****

Zentity (Version 1.0)

Change History Logging User Guide

May 2009

Contents

[Introduction 3](#_Toc228951227)

[SQL Server 2008 CDC Feature 3](#_Toc228951228)

[Overall Architecture 4](#_Toc228951229)

[Zentity DB 4](#_Toc228951230)

[Enabling and Disabling Zentity Change History Logging 5](#_Toc228951231)

[Change History Filegroup 5](#_Toc228951232)

[Capture Tables 5](#_Toc228951233)

[Coupling Tables 5](#_Toc228951234)

[Administration.Changeset 6](#_Toc228951235)

[Administration.Operation 6](#_Toc228951236)

[Administration.ResourceChange 7](#_Toc228951237)

[Administration.PredicateChange 10](#_Toc228951238)

[Administration.PredicatePropertyChange 10](#_Toc228951239)

[Administration.PropertyChange 10](#_Toc228951240)

[Administration.RelationshipChange 11](#_Toc228951241)

[Administration.RelationshipPropertyChange 11](#_Toc228951242)

[Administration.ResourcePropertyChange 12](#_Toc228951243)

[Administration.DataModelModuleChange 12](#_Toc228951244)

[Administration.ResourceTypeChange 13](#_Toc228951245)

[Administration.ScalarPropertyChange 13](#_Toc228951246)

[Administration.NavigationPropertyChange 14](#_Toc228951247)

[Administration.AssociationChange 14](#_Toc228951248)

[Stored Procedures 15](#_Toc228951249)

[Administration.EnableChangeHistory 15](#_Toc228951250)

[Administration.DisableChangeHistory 15](#_Toc228951251)

[Administration.ProcessNextLSN 16](#_Toc228951252)

[Administration.UpdateCaptureInstances 16](#_Toc228951253)

[Scheduled Jobs 16](#_Toc228951254)

[ProcessNextLSN 16](#_Toc228951255)

[Entity Framework based Mapping Layer 16](#_Toc228951256)

[Public API 17](#_Toc228951257)

[Retrieval Scenarios 18](#_Toc228951258)

[Get Changesets 19](#_Toc228951259)

[Get Resource Changes 20](#_Toc228951260)

[Get Relationship Changes 22](#_Toc228951261)

[Get Zentity Data Model Changes 23](#_Toc228951262)

[Additional Notes 25](#_Toc228951263)

[Error 15404 while enabling change history 25](#_Toc228951264)

[Backup-Restore Scenarios 25](#_Toc228951265)

[Data Loss Scenarios 25](#_Toc228951266)

# Introduction

This document describes the Change History Logging feature of Zentity. This feature can be used to capture the create/update/delete operations on resources, relationships, and other entities in the Zentity Core Entity Data Model (EDM) like Predicate, Property etc. The feature also captures any create/update/delete of Zentity Data Model elements like ResourceType, ScalarProperty, NavigationProperty etc.

# SQL Server 2008 CDC Feature

The Zentity Change History Logging is built on top of SQL Server 2008 Change Data Capture (CDC). This section briefly describes this feature. For more details, refer to SQL Server 2008 Books Online topic 'Change Data Capture' (<http://msdn.microsoft.com/en-us/library/bb522489.aspx>).

The SQL Server Change Data Capture tracks the insert, update and delete operations on database tables. While enabling CDC on a database table, a ‘Capture Table’ is created for the source table. The capture table contains all the columns of the source table and some additional metadata columns. Capture tables are populated by a background ‘Capture job’ created while enabling tables for CDC. The capture job asynchronously reads the transaction log and populates the capture tables. Likewise, a ‘Cleanup job’ cleans these capture tables periodically.

To enable the database for change data capture, run the stored procedure sys.sp\_cdc\_enable\_db\_change\_data\_capture in the database context. Run the stored procedure sys.sp\_cdc\_enable\_table\_change\_data\_capture on the source tables to enable change data capture on them. This creates a capture table parallel to each source table in the ‘cdc’ schema of the enabled database. Figure below shows an example of the source and CDC tables for Zentity properties.

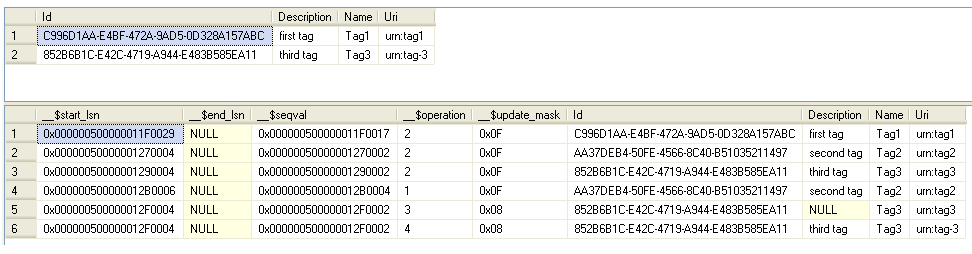


Figure 1. CDC Source and Capture Tables.

Table below (original reference <http://technet.microsoft.com/en-us/library/bb500305.aspx>) summarizes the capture table metadata columns.

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| \_\_$start\_lsn | binary(10) | Log sequence number (LSN) associated with the commit transaction for the change.  All changes committed in the same transaction share the same commit LSN. For example, if a delete operation on the source table removes two rows, the capture table will contain two rows, each with the same \_\_$start\_lsn value. |
| \_\_$end\_lsn | binary(10) | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.  In SQL Server 2008, this column is always NULL. |
| \_\_$seqval | binary(10) | Sequence value used to order the row changes within a transaction. |
| \_\_$operation | Int | Identifies the data manipulation language (DML) operation associated with the change. Can be one of the following:  1 = delete  2 = insert  3 = update (old values)  Column data has row values before executing the update statement.  4 = update (new values)  Column data has row values after executing the update statement. |
| \_\_$update\_mask | varbinary(128) | A bit mask based upon the column ordinals of the capture table identifying those columns that changed. |
| <captured source table columns> | Varies | The remaining columns in the capture table are the columns from the source table that were identified as captured columns when the capture instance was created. If no columns were specified in the captured column list, all columns in the source table are included in this table. |

# Overall Architecture

Zentity Change History Logging is built on top of SQL Server 2008 'Change Data Capture' and is thus available only on Developer, Enterprise and Enterprise Evaluation editions of SQL Server 2008. This feature also requires SQL Server Agent service to be running with the instance of SQL Server. For more details, refer to SQL Server 2008 Books Online topic 'Change Data Capture' (<http://msdn.microsoft.com/en-us/library/bb522489.aspx>).

After enabling change history logging, each major table in Zentity database is enabled for change data capture. SQL Server automatically creates two jobs during this process, 1. to populate capture tables and 2. to periodically clean up the capture tables. Zentity derives its change history data from these capture tables. A background job, ProcessNextLSN, pulls data from the capture tables and populates a separate set of ‘Coupling’ tables. These coupling tables allow us to retain the historical data even after the capture instances are cleaned up. 'Coupling' tables are then mapped to the conceptual model of Zentity Change History Logging. The public API is generated by Entity Framework from the conceptual model. Figure below presents an overall picture.

Figure 2. Zentity Change History Logging Architecture.

# Zentity DB

## Enabling and Disabling Zentity Change History Logging

The stored procedure ‘Administration.EnableChangeHistory’ creates all the required infrastructure to enable change history logging on the Zentity database. The procedure is described in detail in a later section.

Likewise, Administration.DisableChangeHistory can be used to disable the change history logging on Zentity database. The data present in coupling tables is not discarded while disabling the change history logging.

## Change History Filegroup

All CDC tables are created on a separate filegroup, ‘ChangeHistory’.

## Capture Tables

As part of enabling the change history logging, each table in ‘Core’ schema is enabled for change data capture, except for ‘AfterSchemaChangesHandler’, ‘Command’, ‘Configuration’, and ‘Content’. Following is the list of capture tables that get created.

1. Core\_Association\_CT
2. Core\_DataModelModule\_CT
3. Core\_NavigationProperty\_CT
4. Core\_Predicate\_CT
5. Core\_PredicateProperty\_CT
6. Core\_Property\_CT
7. Core\_Relationship\_CT
8. Core\_RelationshipProperty\_CT
9. Core\_Resource\_<Guid>\_CT
10. Core\_ResourceProperty\_CT
11. Core\_ResourceType\_CT
12. Core\_ScalarProperty\_CT

## Coupling Tables

These set of tables help in surfacing the change history data in a manner that the end user understands. To present an idea, consider a change in the ‘Audience’ property on a ‘Zentity.ScholarlyWorks.Lecture’ object. The CDC table captures this information in a row similar to the figure below.



Figure 3. Capture Table Entry.

This row has additional columns such as ‘[cbde5c20-64bf-4c43-b354-6fe6a4cf362e]’ (for AspectRatio) that do not apply to a Lecture. Also, the lecture specific columns, for example ‘[25b52405-1f7a-4c9a-b904-591d4e29c73c]’, do not match exactly in name with ‘Zentity.ScholarlyWorks.Lecture’ .NET properties which in this case is ‘Audience’. Thus some transformation is required to project only the relevant information from the CDC tables that is understood by Zentity user. For this, a ‘coupling table’ parallel to the ‘capture table’ is created that presents the data as shown below.



Figure 4. Coupling Table Entry.

The PropertyChanges column contains an XML that looks like the following. This XML contains only the relevant information applicable to a Lecture object and the column names are translated to .NET property names. Details of this table and the XML are present in a section later in this document.

|  |
| --- |
| <PropertyChanges ResourceId="DFF92785-6ADB-4B3A-90F0-A30D66E52B1A" ResourceTypeFullName="Zentity.ScholarlyWorks.Lecture">  <PropertyChange PropertyName="Title"/>  <PropertyChange PropertyName="DateModified">  <NextValue>Apr 6 2009 11:02AM</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Series"/>  <PropertyChange PropertyName="License"/>  <PropertyChange PropertyName="DateValidUntil"/>  <PropertyChange PropertyName="Notes"/>  <PropertyChange PropertyName="Image"/>  <PropertyChange PropertyName="DateAdded">  <NextValue>Apr 6 2009 11:02AM</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Audience">  <NextValue>Basic level.</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Uri"/>  <PropertyChange PropertyName="Abstract"/>  <PropertyChange PropertyName="Venue"/>  <PropertyChange PropertyName="Copyright"/>  <PropertyChange PropertyName="Description"/>  <PropertyChange PropertyName="DateValidFrom"/>  <PropertyChange PropertyName="Language"/>  <PropertyChange PropertyName="DateEnd"/>  <PropertyChange PropertyName="Scope"/>  <PropertyChange PropertyName="DateStart"/>  <PropertyChange PropertyName="DateAvailableUntil"/>  <PropertyChange PropertyName="DateAvailableFrom"/>  <PropertyChange PropertyName="DateCopyrighted"/>  </PropertyChanges> |

Also, coupling tables allow us to retain change history data even after the capture instances are dropped or cleaned up.

All coupling tables are created in a separate database schema, ‘Administration’. Each coupling table has a surrogate primary key, Id, that uniquely identifies a change in the system. For most of the coupling tables, there is a unique key that includes {ChangeSetId, SequenceNumber, OperationId}.

### Administration.Changeset

This table describes the changeset information and is derived from a special CDC table, cdc.lsn\_time\_mapping, which maps the LSNs to time when the transaction started or completed. LSNs are projected as changeset Id and the time the transaction ended is treated as the changeset creation time.

Take note that the Id is converted to the hexstring representation of its binary value. This is because, Entity Framework does not allow binary columns a primary keys.

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | nvarchar(64) | An identifier for the changeset. This is derived from the start\_lsn column of the cdc.lsn\_time\_mapping table. The value from cdc.lsn\_time\_mapping is converted to its hexstring representation. |
| DateCreated | Datetime | Creation time for the changeset. This is derived from the tran\_end\_time column of the cdc.lsn\_time\_mapping table. |

### Administration.Operation

This table is an enumeration of various operations possible on the entities.

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Int | The identifier of the operation. Possible values are:  1 = Delete.  2 = Insert.  3 = Update. |
| Name | Nvarchar(128) | Operation name. |
| Description | Nvarchar(4000) | Description of the operation. |

### Administration.ResourceChange

This table transforms cdc.Core\_Resource\_<Guid>\_CT capture table into the schema shown below. A scheduled job, ProcessNextLSN, populates this coupling table. The job gathers change history data from all capture instances created on Core.Resource table.

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | An identifier for the changeset. ChangeSetId is derived from the \_\_$start\_lsn column of change data capture table. \_\_$start\_lsn is a binary column. We convert it to its hex string representation while projecting it as ChangeSetId. The conversion was needed because ChangeSetId is used as EntityKey in some of the entities in the Mapping Layer. Entity Framework does not support binary values as entity keys. |
| SequenceNumber | nvarchar(64) | SequenceNumber, within a ChangeSet, represents the order of changes. SequenceNumber is derived from the \_\_$seqval column of change data capture tables. Again, we convert the binary value to its hex string representation since it is being used as EntityKey property in some of the entities in the Mapping Layer. |
| OperationId | Int | Operation defines the type of change.  1 = Delete - PropertyChanges has property values before deletion.  2 = Insert - PropertyChanges has property values after insertion.  3 = Update - PropertyChanges has property values before and after executing the update statement. |
| ResourceId | Uniqueidentifier | The identifier of resource undergoing the change. |
| ResourceTypeFullName | nvarchar(512) | The full .NET name of the resource type. |
| PropertyChanges | nvarchar(max) | Describes the property changes in an XML format with the following schema.   |  | | --- | | <?xml version="1.0" encoding="utf-8"?>  <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">  <xs:element name="PropertyChanges">  <xs:complexType>  <xs:sequence>  <xs:element maxOccurs="unbounded" name="PropertyChange">  <xs:complexType>  <xs:sequence minOccurs="0">  <xs:element minOccurs="0" name="PreviousValue" type="xs:string" />  <xs:element minOccurs="0" name="NextValue" type="xs:string" />  </xs:sequence>  <xs:attribute name="PropertyName" type="xs:string" use="required" />  <xs:attribute name="Changed" type="xs:string" use="optional" />  </xs:complexType>  </xs:element>  </xs:sequence>  <xs:attribute name="ResourceId" type="xs:string" use="required" />  <xs:attribute name="ResourceTypeFullName" type="xs:string" use="required" />  </xs:complexType>  </xs:element>  </xs:schema> |   Empty strings are treated separate from NULL values. An empty PreviousValue or NextValue element represents empty string. Absence of these elements represents NULL values.  For an insert operation, the ‘PreviousValue’ element is not present for the ‘PropertyChange’ element. Absence of the ‘NextValue’ in this case represents that the inserted value is NULL for the property. An example XML is shown below.   |  | | --- | | <PropertyChanges ResourceId="5A732A9F-8824-4BD1-AFF8-FB9A687B1C81" ResourceTypeFullName="Zentity.ScholarlyWorks.Lecture">  <PropertyChange PropertyName="Title">  <NextValue>Lecture1</NextValue>  </PropertyChange>  <PropertyChange PropertyName="DateModified">  <NextValue>Apr 2 2009 10:05PM</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Series" />  <PropertyChange PropertyName="License" />  <PropertyChange PropertyName="DateValidUntil" />  <PropertyChange PropertyName="Notes" />  <PropertyChange PropertyName="Image" />  <PropertyChange PropertyName="DateAdded">  <NextValue>Apr 2 2009 10:05PM</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Audience">  <NextValue>All .NET People</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Uri" />  <PropertyChange PropertyName="Abstract" />  <PropertyChange PropertyName="Venue" />  <PropertyChange PropertyName="Copyright" />  <PropertyChange PropertyName="Description" />  <PropertyChange PropertyName="DateValidFrom" />  <PropertyChange PropertyName="Language">  <NextValue>C#</NextValue>  </PropertyChange>  <PropertyChange PropertyName="DateEnd" />  <PropertyChange PropertyName="Scope" />  <PropertyChange PropertyName="DateStart" />  <PropertyChange PropertyName="DateAvailableUntil" />  <PropertyChange PropertyName="DateAvailableFrom" />  <PropertyChange PropertyName="DateCopyrighted" />  </PropertyChanges> |   For delete operation, the ‘NextValue’ element is not present for the Property. Absence of a ‘PreviousValue’ in this case represents that the property value was NULL before deletion. An example is shown below.   |  | | --- | | <PropertyChanges ResourceId="5A732A9F-8824-4BD1-AFF8-FB9A687B1C81" ResourceTypeFullName="Zentity.ScholarlyWorks.Lecture">  <PropertyChange PropertyName="Title">  <PreviousValue>Lecture1</PreviousValue>  </PropertyChange>  <PropertyChange PropertyName="DateModified">  <PreviousValue>Apr 2 2009 10:05PM</PreviousValue>  </PropertyChange>  <PropertyChange PropertyName="Series" />  <PropertyChange PropertyName="License" />  <PropertyChange PropertyName="DateValidUntil" />  <PropertyChange PropertyName="Notes" />  <PropertyChange PropertyName="Image" />  <PropertyChange PropertyName="DateAdded">  <PreviousValue>Apr 2 2009 10:05PM</PreviousValue>  </PropertyChange>  <PropertyChange PropertyName="Audience">  <PreviousValue>All .NET People Who Knows C# Only</PreviousValue>  </PropertyChange>  <PropertyChange PropertyName="Uri" />  <PropertyChange PropertyName="Abstract" />  <PropertyChange PropertyName="Venue" />  <PropertyChange PropertyName="Copyright" />  <PropertyChange PropertyName="Description" />  <PropertyChange PropertyName="DateValidFrom" />  <PropertyChange PropertyName="Language">  <PreviousValue>C#</PreviousValue>  </PropertyChange>  <PropertyChange PropertyName="DateEnd" />  <PropertyChange PropertyName="Scope" />  <PropertyChange PropertyName="DateStart" />  <PropertyChange PropertyName="DateAvailableUntil" />  <PropertyChange PropertyName="DateAvailableFrom" />  <PropertyChange PropertyName="DateCopyrighted" />  </PropertyChanges> |   For update operation, absence of ‘PreviousValue’ represents that the property value was NULL before update and absence of ‘NextValue’ represents that the property value was NULL after update. However, for BLOB properties (String and Binary with MaxLength = -1), the ‘PreviousValue’ element may not be generated if the property is not changed. The ‘Changed’ attribute on each ‘PropertyChange’ element is set to ‘True’ if the property has undergone any change, otherwise the value of ‘Changed’ attribute is ‘False’.   |  | | --- | | <PropertyChanges ResourceId="DFF92785-6ADB-4B3A-90F0-A30D66E52B1A" ResourceTypeFullName="Zentity.ScholarlyWorks.Lecture">  <PropertyChange PropertyName="Title" Changed="False"/>  <PropertyChange PropertyName="DateModified" Changed="True">  <PreviousValue>Apr 6 2009 11:02AM</PreviousValue>  <NextValue>Apr 6 2009 11:40AM</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Series" Changed="False"/>  <PropertyChange PropertyName="License" Changed="False"/>  <PropertyChange PropertyName="DateValidUntil" Changed="False"/>  <PropertyChange PropertyName="Notes" Changed="False"/>  <PropertyChange PropertyName="Image" Changed="False"/>  <PropertyChange PropertyName="DateAdded" Changed="False">  <PreviousValue>Apr 6 2009 11:02AM</PreviousValue>  <NextValue>Apr 6 2009 11:02AM</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Audience" Changed="True">  <PreviousValue>Basic level.</PreviousValue>  <NextValue>new audience</NextValue>  </PropertyChange>  <PropertyChange PropertyName="Uri" Changed="False"/>  <PropertyChange PropertyName="Abstract" Changed="False"/>  <PropertyChange PropertyName="Venue" Changed="False"/>  <PropertyChange PropertyName="Copyright" Changed="False"/>  <PropertyChange PropertyName="Description" Changed="False"/>  <PropertyChange PropertyName="DateValidFrom" Changed="False"/>  <PropertyChange PropertyName="Language" Changed="False"/>  <PropertyChange PropertyName="DateEnd" Changed="False"/>  <PropertyChange PropertyName="Scope" Changed="False"/>  <PropertyChange PropertyName="DateStart" Changed="False"/>  <PropertyChange PropertyName="DateAvailableUntil" Changed="False"/>  <PropertyChange PropertyName="DateAvailableFrom" Changed="False"/>  <PropertyChange PropertyName="DateCopyrighted" Changed="False"/>  </PropertyChanges> | |

### Administration.PredicateChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| PredicateId | Uniqueidentifier | Predicate identifier. |
| PreviousName | nvarchar(128) | For insert operation this is always NULL.  For delete operation this column specifies the last value.  For update operation this column specifies the value before update. |
| NextName | nvarchar(128) | For insert operation this column specifies the inserted value.  For delete operation this is always NULL.  For update operation this column specifies the value after update. |
| PreviousUri | nvarchar(1024) | For insert operation this is always NULL.  For delete operation this column specifies the last value.  For update operation this column specifies the value before update. |
| NextUri | nvarchar(1024) | For insert operation this column specifies the inserted value.  For delete operation this is always NULL.  For update operation this column specifies the value after update. |

### Administration.PredicatePropertyChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| PredicatePropertyId | Uniqueidentifier | PredicateProperty identifier. |
| PreviousPredicateId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPredicateId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousPropertyId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPropertyId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousValue | nvarchar(max) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextValue | nvarchar(max) | Follows the same semantics as Administration.PredicateChange NextUri. |

### Administration.PropertyChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| PropertyId | Uniqueidentifier | Property Id. |
| PreviousName | nvarchar(50) | Follows the same semantics as Administration.PredicateChange PreviousName. |
| NextName | nvarchar(50) | Follows the same semantics as Administration.PredicateChange NextName. |
| PreviousUri | nvarchar(1024) | Follows the same semantics as Administration.PredicateChange PreviousUri |
| NextUri | nvarchar(1024) | Follows the same semantics as Administration.PredicateChange NextUri |

### Administration.RelationshipChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| RelationshipId | Uniqueidentifier | Relationship Id. |
| PreviousSubjectResourceId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextSubjectResourceId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousObjectResourceId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextObjectResourceId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousPredicateId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPredicateId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousDateAdded | Datetime | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDateAdded | Datetime | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousOrdinalPosition | Int | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextOrdinalPosition | Int | Follows the same semantics as Administration.PredicateChange NextUri. |

### Administration.RelationshipPropertyChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| RelationshipPropertyId | Uniqueidentifier | Relationship property id. |
| PreviousTripletId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextTripletId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousPropertyId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPropertyId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousValue | Nvarchar(max) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextValue | Nvarchar(max) | Follows the same semantics as Administration.PredicateChange NextUri. |

### Administration.ResourcePropertyChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| ResourcePropertyId | Uniqueidentifier | Resource property id. |
| PreviousResourceId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextResourceId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousPropertyId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPropertyId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousValue | Nvarchar(max) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextValue | Nvarchar(max) | Follows the same semantics as Administration.PredicateChange NextUri. |

Next sections describe the coupling tables that capture Zentity Data Model changes.

### Administration.DataModelModuleChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | Nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | Nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| DataModelModuleId | Uniqueidentifier | Id of DataModelModule |
| PreviousNamespace | Nvarchar(150) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextNamespace | Nvarchar(150) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousDescription | Nvarchar(MAX) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDescription | Nvarchar(MAX) | Follows the same semantics as Administration.PredicateChange NextUri. |
| IsMsShipped | Bit | True if the module is a native Zentity type, false otherwise. |

### Administration.ResourceTypeChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| ResourceTypeId | Uniqueidentifier | Id of resource type. |
| PreviousDataModelModuleId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDataModelModuleId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChangeNextUri. |
| PreviousName | Nvarchar(100) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextName | Nvarchar(100) | Follows the same semantics as Administration.PredicateChangeNextUri. |
| PreviousUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousDescription | Nvarchar(4000) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDescription | Nvarchar(4000) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousBaseTypeId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextBaseTypeId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |

### Administration.ScalarPropertyChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| ScalarPropertyId | Uniqueidentifier | Id of scalar property. |
| PreviousResourceTypeId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextResourceTypeId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChangeNextUri. |
| PreviousUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousDescription | Nvarchar(max) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDescription | Nvarchar(max) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousName | Nvarchar(100) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextName | Nvarchar(100) | Follows the same semantics as Administration.PredicateChangeNextUri. |
| PreviousDataType | Nvarchar(100) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDataType | Nvarchar(100) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousNullable | Bit | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextNullable | Bit | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousMaxLength | Int | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextMaxLength | Int | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousScale | Int | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextScale | Int | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousPrecision | Int | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPrecision | Int | Follows the same semantics as Administration.PredicateChange NextUri. |

### Administration.NavigationPropertyChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | Int | Refer Administration.ResourceChange for description. |
| NavigationPropertyId | Uniqueidentifier | Id of Navigation property. |
| PreviousResourceTypeId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextResourceTypeId | Uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousName | Nvarchar(100) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextName | Nvarchar(100) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousUri | nvarchar(1024) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextUri | Nvarchar(1024) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousDescription | Nvarchar(MAX) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextDescription | Nvarchar(MAX) | Follows the same semantics as Administration.PredicateChange NextUri. |

### Administration.AssociationChange

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Id | Uniqueidentifier | Uniquely identifies a change in the system. |
| ChangeSetId | nvarchar(64) | Refer Administration.ResourceChange for description. |
| SequenceNumber | nvarchar(64) | Refer Administration.ResourceChange for description. |
| OperationId | int | Refer Administration.ResourceChange for description. |
| AssociationId | uniqueidentifier | Id of association. |
| PreviousName | nvarchar(100) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextName | nvarchar(100) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousUri | nvarchar(1024) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextUri | nvarchar(1024) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousSubjectNavigationPropertyId | uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextSubjectNavigationPropertyId | uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousObjectNavigationPropertyId | uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextObjectNavigationPropertyId | uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousPredicateId | uniqueidentifier | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextPredicateId | uniqueidentifier | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousSubjectMultiplicity | nvarchar(32) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextSubjectMultiplicity | nvarchar(32) | Follows the same semantics as Administration.PredicateChange NextUri. |
| PreviousObjectMultiplicity | nvarchar(32) | Follows the same semantics as Administration.PredicateChange PreviousUri. |
| NextObjectMultiplicity | nvarchar(32) | Follows the same semantics as Administration.PredicateChange NextUri. |

## Stored Procedures

This section covers the major store procedures of Zentity Change History Logging.

### Administration.EnableChangeHistory

This procedure

1. Creates change history filegroup and files.
2. Enables the database for change data capture and creates capture instances for each of the major tables in Core schema except for ‘AfterSchemaChangesHandler’, ‘Command’, ‘Configuration’, and ‘Content’.
3. Creates Coupling tables described in a previous section. Each Coupling table is created on the change history filegroup.
4. Creates stored procedures to process the capture tables and populate the coupling tables.
5. Creates a scheduled job, ProcessNextLSN, to periodically gather change history data in an asynchronous fashion.
6. Creates an entry in the Core.AfterSchemaChangesHandler table to invoke Administration.UpdateCaptureInstances.
7. Finally, updates the ‘IsChangeHistoryEnabled’ configuration value in Core.Configuration table.

### Administration.DisableChangeHistory

This procedure

1. Removes ProcessNextLSN job.
2. Disables change data capture on the database. This will delete all the capture instances in the database. Any data that is not copied from the capture tables to coupling tables is lost.
3. Finally, updates the ‘IsChangeHistoryEnabled’ configuration value in Core.Configuration table.

Take note that we have not dropped the Coupling tables here. So, the change history data that is present in the coupling tables is intact and can be queried.

### Administration.ProcessNextLSN

This is the master stored procedure invoked by the scheduled job. This procedure queries cdc.lsn\_time\_mapping for the oldest LSN that’s not processed. It then creates an entry in the Administration.ChangeSet table for this LSN and then invokes a separate set of stored procedures to process each capture table. Change history information corresponding to the LSN is then copied to corresponding coupling tables by these stored proecedures. All this is done in a single transaction.

### Administration.UpdateCaptureInstances

This procedure is invoked while enabling change history logging (by Administration.EnableChangeHistory) and after the Zentity Data Model is altered (by invoking Zentity.Core.DataModel.Synchronize method). While enabling change history logging, this method creates a new capture instance for Core.Resource table. During Zentity Data Model updates, this method creates a new capture instance on Core.Resource table if there is any schema change. Since there can be a maximum of two capture instances on a source table, this procedure drops any earlier capture instance if the count is greater than 2. NOTE that this might cause information loss if the capture instance still contains any un-processed change history data. Recall that a background job processes the capture tables and copies the change history information in coupling tables.

## Scheduled Jobs

### ProcessNextLSN

This scheduled job is invoked every 10 seconds to process the capture instances. It invokes Administration.ProcessNextLSN stored procedure with @Count = 100.

NOTE: There is no API or configuration provided to control or monitor this job. Administrators can use SQL Server Management Studio to change the scheduling of this job or the number of LSNs to process in each go.

# Entity Framework based Mapping Layer

Entity Framework tools are used to generate the API and mapping layer to query the change history information. Diagram below shows the entities in the Entity Data Model.

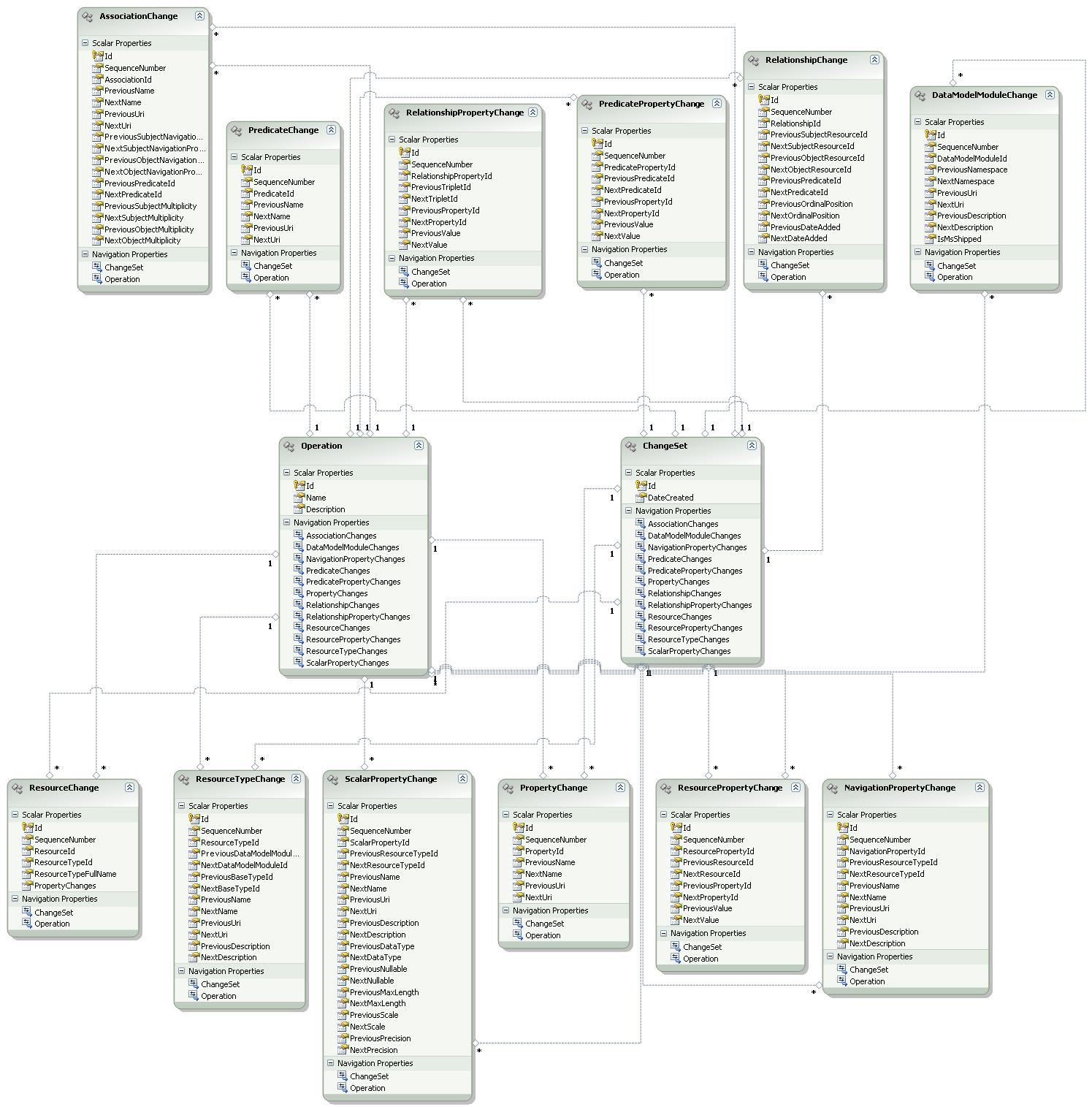


Figure 5. Change History Entity Data Model.

# Public API

The public API is auto-generated using Entity Framework from the conceptual data model. All the properties of each class are made read only. Unfortunately, Entity Framework, at time of this writing, does not allow developer control of the access modifiers on auto-generated methods. So, the auto-generated API produced methods like CreateChangeSet(), AddToChangeSets() etc. The auto-generated API has been modified to remove these methods. The diagram below shows the change history classes. Refer to the API reference for more details.

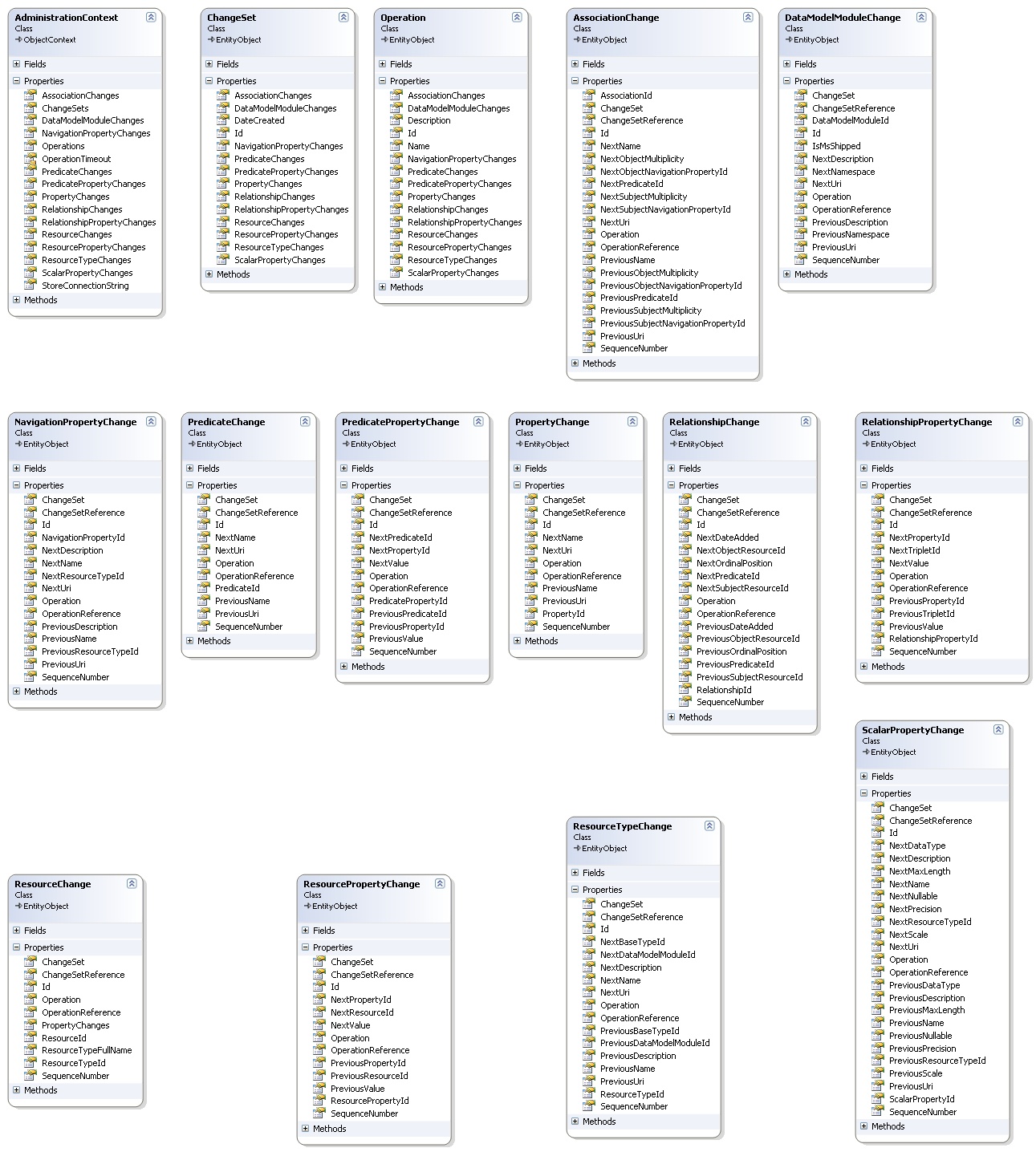


Figure 6. Change History Logging Class Diagram.

# Retrieval Scenarios

This section provides some samples to query change history data. All these samples require Change History Logging to be enabled. Use either Zentity.Administration.AdministrationContext.EnableChangeHistory(string) method or Administration.EnableChangeHistory stored procedure to enable Change History Logging. Add references to Zentity.Core and System.Data.Entity to the sample console applications. Include the application configuration as mentioned in the next section. Finally, ensure that SQL Server Agent is running to collect the change history data for each of these samples.

## Get Changesets

Create a console application and add an application configuration file containing the connection string to be used by the Administration context. The content of application configuration file look something as shown below.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <configuration>  <connectionStrings>  <add name="ZentityContext"  connectionString="metadata=res://Zentity.Core;provider=System.Data.SqlClient;  provider connection string='Data Source=.;Initial Catalog=Zentity;Integrated Security=True;  MultipleActiveResultSets=True'" providerName="System.Data.EntityClient" />  <add name="AdministrationContext"  connectionString="metadata=res://Zentity.Core;provider=System.Data.SqlClient;  provider connection string='Data Source=.;Initial Catalog=Zentity;Integrated Security=True;  MultipleActiveResultSets=True'" providerName="System.Data.EntityClient" />  </connectionStrings>  </configuration> |

Update Program.cs to look as shown below.

|  |
| --- |
| using System;  using System.Linq;  using Zentity.Administration;  using Zentity.Core;  using System.Threading;  namespace ZentitySamples  {  public class Program  {  public static void Main(string[] args)  {  DateTime start = DateTime.Now;  // Use ZentityContext to update repository content.  using (ZentityContext context = new ZentityContext())  {  // Create.  Resource pub = new Resource  {  Title = "sample Resource",  Uri = "urn:famulus-samples:pub1"  };  context.AddToResources(pub);  context.SaveChanges();  // Update.  pub.Title = "new title";  context.SaveChanges();  // Delete.  context.DeleteObject(pub);  context.SaveChanges();  }  // Give some time to the background job to process these changes.  Thread.Sleep(new TimeSpan(0, 0, 20));  // Get all changesets created after a given datetime.  using (AdministrationContext context = new AdministrationContext())  {  foreach (ChangeSet cs in context.ChangeSets.  Where(tuple => tuple.DateCreated >= start))  {  Console.WriteLine("Changeset: [{0}] created on [{1}]", cs.Id, cs.DateCreated);  cs.ResourceChanges.Load();  foreach (ResourceChange rc in cs.ResourceChanges)  {  rc.OperationReference.Load();  Console.WriteLine("ResourceChange ResourceId:[{0}], Operation:[{1}]",  rc.ResourceId, rc.Operation.Name);  }  }  }  }  }  } |

## Get Resource Changes

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Zentity.Core;  using System.Threading;  using Zentity.Administration;  using System.Collections.ObjectModel;  namespace ChangeHistorySamples  {  class ResourceChangeSample  {  static void Main(string[] args)  {  //Create Resource  Guid resourceId = CreateResource();  PrintResource(resourceId);  RetriveLatestResourceChangeset(resourceId);  //Modify Resource  ModifyResource(resourceId);  PrintResource(resourceId);  RetriveLatestResourceChangeset(resourceId);  //Delete Resource  DeleteResource(resourceId);  PrintResource(resourceId);  RetriveLatestResourceChangeset(resourceId);  Console.Read();  }  static Guid CreateResource()  {  Guid resourceId;  using (ZentityContext context = new ZentityContext())  {  Resource resource = new Resource();  resourceId = resource.Id;  resource.Title = "Create Title for SampleResource ";  resource.Uri = "urn:change-history-samples/resource/sample-resource";  context.AddToResources(resource);  context.SaveChanges();  }  //Changeset entries are processed by a background job that is invoked  // every 10 seconds.Wait for a while to let it complete.  Thread.Sleep(new TimeSpan(0, 0, 20));  return resourceId;  }  static void ModifyResource(Guid resourceId)  {  using (ZentityContext context = new ZentityContext())  {  Resource resource = context.Resources.  Where(tuple => tuple.Id == resourceId).First();  resource.Title = "Modify Title for SampleResource ";  context.SaveChanges();  }  Thread.Sleep(new TimeSpan(0, 0, 20));  }  static void DeleteResource(Guid resourceId)  {  using (ZentityContext context = new ZentityContext())  {  Resource resource = context.Resources.  Where(tuple => tuple.Id == resourceId).First();  context.DeleteObject(resource);  context.SaveChanges();  }  Thread.Sleep(new TimeSpan(0, 0, 20));  }  static void PrintResource(Guid resourceId)  {  using (ZentityContext context = new ZentityContext())  {  Resource resource = context.Resources.  Where(tuple => tuple.Id == resourceId).FirstOrDefault();  PrintLine();  if (resource != null)  {  Console.WriteLine("Id [{0}]", resource.Id);  Console.WriteLine("Name [{0}]", resource.Title);  Console.WriteLine("Uri [{0}]", resource.Uri);  }  else  Console.WriteLine("Resource not found.");  PrintLine();  }  }  static void RetriveLatestResourceChangeset(Guid resourceId)  {  using (AdministrationContext adminContext = new AdministrationContext())  {  ResourceChange resourceChange = adminContext.ResourceChanges.  Where(tuple => tuple.ResourceId == resourceId).  OrderByDescending(tuple => tuple.ChangeSet.DateCreated).  FirstOrDefault();  if (resourceChange != null)  {  Console.WriteLine("");  resourceChange.ChangeSetReference.Load();  resourceChange.OperationReference.Load();  Console.WriteLine("Operation [{0}]", resourceChange.Operation.Name);  Console.WriteLine("ResourceId [{0}]", resourceChange.ResourceId);  Console.WriteLine("ResourceTypeFullName [{0}]", resourceChange.ResourceTypeFullName);  Console.WriteLine("PropertyChanges \n {0}", resourceChange.PropertyChanges);  Console.WriteLine("");  }  }  }  static void PrintLine()  {  for (int i = 0; i < 75; i++)  Console.Write("-");  Console.WriteLine("");  }  }  } |

## Get Relationship Changes

|  |
| --- |
| using System;  using System.Linq;  using Zentity.Administration;  using Zentity.Core;  using System.Threading;  namespace ZentitySamples  {  public class Program  {  public static void Main(string[] args)  {  Guid resourceId = Guid.Empty;  using (ZentityContext context = new ZentityContext())  {  Resource pub = new Resource  {  Title = "sample Resource",  Uri = "urn:famulus-samples:pub1"  };  Resource author1 = new Resource { Title = "Alice" };  Resource author2 = new Resource { Title = "Bob" };  Resource author3 = new Resource { Title = "Charlie" };  // Create some relationships.  Predicate authoredBy = new Predicate { Name = "Author", Uri = "urn:" + Guid.NewGuid().ToString("N") };  Relationship rel1 = new Relationship { Subject = pub, Object = author1, Predicate = authoredBy };  Relationship rel2 = new Relationship { Subject = pub, Object = author2, Predicate = authoredBy };  Relationship rel3 = new Relationship { Subject = pub, Object = author3, Predicate = authoredBy };  context.AddToResources(pub);  context.SaveChanges();  // Remove some relationships.  context.DeleteObject(rel1);  context.SaveChanges();  resourceId = pub.Id;  }  // Give some time to the background job to process these changes.  Thread.Sleep(new TimeSpan(0, 0, 20));  // Retrieve all RelationshipChanges for the above Resource.  using (AdministrationContext context = new AdministrationContext())  {  foreach (RelationshipChange rc in context.RelationshipChanges.  Include("Operation").Include("ChangeSet").  Where(tuple => tuple.PreviousSubjectResourceId == resourceId ||  tuple.NextSubjectResourceId == resourceId ||  tuple.PreviousObjectResourceId == resourceId ||  tuple.NextObjectResourceId == resourceId))  {  Console.WriteLine("Relationship Id:[{0}], Operation: [{1}], DateCreated: [{2}]",  rc.RelationshipId, rc.Operation.Name, rc.ChangeSet.DateCreated);  Console.WriteLine("PreviousSubjectId: [{0}], PreviousObjectId: [{1}], PreviousPredicateId: [{2}]",  rc.PreviousSubjectResourceId, rc.PreviousObjectResourceId, rc.PreviousPredicateId);  Console.WriteLine("NextSubjectId: [{0}], NextObjectId: [{1}], NextPredicateId: [{2}]",  rc.NextSubjectResourceId, rc.NextObjectResourceId, rc.NextPredicateId);  }  }  }  }  } |

## Get Zentity Data Model Changes

|  |
| --- |
| using System;  using System.Linq;  using Zentity.Administration;  using Zentity.Core;  using System.Threading;  namespace ZentitySamples  {  public class Program  {  public static void Main(string[] args)  {  Guid moduleId;  using (ZentityContext context = new ZentityContext())  {  string namespaceName = "Namespace" + Guid.NewGuid().ToString("N");  // Create a new module.  DataModelModule module = new DataModelModule { NameSpace = namespaceName };  context.DataModel.Modules.Add(module);  // Create the ScholarlyWork type.  ResourceType resourceTypeResource = context.DataModel.Modules["Zentity.Core"].ResourceTypes["Resource"];  ResourceType resourceTypeScholarlyWork = new ResourceType { Name = "ScholarlyWork", BaseType = resourceTypeResource };  module.ResourceTypes.Add(resourceTypeScholarlyWork);  // Create some Scalar Properties.  ScalarProperty copyRight = new ScalarProperty { Name = "CopyRight", DataType = DataTypes.String, MaxLength = 4000 };  resourceTypeScholarlyWork.ScalarProperties.Add(copyRight);  // Create some Navigation Properties.  NavigationProperty authors = new NavigationProperty { Name = "Authors" };  resourceTypeScholarlyWork.NavigationProperties.Add(authors);  // Create the Contact type.  ResourceType resourceTypeContact = new ResourceType { Name = "Contact", BaseType = resourceTypeResource };  module.ResourceTypes.Add(resourceTypeContact);  ScalarProperty email = new ScalarProperty { Name = "Email", DataType = DataTypes.String, MaxLength = 1024 };  resourceTypeContact.ScalarProperties.Add(email);  NavigationProperty authoredWorks = new NavigationProperty { Name = "AuthoredWorks" };  resourceTypeContact.NavigationProperties.Add(authoredWorks);  // Add SamplesScholarlyWorkAuthoredByContact association.  // Association names should be unique across all the modules in the data model.  Association association = new Association  {  Name = namespaceName + "\_ScholarlyWorkAuthoredByContact",  SubjectNavigationProperty = authors,  ObjectNavigationProperty = authoredWorks,  SubjectMultiplicity = AssociationEndMultiplicity.Many,  ObjectMultiplicity = AssociationEndMultiplicity.Many  };  // Synchronize to alter the database schema.  // This method takes a few minutes to complete depending on the actions taken by  // other modules (such as change history logging) in response to schema changes.  // Provide a sufficient command timeout.  context.CommandTimeout = 300;  context.DataModel.Synchronize();  moduleId = module.Id;  // Delete the module.  context.DataModel.Modules.Remove(module);  context.DataModel.Synchronize();  }  // Give the background job some time to process changes.  Thread.Sleep(new TimeSpan(0, 0, 20));  // Retrieve changes.  using (AdministrationContext context = new AdministrationContext())  {  foreach (DataModelModuleChange dmm in context.DataModelModuleChanges.  Include("Changeset").Include("Operation").  Where(tuple => tuple.DataModelModuleId == moduleId))  {  Console.WriteLine("Changeset: [{0}] created on [{1}].",  dmm.ChangeSet.Id, dmm.ChangeSet.DateCreated);  Console.WriteLine("DataModelModuleChange ModuleId:[{0}], Operation[{1}]",  dmm.DataModelModuleId, dmm.Operation.Name);  foreach (ResourceTypeChange rtc in context.ResourceTypeChanges.  Include("Changeset").Include("Operation").  Where(tuple => tuple.ChangeSet.Id == dmm.ChangeSet.Id &&  (tuple.PreviousDataModelModuleId == dmm.DataModelModuleId ||  tuple.NextDataModelModuleId == dmm.DataModelModuleId)))  {  Console.WriteLine("ResourceTypeChange ResourceTypeId:[{0}], Operation:[{1}]",  rtc.ResourceTypeId, rtc.Operation.Name);  }  foreach (AssociationChange ac in context.AssociationChanges.  Include("Changeset").Include("Operation").  Where(tuple => tuple.ChangeSet.Id == dmm.ChangeSet.Id))  {  Console.WriteLine("AssociationChange AssociationId:[{0}], Operation:[{1}]",  ac.AssociationId, ac.Operation.Name);  }  }  }  }  }  } |

# Additional Notes

## Error 15404 while enabling change history

Sometimes, due to changes in the working environment, the database owner information is no longer valid and you might see error similar to the following while enabling change history.

|  |
| --- |
| Could not update the metadata that indicates database Zentity is enabled for Change Data Capture. The failure occurred when executing the command 'SetCDCTracked(Value = 1)'. The error returned was 15404: 'Could not obtain information about Windows NT group/user 'Domain\username', error code 0x54b.'. Use the action and error to determine the cause of the failure and resubmit the request. |

To fix this, execute a command similar to the following before invoking Administration.EnableChangeHistory.

|  |
| --- |
| exec sp\_changedbowner 'sa','dbo' |

## Backup-Restore Scenarios

During backup and restore of a change-history-enabled Zentity database, all configuration values and capture instances are restored. However, the jobs to populate and cleanup capture instances and 'ProcessNextLSN' are not restored on the target server. So, even though sys.databases and sys.tables show that the database and table are configured for change data capture, no entries are created in the capture tables and hence the coupling tables. To fix this, disable change history logging and then re-enable it again using stored procedures Administration.DisableChangeHistory and Administration.EnableChangeHistory.

## Data Loss Scenarios

When a table is enabled for change data capture, SQL Server tries to keep the structure of capture table intact. If a tracked column is dropped, null values will be supplied for the column in the subsequent change entries. If an existing column undergoes a change in its data type, the change is propagated to the capture table. Finally, the capture process ignores any new columns that are not identified for capture when the source table was enabled for change data capture. This can lead to data losses for the new scalar properties introduced in the custom resource types.

To lessen such losses, we create a new capture instance each time the schema of Core.Resource table is changed. The new capture instance mirrors the new schema of Core.Resource. While processing the change sets, we enumerate all capture instances that host data for the change set. In case, multiple capture instances host the data for the same change set, we pick the latest one since it is the one that mirrors the latest schema. Then entries from this capture instance are copied over to Administration.ResourceChange table. Since, SQL Server allows only two capture instances per source table, we have to drop the oldest capture instance if there are already two capture instances present.

However, there are still some scenarios where change history data might be lost. For example, consider the following sequence of events.

|  |  |  |
| --- | --- | --- |
| S.no. | Event | Effect |
| 1 | Create CustomTypeA  (ZentityContext.DataModel.Synchronize()) | A new capture instance is created cdc.Core\_Resource\_1. The capture instance name is only for illustration, actual name is different than this. |
| 2 | Create instances of CustomTypeA | Entries are created in cdc.Core\_CustomTypeA\_1. |
| 3 | Update the definition of CustomTypeA, add PropertyA | Another capture instance will be created, cdc.Core\_CustomTypeA\_2. |
| 4 | Create some more instances | Both capture instances will be populated. |
| 5 | Update the definition of CustomTypeA, remove PropertyA | New capture instance, cdc.Core\_CustomTypeA\_3, will be created and cdc.Core\_CustomTypeA\_1 will be dropped since there can only be two capture instances on a source table. |

If the scheduled job, ProcessNextLSN, finishes before event 2 and the events 2-5 happen in quick succession before the next run of the job. Since cdc.Core\_CustomType\_1 is now dropped, the information for event 2 will be lost.

Another scenario is while disabling change history. Again consider the following sequence of events:

|  |  |  |
| --- | --- | --- |
| S.no. | Event | Effect |
| 1 | Create instances of Publications | Entries are created in cdc.Core\_Resource\_1. |
| 2 | Disable change history  (AdministrationContext.DisableChangeHistory) | Change Data Capture is turned off on database and all the capture instances are dropped. |

If the scheduled job, ProcessNextLSN, finishes before event 1 and the events 1-2 happen in quick succession before the next run of the job, then all the capture instances are dropped, and the information for event 1 will be lost.

To avoid such scenarios, you can query the cdc.lsn\_time\_mapping and Administration.ChangeSet tables to get a list of change sets that are yet to be processed before performing ‘schema changes’ or ‘disabling change history’. The query looks like the following.

|  |
| --- |
| SELECT [start\_lsn], [tran\_end\_time]  FROM [cdc].lsn\_time\_mapping  WHERE tran\_id <> 0x00  EXCEPT  SELECT Administration.fn\_hexstrtovarbin([Id]) [start\_lsn], [DateCreated] [tran\_end\_time]  FROM [Administration].ChangeSet |