

# Change Data Capture since Last Request Package Sample

SQL Server Technical Article

Writer: Sandra Ward

Published: 11 2008

Applies To: SQL Server 2008

Summary: These are the accompanying notes describing the Change Data Capture since Last Request Package Sample available on Codeplex. The sample demonstrates the use of CDC technology in support of SSIS incremental load packages using LSN based queries.

|  |
| --- |
|   |
|  |

### Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

 2008 Microsoft Corporation. All rights reserved.

Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

### Table of Contents

[Getting Started 1](#_Toc213211161)

[Sample Structure 1](#_Toc213211162)

[The Sample Environment Test Harness 2](#_Toc213211163)

[Package Architecture for Incremental Load 2](#_Toc213211164)

[SetupCDCSample Package 3](#_Toc213211165)

[Initializing the Environment and Enabling Change Data Capture 3](#_Toc213211166)

[Execute SQL Task - Create Tables and Enable Change Data Capture 4](#_Toc213211167)

[Creating Capture Instances for the Source Tables 5](#_Toc213211168)

[Generating the Sample Workload 6](#_Toc213211169)

[Script Tasks – Apply Inserts and Updates to CDC Enabled Tables 6](#_Toc213211170)

[Script Task – Mark Workload Completion 7](#_Toc213211171)

[Verifying the Capture Process is Running 8](#_Toc213211172)

[The Execute SQL Task – Verify Capture Process is Started 8](#_Toc213211173)

[Script Task – Log Capture Not Started Message 9](#_Toc213211174)

[Initializing the Target Tables 11](#_Toc213211175)

[Execute SQL Task – Create Database Snapshot 12](#_Toc213211176)

[Data Flow Tasks to Perform the Initial Load 12](#_Toc213211177)

[Preparing for the Initial Extraction 13](#_Toc213211178)

[Execute SQL Task – Get End-Points for Initial Query Interval 13](#_Toc213211179)

[Extracting and Processing Change Data 15](#_Toc213211180)

[For Loop Container - Cycle Master at 10 Second Intervals 15](#_Toc213211181)

[Validating the Incremental Load and Reporting Completion Status 17](#_Toc213211182)

[Execute SQL Task – Check for Mismatch in Replicas 17](#_Toc213211183)

[VB Script Task – Output Run Completion Status 18](#_Toc213211184)

[SetupCDCSample Package Variables 20](#_Toc213211185)

[MasterCDC Package 21](#_Toc213211186)

[The Change Data Capture Database Validity Interval 22](#_Toc213211187)

[The Change Data Capture Validity Interval for a Capture Instance 22](#_Toc213211188)

[Master Package Configurations and Variables 24](#_Toc213211189)

[Package Configurations for Initializing Package Variables 24](#_Toc213211190)

[Master Package Variables 24](#_Toc213211191)

[Master Package Tasks 25](#_Toc213211192)

[Execute SQL Task – Check for Valid Interval 25](#_Toc213211193)

[Execute Package Tasks – Extract Change Data for CDC Enabled Tables 26](#_Toc213211194)

[Script Task – Log Extract Error 26](#_Toc213211195)

[Script Task – Log Extraction Complete 28](#_Toc213211196)

[Child Packages for the Change Data Capture since Last Request Package Sample 30](#_Toc213211197)

[Child Package Variables 31](#_Toc213211198)

[Child Package Tasks 31](#_Toc213211199)

[Script Task - Generate SQL Data Query 31](#_Toc213211200)

[Data Flow Task – Process Change Data 32](#_Toc213211201)

[Error Logging in the Child Packages 36](#_Toc213211202)

[Running the Change Data Capture since Last Request Package Sample in BI Studio 37](#_Toc213211203)

[*Conclusion* 38](#_Toc213211204)

#### Getting Started

This document goes through the Change Data Capture since Last Request Package Sample in detail, describing how to use CDC change tables to support ETL using SSIS packages. Code for the Change Data Capture LSN since Last Request Package Sample is available through CODEPLEX. The README supplied with the sample provides detailed information on installing the sample files locally, along with the requirements needed to run the sample. This accompanying document is available on MSDN.

The Change Data Capture since Last Request Package Sample makes use of the databases AdventureWorks2008 and AdventureWorksDW2008, both of which are available for download on Codeplex. Follow the instructions at the download site to insure they are properly loaded in your environment.

NOTE: After loading AdventureWorks2008, execute sp\_helpusers to determine whether the database user ‘dbo’ has an associated login. If the LoginName for the returned ‘dbo’ entry is NULL, the database user ‘dbo’ has been orphaned. Run the following command to associate the database user ‘dbo’ with the login ‘sa.’

 exec sp\_changedbowner 'sa'

The database user ‘dbo’ must be associated with a valid login in order for the Change Data Capture capture process to successfully harvest changes from the log and deposit them in the database change tables.

There are two package variables in the SetupCDCSample package that are key to successfully running the sample: **SQLServerInstallPath** and **BasePath**. **SQLServerInstallPath**, which defaults to c:\program files\Microsoft SQL Server\ identifies the install path for SQL Server on the local machine. If this is different in your environment, modify the package variable appropriately. **BasePath**, which defaults to @[SQLServerInstallPath] + “100\Samples\Integration Services\Package Samples\Change Data Capture since Last Request Package Sample\Change Data Capture Sample\”, identifies the standard install path of the sample relative to the SQL Server install path. If you have installed the sample elsewhere, set this path appropriately.

#### Sample Structure

The Change Data Capture feature of SQL Server 2008 allows the DML activity against database tables to be captured in change tables that reside on the source database. This sample demonstrates how CDC technology can be used to systematically harvest those changes using SSIS packages. The technology described is suitable when there are periodic requests for all of the changes that have been logged since the previous request.

##### The Sample Environment Test Harness

A setup package is provided that both initializes the sample environment and provides a test harness for driving the data extraction process. The package begins by initializing the source tables to an initial state. It then enables the source database for Change Data Capture and creates a capture instance for each table to be tracked. The package then launches several tasks to generate DML activity against the source tables. At the same time that DML is being applied to the source tables, the setup package verifies that the capture process is active. If it is not, the Setup package logs a message to the event log to indicate that the capture process has not been started and terminstes. If the capture process is running, a database snapshot is then taken by the Setup package. The snapshot is used to provide data for an initial load of the remote source replicas.

Once the initial load completes, the setup package determines the LSN range to use for the first incremental load. The stored procedure sys.sp\_cdc\_dbsnapshotLSN is used to obtain an appropriate low end-point that will synchronize the first extraction interval with the initial load. The function sys.fn\_cdc\_get\_max\_lsn() is used to determine the high end-point for the first interval. The Setup package then enters a loop that launches the master package to harvest changes at 10 second intervals. It is the action of the master package and its associated child packages to periodically request all of the changes occurring since the last request that represents the principle focus of the sample. The loop logic continues to monitor the progress of the DML tasks through global variables. When the loop logic determines that the DML tasks have all completed and that the window of the next extraction interval has moved beyond the period of time when workload was applied, the loop logic terminates. On completion, SQL CHECKSUM is used to compare the source files to the replicas. Table differences, if present, are noted in an event log entry that is made to record the status of the completed run.

##### Package Architecture for Incremental Load

The core of the sample consists of four packages: one master package and three child packages. The master package obtains the LSN values to mark the extraction interval from the setup package, and then verifies that the interval lies within the current Change Data Capture validity interval for the database. If either the low end-point of the extraction interval is smaller than the current minimum LSN of the database validity interval, or the high end-point is greater than the current maximum LSN for the database validity interval, the master package logs an error to the application event log and terminates. This is generally an indication that change table cleanup has been too aggressive, and the needed change data is no longer available in the change tables. In principle, there should be no need to check the high end-point of the extraction interval, since it was obtained in the Setup package by querying for the current maximum LSN. It is nevertheless checked to verify that the interval has been correctly determined.

The master package then launches the child packages to retrieve the change data and update the source replicas. The master package waits for notification from all packages before writing an informational message to the application event log at the conclusion of the extraction cycle. The log entry identifies the start and end LSNs of the extraction interval as well as other packages variables.

#### SetupCDCSample Package

The package SetupCDCSample.dtsx sets up the test environment for the sample. It uses AdventureWorks2008 as its source database, and AdventureWorksDW2008 as the target. The database snapshot AdventureWorks2008\_dbss is created for AdventureWorks2008 to provide a consistent view of the source tables for the initial load. The diagram below shows the control flow for the setup package.



Figure 1: SETUPCDCSAMPLE Package

##### Initializing the Environment and Enabling Change Data Capture

###### Execute SQL Task - Create Tables and Enable Change Data Capture

The SetupCDC Sample package begins with a single SSIS Execute SQL Task. Its purpose is to run the T-SQL script CDCSetupTables.sql to setup the environment for Change Data Capture.

The script begins by generally cleaning up the sample environment, removing any objects created in previous runs. This makes it straightforward to rerun the sample and still allow the created objects to endure at the end of the run.

The script then enables Change Data Capture for the database AdventureWorks2008. The Change Data Capture feature of SQL Server 2008 that allows the DML activity against database tables to be captured in change tables must initially be enabled at the database level by a member of the fixed server sysadmin role.

You can use the following T-SQL query to determine whether Change Data Capture is already enabled for a database:

 SELECT is\_cdc\_enabled from sys.databases

 WHERE name = 'AdventureWorks2008'

 AND is\_cdc\_enabled = 1

Within the sample, this query is used to determine whether Change Data Capture has already been enabled for the database. If it is, the following stored procedure is run to disable it.

 exec sys.sp\_cdc\_disable\_db

Disabling Change Data Capture at the database level will cleanup all of the Change Data Capture metadata for the database including the metadata associated with capture instances that have have previously been created for tracked tables. This allows each execution of the sample to execute from a clean test environment.

The following stored procedure is then executed to enable Change Data Capture for AdventureWorks2008:

 exec sys.sp\_cdc\_enable\_db

The script then removes preexisting tables and functions from the schema CDCSample in AdventureWorks2008. The schema CDCSample is used for the new tables that will be created to serve as the source tables for the SSIS packages used to extract change data.

The following three tables are created in the CDCSample schema of AdventureWorks2008, each mirroring an existing AdventiureWorks2008 table:

 CDCSample.Customer

 CDCSample.CreditCard

 CDCSample.WorkOrder

A portion of the table data in the original AdventureWorks2008 tables is then used to initialize the source tables in the CDCSample schema. The remaining data will be used later to generate a dynamic workload when tracking is enabled.

With the source tables initialized, the script now creates three destination tables in the database AdventureWorksDW2008, one for each source table. For the purposes of the sample, the column structure of the destination tables is identical to that of the source tables. The function of the sample is to simply apply the changes associated with the source to the destination to allow the destination to reflect changes to the source in a timely fashion.

###### Creating Capture Instances for the Source Tables

Once the source tables are initialized, the setup script creates capture instances for each of the source tables. While the database itself must be enabled for Change Data Capture by a member of the sysadmin server role, capture instances for individual tables can be created by members of the db\_owner database role. The stored procedure sys.sp\_cdc\_enable\_table is used to create a capture instance for a source table. The following calls are made within the setup script to create capture instances for the three source tables.

exec sys.sp\_cdc\_enable\_table

 'CDCSample', 'Customer', 'Customer',

 @supports\_net\_changes = 1, @role\_name = null

exec sys.sp\_cdc\_enable\_table

 'CDCSample', 'CreditCard', ' CreditCard ',

 @supports\_net\_changes = 1, @role\_name = null

exec sys.sp\_cdc\_enable\_table

 'CDCSample', 'WorkOrder', 'WorkOrder',

 @supports\_net\_changes = 1, @role\_name = null

The first two parameters to the stored procedure are the schema and table name of the source table to be tracked. The third parameter is the name chosen for the associated capture instance. Any name can be chosen, but within a given database, the capture instance name must be unique. Since it is used to identify the change data associated with a given source table, it usually makes sense to name the capture instance in a manner that provides cues to its associated source table. If not specified, it will default to the schema name followed by the table name, separated by an underscore. In the sample, the tablename has been used as the capture instance name.

The parameter @supports\_net\_changes is used to indicate that functions to query for net changes should be generated for the capture instance. The source table must have a primary key or a defined unique index if this parameter is set to 1. Here, the term ‘net changes’ is used in a very specific way. The function returning net changes will only return a single row for each changed row for a given query window representing the final state of the row at the end of the interval. The operation returned with the row will be the one needed to correctly apply the row to the destination. In contrast, when a query for all changes is requested, a row is returned in the result set for each committed change to a row of the table.

The parameter @role\_name = null is used to indicate that no gating role is begin used to restrict access to change data. In order to have access to change data, the requestor must have select access to the captured columns of the source table. In addition, if the caller is not sysadmin or db\_owner and a gating role has been defined, the caller must also be a member of the gating role. By default, a gating role must be defined. If, however, the @role\_name parameter is explicitly set to null when the capture instance is created, no gating role is used and select access alone is sufficient to gain access to the change data.

##### Generating the Sample Workload

###### Script Tasks – Apply Inserts and Updates to CDC Enabled Tables

Once the environment has been enabled for Change Data Capture three Execute SQL Tasks are launched in parallel to generate DML activity against the source tables. After each task completes, a second task is launched to generate additional activity. In total, there are six tasks that generate load: one applying inserts and one applying updates, for each of three source tables. The update load only targets table rows populated as part of the initial load. A seeded random number generator is used so that the results are reproducible. The insert load is generated from the portion of the table rows not included within the initial load. By periodically enforcing a 10 second delay between batches of inserts and updates, the load is spread across several minutes.

Structurally, all of the Execute SQL Tasks that are used to generate workload are identical. They take no parameters and do not generate results sets. Each runs a SQL script to apply changes to CDC source tables. The table below shows the common SQL Statement attributes of the workload tasks.

|  |  |
| --- | --- |
| SQL Statement Attribute | Value |
| ConnectionType | OLE DB |
| Connection | AdventureWorks2008 |
| SQLSourceType | File connection |

Below the script files for generating workload are paired with their corresponding tasks.

|  |  |
| --- | --- |
| Execute SQL Task for Workload | SQLStatementSourceExpressions- ConnectionString |
| Insert Customer Table | @[ScriptPath] + ’CDCCustomerInsert.sql’ |
| Modify Customer Table | @[ScriptPath] + ’CDCCustomerModify.sql’ |
| Insert CreditCard Table | @[ScriptPath] + ’CDCCreditCardInsert.sql’ |
| Modify CreditCard Table | @[ScriptPath] + ’CDCCreditCardModify.sql’ |
| Insert WorkOrder Table | @[ScriptPath] + ’CDC WorkOrderInsert.sql’ |
| Modify WorkOrder Table | @[ScriptPath] + ’CDC WorkOrderModify.sql’ |

###### Script Task – Mark Workload Completion

The Script Task, Mark Workload Completion, runs after all the DML workload has been applied to the source tables. It is used to set package variables flagging the completion of the workload tasks and noting the completion time. The loop that periodically launches the master package to harvest change data monitors these package variables and terminates sample execution after the entire workload has been applied to the target environment.

The EntryPoint property which defines the first method that executes when the script task runs is MarkWorkloadCompletion. The ReadWriteVariables property allows the following package variables to be set by the script:

User::WorkloadCompleted,

User::WorkloadEndTime

The VB script code to support setting these package variables is shown below.

 Public Sub MarkWorkloadCompletion()

 Dim varWorkloadEndTime As Variable =

 Dts.Variables("User::WorkloadEndTime")

 Dim varWorkloadCompleted As Variable =

 Dts.Variables("User::WorkloadCompleted")

 Dts.VariableDispenser.LockForWrite("User::WorkloadEndTime")

 Dts.VariableDispenser.LockForWrite("User::WorkloadCompleted")

 varWorkloadEndTime.Value() = Now

 varWorkloadCompleted.Value() = 1

 Dts.Variables.Unlock()

 Dts.TaskResult = ScriptResults.Success

 End Sub

##### Verifying the Capture Process is Running

###### The Execute SQL Task – Verify Capture Process is Started

If there are no entries in cdc.lsn\_time\_mapping, this is an indication that the capture process did not auto-start when Change Data Capture was enabled. This typically occurs because SQL Agent is not running. This condition is detected here so that an event log message can be posted to indicate the failure to auto-start and the run can be terminated. The package variable User::CaptureStarted is used to report this condition.

The following SQL Statement is defined as Direct input:

declare @start\_time datetime, @capturestarted bit

select @start\_time = min(tran\_end\_time)

from cdc.lsn\_time\_mapping

if @start\_time is null

begin

 select @capturestarted = 0

end

else

begin

 select @capturestarted = 1

end

select @capturestarted as CaptureStarted

The SQL statement has no input parameters.

The result set is defined as a single row with the column CaptureStarted:

|  |  |
| --- | --- |
| Result Name | Variable Name |
| CaptureStarted | User::CaptureStarted |

###### Script Task – Log Capture Not Started Message

The Script Task, Log Capture Not Started Message, is used to log an information message to the Windows event log to indicate that the capture process did not auto-start.

The precedence constraint that defines the workflow between the Execute SQL Task Verify Capture Process is Started and this task is the following:

 Completion and

 @CaptureStarted == false evaluates to TRUE

The EntryPoint property which defines the first method that executes when the script task runs is LogCaptureNotStartedMessage. The ReadOnlyVariables property setting makes the following package variables available to the script:

System::PackageName,

System::StartTime,

User::CaptureStarted

The VB script code to support writing to the event log is shown below.

 Public Sub LogCaptureProcessNotStartedMessage()

 Dim varPackageName As Variable =

Dts.Variables("System::PackageName")

 Dim varStartTime As Variable = Dts.Variables("System::StartTime")

 Dim varCaptureStarted As Variable =

Dts.Variables("User::CaptureStarted")

 Dim sLog As String

 Dim sEventMessage As String

 Dim sMachine As String

 Dim sSource As String

 Dts.VariableDispenser.LockForRead("System::PackageName")

 Dts.VariableDispenser.LockForRead("System::StartTime")

 Dts.VariableDispenser.LockForRead("User::CaptureStarted")

 sLog = "Application"

 sSource = varPackageName.Value().ToString

 sEventMessage = "The CDC Capture Process was not started." \_

 & "Make certain that SQL Agent is running." \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "The Package: " + varPackageName.Value().ToString \_

 & Chr(10) \_

 & "Started: " & varStartTime.Value().ToString \_

 & Chr(10) \_

 & "Current Time:" & System.DateTime.Now \_

 & Chr(10) \_

 & "=============================================" \_

 & Chr(10) \_

 & "Capture Started: " & varCaptureStarted.Value().ToString

 sMachine = "."

 Dim ELog As New EventLog(sLog, sMachine, sSource)

 ELog.WriteEntry(sEventMessage, EventLogEntryType.Information)

 Dts.Variables.Unlock()

 Dts.TaskResult = ScriptResults.Success

 End Sub

The event log message posted will be similar to that shown in Figure 2:



Figure 2: Event Log Message Signaling Capture Process Not Started

##### Initializing the Target Tables

One of the principle issues to address when applying change data to a target environment is the synchronization of the stream of change data to the initial load. Simply put, the initial load of the target reflects a snapshot of the source at some point in time. The challenge is to determine the last change represented within the snapshot in order to have a basis for determining the first incremental change that needs to be applied.

For Change Data Capture, the recommended strategy for synchronization makes use of a database snapshot created after the source tables are enabled. Use of the snapshot as the source for the initial load insures the cross table consistency of the tracked tables. More importantly, however, the metadata maintained for the snapshot allows you to precisely determine the LSN to use when you apply the first incremental load.

###### Execute SQL Task – Create Database Snapshot

As described above, a database snapshot is created to be used as the source for the initial load of the target tables. The snapshot is created concurrently with the execution of the workload tasks to demonstrate that the content of the target tables subsequent to the initial load is not explicitly predetermined. Snapshot metadata will be used to determine the appropriate starting point for the incremental loads that follow.

The precedence constraint that defines the workflow between the Execute SQL Task Verify Capture Process is Started and this task is the following:

 Completion and

 @CaptureStarted == true evaluates to TRUE

The package variable User::CreateSnapshot contains the statement used to create the database snapshot. The content of the variable is derived from an expression that references the package variable User::BasePath to construct the path where the snapshot file is to be located.

"CREATE DATABASE @AdventureWorks2008\_dbss ON

(NAME = AdventureWorks2008\_Data, FILENAME = '" +

@[BasePath] + "AdventureWorks2008\_data.ss')

AS SNAPSHOT OF AdventureWorks2008;"

The SQL statement has no input parameters and no result set is returned.

###### Data Flow Tasks to Perform the Initial Load

Once the snapshot is created it can be queried concurrently to obtain a consistent initial load for the target tables. The sample makes use of data flow tasks to perform the initial load, but any technology that uses the snapshot as the source for the data can be used. Use of the snapshot as the source guarantees the cross table consistency of the target tables that are loaded.

The three Data Flow Tasks used to perform the initial load of the target tables are all structured identically. What is different in each of the three tasks are the names of the source and destination tables that provide the end points of the data flow. Each consists of an OLE DB source referencing the database snapshot, and an OLE DB destination that references AdventureWorksDW2008, the target database. All of the columns of the source table are output to the data flow. In general, column names associated with the source are mapped to identical names in the destination. Columns defined as computed columns, however, are ignored.

Below is the association between individual Data Flow Tasks and the source and destination tables.

|  |  |
| --- | --- |
| OLE DB Source Attribute | Value |
| OLE DB connection manager | AdventureWorks2008\_dbss |
| Data access mode | Table or view |

|  |  |
| --- | --- |
| OLE DB Destination Attribute | Value |
| OLE DB connection manager | AdventureWorksDW2008 |
| Data access mode | Table or view |

The table below shows the common attributes of the OLE DB source and destination Data Flow Components:

|  |  |
| --- | --- |
| Data Flow Task | Source/Destination Table |
| Load Target Customer Table From Snapshot | CDCSample.Customer |
| Load CreditCard Table From Snapshot | CDCSample.CreditCard |
| Load WorkOrder Table From Snapshot | CDCSample.WorkOrder |

##### Preparing for the Initial Extraction

Once the initial load of the target tables has completed, the Setup package can establish the boundaries for the first incremental load. Meta data from the database snapshot is used to determine the LSN that will anchor the initial incremental load while the current maximum LSN processed by the capture process serves as the initial upper bound.

###### Execute SQL Task – Get End-Points for Initial Query Interval

The Execute SQL Task – Get End-Points for Initial Query Interval is used to establish the query interval for the first interval that must synchronize with the initial load. The stored produre sys.sp\_cdc\_dbsnapshotLSN is used to determine the low end-point for the synchronization interval. The function sys.fn\_cdc\_get\_max\_lsn() is used to determine an appropriate high end-point. Note that the binary LSN values need to be converted to hexadecimal strings to be saved as package variables. The built in delay insures that the incremental load waits until the capture process has processed through the returned snapshot LSN.

The following SQL Statement is defined as Direct input:

declare @command nvarchar(max),

@database\_name nvarchar(1000),

@lastLSN binary(10),

@startLSN binary(10),

@endLSN binary(10),

@lastLSN\_str nvarchar(42),

@startLSN\_str nvarchar(42),

@endLSN\_str nvarchar(42),

@commit\_time datetime

exec sys.sp\_cdc\_dbsnapshotLSN 'AdventureWorks2008\_dbss',

@lastLSN output, @lastLSN\_str output

select @startLSN = sys.fn\_cdc\_increment\_lsn(@lastLSN)

select @endLSN = sys.fn\_cdc\_get\_max\_lsn()

while (@startLSN >= @endLSN)

begin

 waitfor delay '00:00:10'

 select @endLSN = sys.fn\_cdc\_get\_max\_lsn()

end

select @commit\_time = (select min(tran\_end\_time) from cdc.lsn\_time\_mapping where start\_lsn >= @startLSN)

select @startLSN\_str = upper(sys.fn\_varbintohexstr(@startLSN))

select @endLSN\_str = upper(sys.fn\_varbintohexstr(@endLSN))

select @startLSN\_str as ExtractStartLSN,

@commit\_time as ExtractStartTime,

@endLSN\_str as ExtractEndLSN

The SQL statement has no input parameters.

The result set is defined as a single row with the columns CustomerMismatch, CreditCardMismatch, and WorkOrder Mismatch.

|  |  |
| --- | --- |
| Result Name | Variable Name |
| ExtractEndLSN | User:: ExtractEndLSN |
| ExtractStartLSN | User:: ExtractStartLSN |
| ExtractStartTime | User:: ExtractStartTime |

##### Extracting and Processing Change Data

After the boundaries of the first incremental load have been determined, the setup package then enters a loop that repeatedly invokes the master package to gather and apply the latest group of changes. The loop terminates when all the changes from the generated workload have been applied to the target tables. After each execution of the master package, the Setup package delays for 10 seconds and then determines the LSN values to use to bound the next query for changes.

###### For Loop Container - Cycle Master at 10 Second Intervals

The Loop Container logic itself is designed to execute the container tasks until all of the changes in the generated workload have been extracted and applied to the target and then terminate. Two conditions need to be satisfied in order for the loop to terminate. First the workload generation tasks need to have all completed. This is determined by checking the package variable WorkloadCompleted, which is set by the Mark Workload Completion task that runs after all of the load generation tasks run. Second, all of the changes associated with the workload must have been applied to the target tables. This is determined by checking the proposed start time for the next extraction interval. If this value is greater than the time when the Workload Completion Task ran, then we are assured that the extraction intervals already processed fully cover the time when changes were applied to the source tables and the run can terminate.

The loop container also maintains a package variable that is used as an interval counter. It initializes the variable to 0 and then increments it by one each iteration. Since the interval counter is a package variable, it can be passed to the master package launched from within the loop.

For Loop Properties

|  |  |
| --- | --- |
| Property Name | Property Value |
| InitExpression | @IntervalID = 0 |
| EvalExpression | !((@WorkloadCompleted == 1) && (@ExtractStartTime > @WorkloadEndTime)) |
| AssignExpression | @IntervalID = @IntervalID + 1 |

Within the for loop container are two tasks - an Execute Package Task that invokes the master package driving the extraction cycle and an Execute SQL Task that first delays for 10 seconds and then computes the new end-points for the next extraction interval.

Execute Package Task – Run Master to Extract Data

The Execute Package Task Run Master to Extract Data functions as the test harness for the ETL extraction cycle. Its’ sole function is to launch the master package that will in turn launch the individual extraction packages for the individual tables.

Execute SQL Task – Delay 10 Seconds and Set New Query Interval

The Execute SQL Task Delay 10 Seconds Set New Query Interval is used to first delay for 10 seconds and then determine the bounds of the next query window. The previous upper bound is incremented by 1 to obtain the new lower bound. The function sys.fn\_cdc\_get\_max\_lsn is then used to obtain the high end-point for the query window. If 10 seconds is not sufficient to insure that the new lower bound of the query is less than or equal to the current maximum LSN, delays of 10 seconds are repeated until the condition is true. Note that since the LSN values are maintained in package variables as strings, they must be converted to binary values before they are used and reconverted to strings before they are saved.

declare @start\_lsn binary(10), @end\_lsn binary(10)

declare @start\_lsn\_str nvarchar(42), @end\_lsn\_str nvarchar(42),

@start\_time datetime

set @start\_lsn\_str = ?

set @start\_lsn = sys.fn\_cdc\_hexstrtobin(@start\_lsn\_str)

set @start\_lsn = sys.fn\_cdc\_increment\_lsn(@start\_lsn)

waitfor delay '00:00:10'

set @end\_lsn = sys.fn\_cdc\_get\_max\_lsn()

while (@start\_lsn > @end\_lsn)

begin

 waitfor delay '00:00:10'

 select @end\_lsn = sys.fn\_cdc\_get\_max\_lsn()

end

set @start\_lsn\_str = upper(sys.fn\_varbintohexstr(@start\_lsn))

set @end\_lsn\_str = upper(sys.fn\_varbintohexstr(@end\_lsn))

set @start\_time = (select min(tran\_end\_time) from cdc.lsn\_time\_mapping where start\_lsn >= @start\_lsn)

select @start\_lsn\_str as ExtractStartLSN,

@end\_lsn\_str as ExtractEndLSN,

@start\_time as ExtractStartTime

The SQL statement has one input parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Direction | Data Type | Parameter Name |
| User::ExtractEndLSN | Input | NVARCHAR | 0 |

The result set is defined as a single row with the columns ExtractEndLSN, ExtractStartLSN, and ExtractStartTime.

|  |  |
| --- | --- |
| Result Name | Variable Name |
| ExtractEndLSN | User:: ExtractEndLSN |
| ExtractStartLSN | User:: ExtractStartLSN |
| ExtractStartTime | User:: ExtractStartTime |

##### Validating the Incremental Load and Reporting Completion Status

Once the generated workload has been applied to the target environment, the state of the replicas should match that of the source tables. The Execute SQL Task Check for Mismatch in Replicas uses SQL CHECKSUM to verify the contents of the source and target tables match, setting package variables to record the status of the run. Finally the VB Script task Output Run Completion Status is launched to output the run status to the event log.

###### Execute SQL Task – Check for Mismatch in Replicas

The Execute SQL Task Check for Mismatch in Replicas is used to determine whether the replicas created match the source tables. CHECKSUM is used to compare the table contents and package variables are set for each of the tracked tables to indicate whether differences were detected.

The following SQL Statement is defined as Direct input:

declare @CustomerMismatch int,

 @CreditCardMismatch int,

 @WorkOrderMismatch int,

 @Checksum bigint,

 @ChecksumDW bigint

select @CustomerMismatch = 0,

 @CreditCardMismatch = 0,

 @WorkOrderMismatch = 0,

 @Checksum = 0,

 @ChecksumDW = 0

select @Checksum = CHECKSUM(\*)

from AdventureWorks2008.CDCSample.Customer

select @ChecksumDW = CHECKSUM(\*)

from AdventureWorksDW2008.CDCSample.Customer

if (@Checksum <> @ChecksumDW)

begin

 set @CustomerMismatch = 1

end

select @Checksum = CHECKSUM(\*)

from AdventureWorks2008.CDCSample.CreditCard

select @ChecksumDW = CHECKSUM(\*)

from AdventureWorksDW2008.CDCSample.CreditCard

if (@Checksum <> @ChecksumDW)

begin

 set @CreditCardMismatch = 1

end

select @Checksum = CHECKSUM(\*)

from AdventureWorks2008.CDCSample.WorkOrder

select @ChecksumDW = CHECKSUM(\*)

from AdventureWorksDW2008.CDCSample.WorkOrder

if (@Checksum <> @ChecksumDW)

begin

 set @WorkOrderMismatch = 1

end

select @CustomerMismatch as CustomerMismatch,

@CreditCardMismatch as CreditCardMismatch,

@WorkOrderMismatch as WorkOrderMismatch

The SQL statement has no input parameters.

The result set is defined as a single row with the columns CustomerMismatch, CreditCardMismatch, and WorkOrderMismatch.

|  |  |
| --- | --- |
| Result Name | Variable Name |
| CustomerMismatch | User::CustomerMismatch |
| CreditCardMismatch | User::CreditCardMismatch |
| WorkOrderMismatch | User::WorkOrderMismatch |

###### VB Script Task – Output Run Completion Status

The VB Script Task Output Run Completion Status writes a status entry to the Event Log indicating the result of the comparisons made between the source tables and the target tables that were updated using Change Data Capture.

The EntryPoint property which defines the first method that executes when the script task runs is OutputRunCompletionStatus. The ReadOnlyVariables property setting makes the following package variables available to the script:

 System::PackageName

 System::StartTime

 User::CreditCardMismatch

User::CustomerMismatch

User::WorkOrderMismatch

The VB script code to log completion status is shown below. Note that a comparison failure for any of the tables will cause the task to complete with a failure status.

 Public Sub OutputRunCompletionStatus()

Dim varPackageName As Variable =

Dts.Variables("System::PackageName")

 Dim varStartTime As Variable =

Dts.Variables("System::StartTime")

 Dim varCustomerMismatch As Variable =

Dts.Variables("User::CustomerMismatch")

 Dim varCreditCardMismatch As Variable =

Dts.Variables("User::CreditCardMismatch")

 Dim varWorkOrderMismatch As Variable =

Dts.Variables("User::WorkOrderMismatch")

 Dim sLog As String

 Dim sEventMessage As String

 Dim sMachine As String

 Dim sSource As String

 Dim sCustomer As String

 Dim sCreditCard As String

 Dim sWorkOrder As String

 sCustomer = "Customer replica is identical."

 sCreditCard = "CreditCard replica is identical."

 sWorkOrder = "WorkOrder replica is identical."

 Dts.VariableDispenser.LockForRead("System::PackageName")

 Dts.VariableDispenser.LockForRead("System::StartTime")

 Dts.VariableDispenser.LockForRead("User::CustomerMismatch")

 Dts.VariableDispenser.LockForRead("User::CreditCardMismatch")

 Dts.VariableDispenser.LockForRead("User::WorkOrderMismatch")

 Dts.TaskResult = ScriptResults.Success

 If varCustomerMismatch.Value = 1 Then

 sCustomer = "Customer replica does not match source."

 Dts.TaskResult = ScriptResults.Failure

 End If

 If varCreditCardMismatch.Value = 1 Then

 sCreditCard = "CreditCard replica does not match source."

 Dts.TaskResult = ScriptResults.Failure

 End If

 If varWorkOrderMismatch.Value = 1 Then

 sWorkOrder = "WorkOrder replica does not match source."

 Dts.TaskResult = ScriptResults.Failure

 End If

 sLog = "Application"

 sSource = varPackageName.Value().ToString

 sEventMessage = "CDC LSN Based Sample Completion Status" \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "The Package: " + varPackageName.Value().ToString \_

 & Chr(10) \_

 & "Started: " & varStartTime.Value().ToString \_

 & Chr(10) \_

 & "Current Time:" & System.DateTime.Now \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "Customer table: " & sCustomer \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "CreditCard table: " & sCreditCard \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "WorkOrder table: " & sWorkOrder \_

 & Chr(10) \_

 & "============================================="

 sMachine = "."

 Dim ELog As New EventLog(sLog, sMachine, sSource)

 ELog.WriteEntry(sEventMessage, EventLogEntryType.Information)

 Dts.Variables.Unlock()

 End Sub

##### SetupCDCSample Package Variables

The following package variables are defined for the setup package:

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description and Default Value |
| BasePath | String | Install path for Change Data Capture since Last Request Package Sample packagesDefault: @[SQLServerInstallPath] + "100\\Samples\\Integration Services\\Package Samples\\Change Data Capture since Last Request Package Sample\\Change Data Capture Sample\\" |
| ScriptPath | String | Install path for Change Data Capture since Last request Package Sample scriptsDefault: @[BasePath] + “Scripts\\” |
| SQLServerInstallPath | String | Install path for SQL ServerDefault: c:\program files\Microsoft SQL Server\ |
| CaptureStarted | Boolean | Flag indicating capture process has startedDefault: False |
| CreateSnapshot | String | SQL statement to create database snapshotDefault: "CREATE DATABASE AdventureWorks2008\_dbss ON( NAME = AdventureWorks2008\_Data, FILENAME = '" + @[BasePath] + "AdventureWorks2008\_data.ss' )AS SNAPSHOT OF AdventureWorks2008;" |
| ExtractStartLSN | String | Start LSN of next extractionDefault: '0x00000000000000000000' |
| ExtractEndLSN | String | End LSN of next extractionDefault: '0x00000000000000000000' |
| ExtractStartTime | datetime | Start time of next extractionDefault: 5/6/2007 8:54 AM |
| IntervalID | Int32 | Extraction Interval identifierDefault: 0 |
| LastLSN | String | LSN anchor for first extractionDefault: '0x00000000000000000000' |
| WorkloadCompleted | Int32 | 0 if Customer replica is identical at the end of the run: 1 if validation detected a mismatchDefault: 0 |
| WorkloadEndTime | datetime | Time of workload completionDefault: 2/21/2008 12:31 PM |
| CustomerMismatch | Int32 | 0 if Customer replica is identical at the end of the run: 1 if validation detected a mismatchDefault: 0 |
| CreditCardMismatch | Int32 | 0 if CreditCard replica is identical at the end of the run: 1 is validation detected a mismatchDefault: 0 |
| WorkOrderMismatch | Int32 | 0 if WorkOrder replica is identical at the end of the run: 1 is validation detected a mismatchDefault: 0 |

#### MasterCDC Package

The package MasterCDC is responsible for launching all of the individual packages used to extract change data for the SQL Server source tables and apply the changes to the Data Mart. The principle task of the master package is to verify that the extraction interval passed from the parent lies with the current Change Data Capture validity interval for the database.

##### The Change Data Capture Database Validity Interval

The Change Data Capture validity interval for a database is simply the time interval for which change data is currently available for its capture instances. In principle, it begins when the first capture instance is created for a database table, and extends forward in time to the present. In practice, the validity interval is a moving window just as the extraction interval is a moving window. Change data deposited in tables would grow unmanageably if it wasn’t periodically and systematically pruned. By default, only three days of data are retained in the change tables. Hence, the period of time covered by the validity interval is typically 72 hours.

While the cleanup process works to move the low end-point of the validity interval to the right on the Change Data Capture timeline, the capture process does the same for the high end-point. Since the capture process extracts change data from the transaction log, there is a built in latency between the time that a change is committed to a source table and the time that the change appears within its associated change table. While this latency is typically small, it is nevertheless important to remember that change data is not available until the capture process has processed the related log entries.

For each transaction that results in one or more entries appearing within database change tables, an entry is logged to the cdc.lsn\_time\_mapping table. Each entry in the mapping table contains both a commit Log Sequence Number or commit LSN, and a transaction commit time (columns start\_lsn and tran\_end\_time respectively.) Change Data Capture uses the table cdc.lsn\_time\_mapping to identify the current bounds of the database validity interval, with the smallest and largest commit times represented in the entries of cdc.lsn\_time\_mapping table denoting the low and high end-points of the validity interval for the database.

##### The Change Data Capture Validity Interval for a Capture Instance

While it is often the case that the database validity interval and an individual capture instance’s validity interval will coincide, this is not always true. The validity interval of the capture instance begins when the capture process recognizes the capture instance and begins logging associated changes to its change table. As a result, if capture instances are created at different times, each will initially have a different low end-point. The start\_lsn column of the result set returned by sys.sp\_cdc\_help\_change\_data\_capture shows the current low end-point for each defined capture instance. When the cleanup process cleans up change table entries, it adjusts the start\_lsn values for all capture instances to reflect the new low water mark for available change data. Only those capture instances with start\_lsn values currently less than the new low water mark are adjusted. Over time, if no new capture instances are created, the validity intervals for all individual instances will coincide with the database validity interval.

The validity interval is important to consumers of change data because the extraction interval for a request must be fully covered by the current Change Data Capture validity interval for the capture instance. If the low end-point of the extraction interval is to the left of the low end-point of the validity interval, there could be missing change data due to aggressive cleanup. If the high end-point of the extraction interval is to the right of the high end-point of the validity interval, the capture process has not yet processed through the time period represented by the extraction interval and change data could also be missing. This relationship is illustrated in the diagram below.



Figure 3: Change Data Capture Validity Intervals

It is important to note that the CDC query functions will fail if the request interval is not fully covered by the validity interval. In this sample, validity checks are systematically performed up front in the master package before launching the child packages used to gather change data.

The first task of the master package is to verify the extraction interval that it is passed. If either the low end–point or the high end-point of the extraction interval is outside the validity interval, the package logs an error to the event log and exits with error status. If the query interval is in range, each child package picks up the LSN values defining the extraction interval from its parent through Package Configurations. When all client packages have completed, the master package logs an event message indicating completion.

The control flow diagram for the master package is shown below.



Figure 4: Master Package for Incremental Data Extraction

##### Master Package Configurations and Variables

###### Package Configurations for Initializing Package Variables

The master package uses Package Configurations to initialize several runtime parameters. The table below shows the parent variables that are used to initialize local package variables of the master package.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration Name | Configuration Type | Configuration String | Target Object | Target Property |
| Configure ExtractStartLSN | Parent Package Variable | User::ExtractStartLSN | ExtractStartLSN | Value |
| Configure ExtractEndLSN | Parent Package Variable | User::ExtractEndLSN | ExtractEndLSN | Value |
| Configure IntervalID | Parent Package Variable | User::IntervalID | IntervalID | Value |
| Configure Base Path | Parent Package Variable | User::BasePath | BasePath | Value |

###### Master Package Variables

The following package variables are defined for the master package:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Data Type | Scope | Description |
| BasePath | String | MasterCDC | Path where Change Data Capture since Last Request Package Sample packages are installed |
| DataReady | Int32 | MasterCDC | Code indicating query status0 = Extract Interval is Invalid1 = Extract Interval is Valid |
| ExtractStartLSN | String | MasterCDC | Start LSN of extraction interval  |
| ExtractEndLSN | String | MasterCDC | End LSN of extraction interval |
| IntervalID | Int32 | MasterCDC | Interval ID  |

##### Master Package Tasks

###### Execute SQL Task – Check for Valid Interval

The Execute SQL Task, Check for Valid Interval, is used by the master package to determine how to proceed based upon the relationship of the requested extraction interval to the current Change Data Capture validity interval for the database. It assigns one of 2 possible values to the package variable DataReady to indicate whether or not the specified query interval is valid.

The following SQL Statement is defined as Direct input:

declare @start\_lsn binary(10), @end\_lsn binary(10)

declare @start\_lsn\_str nvarchar(42),

@end\_lsn\_str nvarchar(42)

declare @DataReady int

set @start\_lsn\_str = ?

set @end\_lsn\_str = ?

set @start\_lsn = sys.fn\_cdc\_hexstrtobin(@start\_lsn\_str)

set @end\_lsn = sys.fn\_cdc\_hexstrtobin(@end\_lsn\_str)

set @DataReady = 1

if not exists

( select start\_lsn

 from cdc.lsn\_time\_mapping

 where start\_lsn <= @start\_lsn )

select @DataReady = 0

if not exists

( select start\_lsn

 from cdc.lsn\_time\_mapping

 where start\_lsn >= @end\_lsn )

select @DataReady = 0

select @DataReady as DataReady

The SQL statement has two input parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Direction | Data Type | Parameter Name |
| User::ExtractStartLSN | Input | NVARCHAR | 0 |
| User::ExtractEndLSN | Input | NVARCHAR | 1 |

The result set is defined as a single row with the column DataReady:

|  |  |
| --- | --- |
| Result Name | Variable Name |
| DateReady | User::DataReady |

The SQL query first converts the passed string representations for the starting and ending LSN values to binary values using the function sys.fn\_cdc\_hexstrtobin. This is required since the binary values cannot be directly saved as package variables. The low end-point and high end-point of the query interval are then checked to insure that they lie within the database validity interval. If no entries are found in the cdc.lsn\_time\_mapping table with a start\_lsn value less than or equal to the low end-point or a start\_lsn value greater than or equal to the high end-point, @DataReady is returned as 0. Otherwise, it is returned as 1.

After execution of the Execute SQL Task Check for Valid Interval by the package, there are two possible paths that can be taken: If DataReady is 0 the passed query interval is invalid and the VB Script Task Log Extract Error is called. If dataReady is 1, the passed interval is valid and the three extraction packages are launched.

###### Execute Package Tasks – Extract Change Data for CDC Enabled Tables

The Execute Package Tasks Extract Cutomer Data, Extract CreditCard Data and Extract WorkOrder Data launch the individual extraction packages for the individual tables.

###### Script Task – Log Extract Error

The Script Task Log Extract Error is used to log an error to the Windows event log.

The precedence constraint that defines the workflow between the Execute SQL Task Check Data and this task is the following:

 Success Completion Code and

 @DataReady == 0 evaluates to TRUE

The EntryPoint property which defines the first method that executes when the script task runs is LogExtractError. The ReadOnlyVariables property setting makes the following package variables available to the script:

 System::ExecutionInstanceGUID,

User::ExtractStartLSN,

User::ExtractEndLSN,

System::PackageName,

System::StartTime

The VB script code to support writing to the event log is shown below.

 Public Sub LogExtractError()

 Dim varPackageName As Variable =

Dts.Variables("System::PackageName")

 Dim varStartTime As Variable = Dts.Variables("System::StartTime")

 Dim varInstanceID As Variable =

Dts.Variables("System::ExecutionInstanceGUID")

Dim varExtractStartLSN As Variable =

Dts.Variables("User::ExtractStartLSN")

Dim varExtractEndLSN As Variable =

Dts.Variables("User::ExtractEndLSN")

 Dim sLog As String

 Dim sEventMessage As String

 Dim sMachine As String

 Dim sSource As String

 Dts.VariableDispenser.LockForRead("System::PackageName")

 Dts.VariableDispenser.LockForRead("System::StartTime")

 Dts.VariableDispenser.LockForRead("System::ExecutionInstanceGUID")

 Dts.VariableDispenser.LockForRead("User::ExtractStartLSN")

 Dts.VariableDispenser.LockForRead("User::ExtractEndLSN")

 sLog = "Application"

 sSource = varPackageName.Value().ToString

 sEventMessage = "Invalid Query Interval"

 sEventMessage = sEventMessage \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "The Package: " + varPackageName.Value().ToString \_

 & Chr(10) \_

 & "Started: " & varStartTime.Value().ToString \_

 & Chr(10) \_

 & "Current Time:" & System.DateTime.Now \_

 & Chr(10) \_

 & "=============================================" \_

 & Chr(10) \_

 & "Extract Start LSN: " \_

& varExtractStartLSN.Value().ToString \_

 & Chr(10) \_

 & "Extract End LSN: " \_

& varExtractEndLSN.Value().ToString \_

 & Chr(10) \_

 & "Execution GUID: " & varInstanceID.Value().ToString

 sMachine = "."

 Dim ELog As New EventLog(sLog, sMachine, sSource)

 ELog.WriteEntry(sEventMessage, EventLogEntryType.Error)

 Dts.Variables.Unlock()

 Dts.TaskResult = ScriptResults.Failure

 End Sub

###### Script Task – Log Extraction Complete

The Script Task Log Extraction Complete is used to log a completion message to the Windows event log.

The precedence constraint that defines the workflow between the three Execute Package Tasks is Completion.

The EntryPoint property that defines the first method that executes when the script task runs is LogExtractionCompletion. The ReadOnlyVariables property setting makes the following package variables available to the script:

User::DataReady,

System::ExecutionInstanceGUID,

User::ExtractEndLSN,

User::ExtractStartLSN,

User::IntervalID,

System::PackageName,

System::StartTime

The VB script code to support writing the completion notification to the event log is shown below.

Public Sub LogExtractionCompletion()

 Dim varPackageName As Variable = Dts.Variables("PackageName")

 Dim varStartTime As Variable = Dts.Variables("StartTime")

 Dim varInstanceID As Variable =

 Dts.Variables("ExecutionInstanceGUID")

 Dim varExtractStartLSN As Variable =

 Dts.Variables("ExtractStartLSN")

 Dim varExtractEndLSN As Variable =

 Dts.Variables("ExtractEndLSN")

 Dim varIntervalID As Variable = Dts.Variables("IntervalID")

 Dim varDataReady As Variable = Dts.Variables("DataReady")

 Dim sLog As String

 Dim sEventMessage As String

 Dim sMachine As String

 Dim sSource As String

 Dts.VariableDispenser.LockForRead("PackageName")

 Dts.VariableDispenser.LockForRead("StartTime")

 Dts.VariableDispenser.LockForRead("ExecutionInstanceGUID")

 Dts.VariableDispenser.LockForRead("ExtractStartLSN")

 Dts.VariableDispenser.LockForRead("ExtractEndLSN")

 Dts.VariableDispenser.LockForRead("IntervalID")

 Dts.VariableDispenser.LockForRead("DataReady")

 sLog = "Application"

 sSource = varPackageName.Value().ToString

 sEventMessage = "Extract Complete" \_

 & Chr(10) \_

 & "=============================================" & Chr(10) \_

 & "The Package: " + varPackageName.Value().ToString \_

 & Chr(10) \_

 & "Started: " & varStartTime.Value().ToString \_

 & Chr(10) \_

 & "Current Time:" & System.DateTime.Now \_

 & Chr(10) \_

 & "=============================================" \_

 & Chr(10) \_

 & "Extract Start LSN: " & varExtractStartLSN.Value().ToString \_

 & Chr(10) \_

 & "Extract End LSN: " & varExtractEndLSN.Value().ToString \_

 & Chr(10) \_

 & "Interval ID: " & varIntervalID.Value().ToString \_

 & Chr(10) \_

 & "Data Ready: " & varDataReady.Value().ToString \_

 & Chr(10) \_

 & "Execution GUID: " & varInstanceID.Value().ToString

 sMachine = "."

 Dim ELog As New EventLog(sLog, sMachine, sSource)

 ELog.WriteEntry(sEventMessage, EventLogEntryType.Information)

 Dts.Variables.Unlock()

 Dts.TaskResult = ScriptResults.Success

 End Sub

#### Child Packages for the Change Data Capture since Last Request Package Sample

The sample child packages all use the CDC generated query functions to query for change data. The LSN boundaries of the extraction interval are obtained from parent package variables when the packages are launched.

The basic control flow for the packages is straightforward. A Script Task is called first to construct the SQL query that will be used to query for change data. Control is then passed to the Data Flow Task to request and process the change data. The data flow task uses an OLE DB source component to perform the query, directing the returned result set to a conditional split transformation. The conditional split uses the operation returned in each result set row to direct the rows to appropriate transformations: Deletes and updates are sent to OLE DB command transformations, while inserts are directed to an OLE DB destination.

The control and data flow for these child packages is shown in the figure below.



Figure 5: Child Packages Using CDC Enumeration TVFs for Data Access

The Configurations used by the child extract packages use package variables from the launching master package to initialize their own package variables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration Name | Configuration Type | Configuration String | Target Object | Target Property |
| Configure StartLSN | Parent Package Variable | User::ExtractStartLSN | StartLSN | Value |
| Configure EndLSN | Parent Package Variable | User::ExtractEndLSN | EndLSN | Value |

##### Child Package Variables

The following package variables are defined for the child packages:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Data Type | Scope | Description |
| StartLSN | String | CDCWorkOrderExtract | Low end-point for the query window |
| EndLSN | String | CDCWorkOrderExtract | High end-point for the query window |
| SQLDataQuery | String | CDCWorkOrderExtract | Query to use to obtain change data |

##### Child Package Tasks

###### Script Task - Generate SQL Data Query

The Execute SQL Task Generate SQL Query is used to set up the query for change data. It allows us to work around the inability to pass parameters directly to table valued functions in an OLE DB source. The parameters are passed to the VB script code which is used to compose the query string and return the desired select statement in the package variable SQLDataQuery.

One additional issue bears mentioning. The general strategy for systematically processing a stream of change data is to use the high end-point of the previous interval to determine the low end-point of the subsequent interval, and to compute a new high end-point based upon the needs of the application environment. This strategy works well for all intervals except the initial interval, when there is no previous interval.

In general, the destination and source will not be synchronized when a decision is made to use CDC technology to apply incremental loads and the first task will be to identify an anchor for the destination. The anchor is defined as an LSN lying within the Change Data Capture validity interval for a capture instance such that (1) All changes with start LSN values up to and including that anchor are already reflected in the destination and (2) All changes with start LSN values greater than the anchor have yet to be applied. Once the anchor is determined, it is used to explicitly set the LSN boundary for the initial extraction.

For this sample, a database snapshot provides data for the initial load, allowing the anchor LSN to be determined directly from snapshot metadata. While use of a database snapshot in preparing the initial load is not a requirement, it greatly simplifies the synchronization process.

The Script Task Generate SQL Data Query is used to generate the query to be used to extract change data. It uses the passed package variables StartLSN and EndLSN to define the query interval for the function call to obtain change data.

The EntryPoint property that defines the first method that executes when the script task runs is GenerateSQLQuery. The ReadOnlyVariables property setting makes the following package variables available to the script:

User::StartLSN,

User::EndLSN

The ReadWriteVariables property setting allows the following package variableto be set by the script:

User::SQLDataQuery

The VB script code to support constructing the data query is shown below.

 Public Sub GenerateDataQuery()

 Dim varStartLSN As Variable = Dts.Variables("User::StartLSN")

 Dim varEndLSN As Variable = Dts.Variables("User::EndLSN")

 Dim varSQLDataQuery As Variable =

Dts.Variables("User::SQLDataQuery")

 Dim sStartLSN As String

 Dim sEndLSN As String

 Dts.VariableDispenser.LockForRead("User::StartLSN")

 Dts.VariableDispenser.LockForRead("User::EndLSN")

 Dts.VariableDispenser.LockForWrite("User::SQLDataQuery")

 sStartLSN = varStartLSN.Value().ToString

 sEndLSN = varEndLSN.Value().ToString

 varSQLDataQuery.Value =

"select \* from cdc.fn\_cdc\_get\_net\_changes\_WorkOrder(" \_

 & sStartLSN & "," & sEndLSN & ", 'all')"

 Dts.Variables.Unlock()

 Dts.TaskResult = ScriptResults.Success

 End Sub

###### Data Flow Task – Process Change Data

The Data Flow Task Process Change Data calls a table valued function to query the source for change data and then applies the change data returned in the result set to the destination.

OLE DB Source

The query used by these child processes is obtained from a package variable and makes use of the generated CDC query function.

|  |  |
| --- | --- |
| Data Access Mode | SQL Command from variable |
| Variable Name | User::SQLDataQuery |

The result set returned by the query includes the metadata column \_\_$operation that identifies the operation to be used when applying the change to the destination. A conditional split transformation is used to direct the rows to one of three possible components based upon the following defined conditions:

|  |  |  |
| --- | --- | --- |
| Order | Output Name | Condition |
| 1 | Inserts | \_\_$operation == 2 |
| 2 | Updates | \_\_$operation == 4 |
| 3 | Deletes | \_\_$operation == 1 |

OLE DB Command Data Mart Deletes

The Delete flow is directed to the OLE DB Command Data Mart Deletes. The following command is used to apply the changes within this flow to the destination:

delete from CDCSample.WorkOrder where WorkOrderID = ?

The command requires a column from the result set in order to successfully apply the delete. This is the primary key columns for the table. The delete command above is for the child package processing changes to the WorkOrder table, which has a single primary key column WorkOrderID.

The mapping of result set columns to command parameters can be seen in the following diagram:



Figure 6: Result Set Mapping for Deletes

OLE DB Command Data Mart Updates

The Update flow is directed to the OLE DB Command Data Mart Updates. The following command is used to apply the changes within this flow to the destination:

update CDCSample.WorkOrder set

 ProductID = ?,

 OrderQty = ?,

 ScrappedQty = ?,

 StartDate = ?,

 EndDate = ?,

 DueDate = ?,

 ScrapReasonID = ?,

 ModifiedDate = ?

where WorkOrderID = ?

This command requires all of the source columns from the result set in order to successfully apply the update.

The mapping of result set columns to command parameters can be seen in the following diagram:



Figure 7: Result Set Mapping for Updates

OLE DB Destination Data Mart Inserts

The Insert flow is directed to the OLE DB Destination. The table name alone identifies the target for the insert. The following diagram shows how the result set columns are mapped to the table columns:



Figure 8: Result Set Mapping for Inserts

##### Error Logging in the Child Packages

Each of the child packages makes use of the SSIS Log Provider for Windows Event Log to allow logging for the OnError event. Logging is configured using the Configure SSIS Logs dialog box. This box appears in the designer when right clicking on a select package and choosing Logging. This dialog box also allows the current logging settings to be examined. For the child packages, this dialog box shows the SSIS Log Provider for Windows Event Log as the Provider type on the Providers and Logs tab. Under the Details tab, the OnError condition is set.

Even when range validation is done up front, it is always possible that conditions will change between the time the check is performed and the time the TVF executes. Applications must always be prepared to deal with the possibility of range errors. The CDC TVFs check to insure that the extraction interval defined is fully covered by the validity interval for the capture instance and errors if this requirement is not met.

In the sample, if range errors are encountered by any of the child packages, OLE DB will log the error to the event log producing an entry similar to the following.



Figure 9: Range Error Posted from OLEDB

The error returned for range errors is error 313, “An insufficient number of arguments were supplied for the procedure or function cdc.fn\_cdc\_get\_net\_changes\_ …”. The figure above shows this error.

#### Running the Change Data Capture since Last Request Package Sample in BI Studio

To run the Change Data Capture since Last Request Package Sample in BI Studio, open the Change Data Capture since Last Request Package Sample project and execute the package SetupCDCSample.dtsx. It should take several minutes to run. Once the execution environment has been set up, three of the workload tasks are launched to generate DML against the AdventureWorks2008 tables that are being tracked. These workload tasks have built in 10 second delays to guarantee that the load is spread over several minutes, regardless of the hardware that it is run on.



Figure 10: Setup Package Performing Initial Load and Workload Generation Concurrently

While the workload is being applied, a task to generate the database snapshot for AdventureWorks2008 is also launched. After it completes the dataflow tasks are allowed to run against the snapshot database to initially load the target tables in AdventureWorksDW2008. Once the initial load completes, the anchor LSN for the initial extraction interval is determined, and from it, the low end point of the interval that will synchronize to the initial load. With the LSN range for the synchronization interval determined, the master incremental load package can be invoked.

Each time the master package is invoked, it verifies that the desired extraction interval is contained within the validity interval for the database before invoking the child packages to query for the changes and apply them to the target. A delay of 10 seconds is imposed before a new query interval is determined and the master package is again invoked. When the workload finishes, the Mark Workload Completion task sets package variables to indicate that the workload has completed and to identify a completion time. When the start time of an extraction interval exceeds that time, the loop cycling the master package will terminate.



Figure 11: Setup package at Successful Run Completion

The control flow diagram above shows the setup package at the end of a successful run.

#### *Conclusion*

The Change Data Capture feature of SQL Server 2008 makes change data available in a relational format. This document describes how SSIS packages can leverage this feature to handle incremental loads to a Data Mart using a sample available on Codeplex as the vehicle to drive the discussion.

For more information:

SQL Server Web site: <http://www.microsoft.com/sql/default.mspx>

SQL Server TechCenter: <http://technet.microsoft.com/en-us/sqlserver/default.aspx>

SQL Server DevCenter: <http://msdn2.microsoft.com/en-us/sqlserver/default.aspx>

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

* Are you rating it high due to having good examples, excellent screenshots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screenshots, unclear writing?

This feedback will help us improve the quality of white papers we release. Send feedback.