**CDC in SSIS for SQL Server 2012**

SQL Server 2012 introduces new components that make it easier to do [Change Data Capture (CDC)](http://msdn.microsoft.com/en-us/library/bb522489.aspx) using SSIS. This blog post provides a quick walkthrough of how to use them.

***Note:*** *The documentation for the CDC components is not in Books Online yet. It will be appearing in the RTM refresh.*

**New Task and Components**

**CDC Control Task**

The CDC Control task is used to control the life cycle of change data capture (CDC) packages. It handles CDC package synchronization with the initial load package, the management of [Log Sequence Number (LSN)](http://msdn.microsoft.com/en-us/library/ms190411.aspx) ranges that are processed in a run of a CDC package. In addition, the CDC Control task deals with error scenarios and recovery.

**CDC Source**

The CDC source reads a range of change data from CDC change tables and delivers the changes downstream to other SSIS components.

**CDC Splitter**

The CDC splitter splits a single flow of change rows from a CDC Source component into different data flows for Insert, Update and Delete operations. It is essentially a “smart” Conditional Split transform that automatically handles the standard values of the **\_\_$operation** column.

**Walkthrough**

**Database Setup**

For sample data, we will create a new database (**CDCTest**), and select a subset of rows from the [AdventureWorksDW](http://msftdbprodsamples.codeplex.com/) **DimCustomer** table into a sample table (**DimCustomer\_CDC**). This will become the *Source* table for this demo.



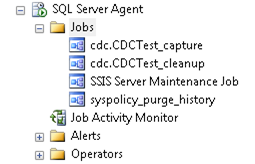
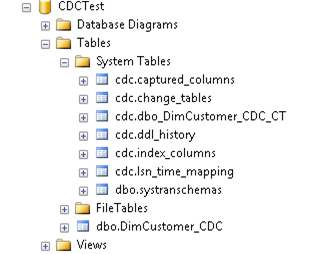
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | USE [CDCTest]  GO  SELECT \* INTO DimCustomer\_CDC  FROM [AdventureWorksDW].[dbo].[DimCustomer]  WHERE CustomerKey < 11500 |

We then [enable CDC on the database](http://technet.microsoft.com/en-us/library/cc627369.aspx), and [create a capture instance](http://technet.microsoft.com/en-us/library/bb522475.aspx) for the DimCustomer\_CDC table.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | USE [CDCTest]  GO  EXEC sys.sp\_cdc\_enable\_db  GO  -- add a primary key to the DimCustomer\_CDC table so we can enable support for net changes  IF NOT EXISTS (SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID(N'[dbo].[DimCustomer\_CDC]') AND name = N'PK\_DimCustomer\_CDC')  ALTER TABLE [dbo].[DimCustomer\_CDC] ADD CONSTRAINT [PK\_DimCustomer\_CDC] PRIMARY KEY CLUSTERED  (  [CustomerKey] ASC  )  GO  EXEC sys.sp\_cdc\_enable\_table  @source\_schema = N'dbo',  @source\_name = N'DimCustomer\_CDC',  @role\_name = N'cdc\_admin',  @supports\_net\_changes = 1  GO |

We can see that a number of tables have been added under the **cdc** schema, and that SQL agent jobs have been created to capture changes being made to this table.



For the *Destination*, we’ll create a separate table – **DimCustomer\_Destination –** with the same structure as the Source.



|  |  |
| --- | --- |
| 1  2 | SELECT TOP 0 \* INTO DimCustomer\_Destination  FROM DimCustomer\_CDC |

In real life this would be in a separate database, and usually on a completely different server (otherwise, why are you mirroring the changes?), but for the purposes of this walkthrough, we’ll keep it all together.

We’re ready to start consuming changes with SSIS.

**SSIS Packages**

Our processing logic will be split into two packages – an **Initial Load** package that will read all of the data in the source table, and an **Incremental Load** package that will process change data on subsequent runs.

**Initial Load**

This package will only be run once, and handles the initial load of data in the source table (DimCustomer\_CDC). The package uses the following logic:

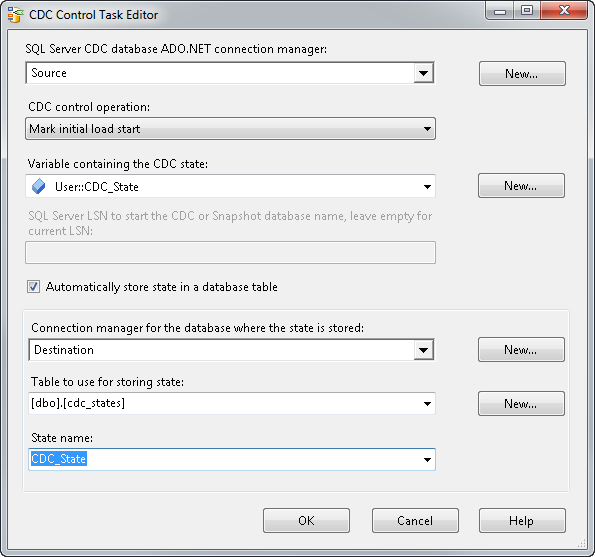
1. Use the CDC Control Task to mark the initial load start LSN
2. Transfer all of the data from the source table into our destination table
3. Use the CDC Control Task to mark the initial load end LSN

Package creation steps:

Create a new SSIS package

Add a CDC Control Task. Double click the Control Task to bring up the editor.

* Add a new ADO.NET connection manager for the Source database
* Set CDC Control Operation to **Mark initial load start**
* Create a new package variable (CDC\_State) to hold the CDC state information.
* Set the connection manager for the Destination database
* Create a table for storing the state (**[cdc\_states]**). This table will be used to track the CDC load information, so that you only pick up new changes each time the incremental load package is run. It will be created in the Destination database.
* Set the state name (**CDC\_State**). This value acts as a key for the CDC state information. Packages that are accessing the same CDC data should be using a common CDC state name.

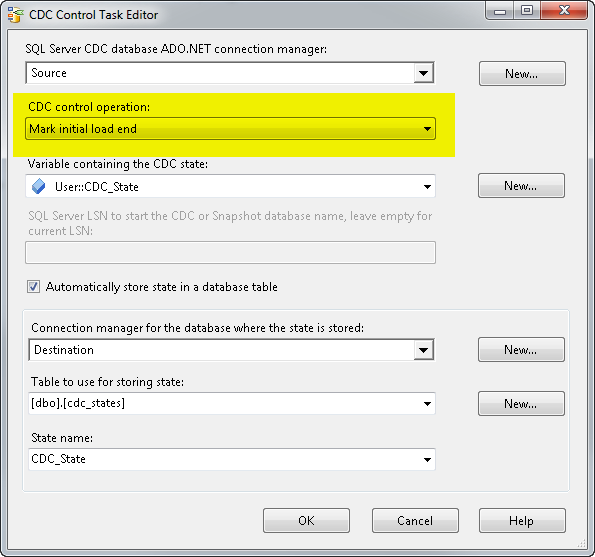


Add a Data Flow Task, and connect it to the CDC Control Task

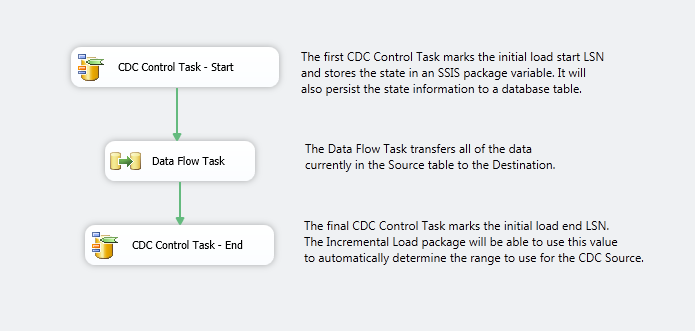
* Configure the Data Flow task to transfer all of the data from the Source to the Destination

Add a second CDC Control Task. Connect the success constraint of the Data Flow Task to it.

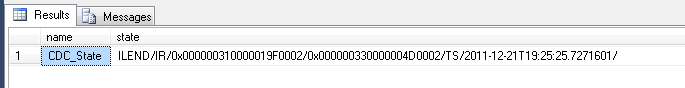
* Configure the second CDC Control Task with the same settings as the first one, except the CDC Control Operation should be set to **Mark initial load end**.



The package will now look like this:



When we run the package, all of the data currently in the Source table will be transferred to the Destination, and the initial CDC state markers will be created. If we select from the **cdc\_states** table, we can see that there is now a “CDC\_State” entry. Note, the state entry is an encoded string that is used by the CDC components – you should not have to edit or deal with it directly.



**Incremental Load**

This package will be run every time we want to grab the latest changes from our Source table. It will store the CDC state every time it runs, ensuring that we only pick up new changes every time we run the package. It will use the following logic:

1. Create staging tables for updated and deleted rows (so we can process the changes in a batch – more about that below)
2. Use a CDC Control Task to retrieve the CDC state from the destination table
3. Use a CDC Source to retrieve our change data
4. Use a CDC Splitter transform to redirect the rows based on their operation (New, Updated, and Deleted)
5. Insert the new rows into the Destination table
6. Stage the Updated and Deleted rows
7. Process the Updated and Deleted rows using Execute SQL Tasks
8. Use a CDC Control Task to update the CDC state

Package creation steps:

Add an Execute SQL Task to create staging tables

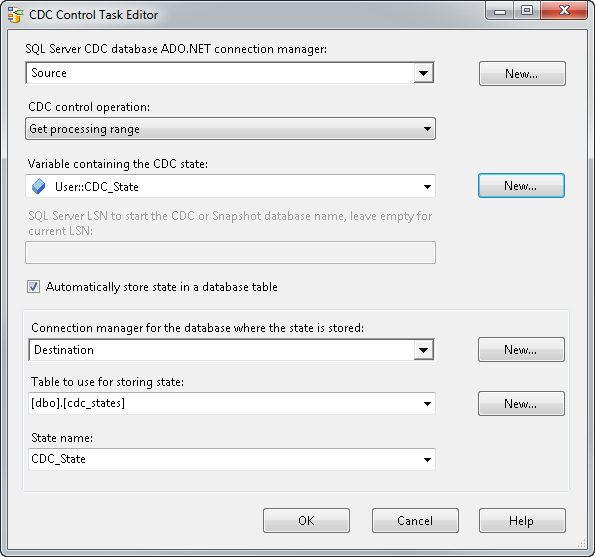
* Create a connection manager for the Destination database (set the ConnectionType to ADO.NET to reuse the same connection manager used by the CDC Control Task)
* Enter the SQL statements to create two staging tables that match the Destination table. For example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | IF NOT EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID(N'[dbo].[stg\_DimCustomer\_UPDATES]') AND type in (N'U'))  BEGIN  SELECT TOP 0 \* INTO stg\_DimCustomer\_UPDATES  FROM DimCustomer\_Destination  END  IF NOT EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID(N'[dbo].[stg\_DimCustomer\_DELETES]') AND type in (N'U'))  BEGIN  SELECT TOP 0 \* INTO stg\_DimCustomer\_DELETES  FROM DimCustomer\_Destination  END |

Add a CDC Control Task. Connect the Execute SQL task to the CDC Control Task

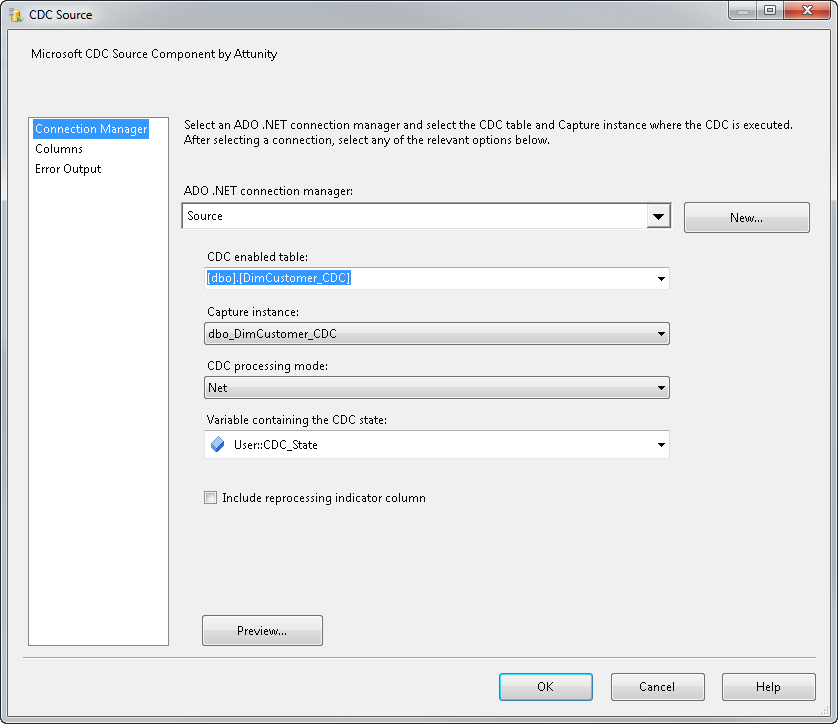
* Create a connection manager for the Source database
* Set the CDC Control Operation to **Get processing range**
* Create a new CDC state variable (**CDC\_state**)
* Create a connection manager for the Destination database
* Select the state table (this was created by the Initial Load package) – **[dbo].[cdc\_states]**
* Set the State name – this must match what was used in the Initial Load package (CDC\_State)



Add a Data Flow Task. Connected it to the CDC Control Task.

Add a CDC Source component

* Set the Connection Manager to the Source database
* Select the source table (DimCustomer\_CDC)
* Set the CDC processing mode to **Net**
* Select the CDC\_State variable
* Click the Columns tab to make sure we’re pulling back all of the right information, then click OK.

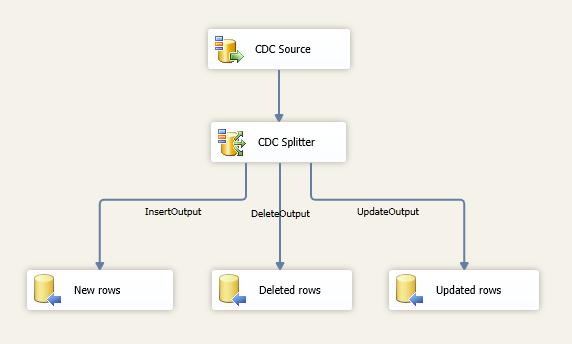


Add a CDC Splitter transform

Add an ADO.NET Destination – rename it to “New rows”

* Connect the InsertOutput of the CDC Splitter to the “New rows” destination
* Double click the “New rows” destination to bring up its editor
* Set the Destination connection manager, and select the main destination table (**DimCustomer\_Destination**)
* Click the Mappings tab. The columns should automatically match by name. The CDC columns (the ones starting with \_\_$) can be ignored

Add two more ADO.NET Destinations, mapping the **DeleteOutput** to the **stg\_DimCustomer\_DELETES** table, and **UpdateOutput** to **stg\_DimCustomer\_UPDATES**. We will update the final Destination table using batch SQL statements after this data flow. An alternative design here would be to use an OLE DB Command transform to perform the updates and deletes. The OLE DB Command approach has some performance problems though, as the transform operates on a row by row basic (i.e. it issues one query per row).



Back in the Control Flow, add two Execute SQL tasks. These tasks will perform the batch update/delete using the data we loaded into the staging tables. The queries look like this (note, I took columns out of the update statement to keep things short – normally you’d include all of the columns here):

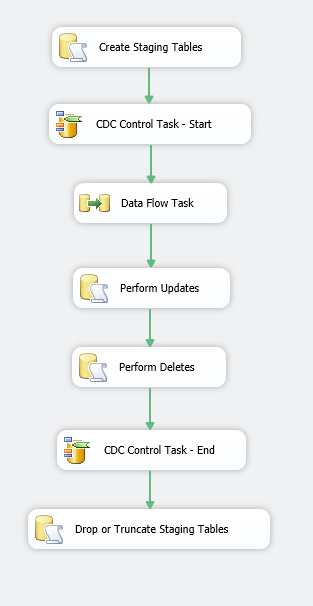


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | --  -- These queries go into the incremental load package, and do not need to be run directly  --  -- batch update  UPDATE dest  SET  dest.FirstName = stg.FirstName,  dest.MiddleName = stg.MiddleName,  dest.LastName = stg.LastName,  dest.YearlyIncome = stg.YearlyIncome  FROM  [DimCustomer\_Destination] dest,  [stg\_DimCustomer\_UPDATES] stg  WHERE  stg.[CustomerKey] = dest.[CustomerKey]  -- batch delete  DELETE FROM [DimCustomer\_Destination]  WHERE[CustomerKey] IN  (  SELECT [CustomerKey]  FROM [dbo].[stg\_DimCustomer\_DELETES]  ) |

Add a CDC Control Task. It should have the same settings as the first CDC Control Task in the package, except the CDC control operation is **Mark processed range**.

Finally, add an Execute SQL Task to drop the staging tables. Alternatively, you can leave the staging tables in place, just truncate them.

Your package should look like this:



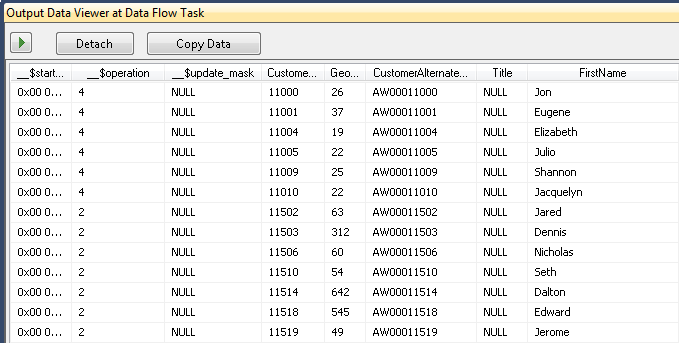
**Running the Incremental Load Package**

If we run the Incremental Load package at this point, it should run successfully, but not transfer any rows. That’s because we haven’t made any changes yet to the Source table. Let’s do that now by running the following script against the Source table:

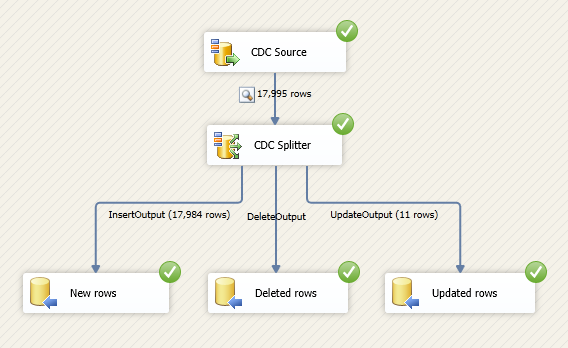


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | USE [CDCTest]  GO  -- Transfer the remaining customer rows  SET IDENTITY\_INSERT DimCustomer\_CDC ON  INSERT INTO DimCustomer\_CDC  (  CustomerKey, GeographyKey, CustomerAlternateKey, Title, FirstName,  MiddleName, LastName, NameStyle, BirthDate, MaritalStatus,  Suffix, Gender, EmailAddress, YearlyIncome, TotalChildren,  NumberChildrenAtHome, EnglishEducation, SpanishEducation,  FrenchEducation, EnglishOccupation, SpanishOccupation,  FrenchOccupation, HouseOwnerFlag, NumberCarsOwned, AddressLine1,  AddressLine2, Phone, DateFirstPurchase, CommuteDistance  )  SELECT CustomerKey, GeographyKey, CustomerAlternateKey, Title, FirstName,  MiddleName, LastName, NameStyle, BirthDate, MaritalStatus,  Suffix, Gender, EmailAddress, YearlyIncome, TotalChildren,  NumberChildrenAtHome, EnglishEducation, SpanishEducation,  FrenchEducation, EnglishOccupation, SpanishOccupation,  FrenchOccupation, HouseOwnerFlag, NumberCarsOwned, AddressLine1,  AddressLine2, Phone, DateFirstPurchase, CommuteDistance  FROM [AdventureWorksDW].[dbo].[DimCustomer]  WHERE CustomerKey &lt; 500  SET IDENTITY\_INSERT DimCustomer\_CDC OFF  GO  -- give 10 people a raise  UPDATE DimCustomer\_CDC  SET  YearlyIncome = YearlyIncome + 10  WHERE  CustomerKey &lt;= 11000 AND CustomerKey &gt;= 11010  GO |

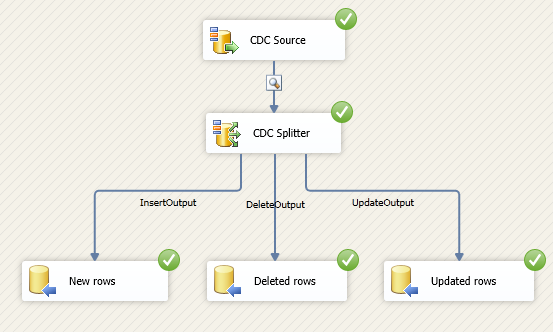
If we enable a Data Viewer in the Incremental Load package and run it, we’ll see that the CDC Source picks up all of the rows we’ve changed. We can see that some of the rows are \_\_$operation = 4 (update), while the rest are 2 (new rows).



When the package completes, we see that the data flow moved a total of 17,995 rows (11 updates, and the rest are inserts).



Because the CDC Control Task updated LSN values stored in the CDC state table, if we run the package a second time (without making any changes to the source table), we see that no rows get transferred in the data flow.



**Wrap Up**

I hope you found this walkthrough of the new CDC components in SQL Server 2012 helpful. I will continue posting more information about the CDC Components in follow up posts. Please let me know if you have any specific questions / topics you’d like me to describe further.