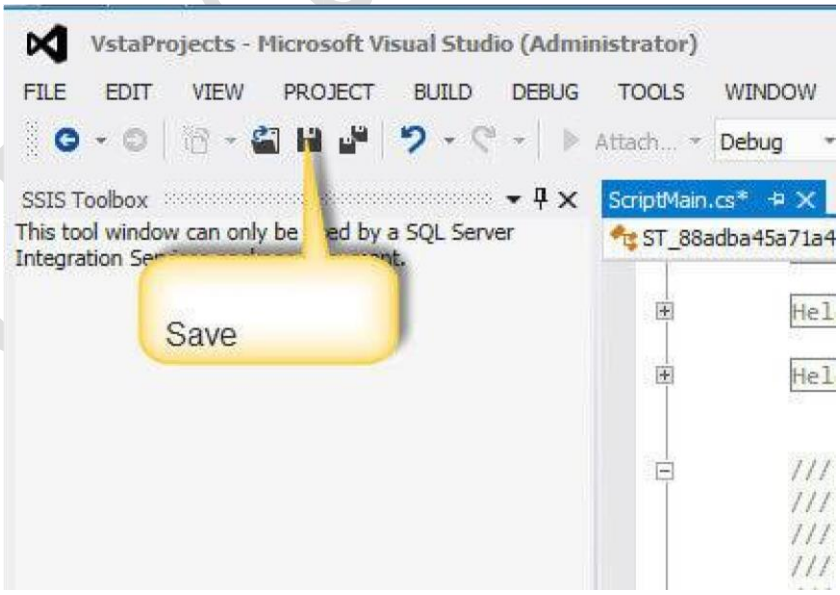


A new Window will be displayed to allow you to write the code. Go to the main procedure, by default you will create your code there.

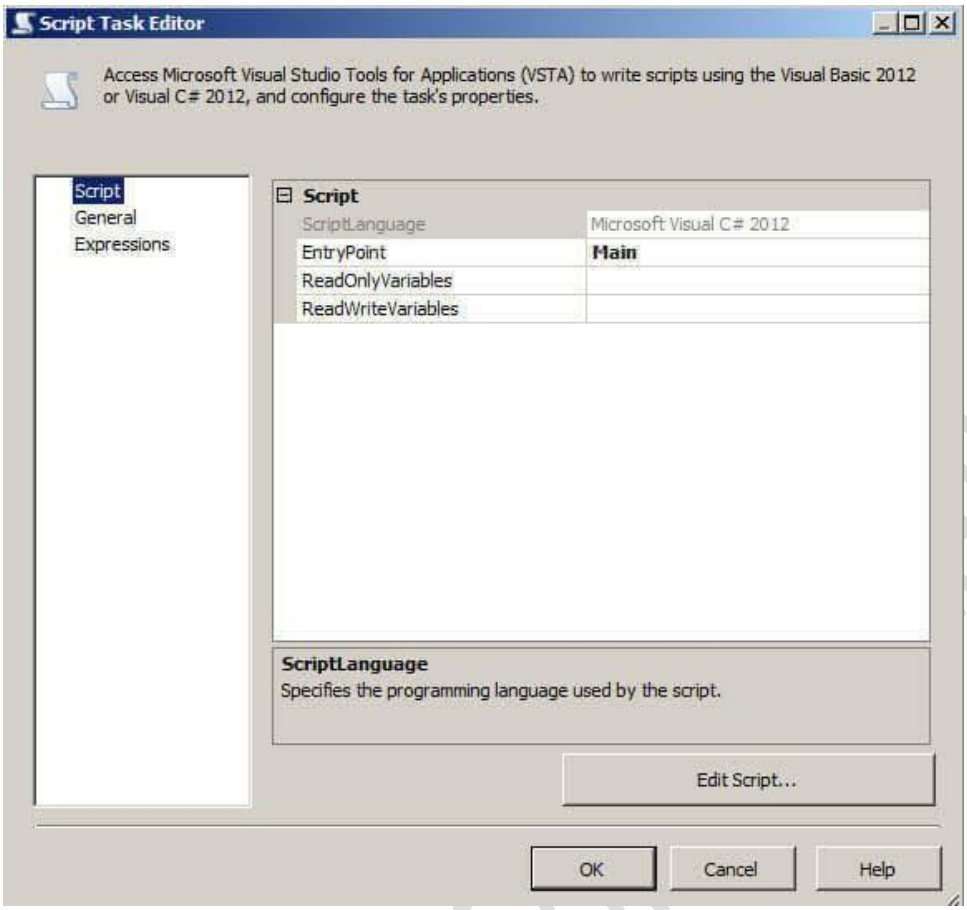
Add the following code in the Main section. This code will display a message with the Hello World message.

```
public void Main()
{
   // TODO: Add your code here
   MessageBox.Show("Hello World"); }
```

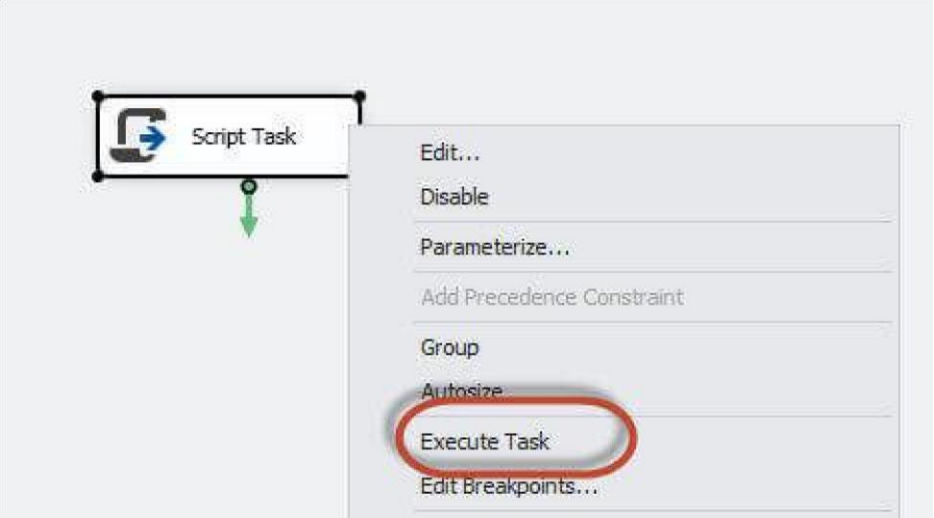
Save the code.



In the Script Task Editor, press OK.



Right click on the Script Task and select the Execute Task option.



If everything is done correctly, you will receive the following pop-up message:

Once finished, you can stop the package as shown below.

Example 2 - Help, Variables and Handling Errors

In this example, we will show how to work with variables and how to handle errors. Open the Script Task and press the Edit script button.

There are regions that can be expanded. For example, the Help introduction explains how to use the help.

```
#region Help:  Introduction to the script task
/* The Script Task allows you to perform virtually any operation that can be accomplished in
 * a .Net application within the context of an Integration Services control flow.
 *
 * Expand the other regions which have "Help" prefixes for examples of specific ways to use
 * Integration Services features within this script task. */
#endregion
```

By default, there are help samples to use SSIS variables, parameters, firing events and using the connection managers. In this tip, we will work with variables. In future tip, we will talk about firing events and the connection manager. Expand the help using the Integration Services variables tree as shown below.

```
[Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
public partial class ScriptMain : Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
{
    Help:  Using Integration Services variables and parameters in a script

    Help:  Firing Integration Services events from a script

    Help:  Using Integration Services connection managers in a script
```

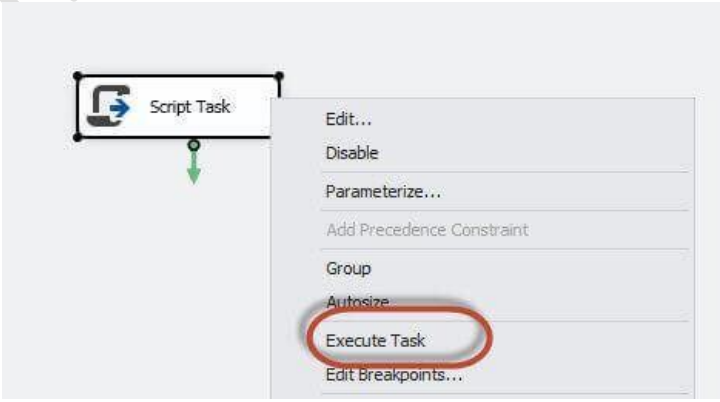
There are samples to save SSIS variable values in a C# or VB variable. There are also samples to save values in an SSIS variable. There are also similar samples to work with parameters. Parameters are a new feature in SQL Server 2012 and they can work at the project level.

```
/* To use a variable in this script, first ensure that the variable has been added to
 * either the list contained in the ReadOnlyVariables property or the list contained in
 * the ReadWriteVariables property of this script task, according to whether or not your
 * code needs to write to the variable.  To add the variable, save this script, close this instance of
 * Visual Studio, and update the ReadOnlyVariables and
 * ReadWriteVariables properties in the Script Transformation Editor window.
 * To use a parameter in this script, follow the same steps. Parameters are always read-only.
 *
 * Example of reading from a variable:
 *  DateTime startTime = (DateTime) Dts.Variables["System::StartTime"].Value;
 *
 * Example of writing to a variable:
 *  Dts.Variables["User::myStringVariable"].Value = "new value";
 *
 * Example of reading from a package parameter:
 *  int batchId = (int) Dts.Variables["$Package::batchId"].Value;
 *
 * Example of reading from a project parameter:
 *  int batchId = (int) Dts.Variables["$Project::batchId"].Value;
 *
 * Example of reading from a sensitive project parameter:
 *  int batchId = (int) Dts.Variables["$Project::batchId"].GetSensitiveValue();
 * */
```

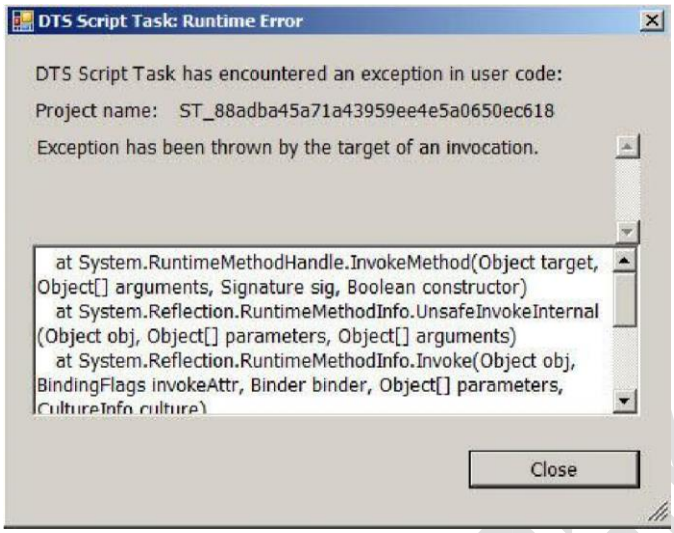
In this example, we are going to display the system time from a variable. Use the code below. The code is assigned to the C# variable named startTime from the SSIS system variable **system::Starttime.** The variable is converted from DateTime to Text using the ToString function and then displayed in a MessageBox. Finally, the TaskResult shows the task as a success (the green color in SSIS).

```
DateTime startTime = (DateTime)Dts.Variables["System::StartTime"].Value;
MessageBox.Show(startTime.ToString());
Dts.TaskResult = (int)ScriptResults.Success;
```

Save the script and close it and then right click the Script Task and execute the task.



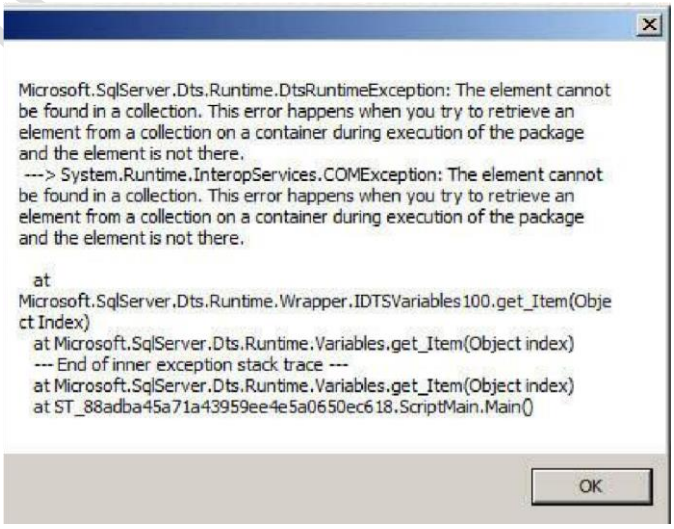
You will receive an error similar to this one: DTS Script Task has encountered an exception in user code: Project name: xxxxx Exception has been thrown by the target of an invocation.



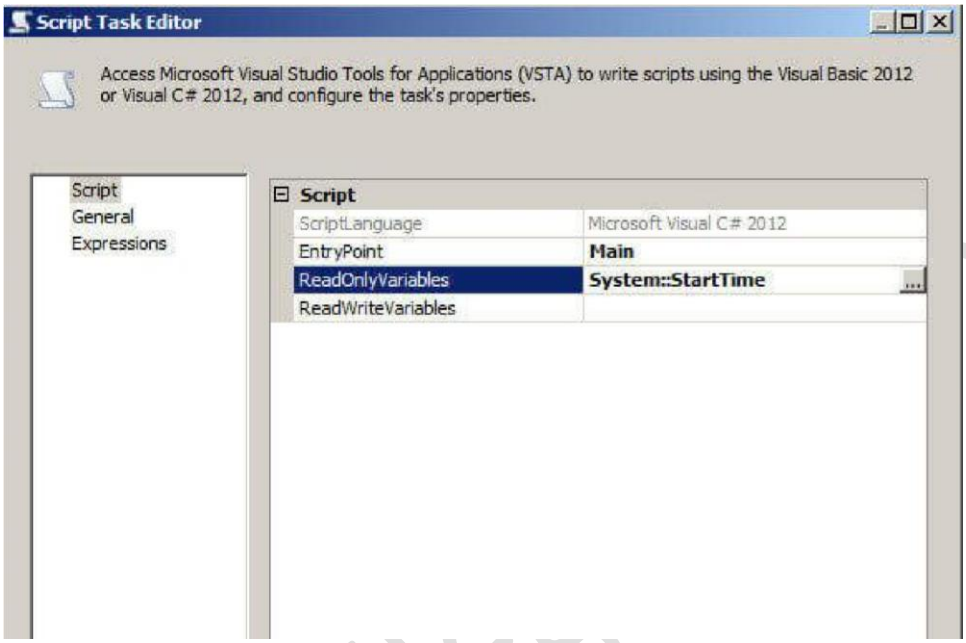
By default, error messages are not really descriptive. That is why it is a good practice to use try and catch logic in your code. The try block contains the code to be executed and the catch code handles the exception. Stop the task and modify the code as follows:

```
try {
  DateTime startTime = (DateTime)Dts.Variables["System::StartTime"].Value;
  MessageBox.Show(startTime.ToString());
  Dts.TaskResult = (int)ScriptResults.Success;
}
catch (Exception ex)
{
  MessageBox.Show(ex.ToString());
  Dts.TaskResult = (int)ScriptResults.Failure; }
```

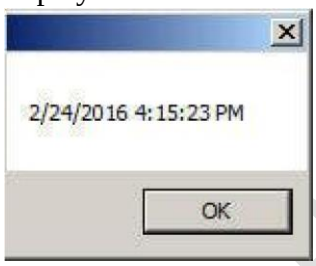
Save the script, close it and execute again. You will receive the following error message this time. The error message is now more descriptive. It says that the element cannot be found and this item is a variable. The error is that we did not include the variable in the Script Task.



Stop the task and double click on the Script Task to make some changes. Add the **System:Starttime** in the ReadOnlyVariables property. This variable is a system variable used to get the time when the package started to run.

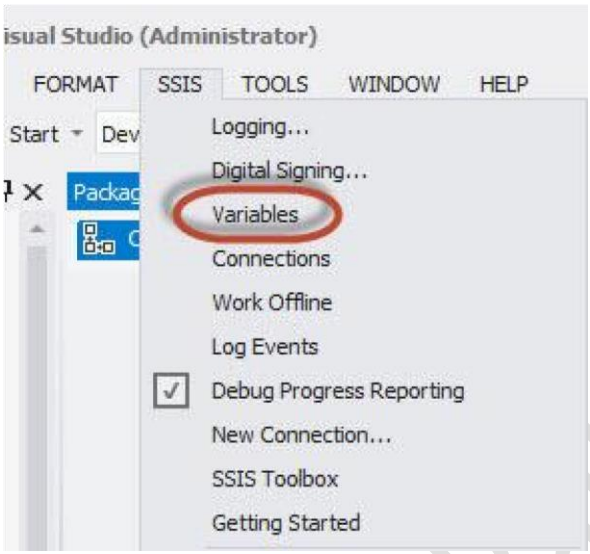


Press OK on the Script Task and execute the task again. If everything is done correctly, a MessageBox will be displayed with the start time of the package as shown below.

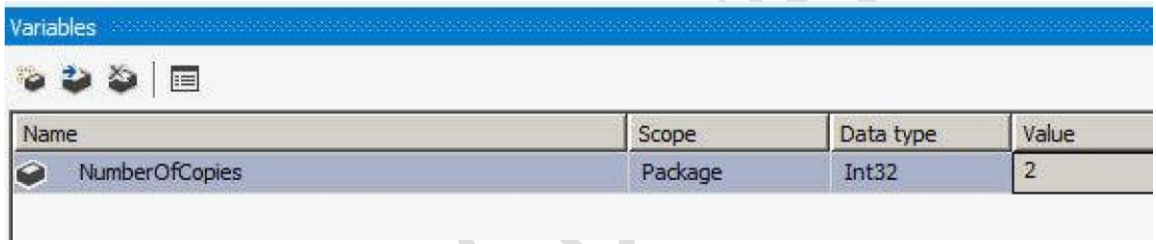


Example 3 - User Variables and Loops

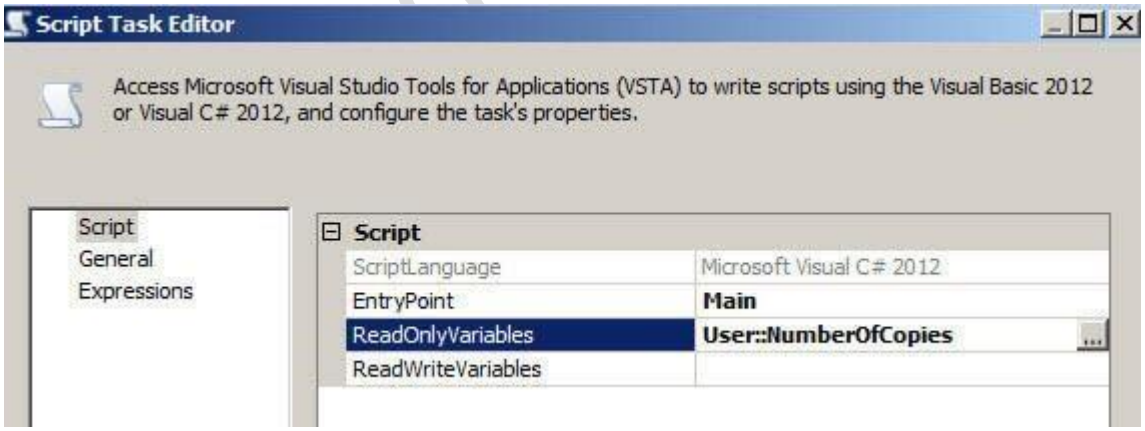
In this example, we are going to create a variable, assign a number and create several copies of a SQL Server backup. For example, if we assign a value of 100, it will create 100 copies. In order to start, we are going to create a new SSIS user variable. Go to the menu, **SSIS > Variables** as shown below.



Create a variable named **NumberOfCopies** of type Int32 (integer) and assign a value. In this example, we are specifying the number 2 to create two copies of the SQL Server backup.



Double click on the Script Task and select the new variable we created for the ReadOnlyVariables section as shown below.



Edit the code in the Script Task and expand the Namespaces region and add the System.IO namespace. This namespace is used to copy, replace, read and write files, show directory and file information as well as other functions:

```
using System.IO;
```

```
#region Namespaces
using System;
using System.IO;
using System.Data;
using Microsoft.SqlServer.Dts.Runtime;
using System.Windows.Forms;
#endregion
```

Add the following code to the script in the Main section which will create copies of the backup files:

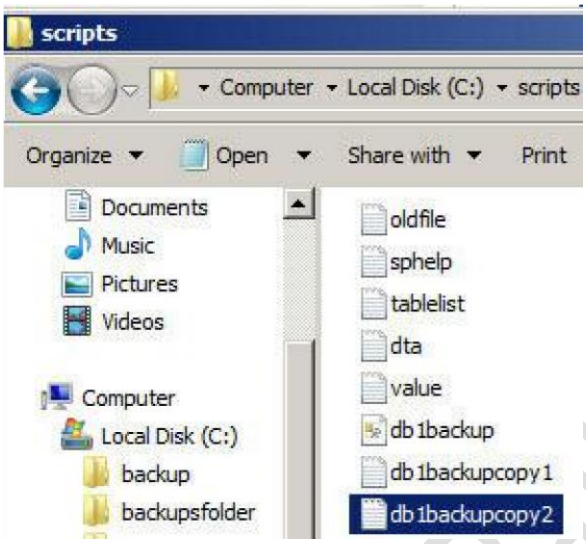
```
try {
   int numberofcopies = Convert.ToInt16(Dts.Variables ["User::NumberOfCopies"].Value);
for (int i = 1; i <= numberofcopies; i++)
   {
     File.Copy(@"C:\scripts\db1backup.bak", @"C:\scripts\db1backupcopy" + i.ToString() +
".bak");
   }

   Dts.TaskResult = (int)ScriptResults.Success; }

catch (Exception ex)
{
   MessageBox.Show(ex.ToString());
   Dts.TaskResult = (int)ScriptResults.Failure; }
```

The code stores the values of the SSIS variable in a C# variable. We use the Convert.ToInt16 function to convert the SSIS variable value to a integer in C#. The for (int i = 1; i <= numberofcopies; i++) is a loop used to copy the backup a specified number of times defined by the NumberOfCopies value. If the NumberOfCopies is 100, it will create 100 copies. Finally, the File.Copy function will create copies with a number at the end of the file name. For example, if I create 3 copies the files will be named**db1backupcopy1.bak**, **db1backupcopy2.bak, db1backupcopy3.bak**.

Save the code and close it and accept the changes and execute the Script Task. As you can see, two copies of the backup were created.

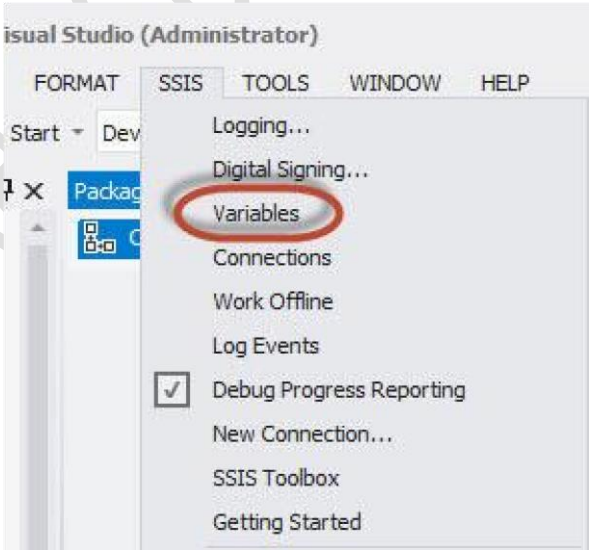


Conclusion

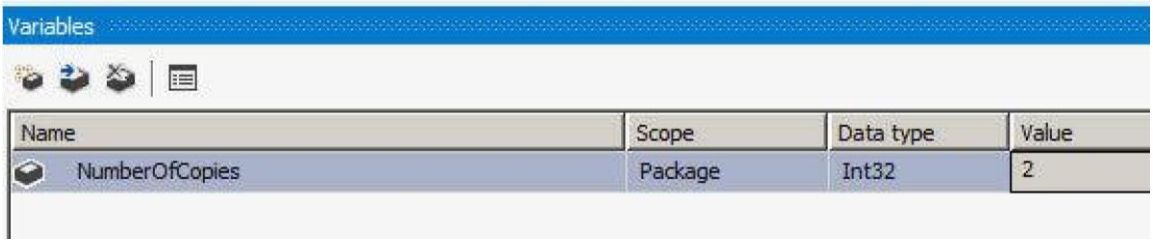
The Script Task is a very useful and powerful tool to accomplish whatever you need in your daily tasks.

Example 3 - User Variables and Loops

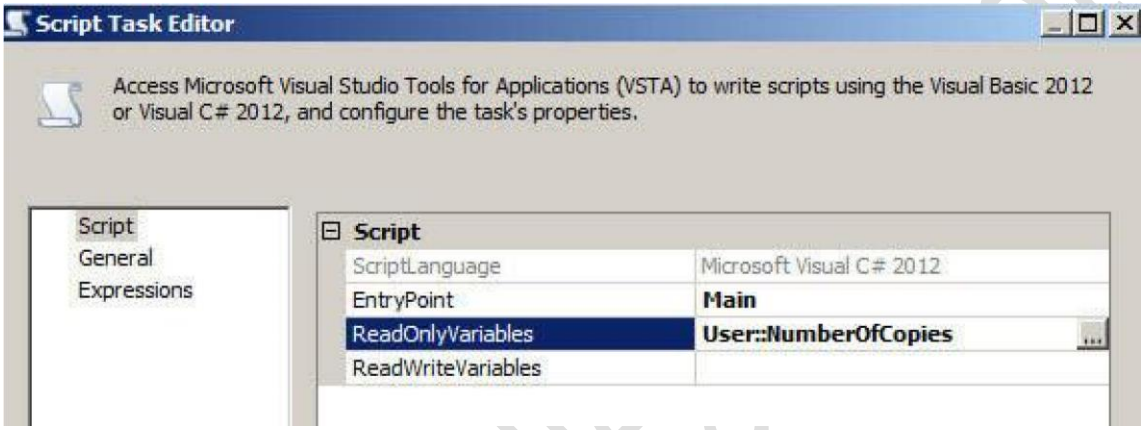
In this example, we are going to create a variable, assign a number and create several copies of a SQL Server backup. For example, if we assign a value of 100, it will create 100 copies. In order to start, we are going to create a new SSIS user variable. Go to the menu, **SSIS > Variables** as shown below.



Create a variable named **NumberOfCopies** of type Int32 (integer) and assign a value. In this example, we are specifying the number 2 to create two copies of the SQL Server backup.



Double click on the Script Task and select the new variable we created for the ReadOnlyVariables section as shown below.



Edit the code in the Script Task and expand the Namespaces region and add the System.IO namespace. This namespace is used to copy, replace, read and write files, show directory and file information as well as other functions:

```
using System.IO;
```

Add the following code to the script in the Main section which will create copies of the backup files:

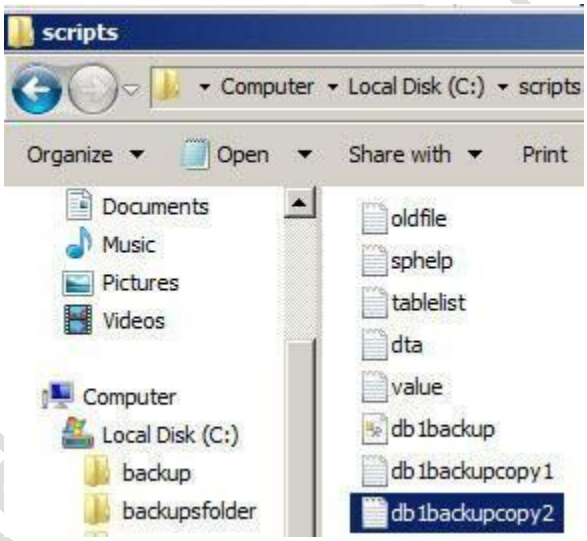
```
try
{
   int numberofcopies = Convert.ToInt16(Dts.Variables ["User::NumberOfCopies"].Value);
for (int i = 1; i <= numberofcopies; i++)
   {
      File.Copy(@"C:\scripts\db1backup.bak", @"C:\scripts\db1backupcopy" + i.ToString() +
".bak");
   }
```

```
   Dts.TaskResult = (int)ScriptResults.Success; }

catch (Exception ex)
{
   MessageBox.Show(ex.ToString());
   Dts.TaskResult = (int)ScriptResults.Failure; }
```

The code stores the values of the SSIS variable in a C# variable. We use the Convert.ToInt16 function to convert the SSIS variable value to a integer in C#. The for (int i = 1; i <= numberofcopies; i++) is a loop used to copy the backup a specified number of times defined by the NumberOfCopies value. If the NumberOfCopies is 100, it will create 100 copies. Finally, the File.Copy function will create copies with a number at the end of the file name. For example, if I create 3 copies the files will be named **db1backupcopy1.bak**, **db1backupcopy2.bak, db1backupcopy3.bak**.

Save the code and close it and accept the changes and execute the Script Task. As you can see, two copies of the backup were created.



The SSIS Script Component is one of the most important, and powerful component in SQL Server Integration Services. It can act as a Source, Transformation, and Destination.

Before we start creating the SSIS package, Let us see the SQL table and the data that we are going to use.

```
USE [SSIS Tutorials]
GO

SELECT [FirstName]
      ,[LastName]
      ,[Education]
      ,[Occupation]
      ,[YearlyIncome]
      ,[Sales]
  FROM [MyEmployees]
```

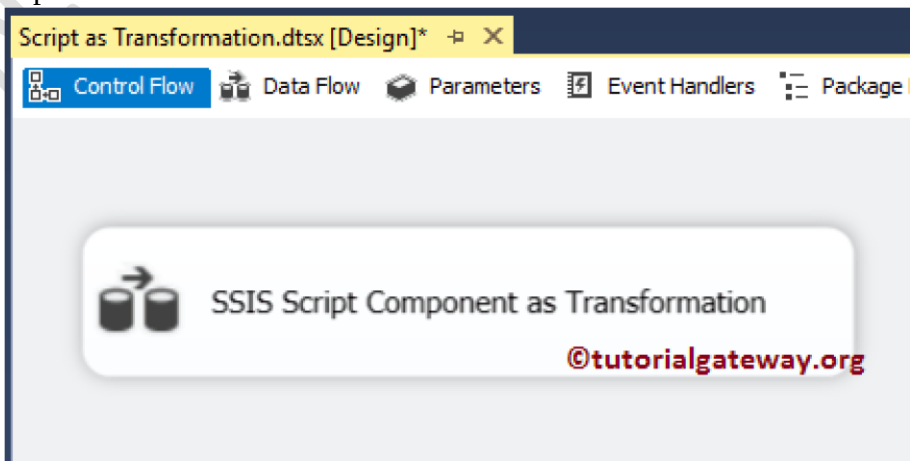
100 %

Results | Messages

	FirstName	LastName	Education	Occupation	YearlyIncome	Sales
1	tutorial	gateway	Masters Degree	Management	90000	3578.27
2	rob	johnson	Bachelors	Management	80000	3399.99
3	ruben	torres	Partial College	Skilled Manual	50000	699.0982
4	christy	zhu	Bachelors	Professional	80000	3078.27
5	rob	huang	High School	Skilled Manual	60000	2319.99
6	john	ruiz	Bachelors	Professional	70000	539.99
7	john	yang	Bachelors	Professional	80000	2320.49
8	christy	mehta	Partial High School	Clerical	50000	24.99
9	rob	verhoff	Partial High School	Clerical	45000	24.99
10	christy	carlson	Graduate Degree	Management	70000	2234.99
11	gail	erickson	Education	Professional	90000	4319.99
12	barry	johnson	Education	Management	80000	4968.59
13	peter	krebs	Graduate Degree	Clerical	50000	59.53
14	greg	alderson	Partial High School	Clerical	45000	23.5

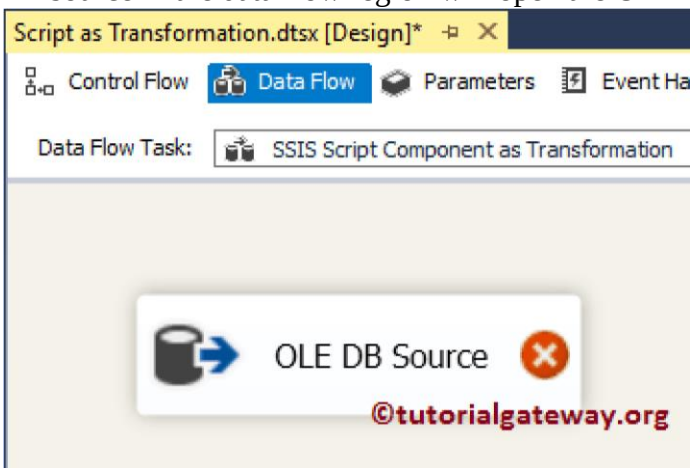
Configuring SSIS Script Component as Transformation

STEP 1: Drag and drop the Data Flow Task from toolbox to control flow region, and rename it as the SSIS Script Component as Transformation.

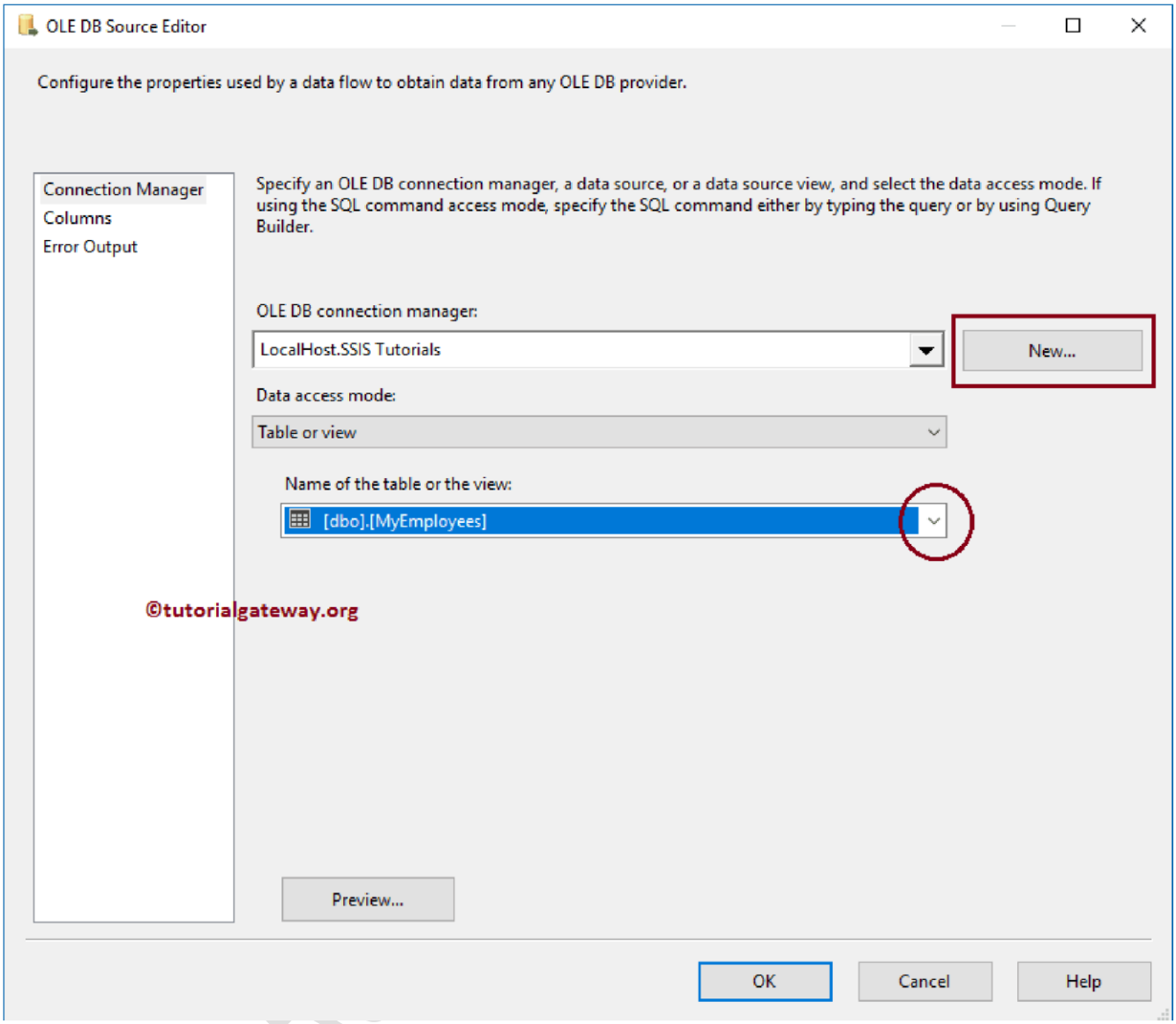


Double click on the data flow task will open the data flow tab.

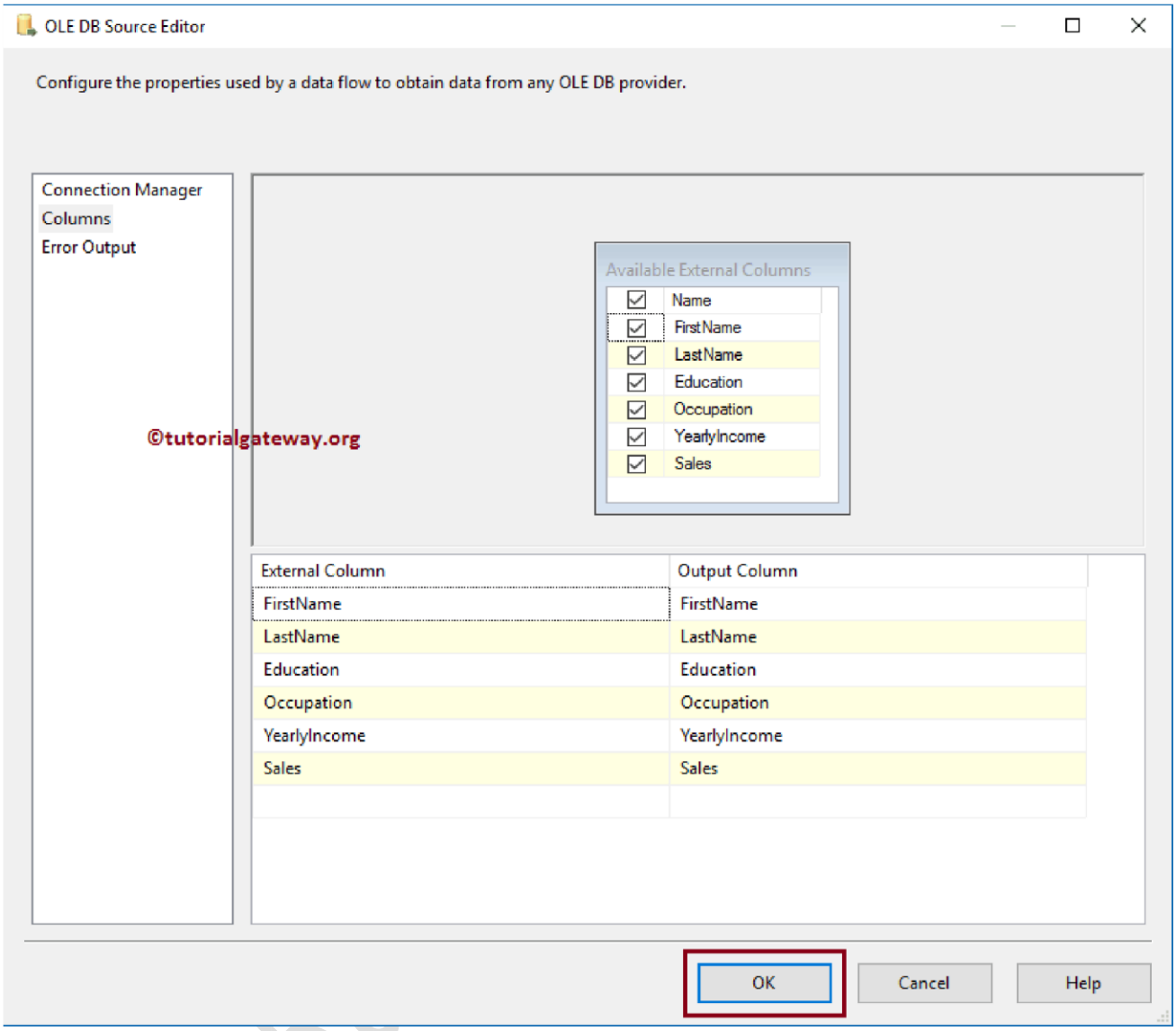
STEP 2: Drag and drop OLE DB Source from toolbox to data flow region. Double click on OLE DB source in the data flow region will open the OLE DB Connection Manager settings



From the below screenshot you can observe that, We selected [SSIS Tutorials] Database as source database and [MyEmployees] table as source table



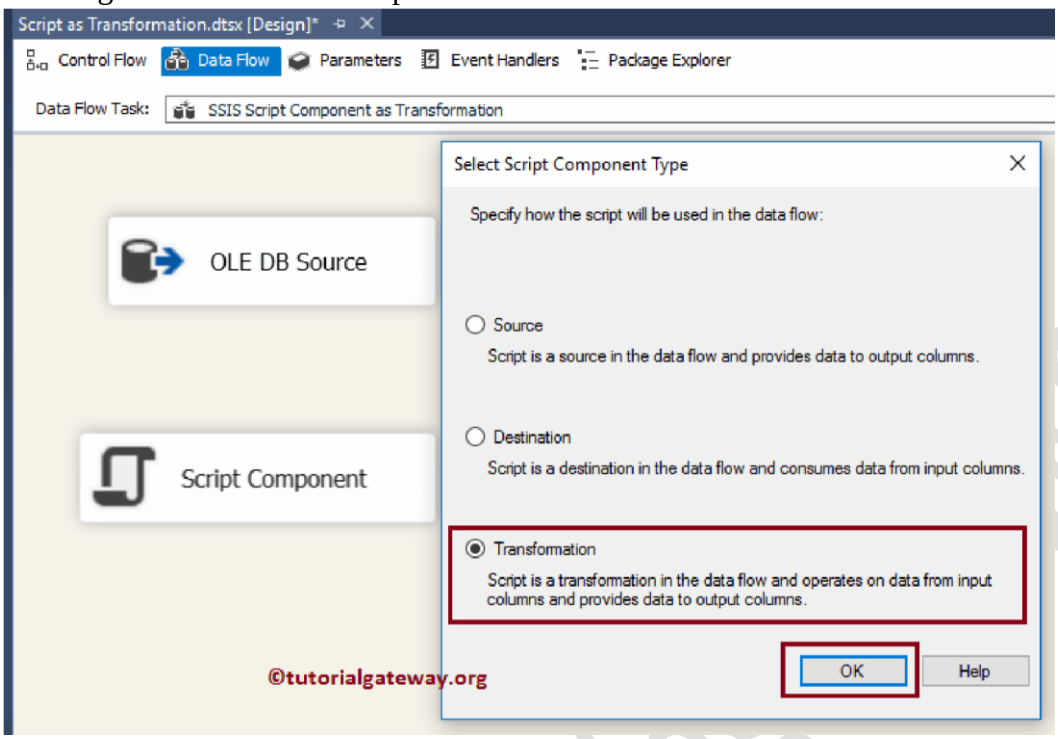
STEP 3: Click on columns tab to verify the columns. In this tab we can uncheck the unwanted columns also.



STEP 4: Drag and drop Script Component in SSIS toolbox to data flow region. Once you drop the Script component, a new pop up window called Select Script Content Type will be opened as shown below.

Here we want to demonstrate about the ssis script component as a transformation. So, we are

selecting the Transformation option



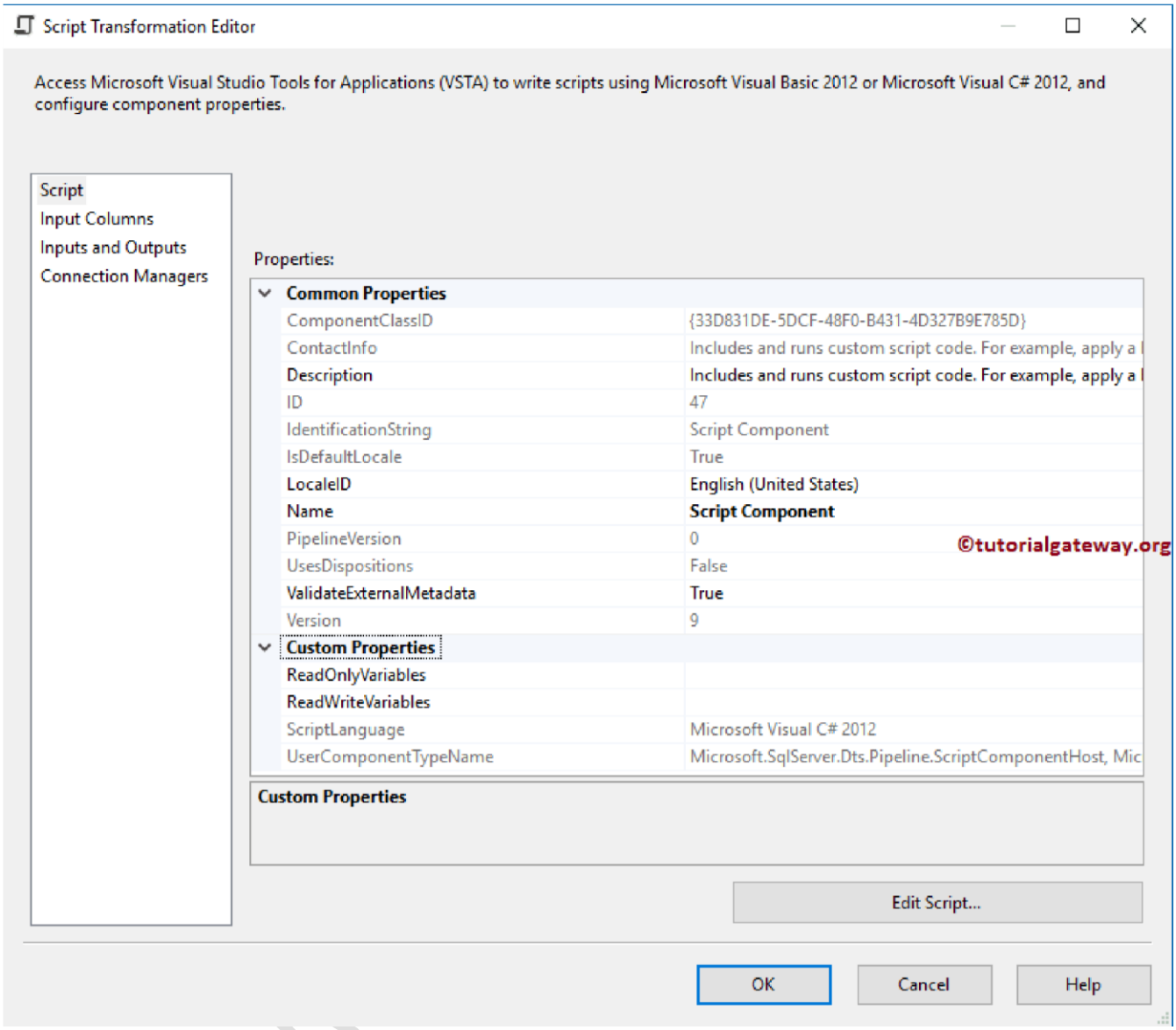
STEP 5: Double click on the SSIS Script component will open the following editor to configure the properties. Though there are many properties, we will explore few important properties that we use in our daily coding

Name: Please provide the Unique Name

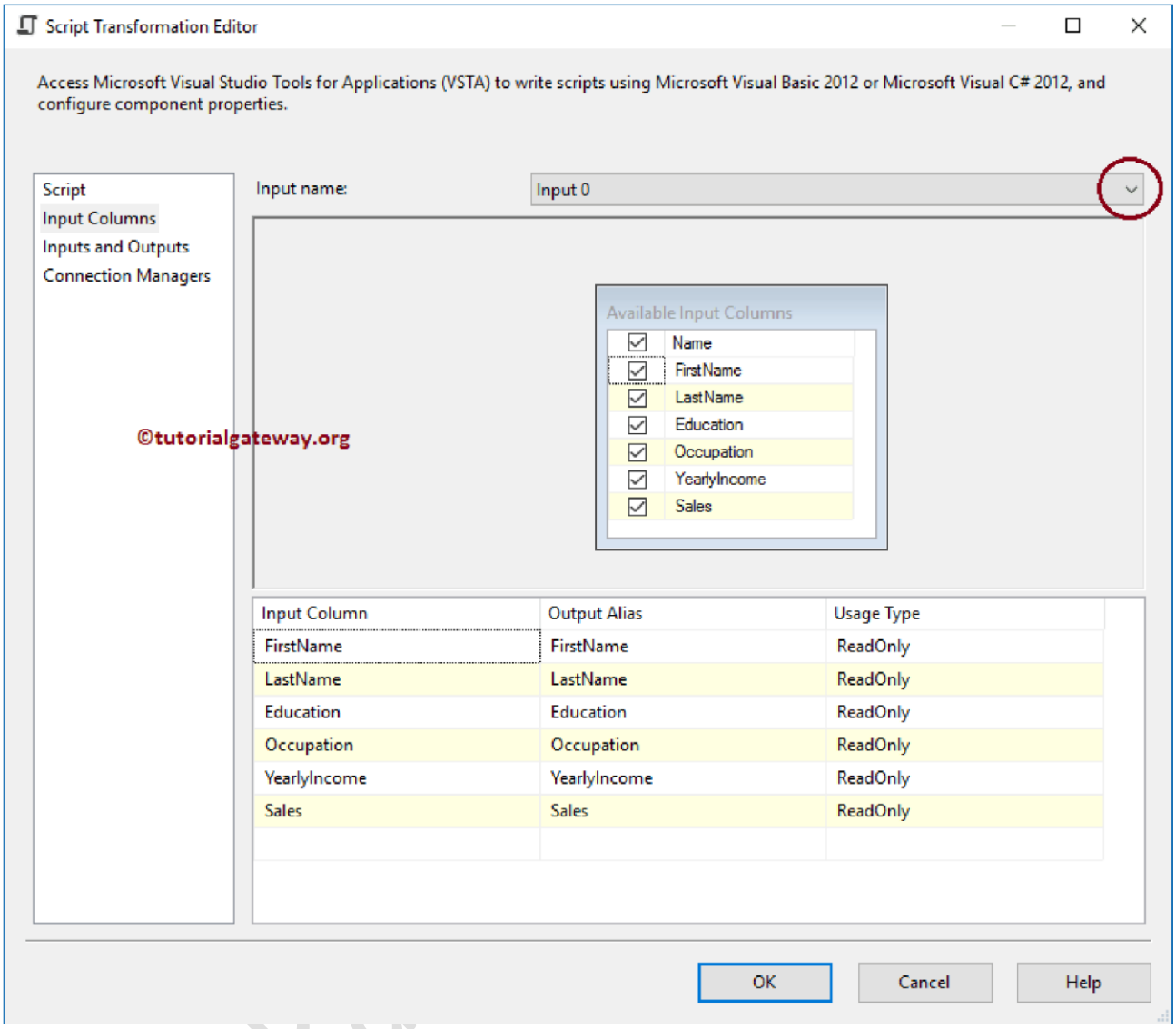
Description: Briefly describe the Script Functionality. It is always a good practice to provide the valid description.

ReadOnlyVariables: Please select the variables that you want to use in the Script, and they may be user defined variables or System default variables. Remember, variables selected as ReadOnlyVariables are used for Read-only purpose (we can‘t alter them)

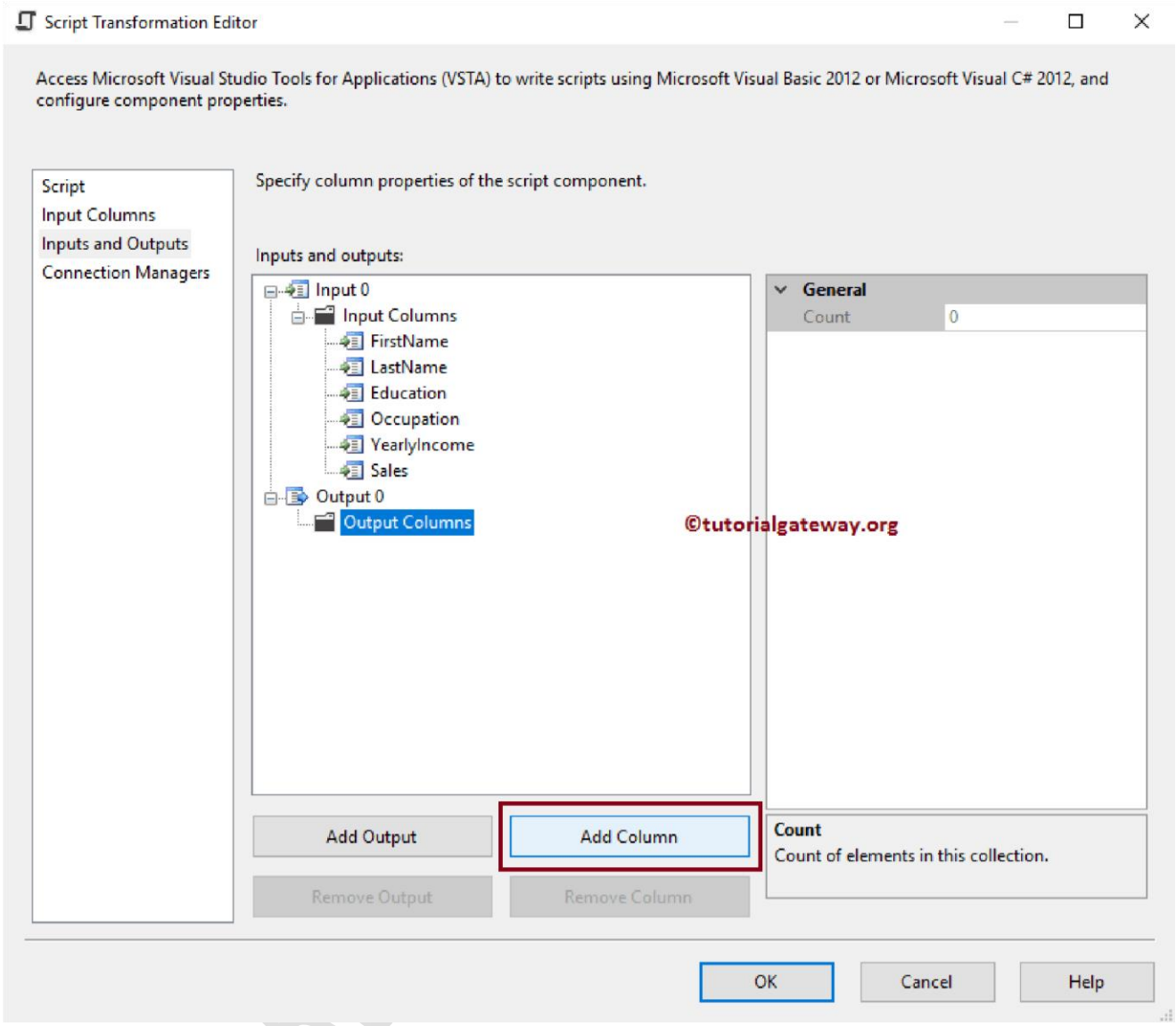
ReadWriteVariables: Please select the variables you want to use in the Script. Remember, variables selected as ReadWriteVariables can be altered according to our requirement



STEP 6: Within the Input Columns tab, you can cross check the input columns.

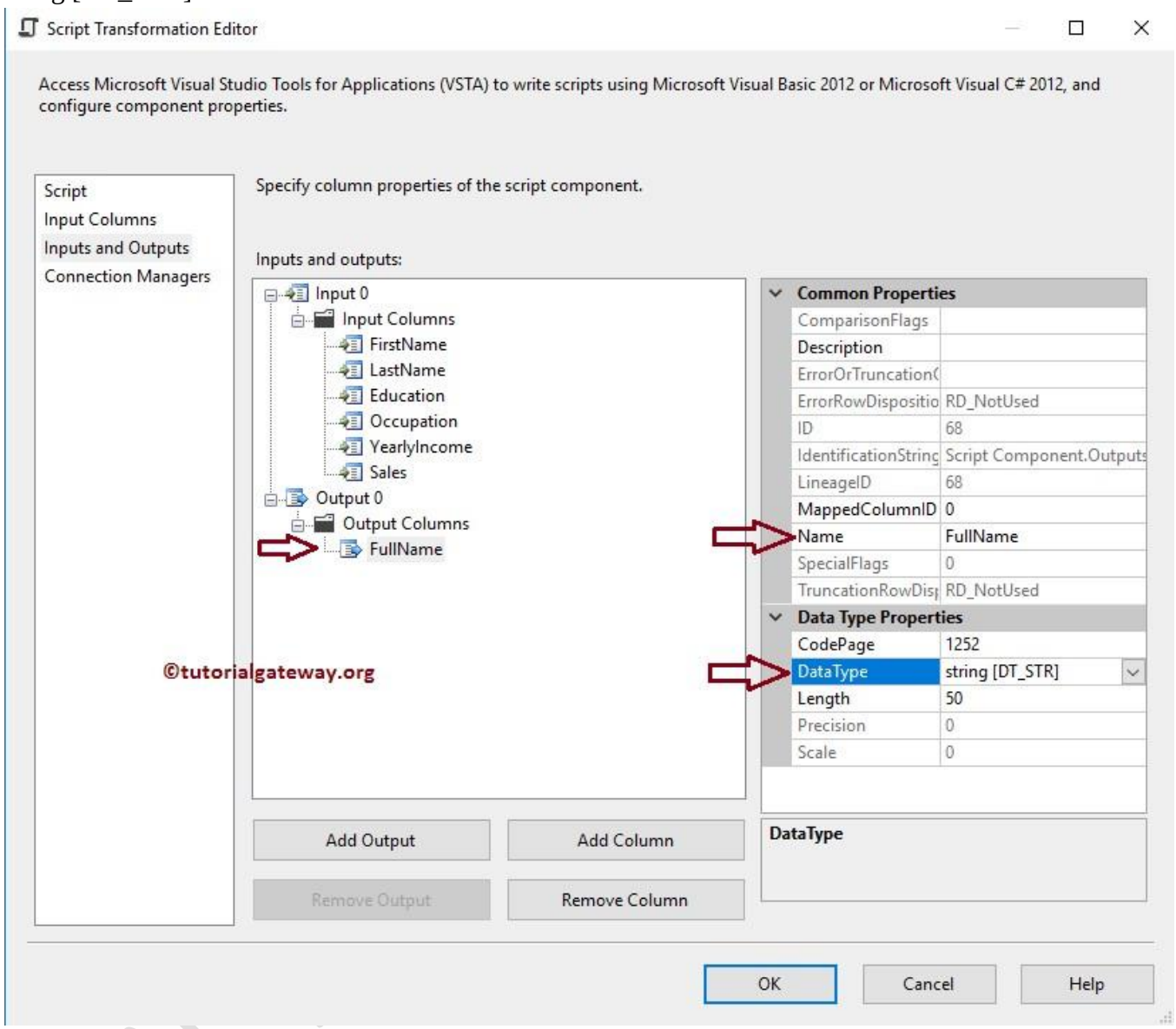


STEP 7: Within the Input and Outputs tab, Go to Output Columns, and under output columns we are going add one column called FullName using Add Columnbutton



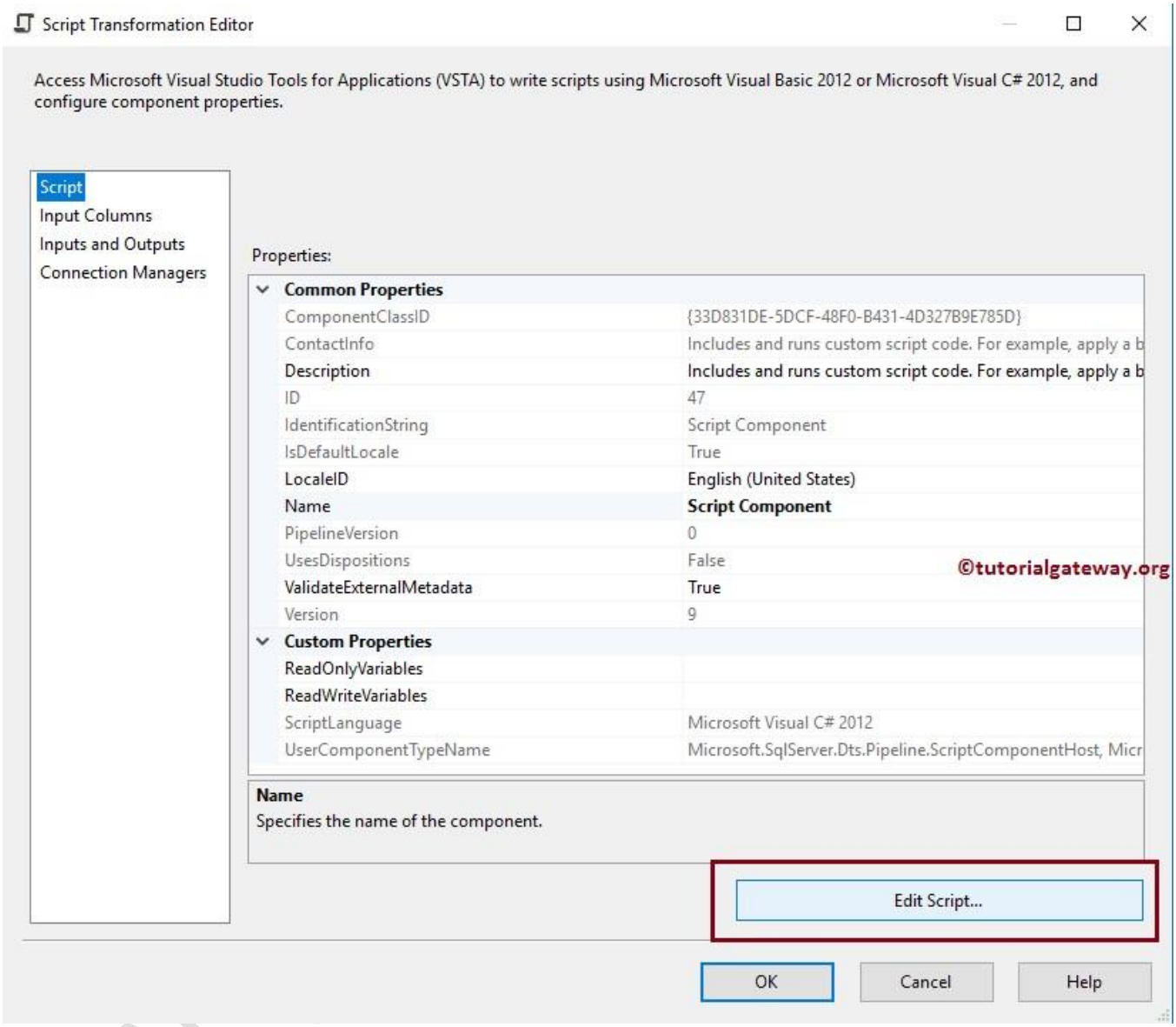
Here we added the FullName Column, and then changed the Data type from Integer (default) to

string [DT_STR]

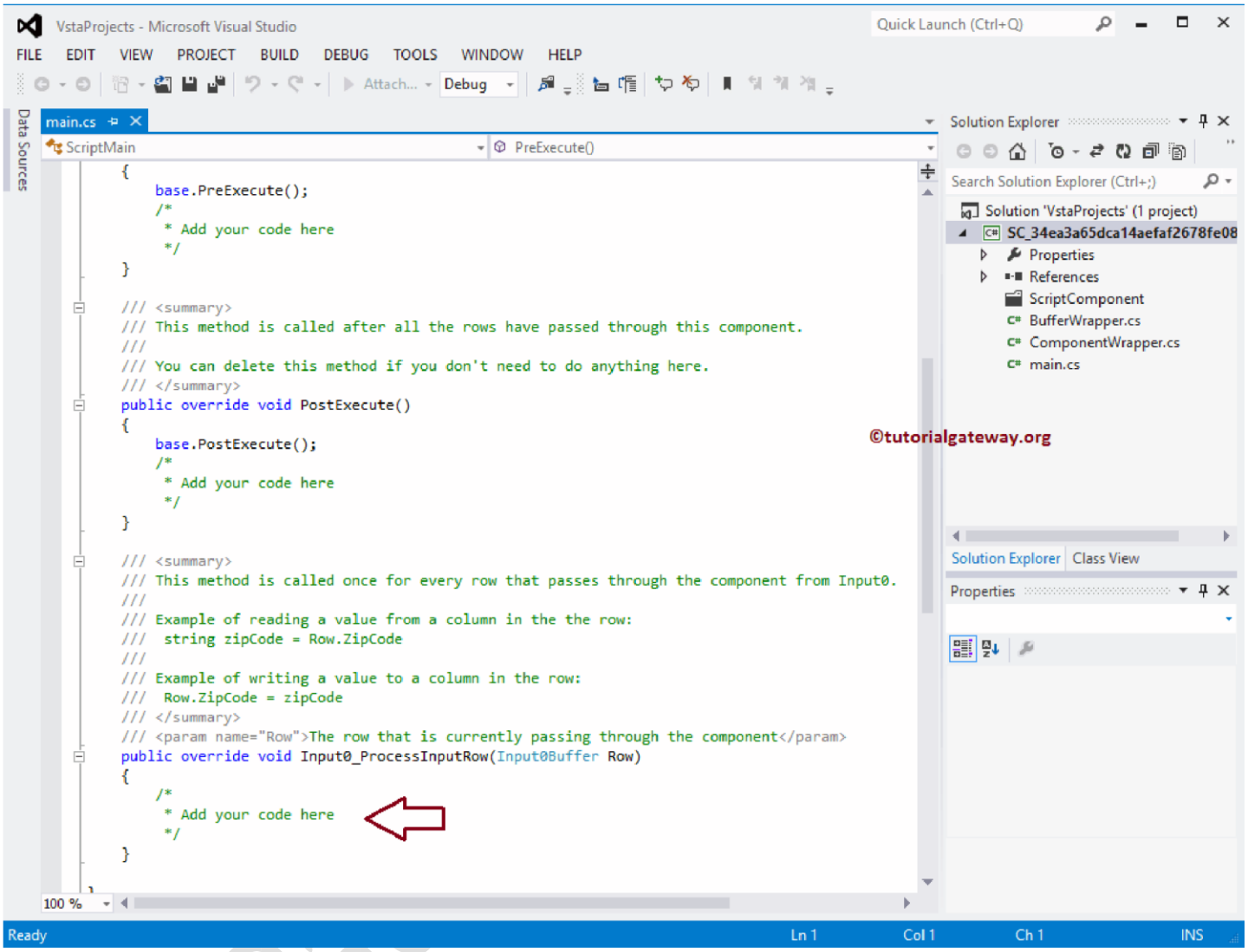


STEP 8: Within the Script tab, please click on the Edit Script.. button to write the actual C# Script

TIP: You can change the language to VB.Net using ScriptLanguageproperty.

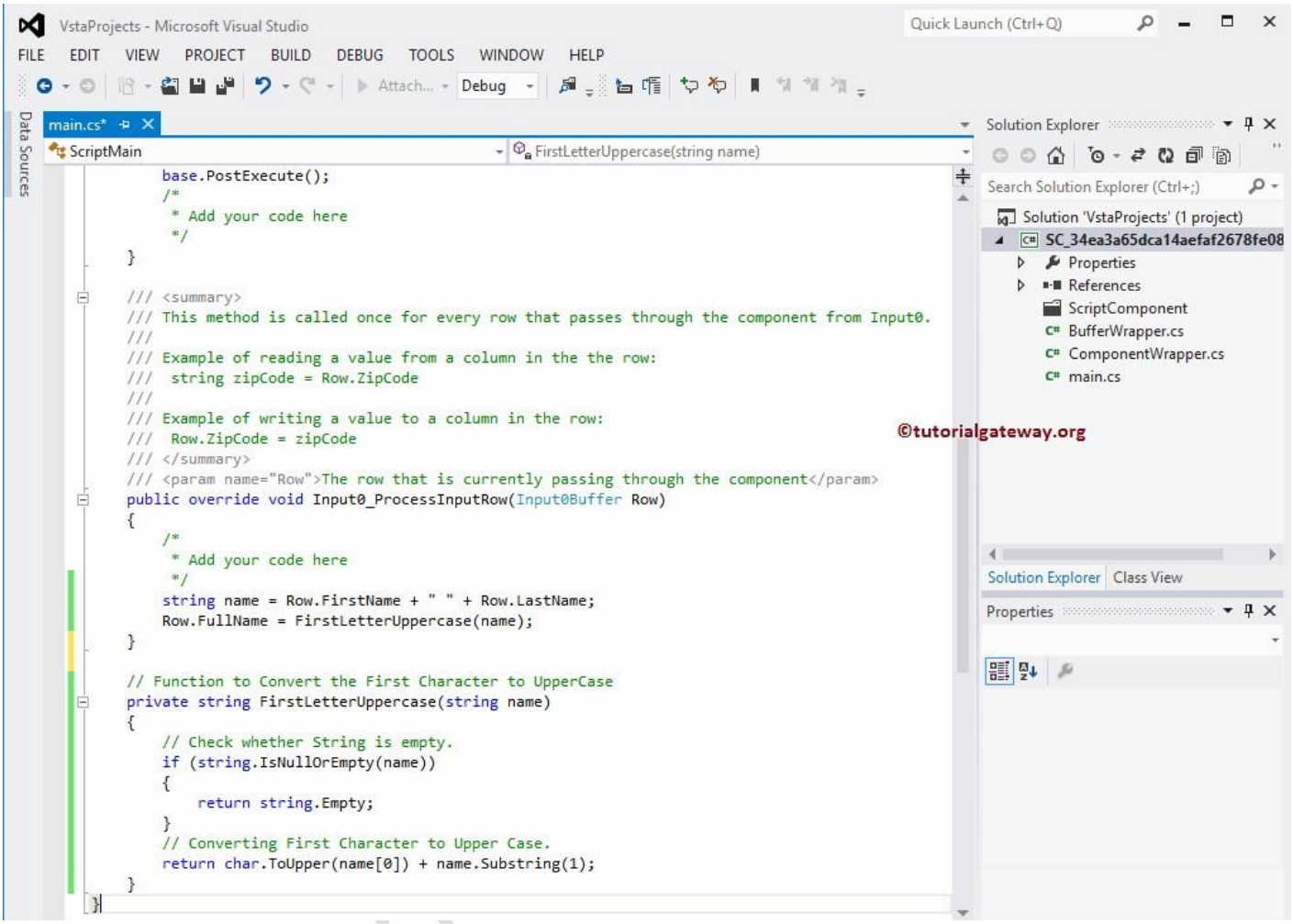


Once you click on the Edit Script, it will open the main.cs class file to write the C# code. Please write your custom code inside the Input0_ProcessInputRow(Input0Buffer Row) function



STEP 9: Add your custom C# code here. For this example, we are concatenating First name, Last

Name, and then we are converting the First Letter to Uppercase.



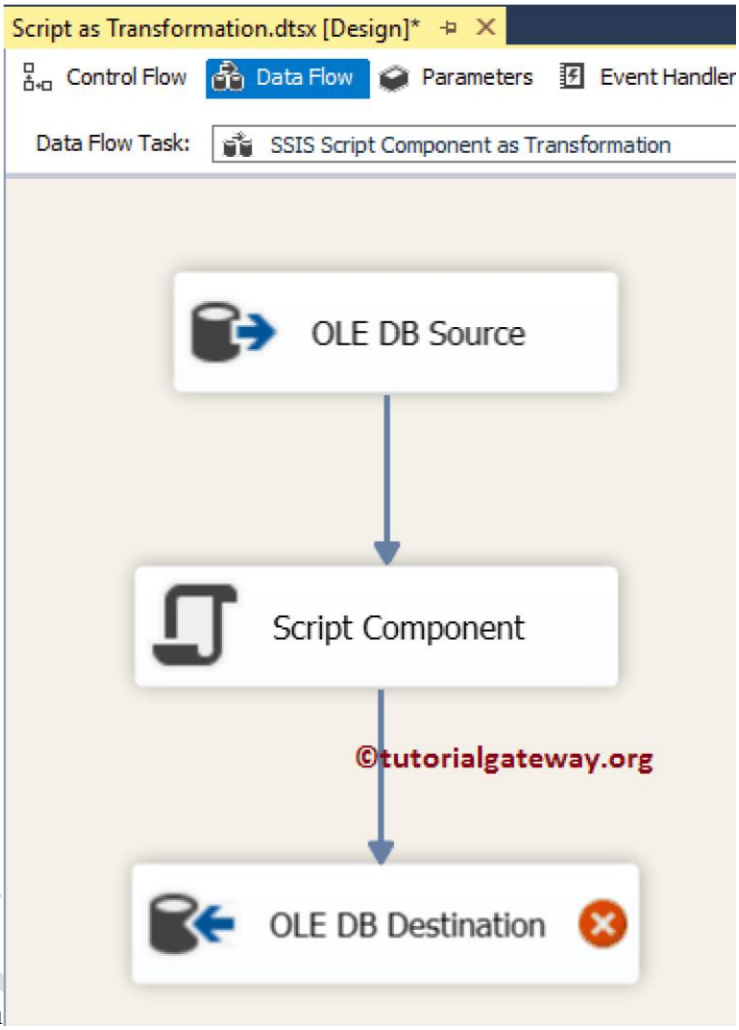
Code that we used in the above SSIS Script Component as Transformation screenshot is:

C# CODE

```
public override void Input0_ProcessInputRow(Input0Buffer Row)
{
    /*
     * Add your code here
     */
    string name = Row.FirstName + " " + Row.LastName;
    Row.FullName = FirstLetterUppercase(name);
}
```

```
// Function to Convert the First Character to UpperCase
private string FirstLetterUppercase(string name)
{
    // Check whether String is empty.
if (string.IsNullOrEmpty(name))
    {
        return string.Empty;
    }
    // Converting First Character to Upper Case.
    return char.ToUpper(name[0]) + name.Substring(1);
}
```

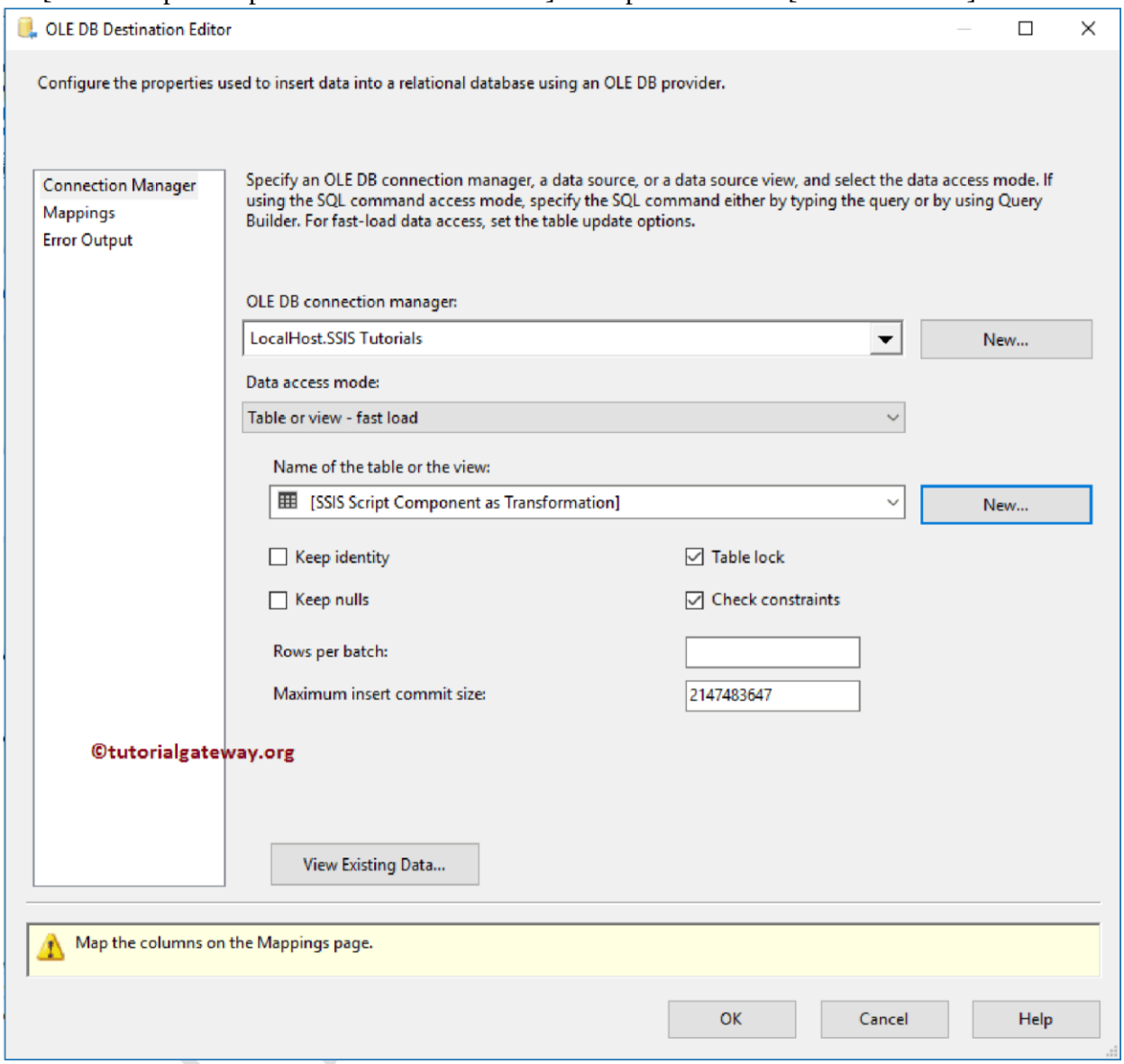
STEP 10: Once you finished editing the Script, Please close the main.cs file. Next, drag and drop OLE DB Destinations on to the data flow region, and then drag the script component Output



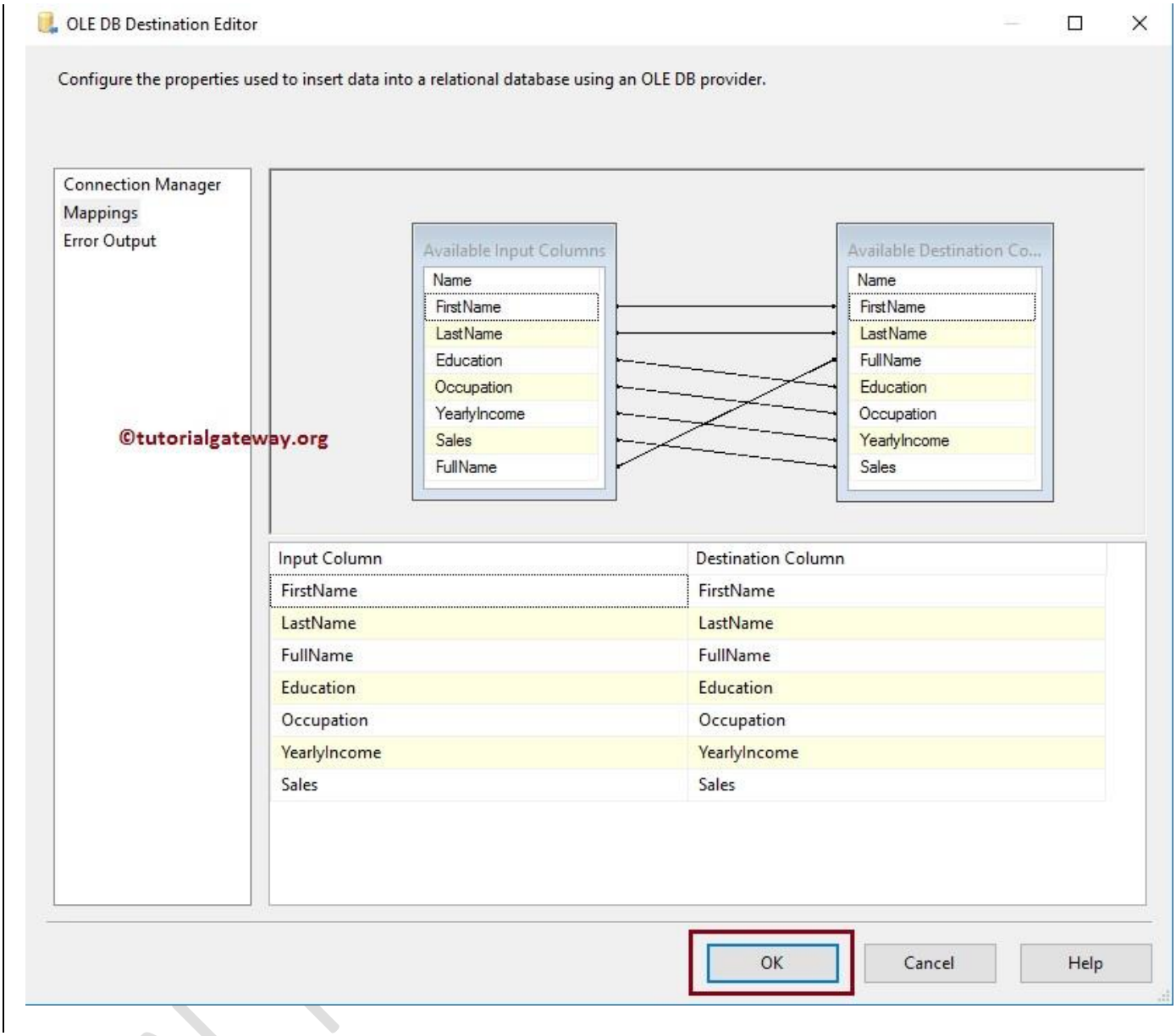
Arrow to this new OLE DB Destination

STEP 11: Double click on OLE DB Destination will open the OLE DB Destination Editor. Please Select the OLE DB Connection manager that you already created. Here we are selecting

the [SSIS Script Component as Transformation] table present in the [SSIS Tutorials] Database.

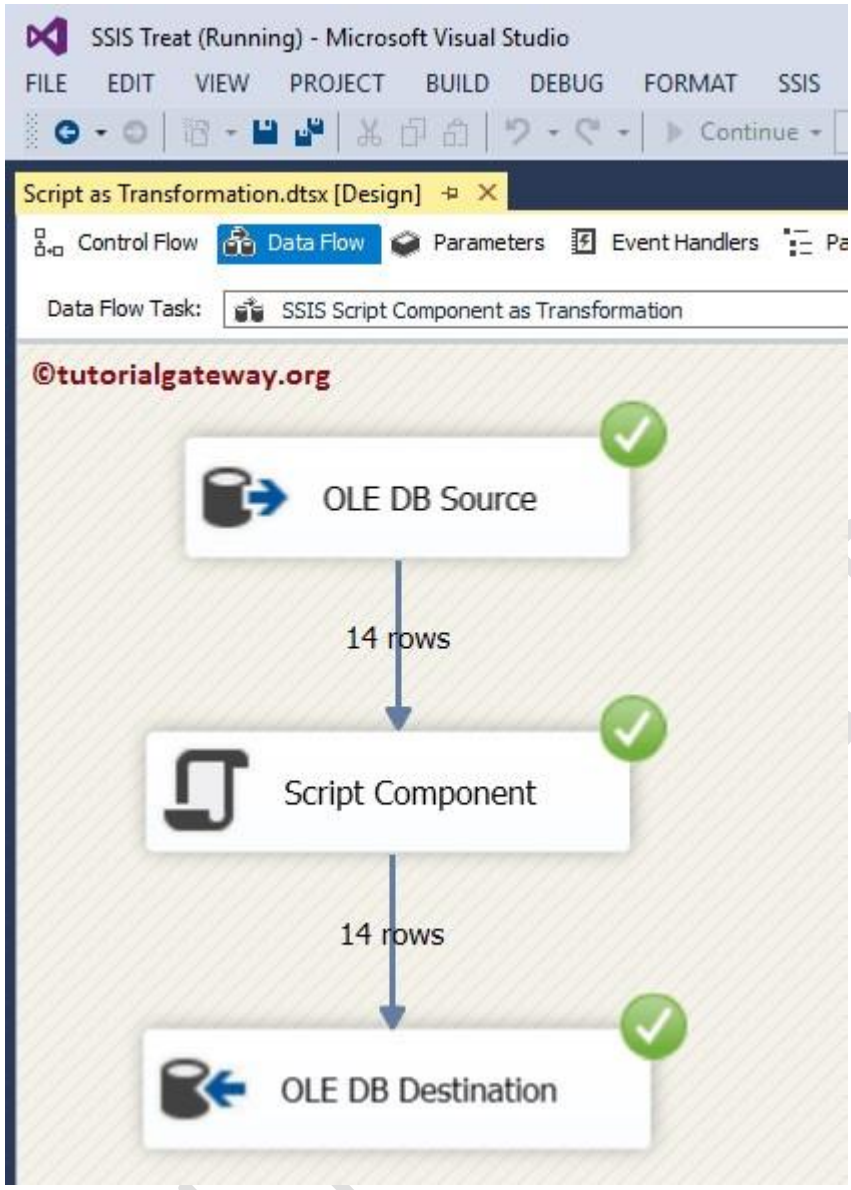


STEP 12: Click on Mappings tab to check whether the source columns are exactly mapped to the destination columns.



Click OK to finish creating our Package.

STEP 13: Right click on the SSIS Script Component as Transformation Package in the Solution Explorer, and select Execute Package.



From the above screenshot you can observe that our Package has executed successfully. Let‘s open the SQL Server Management Studio and write the following query to view the data

T-SQL

```
USE [SSIS Tutorials]

GO

SELECT [FirstName]

    ,[LastName]
    ,[FullName]
```

```
,[Education]
,[Occupation]
,[YearlyIncome]
,[Sales]
FROM [SSIS Script Component as Transformation]
```

OUTPUT

```
USE [SSIS Tutorials]
GO
SELECT [FirstName]
      ,[LastName]
      ,[FullName]
      ,[Education]
      ,[Occupation]
      ,[YearlyIncome]
      ,[Sales]
  FROM [SSIS Script Component as Transformation]
```

100 %

Results | Messages

	FirstName	LastName	FullName	Education	Occupation	YearlyIncome	Sales
1	tutorial	gateway	Tutorial gateway	Masters Degree	Management	90000	3578.27
2	rob	johnson	Rob johnson	Bachelors	Management	80000	3399.99
3	ruben	torres	Ruben torres	Partial College	Skilled Manual	50000	699.0982
4	christy	zhu	Christy zhu	Bachelors	Professional	80000	3078.27
5	rob	huang	Rob huang	High School	Skilled Manual	60000	2319.99
6	john	ruiz	John ruiz	Bachelors	Professional	70000	539.99
7	john	yang	John yang	Bachelors	Professional	80000	2320.49
8	christy	mehta	Christy mehta	Partial High School	Clerical	50000	24.99
9	rob	verhoff	Rob verhoff	Partial High School	Clerical	45000	24.99
10	christy	carlson	Christy carlson	Graduate Degree	Management	70000	2234.99
11	gail	erickson	Gail erickson	Education	Professional	90000	4319.99
12	barry	johnson	Barry johnson	Education	Management	80000	4968.59
13	peter	krebs	Peter krebs	Graduate Degree	Clerical	50000	59.53
14	greg	alderson	Greg alderson	Partial High School	Clerical	45000	23.5

Using Custom components in SSIS

In contrast to Script tasks and components, custom tasks and components can be developed, deployed, and maintained independently of the SSIS package in which they are going to be used, meaning that any updates to the task or component can be performed without having to redesign and redeploy every package in which it is being used—of course, as long as the change has not affected any of the exposed interfaces.

On the other hand, compared to Script task and component development, the approach to custom task and component development is a bit more complex and requires more development efforts.

Planning a Custom Component

After you have determined that the business problem cannot be solved by using any of the standard, built-in SSIS data flow components, and after you have determined that due to complexity, dependency, or reusability requirements (or restrictions), a Script component may also not be appropriate, you can use the following guidelines to plan the design of the custom component:

■■ **Role** Is the component going to be used as a data source, a data destination, or to transform data?

Typically, a *custom source* would be needed if none of the existing sources support the specific connection manager that you are using, or if an appropriate connection manager is not available. For instance, if the source data is extracted from an incompatible source or is stored in an incompatible format, you could develop a custom data source. Similarly, a *custom destination* could be designed if such an incompatible data store is used as the data flow‘s destination.

In most cases, when custom development is needed, it is to design a *custom transformation*— to support a particular data management operation that is not supported by any of the standard, built-in transformations.

■■ **Usage** Is the source or transformation component going to use multiple outputs?

Is the transformation or destination component going to use multiple inputs? A source or a transformation component can send data to multiple outputs, and a destination or a transformation component can receive data from multiple inputs. For instance, a transformation component used in merging data from multiple sources would have to support multiple inputs. A source component accessing a composite data set could be programmed to produce multiple, normalized row sets, instead of a single de-normalized one.

■■ **Access to external data** Is the component going to use additional data sources, or will it consume only data in the data flow buffer?

If the component will perform lookup operations or will need to access data that is not available in the current data flow, it will require access to external data sources. To access data stored in variables or parameters, the component will also need access to those variables and parameters. ■■ **Behavior** Is the component going to be a blocking, a partially blocking, or a nonblocking component?

Is it going to use synchronous or asynchronous outputs? If the component is going to pass rows to the destination without having to retain them, such as to calculate running totals (partially blocking), or to sort them (blocking), the component will not block the data flow (and is a nonblocking transformation). If the transformation produces a single output row for each input row, where the result of the transformation can be written to one or more columns of the source row, a synchronous output can be used. However, if the transformation could produce one or more output rows for each input row, or even not produce a row at all, an asynchronous output would have to be used. New rows cannot be added to a synchronous output and cannot be removed from it.

■■ **Configuration** How will the component be configured?
To improve the reusability of a custom component, specific settings used to control its operation should be exposed, allowing the developer to set them at design time, or even expose them to the environment. For instance, a transformation performing data extraction or data validation using Regular Expressions should allow the developer to set the expressions at design time, or even allow them to be determined automatically at run time.

Developing a Custom Component
Custom components are built in Visual Studio 2010, using the Class Library template, or even a blank solution template. The only vital prerequisite for custom SSIS development using the .NET Framework is to create references to the appropriate SSIS libraries:

■■ Microsoft.SqlServer.DTSPipelineWrap, containing classes and interfaces used to create data flow objects and automate data flow operations
■■ Microsoft.SQLServer.DTSRuntimeWrap, containing classes and interfaces used to create control flow objects and automate control flow operations
■■ Microsoft.SqlServer.PipelineHost, containing managed classes to access SSIS data flow objects and methods

Typically, custom components are developed separately from SSIS packages; mostly due to the fact that they are developed by a different developer or developer team. Nonetheless, they could also be part of the same solution (for example, they could be developed together with other SSIS projects) but placed in separate projects.

There are two principal classes that you should be familiar with when designing components:
■■ All data flow components are derived from the Microsoft.SqlServer.Dts.Pipeline .PipelineComponent base class, available in the Microsoft.SqlServer.PipelineHost class library. This class contains all built-in component properties, as well as design-time methods used to configure the component at design time and run-time methods to perform its operations at run time.
■■ The Microsoft.SqlServer.Dts.Pipeline.PipelineBuffer class, also available in the Microsoft .SqlServer.PipelineHost class library, provides access to the data being passed into, or being created by, the component. The PipelineBuffer represents an in-memory data store organized in rows and columns.

Design Time and Run Time
Custom tasks and components are used at design time, when developers are designing SSIS packages, as well as at run time, when an SSIS package using the component is being executed.

This duality is reflected in the SSIS component object model and should be considered throughout development.

The customization of custom tasks and components is achieved when existing members of the component base class are overridden with custom code. There are two groups of methods that need to be customized when designing custom components:

■■ The *design-time methods* are used at design time, when the SSIS package developer adds the component to a data flow task and configures it.
Design-time methods allow the developer to access the component and its properties when designing the SSIS package. They are used to create one or more instances of the component inside the data flow, to configure an instance, to validate it, and/or to reset the component to its initial state.
■■ The *run-time methods* are invoked by the SSIS engine when the package is run. Run-time methods allow the computer to perform the operations when packages implementing the component are being executed. They contain the actual business logic required by the particular data management process.

Design-Time Methods
Design-time methods facilitate the interaction between the SSIS developer and the data flow component, allowing the component to be placed in the data flow and configured appropriately.

The principal design-time methods are listed in this section, and you can find out more about the rest of the design-time methods in Books Online for SQL Server 2012.

ProvideComponentProperties
This method is used to initialize the component and is invoked when the component is dragged from the SSIS Toolbox and placed inside the data flow task. This method should provide all the elements needed by the component at design time: inputs and outputs, custom component properties, and any external data sources.

Validate
This method is used to validate the state of the component and is invoked automatically when the component‘s editor is accessed and when changes to its configuration are confirmed by the SSIS developer.

The method is also called once at run time, before the component is executed. This method should provide the programmatic logic used to verify whether the component has been configured correctly and can therefore be used at run time. It should not change the component‘s metadata, such as by adding, modifying, or removing any of its elements.

TABLE 19-1 Validation Results

Validation status	Description
VS_ISVALID	The component is correctly configured and ready for execution. Use this status only if the component is configured correctly and can be executed.
VS_ISBROKEN	The component is incorrectly configured; for instance, a property is set incorrectly or an expected element is missing. Use this status to help the package developer configure the component appropriately.
VS_NEEDSNEWMETADATA	The component's metadata is outdated or corrupt, which can be corrected by invoking the ReinitializeMetaData method. Use this status to prompt the package developer to refresh the component's metadata.
VS_ISCORRUPT	The component is irreparably damaged and must be completely reset by invoking the ProvideComponentProperties method. Use this status when the component has been misconfigured and should be reinitialized.

ReinitializeMetaData

This method is used to update metadata describing the external data sources referenced by the component.

This method is invoked automatically when the component editor has been opened, after the VS_NEEDSNEWMETADATA result has been returned in an earlier validation.

Use this method to correct the metadata when changes have been made in the underlying data stores referenced by the component, and the changes have not yet been reflected in the component's metadata.

FireError, FireWarning, and FireInformation

These methods are used to facilitate communication between the developer who designed the component and the developer using the component when designing an SSIS package. Some of these methods may be invoked automatically by the base Validate method and should be invoked by the overridden Validate method to convey information about the state of the component to the package developer.

Package developers should understand how the component needs to be configured in order to perform the operations as expected, and the SSIS engine should prevent a misconfigured component from being executed at all.

These methods could also be invoked at run time: all of them are reported to the calling environment, but FireError will even prevent or stop execution.

By using FireInformation and FireWarning, the component developer can communicate to the package developer that the component has been set up in a certain way, or that it could have been set differently. By using the FireError method, the component developer can prevent the execution of the component until it has been configured correctly.

Run-Time Methods
Run-time methods represent the component‘s operational programmatic logic, which is executed when the component is used at run time. These methods provide the complete operational capabilities of the component; without them, the component performs no actions other than being validated and returning a validation result.

The run-time methods in this section are listed in the order of operation—from validation, through row processing, to cleanup.

AcquireConnections (for Validation)
The method is invoked at design time whenever connections are needed—for instance for validation—and at run time to establish connections with external data sources. This should be the only method used in establishing connections; once established, each connection should then be cached in a local variable and reused without the need to be re-established.

Connections should be released by using the ReleaseConnections method, described later in this lesson.

Validate
This method has already been described earlier in this lesson. At run time, it is executed at the beginning of the execution.

ReleaseConnections (after Validation)
This method is invoked after the validation has completed and is used to release the connections. It is executed again later, after the execution has completed.

PrepareForExecute
After the validation has completed successfully, execution begins with this method. The BufferManager is not yet available when this method is executed, so PrepareForExecute can be used to initialize all the settings and variables that can be set before the data from the data flow is available to the component.

Typically, PrepareForExecute is used to initialize variables that will be used throughout the execution and that are not usually affected by the data received from the buffers.

AcquireConnections (for Execution)
After the execution has actually started, the connections are acquired again, this time to be used for data processing.

PreExecute
This method is invoked after the connections have been established and after the BufferManager has been initialized, allowing access to the data in the data flow.

This method should be used to complete the initialization of variables and other settings needed to perform the operations.

If data sources provide access to sets of data that are small enough to fit into memory completely, the data they provide could now be cached in a variable, and the corresponding connections released.

PrimeOutput

If the component uses asynchronous outputs, these are initialized by using the PrimeOutput method; after this initialization, rows can be added to the output.

If multiple asynchronous outputs are used, appropriate programmatic logic must be implemented to prepare all outputs. Outputs that have not been primed cannot be written to.

This is the final method before row processing begins. Any resources that have already been consumed and will not be needed later—especially if they are not required for row processing—should now be released.

ProcessInput

This is the principal method used in processing the rows in the data flow; it is invoked after all preceding methods have completed successfully and when rows from an input buffer become available to the component.

The input buffers contain rows passed from upstream components and allow access to all columns returned by the upstream component. Synchronous outputs share the buffer with the corresponding input; therefore, the output columns are also already available when processing inputs with a corresponding synchronous output. Rows must be added manually to asynchronous outputs by using the NewRow() buffer method.

The ProcessInput method will be called repeatedly as data is received from upstream components, until the last row has been processed, which is reflected in the EndOfRowset property. Data processing is performed in batches, which means that multiple buffers will be sent to the component. Therefore, you should not assume that the processing has been completed solely based on the fact that all the rows in the current buffer have been processed. This method will also be called multiple times for multiple inputs; therefore, appropriate programmatic logic must be implemented to process all inputs.

To process rows continuously as they are passed to the component, the data management operation must be implemented as a loop by using the NextRow() buffer method. This method returns each row from the buffer, to be processed one at a time, until the entire buffer has been consumed. As long as new buffers are being sent from upstream components, row processing continues.

When the entire input has been exhausted, synchronous outputs are completed automatically, whereas asynchronous outputs must be completed manually by using the SetEndOfRowSet() method. Only by understanding the nature of SSIS data flow processing well can you ensure prevention of data loss or data corruption .

PostExecute

This method is called after all the data from the data flow has been processed. If any data needs to be passed to the package—via variables—or passed to external data sources, these operations should be performed in PostExecute.

Any resource that has not yet been released (such as component variables), except for external connections, should also now be released.

ReleaseConnections (after Execution)
This method is used to release any connections to external data sources used by the component that have not already been released earlier in the process.

This method should be used to release all connections acquired by using the Acquire-Connections method; even if the connections have been released in other methods, the ReleaseConnections method should guarantee that no resources will remain active after the processing has completed.

In fact, whenever the AcquireConnections method is used in a component, the

ReleaseConnections
Method must also be implemented; otherwise, a run time error will occur during each validation, and execution will not be possible.

Cleanup
After the processing has completed and any external connections have been released, the rest of the resources should also be released. This method can also be used to assign values to package variables.

Designing, Deploying, and Using a Custom Data Flow Component
Exercise 1 Prepare the Environment
1. Start SSMS, and on the File menu, under Open, select File, then navigate to the C:\TK463\Chapter19\Code folder, and open the TK463Chapter19.sql Transact-SQL script.
2. After you have reviewed the script, execute the part for Lesson 3. The script creates the database and the objects you will be using in this lesson.

Exercise 2 Develop a Custom Data Flow Transformation
1. Start SSDT. On the Start page, select Open Project, navigate to the C:\TK463 \Chapter19\Lesson3\Starter folder, and open the TK 463 Chapter 19.sln solution.
2. On the File menu, select Add | New Project, and create a new Visual C# Class Library, using the Class Library template found under Installed Templates | Visual C# | Windows. Use the information provided in Table 19-2 to configure the new project.

TABLE 19-2 New Class Library

Property	Value
Name	TK463.CalculateCheckSum
Location	C:\TK463\Chapter19\Lesson3\Starter\
Solution Name	TK463 Chapter 19
Create Directory For Solution	(Selected)

3. In the Solution Explorer pane, locate the class file with the default name Class1.cs, right-click it, and on the shortcut menu select Delete to remove it.

4. In the Solution Explorer pane, right-click the newly created project, and on the shortcut menu, select Add | Existing Item to add a class file to the project. Use the Add Existing Item dialog box to navigate to the C:\TK463\Chapter19\Code folder, and then select the CalculateCheckSum.cs file. When you are ready, click OK to add the file.

5. In the Solution Explorer pane, right-click the newly created project, and the select Properties on the shortcut menu to access the project properties. On the Application page of the properties editor, set the project properties, using the information provided in Table 19-3.

TABLE 19-3 Project Properties

Property	Value
Assembly Name	TK463.CalculateCheckSum
Default Namespace	Microsoft.TK463
Target Framework	.NET Framework 4
Output Type	Class Library
Startup Object	(Not set)

6. When you are done, save the project, and then close the properties editor.

7. In the Solution Explorer pane, under the newly created project, right-click References, and select Add Reference on the shortcut menu to add references to the following SQL Server 2012 Integration Services libraries:

■■ Microsoft.SqlServer.DTSPipelineWrap
■■ Microsoft.SQLServer.DTSRuntimeWrap
■■ Microsoft.SqlServer.PipelineHost

You can find all these libraries among the .NET components, on the .NET tab of the Add Reference dialog box.

8. Open the CalculateCheckSum.cs file and review the definition. The component implements the following design-time methods:

■■ ProvideComponentProperties
■■ Validate
■■ DeleteOutputColumn

The DeleteOutputColumn method is invoked when output columns are removed at design time. By overriding it you can, for instance, control which columns can actually be removed. The component implements the following run-time methods:

■■ PrepareForExecute
■■ PreExecute
■■ ProcessInput ■■
PostExecute

Follow the comments in the code to see how the methods are implemented, what other functions are used when the component is configured, and when the operation is executed.

9. When done, save the solution and build the TK463.CalculateCheckSum project in debug mode.
10. Leave the solution open, because you will need it in the following exercises.

Exercise 3 Deploy a Custom Data Flow Component

Before you can deploy the component, it needs to be released. If you were able to build the project successfully in the preceding exercise, you can now complete the building process by creating a release build.

1. Switch the solution configuration to Release, and build the TK463.CalculateCheckSum project again.
2. By default, the release version of the assembly should be placed in the C:\TK463 \Chapter19\Lesson3\Solution\TK463 Chapter 19\TK463.CalculateCheckSum\bin \Release folder. Use Windows Explorer to verify this.
3. To deploy an SSIS custom task or component, you need to copy its assembly to the SSIS task or component folders and then register it in the Global Assembly Cache.

By default, the destination folders for task assemblies are:

■■ %ProgramFiles(x86)%\Microsoft SQL Server\110\DTS\Tasks for the 32-bit edition
■■ %ProgramFiles%\Microsoft SQL Server\110\DTS\Tasks for the 64-bit edition By default, the destination folders for component assemblies are:
■■ %ProgramFiles(x86)%\Microsoft SQL Server\110\DTS\PipelineComponents for the 32-bit edition
■■ %ProgramFiles%\Microsoft SQL Server\110\DTS\PipelineComponents for the 64-bit edition

4. To help you with the deployment, an appropriate batch file is available in the C:\TK463 \Chapter19\Lesson3\Solution folder. You can use it to perform all the tasks described in Step 3 automatically.

Open the Deploy.bat file for editing in Notepad, and observe the script.
The deployment script uses the gacutil utility, which is part of Windows SDK.
If Windows SDK is not installed on your machine, you should install it before attempting the deployment. Alternatively, you can use Visual Studio Command Prompt (2010) to register the library in the Global Assembly Cache.

5. When you are ready, run the batch file to deploy the component.

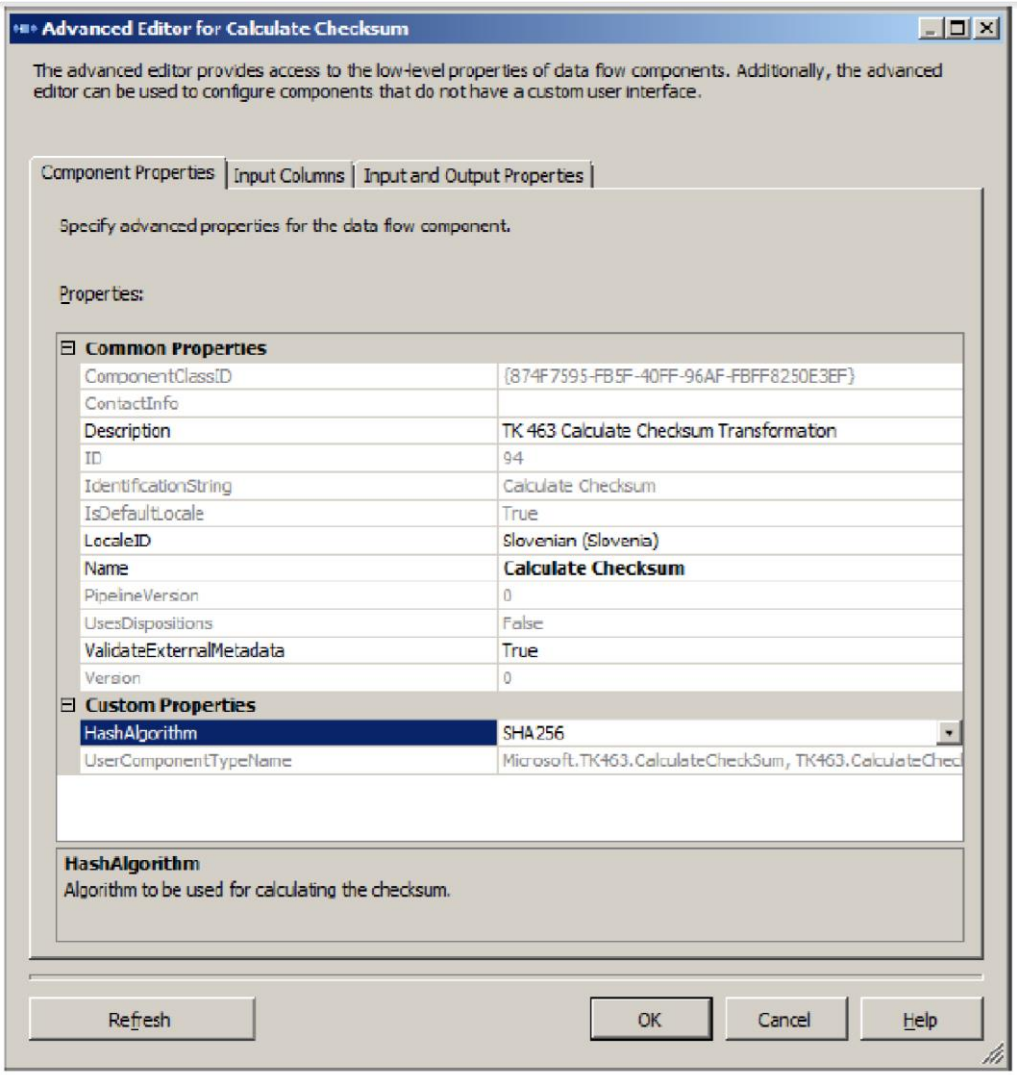
Exercise 4 Configure and Use a Custom Data Flow Component
After the component has been deployed successfully, it can be used in SSIS packages.

1. In SSDT, open the FillStageTables.dtsx package of the TK463 Chapter 19 project, which is part of the TK 463 Chapter 19.sln solution that you used in Exercise 1 earlier in this lesson.
2. Open the Insert stgCustomer data flow component in the data flow editor, and remove the data path leading from the Customer data flow source to the Data Conversion data flow transformation.
3. Drag the Calculate Checksum transformation from the SSIS Toolbox onto the data flow editor surface.

The Calculate Checksum component should be listed in the Common group. If the component is not displayed, right-click the SSIS Toolbox and select Refresh Toolbox on the shortcut menu to refresh the toolbox.

4. Connect the output of the Customer data source to the input of the Calculate Checksum transformation. Double-click the newly added transformation to open the Advanced Editor.

Use the Advanced Editor to configure the component. On the Component Properties page, use SHA256 as the HashAlgorithm property value. On the Input Columns page, select all input columns to be processed and passed to downstream components. On the Input and Output Properties page, explore the available settings, but make no additional changes. The dialog box should look like the one shown in Figure 19-4.



5. When you are done, click OK to confirm the configuration.
6. Connect the output of the Calculate Checksum transformation to the input of the Data Conversion transformation.
7. Edit the stgCustomer data flow destination, and correct the column mapping so that the results of the Calculate Checksum transformation can be stored in the *RowCheck- Sum* column of the destination table.
8. When you are done, save the solution, and then execute the FillStageTables.dtsx package in debug mode.
9. After the execution has completed, review the contents of the *stg.Customer* table by using SSMS.
10. Optionally, you can repeat steps 2 through 8 of this exercise to implement the Calculate Checksum component in the Insert stgPerson and Insert stgCustomerInformation data flows as well.

Developing a Custom SSIS Source Component

SSIS was designed to be extensible. Although you can create tasks that will take data from a wide variety of sources, transform the data is a number of ways and write the results a wide choice of

destinations, using the components provided, there will always be occasions when you need to customise your own SSIS component.

Overview

SQLServer Integration Services (SSIS) is the Microsoft platform for building enterprise-level data integration and data transformation solutions. Integration Services can extract and transform data from a wide variety of sources such as XML data files, flat files, and relational data sources, and then load the data into one or more destinations. Integration Services is so versatile because it has the ability to connect to disparate data sources built into its object model. In this article, we will look at how to extend this object model by building a custom data source, using a working example.

What is an SSIS component?

SSIS components are one of the basic building blocks of the SSIS framework, and can be found in the SSIS toolbox in SQLServer Data Tools. In this article we will be specifically looking at data flow components. These can read data from external data sources, transform it, and write it back to other data destinations. SQLServer Data Tools have some examples of data flow components, including the OLE DB Source, Lookup transformation and the Character Map transformation components.

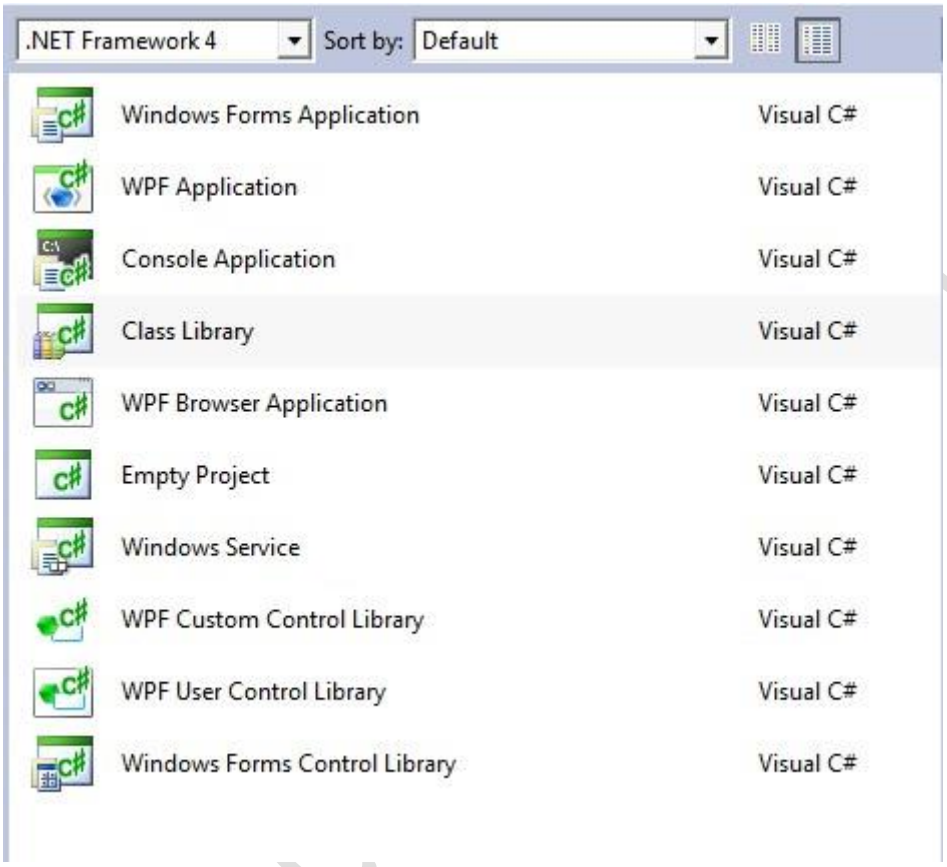
Under the hood, an SSIS data flow component is a .Net class that inherits from the PipelineComponent class, which is part of the overall SSIS Object Model. It overrides the various methods of PipelineComponent , such as those for creating inputs/outputs, adding custom properties to the component etc. The compiled class library is placed in a subfolder in the SQLServer installation folder. SQLServer Data Tools automatically recognizes any class libraries in this folder that implement PipelineComponent , and makes the component available in the SSIS Toolbox.

Building the component class

We will start by first building a bare-bones data source component, and then deploying it so that you can use it in your Integration Services package. I‘ve already mentioned that a data source component is used to read data from an external data source, but at this stage it won‘t actually do anything. In the next stage, once we are satisfied that we can deploy it, we‘ll get it doing something useful.
Before we get into the details of building the class, note that the following description assumes that you are using Visual Studio 2012 and SQLServer Data Tools 2012. If you are using an older version of either tool, you would have to adjust accordingly. I have tried to highlight specific tools between different versions wherever possible.

With that in mind, start by launching Visual Studio, and create a new class library project, as shown in Figure 1. You can name the project CustomSSISComponent . Make sure that you select .Net

Framework 4 as the .net version. The code I have presented here uses C#, but you can write it just as easily in Visual Basic also.



Rename the Class1.cs file in the project to CustomSSISComponent.cs . You will also need to reference the following assemblies in your project:

Assembly to import	Namespace to import	Can be found at
Microsoft.SqlServer.PipelineHost	Microsoft.SqlServer.Dts.Pipeline	Under the GAC_MSIL folder in the GAC
Microsoft.SqlServer.DTSPipelineWrap	Microsoft.SqlServer.Dts.Pipeline.Wrapper	Under the GAC_MSIL folder in the GAC
Microsoft.SqlServer.ManagedDTS	Microsoft.SqlServer.Dts.Runtime	Under the GAC_MSIL folder in the

Microsoft.SqlServer.DTSRuntimeWrap	Microsoft.SqlServer.Dts.Runtime.Wrapper	Under the GAC GAC_32 folder in the GAC
System.ServiceModel	System.ServiceModel	Part of the .Net framework.

* *Note that the GAC is at %windowsfolder%\Microsoft.Net\Assembly in .Net framework 4.0.*

* *Since we are building the component with .Net framework 4.0, make sure you refer to the assemblies at %windowsfolder%\Microsoft.Net\Assembly.*

Modify the CustomSSISComponent.cs file to add the following class definition for a custom Integration services component:

```
using System;
using System.Collections.Generic;
using System.Linq;     using
System.Text;

using Microsoft.SqlServer.Dts.Pipeline;
using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
using Microsoft.SqlServer.Dts.Runtime;       using
Microsoft.SqlServer.Dts.Runtime.Wrapper;       using
System.ServiceModel.Syndication;       using
System.Data;       using System.Xml;


namespace CustomSSISComponent
{
```

```
[DtsPipelineComponent(DisplayName = "CustomSSISComponent", ComponentType = ComponentType
public class CustomSSISComponent : PipelineComponent
{
    public override void AcquireConnections(object transaction)
    {
        base.AcquireConnections(transaction);
    }
    public override void PrimeOutput(int outputs, int[] outputIDs, PipelineBuffer[] buffers)
    {
        base.PrimeOutput(outputs, outputIDs, buffers);
    }
    public override void PreExecute()
    {
        base.PreExecute();
    }
    public override DTSValidationStatus Validate()
    {
        return base.Validate();
    }
    public override IDTSCustomProperty100 SetComponentProperty(string propertyName, object prope
    {
        return base.SetComponentProperty(propertyName, propertyValue);
    }
    public override void ProvideComponentProperties()
    {
```

```
        base.ProvideComponentProperties();
    }


}
}
```

This class inherits from PipelineComponent , which is the base class for all Integration Services data flow components. The DTSPipelineComponent attribute provides the name of the component to be shown in the SSIS Toolbox in SQLServer Data Tools, and also specifies the type of the component. We will delve into the details of the overridden methods later in this article. Since at this stage we have not provided implementations for any of the overridden methods, the component will not actually do anything. However, we can still deploy it and add it to the SSIS Toolbox.

Special considerations for VS 2012

If you are using Visual Studio 2012 to launch your SSIS packages, then follow these additional steps: In solution explorer, right-click on the project name and select *_Add_ _New Item,,*. In the dialog box that comes up, select *—Resources File—*, and click *"Add"*. This will add a new resource file to your project, called Resource1.resx , and launch the resource editor.

- In the resource editor, click on *—Add Resource‖* from the top menu, and select *—Add New Icon‖* from the dropdown list.
- Leave the name asIcon1 and click *_Add_*. This will add a new icon file Icon1.ico under the *—Resources‖* folder in your project.
- In the class library file created earlier, replace the line

```
DtsPipelineComponent(DisplayName                    = "CustomSSISComponent", ComponentType =  Com
```

- with the line below :

```
        [DtsPipelineComponent(DisplayName =    "CustomSSISComponent",          ComponentT
    "CustomSSISComponent.Resources.Icon1.ico")]
```

- In the solution explorer, right-click on your project name and select *_Properties_*. On the Application tab, click the *—Icon and manifest‖* radio button, and select the Icon1.ico file from the drop-down list. Save and rebuild the project.

Deploying the custom component

Follow these steps to deploy your custom SSIS component:-

- Sign the assembly with a strong name. You can either do this after building the assembly, or at build time by providing the key/value file (.snk file) in the project properties. You can find details on signing an assembly on MSDN.
- Copy the strongly-named assembly to {{ SQLServer Installation Folder}}\110\DTS\ PipelineComponents . On 32-bit machines, if your SQLServer installation is at the default path, this path translates to C:\Program Files\Microsoft SQLServer\110\DTS\ PipelineComponents. On 64-bit machines, the path is C:\Program Files (x86)\Microsoft SQLServer\110\DTS\ PipelineComponents

  If you are developing a component for SQLServer 2008, replace the folder 110 with 100 in the above path.

- Next, launch the Visual Studio developer console, and install the strongly-named assembly to the GAC using the following command –

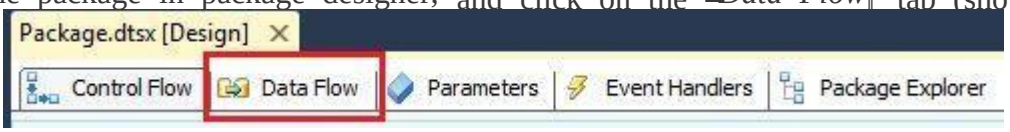
  C:\gacutil -u CustomSSISComponent

  C:\gacutil -i {Solution Root folder}\CustomSSISComponent\CustomSSISComponent\bin\Debug\CustomS

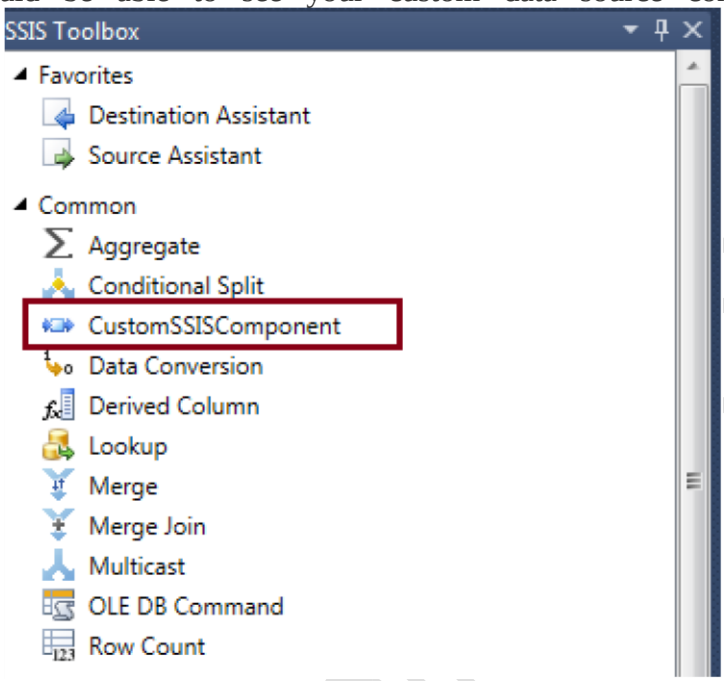
- The above uninstalls any previous versions of the assembly and reinstalls the newer version of the DLL.

- If you have multiple versions of the .Net framework installed on your computer, it is important that you use the correct version of gacutil.exe to add the file to the GAC. Since we developed the component using .Net framework 4.0, the gacutilversion should also be.Net framework 4.0.
- Launch a new instance of SQLServer Data Tools, and create a blank Integration Services package.

- Open the package in package designer, and click on the _Data Flow‖ tab (shown below).



- You should be able to see your custom data source component in the SSIS Toolbox.



If everything was configured correctly, the custom component should automatically appear in the SSIS toolbox. If you don‘t see it, try adding it manually, via _*Tools*_ _*Choose Toolbox Items*,, _*Browse to your component DLL*_.

Reading from a data source

Now that we are confident that we can build and deploy a custom Integration Services component, we will be creating a sample SSIS source custom component that reads from an RSS data feed. Conceptually, it is similar to the other source components that ship with Integration Services, such as the OLE DB Source or ADO .Net Source component, but the data source in this case is an RSS data feed.

An RSS feed is basically data served in a well-formed XML feed over the internet. I will not get into the format, structure etc. of RSS since that is outside the scope of this article. But let us take a quick look at the .Net helper classes that we will be using to read from an RSS source.

System.ServiceModel.SyndicationFeed is the .Net class that represents an RSS feed (e.g. a blog feed, newspaper feed etc.).

A SyndicationFeed consists of one or more SyndicationItem objects -examples of SyndicationItem are blog posts, news articles etc.

SyndicationItem

represents an individual RSS item. We will be accessing the following properties on the SyndicationItem :-

- Title – The title of the feed item.
- PublishDate – Publication date of the item
- URL – the url for the item

The component will be reading data from the SyndicationItem objects, and the properties mentioned above (Title, PublishDate, URL), will be available in the SSIS data flow.

With this basic information, add the following method to your CustomSSISComponent class. This method reads syndication items from an RSS endpoint, and returns all the items for that feed.

```
public DataTable GetRSSDataTable(String propertyValue)
{
    DataTable dt = new DataTable();
    try
    {
        XmlReader reader = XmlReader.Create(propertyValue);
        SyndicationFeed feed = SyndicationFeed.Load(reader);

        dt.Columns.Add("Title", Type.GetType("System.String"));
dt.Columns.Add("PublishDate", Type.GetType("System.DateTime"));
dt.Columns.Add("URL", Type.GetType("System.String"));

        foreach (SyndicationItem item in feed.Items)
        {
            DataRow dRow = dt.NewRow();
```

```
            dRow["Title"] = item.Title.Text;
            dRow["PublishDate"] = item.PublishDate.DateTime;

            dRow["URL"] = item.Id;

dt.Rows.Add(dRow);

        }


        return dt;

    }

    catch (Exception e)

    {

        throw e;

    }


}
```

Save the updated file – we will be adding more code in a little bit. The above method reads data from an RSS data source, and returns a datatable with all the data from the source.

Note: Although I have used RSS as a data source here, you can read data from a third party application, or even a cloud-based platform like Salesforce.com.

In the subsequent sections below, I will be covering the individual structures that have to be built when building a custom SSIS component. On a very high level, it involves the following –

- Creating a custom property on the component for the URL of the RSS feed.
- Create an output and add output columns to it. Remember that we are creating an ―input‖ component that reads from an external data source. Consequently, we need to explicitly create the ―output‖ on which the component can send data to the next component in the SSIS pipeline.
- Send the data read from the external data source to the output.

Create component outputs and properties

All Integration Services components have 0 – N inputs and outputs. An input represents a connection point between two components. When an output of one component is connected to the input of another component, it establishes the flow of data between components. We need to explicitly create any inputs/outputs and custom properties for our component. The best place to do this is the overridden ProvideComponentProperties method. This method is called only once, when the component is added to the SSIS package designer. In the CustomSSISComponent.cs file created above, modify the method definition as below:-

```
public override void ProvideComponentProperties()

{
    base.ProvideComponentProperties();
base.RemoveAllInputsOutputsAndCustomProperties();

    IDTSCustomProperty100 rsspath = ComponentMetaData.CustomPropertyCollection.New();
    rsspath.Description = "The path of the RSS feed";
    rsspath.Name = "RSS Path";
rsspath.Value = String.Empty;

    IDTSOutput100 output = ComponentMetaData.OutputCollection.New();
output.Name = "Output";
}
```

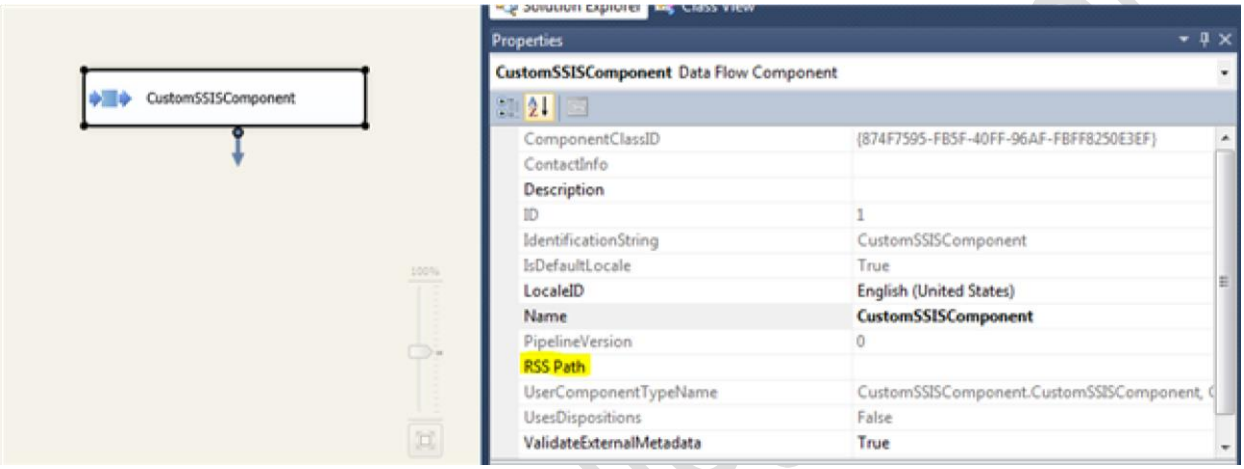
The method is adding a custom property called RSS Path to store the URL of the RSS feed that the component will read from. Also, since we are developing a custom source component, we are also adding one output to the component.

Couple of things to note here:-

- Though we have added an output, we have not added any output columns yet. We will be adding these later in the article.

- The data type of the RSS Path custom property is determined the first time you assign a value to the property – in this case since the initial value was an empty string, the property data type is string.

Compile and re-deploy the component using the steps listed in the *"Deploying the custom component"*section above. Close SQLServer Data Tools and re-open the empty Integration Services package that you created above, and add a new Data Flow component. You should now be able to drag your custom component on the package designer, and also update the *RSS Path* custom property.



Building output columns

In the previous section, we saw how to add an output to our custom Integration Services source component. The next step is to add columns to the output that we added earlier. There are two rules to keep in mind when adding output columns:-

- The number of columns should match with that being read from the data source. Since we will be reading three fields from the RSS feed (Title, Publish Date, and URL), we need three output columns in the output.
- The data types of the columns should be compatible with that being read from the data source.

Replace the *SetComponentProperty* method with the below:-

```
public void AddOutputColumns(String propertyValue)
{
   DataTable dt = GetRSSDataTable(propertyValue);
   if (dt != null)
```

```
                {
            //Check if there are any rows in the datatable
        if (dt.Rows != null && dt.Rows.Count > 0)
                {
                    DataTable schemaDT = dt.CreateDataReader().GetSchemaTable();
                foreach (DataRow row in schemaDT.Rows)
                    {
            IDTSOutputColumn100 outputCol = ComponentMetaData.OutputCollection[0].

                    bool isLong = false;
                    DataType dType = DataRecordTypeToBufferType((Type)row["DataType"]);
                dType = ConvertBufferDataTypeToFitManaged(dType, ref isLong);
int length = ((int)row["ColumnSize"]) == -1 ? 1000 : (int)row["ColumnSize"];
int precision = row["NumericPrecision"] is System.DBNull ? 0 : (short)row["Nu
int scale = row["NumericScale"] is System.DBNull ? 0 : (short)row["NumericSc
int codePage = schemaDT.Locale.TextInfo.ANSICodePage;

                    switch (dType)
                    {
                        case DataType.DT_STR:
                    case DataType.DT_TEXT:
                            precision = 0;
                scale = 0;
```

```
break;                                  case
DataType.DT_NUMERIC:
                                        length = 0;
                        codePage = 0;
if (precision > 38)                     precision = 38;
if (scale > precision)
                                            scale = precision;
                                        break;
 case DataType.DT_DECIMAL:
                                        length = 0;
precision = 0;                                      codePage = 0;
                                        if (scale > 28)
                            scale = 28;
break;                                  case
DataType.DT_WSTR:
                                        precision = 0;
                                        scale = 0;
                                codePage = 0;
                                        break;
                        default:
length = 0;
precision = 0;
scale = 0;
```

```
codePage = 0;
break;
                    }
                        outputCol.Name = row["ColumnName"].ToString();
                    outputCol.SetDataTypeProperties(dType, length, precision, scale, codePage);
                    }
                }
            }
        }

    public override IDTSCustomProperty100 SetComponentProperty(string propertyName, ob
        {
            if (propertyName == "RSS Path" &&
        ComponentMetaData.OutputCollection[0].OutputColumnCollection.Count == 0)
            {
                AddOutputColumns(propertyValue.ToString());
            }

            return base.SetComponentProperty(propertyName, propertyValue);
        }
```

The goal of the AddOutputColumns ( ) method above is to identify the columns coming from the data source (GetRSSDataTable ()), and add the corresponding output columns to the Output. Here is a step by step description of what it is doing:

- Read the data from the RSS data source using the GetRSSDataTable ( ) method that we defined earlier, and extract the schema of the resulting data table – we use the GetSchemaTable () method of the datatable to get the schema.
- Each row in the schema describes one column in the datatable . For each such row, add an output column to the component output using ComponentMetaDataÂ.OutputCollection[0]Â.OutputColumnCollectionÂ.New().
- Next, we map the datatable column data type to a corresponding Integration Services data type. Integration Services has its own data type model, which is separate from the .Net data types. For e.g., a String data type in .Net can be a DT_STR or DT_WSTR type in Integration Services. Since output columns on Integration Services components use only Integration Services data types, we need to assign a compatible data type to the output column, depending on the type of data that will be flowing through it. We use the following two methods to map the column data type to a corresponding type in the Integration Services object model:-
    - DataRecordTypeToBufferType – given a managed data type (of type System.Type), this method returns the corresponding Integration Services data type (of type MicrosoftÂ.SQLServerÂ.DtsÂ.RuntimeÂ.WrapperÂ-.Datatype).
    - ConvertBufferDataTypeToFitManaged – accepts an Integration Services data type as argument, and returns the Integration Services data type that is appropriate for managed code.
- The two-step approach shown in the code above is required to get the correct Integration Services data type for a managed type. The reason we have two steps is that the DataRecordTypeToBufferType does not always return a type that can be used – for e.g., for a Boolean data type, the DataRecordTypeToBufferType method returns the Integration Services type DT_ BOOL . However, the call to ConvertBufferDataTypeToFitManaged gets the correct type, which is DT_I4 .
- In addition to identifying the correct Integration Services data type, your code will also have to set the precision, scale, length and codepage properties on the output column. All these properties, except codepage, are available in the datatable schema object. However, certain properties (e.g. length) may not be available for some columns, in this case, make sure you assign a default value (in this case, I've added a default length of 1000).
- Next, we use a switch statement to assign the correct values for length, precision, scale and codepage. A complete list of requirements for all applicable data types can be found here. The switch is required because different data types have different requirements of length, precision and scale, as shown below:

DataType	Length	Scale	Precision	CodePage
DT_DECIMAL	0	Greater than 0 and less than or equal to 28.	0	0
DT_CY	0	0	0	0

a

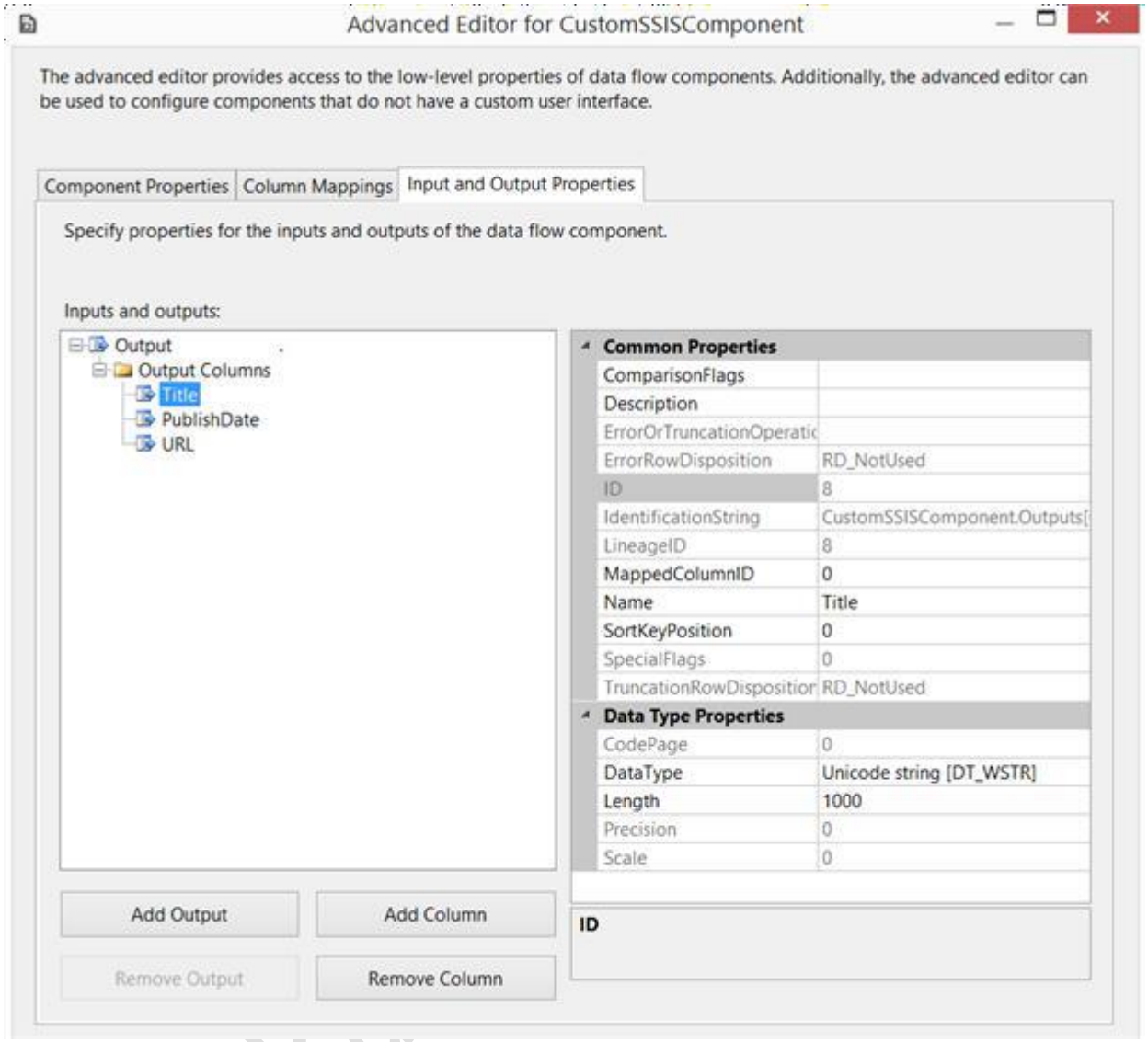
DT_NUMERIC	0	Greater than 0 and less than or equal to 28, and equal to Precision.	Greater than or 0 than 1 and less less than than or equal to 38.
DT_BYTES	Greater than 0.	0	0
DT_STR	Greater than 0 and less	0 valid code page.	Not 0, and than 8000.
DT_WSTR	Greater than 0 and less than 4000.	0	0

- Finally, set the output column name to the column name from the data table, and call the SetDataTypeProperties method to set the precision, length, scale and codepage properties for the output column.

The AddOutputColumns ( ) method is invoked from the SetComponentProperty () method because I want to add output columns only when the ―RSS Path‖ property is set to a legitimate RSS path. Alternatively, since we already know the three columns we will be reading from the RSS feed, we could also have added the output columns in the ProvideComponentProperties method itself.

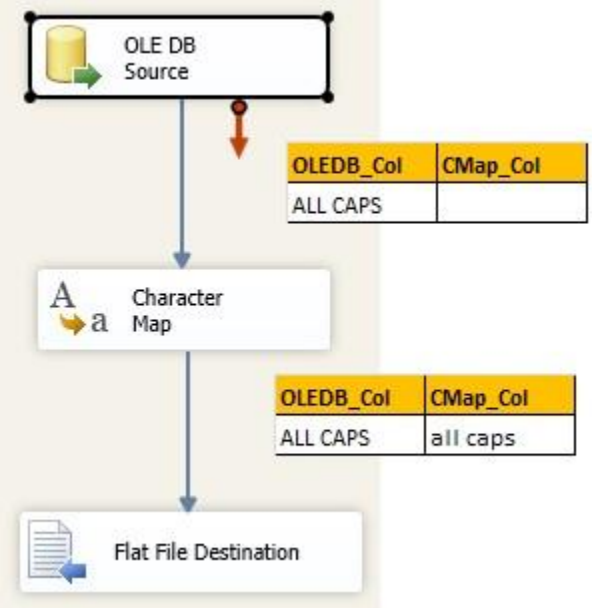
At this stage, build and redeploy your custom component using the steps in the ―Deploying your custom component" section above, and add it again to a test Integration Services package. You should be able to see the RSS Path custom property in the component properties – set the RSS Path property to http://rss.cnn.com/rss/cnn_topstories.rss – this will trigger the call to AddOutputColumns() .

Inspect your component properties, and you should be able to see the output columns on the Input and Output properties tab. Note how the data types for each column are Integration Services data types as a result of the type conversion that we discussed earlier.



Output column mapping

During package execution, a data source component extracts data from an external source and writes it to one or more similar data buffers. The data buffers are supplied by the Integration Services runtime, and are tabular structures with rows and columns. However, no direct mapping exists between the component output columns and the columns in the data buffers. To understand why, look at the sample package below:



In this sample package, we have an OLE DB Source that reads data – in this case, it reads a single column of data called OLEDB_Col . It is followed by a Character Map transformation that adds a new column called CMap_Col that converts the data in OLEDB_Col to all lower case letters. Finally, data is written to a file via the Flat File Destination component.

A single data buffer is created for all 3 components in the package – this is so that the Integration Services runtime does not have to incur the overhead of adding new columns during package execution for each component to the buffer. The downside of this is that the buffer has to accommodate the columns used by all the components – in this example, the buffer for OLE DB Source contains the column CMap_Col even though the OLE DB Source has nothing to do with it.

Because the columns in the buffer may be different than the output columns on the component, your component should always explicitly map its output columns to columns in the data buffer, so that it does not accidentally overwrite columns that are meant for other components. The best place to do this is in the PreExecute method, since it is called only once just prior to package execution. Modify the PreExecute method in the component code as below:-

```
public int[] mapOutputColsToBufferCols;


public override void PreExecute()

{

   base.PreExecute();
   IDTSOutput100 output = ComponentMetaData.OutputCollection[0];

mapOutputColsToBufferCols = new int[output.OutputColumnCollection.Count];
```

```
for (int i = 0; i < ComponentMetaData.OutputCollection[0].OutputColumnCollection.Count; i++
{
    // Here, "i" is the column count in the component's outputcolumncollection
    // and the value of mapOutputColsToBufferCols[i] is the index of the corresponding column i
// buffer.
    mapOutputColsToBufferCols[i] = BufferManager.FindColumnByLineageID(output.Buffer, o
}
}
```

In this method, we iterate through the output columns for our source component, and add the index of the corresponding column in the data buffer to the map – an array in this case. The key method here is BufferManager.FindColumnByLineageId , which finds the matching column in the data buffer based on the output column‘s Lineage Id.

Sending rows to the output

The final step in building our component is to define the runtime method PrimeOutput , which is responsible for sending rows to the output buffers during package execution. Note that the PrimeOutput method is a runtime method, and is called only at package execution time. The process of reading data from the external data source, and writing it to the data buffers should be implemented in this method. Replace the blank PrimeOutput method in your implementation withwith the below method definition:- to your component:-

```
public override void PrimeOutput(int outputs, int[] outputIDs, PipelineBuffer[] buffers)
{
    base.PrimeOutput(outputs, outputIDs, buffers);

    IDTSOutput100 output = ComponentMetaData.OutputCollection.FindObjectByID(outputIDs[0
    PipelineBuffer buffer = buffers[0];

DataTable dt = GetRSSDataTable(ComponentMetaData.CustomPropertyCollection["RSS Path "
```

```
            foreach (DataRow row in dt.Rows)
            {
               buffer.AddRow();

               for (int x = 0; x < mapOutputColsToBufferCols.Length;  x++)
               {
                  if (row.IsNull(x))
buffer.SetNull(mapOutputColsToBufferCols[x]);
                  else
                     buffer[mapOutputColsToBufferCols[x]] = row[x];
               }
            }

            buffer.SetEndOfRowset();
         }
```

- The PrimeOutput method takes as parameters the total number of outputs on the component, an array with the IDs of all outputs, and an array of buffers which are used by these outputs.
- Since we have only one output on our component, we locate the only output with the FindObjectById method.
- Read data from the RSS feed into a datatable , and iterate through the rows.
- For each row from the datatable , create a new row in the buffer and write its contents. Locate the corresponding column in the buffer using the mapOutputColsToBufferCols array created during PreExecute .
- When all the rows have been written to the buffer, call the SetEndOfRowset method to signal that the component is done writing rows to the output.

When SetEndOfRowset is called, the Integration Services runtime flushes out any remaining rows in the buffer and moves processing to the next component in the package.

Deploy the component using the steps detailed earlier – you should now be able to use it just like any other data source component in an SSIS package. For example, in the below screenshot, I'm using the component to read data from an RSS data source and load it into a SQLServer table.