**Using DDL Triggers in SQL Server 2005 to Capture Schema Changes**

By [**Chris Rock**](http://www.sqlteam.com/author/chris-rock) on **13 August 2007** | [12 Comments](http://www.sqlteam.com/forums/topic.asp?TOPIC_ID=87732) | Tags: [Triggers](http://www.sqlteam.com/tag/sql-server-triggers)

This article shows how to use DDL triggers to capture schema changes. This solutions works a little differently than most DDL triggers examples. This script captures the old object that was replaced.

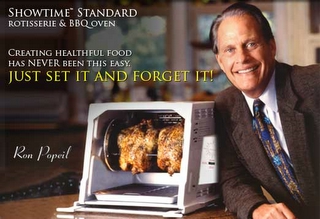
A while back I was assigned to the role of backup DBA. I referred to myself as the pseudo-DBA. I liked it because it was a challenge to me. Something new is always good for a geek. I learned a lot during my tenure which ended when we hired a new DBA. Looking back the one task I feared most during that time was updates from the other developers. Not because they gave me crappy code (well, some of them did) but because of the tedious nature of the system we had in place.

We had a system setup to send all change requests through me. This was in place as to not disturb the real DBA who responsibilities were to keep the enterprise running. More than a few times a day I'd get an email with change request. No big deal right? Well it wasn't a big deal to actually deploy the changes. The problem was keeping a backup of the old object in a change log. This required me to script out the old object and then save that data to a table. I had a little ASP app setup to paste the text of the script, the name of the database and the name of the object into the form. When I clicked the "Save" button the data was inserted into the change log table. Even with the little app setup I'd sometimes forget, be in a rush, or for some other stupid reason I wouldn't back up the old object. That led to trouble.

As I said before, this was a tedious and sometimes annoying part of the job. I tried a few source safe systems but was always disappointed with their performance or they required you to use their environment for making changes. There wasn't any integration into query analyzer. I eventually gave up and just accepted that this was the way it was going to be.

This is where you are lucky. I think I found a way for you, the DBA, psuedo-DBA, forced DBA, etc to get the same type of change log system with just a little setup.

There isn't a rotisserie involved but with this system, you can "set it and forget it!".



FYI, Those rotisseries rock!

**DDL Triggers**

SQL Server 2005 introduced a new trigger called the DDL Trigger. For you newbies, DDL means "Data Definition Language". (e.g. create table, alter table, create proc, etc.) What's great about these new triggers is if they are setup on the database/server, you are able to capture the DDL statement and automatically log it to a change log. You have no idea how much easier that is than doing it manually. Also, the laziness in all of us gets a little boost.

**Trigger Scope**

The DDL Triggers have two scope levels; database and server. The server-scoped statements that can be captured are:

|  |  |  |
| --- | --- | --- |
| ALTER\_AUTHORIZATION\_SERVER |  |  |
| CREATE\_DATABASE | ALTER\_DATABASE | DROP\_DATABASE |
| CREATE\_ENDPOINT | DROP\_ENDPOINT |  |
| CREATE\_LOGIN | ALTER\_LOGIN | DROP\_LOGIN |
| GRANT\_SERVER | DENY\_SERVER | REVOKE\_SERVER |

The database-scoped statements are:

|  |  |  |
| --- | --- | --- |
| CREATE\_APPLICATION\_ROLE | ALTER\_APPLICATION\_ROLE | DROP\_APPLICATION\_ROLE |
| CREATE\_ASSEMBLY | ALTER\_ASSEMBLY | DROP\_ASSEMBLY |
| ALTER\_AUTHORIZATION\_DATABASE |  |  |
| CREATE\_CERTIFICATE | ALTER\_CERTIFICATE | DROP\_CERTIFICATE |
| CREATE\_CONTRACT | DROP\_CONTRACT |  |
| GRANT\_DATABASE | DENY\_DATABASE | REVOKE\_DATABASE |
| CREATE\_EVENT\_NOTIFICATION | DROP\_EVENT\_NOTIFICATION |  |
| CREATE\_FUNCTION | ALTER\_FUNCTION | DROP\_FUNCTION |
| CREATE\_INDEX | ALTER\_INDEX | DROP\_INDEX |
| CREATE\_MESSAGE\_TYPE | ALTER\_MESSAGE\_TYPE | DROP\_MESSAGE\_TYPE |
| CREATE\_PARTITION\_FUNCTION | ALTER\_PARTITION\_FUNCTION | DROP\_PARTITION\_FUNCTION |
| CREATE\_PARTITION\_SCHEME | ALTER\_PARTITION\_SCHEME | DROP\_PARTITION\_SCHEME |
| CREATE\_PROCEDURE | ALTER\_PROCEDURE | DROP\_PROCEDURE |
| CREATE\_QUEUE | ALTER\_QUEUE | DROP\_QUEUE |
| CREATE\_REMOTE\_SERVICE\_BINDING | ALTER\_REMOTE\_SERVICE\_BINDING | DROP\_REMOTE\_SERVICE\_BINDING |
| CREATE\_ROLE | ALTER\_ROLE | DROP\_ROLE |
| CREATE\_ROUTE | ALTER\_ROUTE | DROP\_ROUTE |
| CREATE\_SCHEMA | ALTER\_SCHEMA | DROP\_SCHEMA |
| CREATE\_SERVICE | ALTER\_SERVICE | DROP\_SERVICE |
| CREATE\_STATISTICS | DROP\_STATISTICS | UPDATE\_STATISTICS |
| CREATE\_SYNONYM | DROP\_SYNONYM |  |
| CREATE\_TABLE | ALTER\_TABLE | DROP\_TABLE |
| CREATE\_TRIGGER | ALTER\_TRIGGER | DROP\_TRIGGER |
| CREATE\_TYPE | DROP\_TYPE |  |
| CREATE\_USER | ALTER\_USER | DROP\_USER |
| CREATE\_VIEW | ALTER\_VIEW | DROP\_VIEW |
| CREATE\_XML\_SCHEMA\_COLLECTION | ALTER\_XML\_SCHEMA\_COLLECTION | DROP\_XML\_SCHEMA\_COLLECTION |

Yeah, that's a lot of events that can be captured. For our purposes we're just going to focus on a select group of events that surround tables, stored procedures and functions.

**Creating a DDL Trigger**

It's pretty simple to create a DDL trigger. It's very similar to a table trigger:

CREATE TRIGGER [name of trigger]

ON [scope (database|server)]

FOR [event]

As

--Trigger definition here…

Here is an example of a real DDL trigger:

create trigger backup\_procs

on database

for create\_procedure, alter\_procedure, drop\_procedure

as

set nocount on

declare @data xml

set @data = EVENTDATA()

insert into dbo.eventslog(eventtype, objectname, objecttype, sqlcommand, username)

values(@data.value('(/EVENT\_INSTANCE/EventType)[1]', 'varchar(50)'),

@data.value('(/EVENT\_INSTANCE/ObjectName)[1]', 'varchar(256)'),

@data.value('(/EVENT\_INSTANCE/ObjectType)[1]', 'varchar(25)'),

@data.value('(/EVENT\_INSTANCE/TSQLCommand)[1]', 'varchar(max)'),

@data.value('(/EVENT\_INSTANCE/LoginName)[1]', 'varchar(256)')

)

go

The script above will create a DDL trigger called backup\_procs. It will insert a row into the eventslog table when a proc is created, altered or dropped. You'll see a more in depth explanation of the DDL trigger used in the change log system in a few minutes.

**What's that EVENTDATA() thingy?**

EVENTDATA is where we're going to get our data for the change log system. It's basically an XML datatype that looks like this:

<**EVENT\_INSTANCE**>

<**EventType**>event </**EventType**>

<**PostTime**>date-time</**PostTime**>

<**SPID**>spid</**SPID**>

<**ServerName**>name </**ServerName**>

<**LoginName**>login </**LoginName**>

<**UserName**>name</**UserName**>

<**DatabaseName**>name</**DatabaseName**>

<**SchemaName**>name</**SchemaName**>

<**ObjectName**>name</**ObjectName**>

<**ObjectType**>type</**ObjectType**>

<**TSQLCommand**>command</**TSQLCommand**>

<**/EVENT\_INSTANCE**>

The XML contains all of the information you'll need to create an entry into your changelog table.

**The change log system**

Ok. Now we're getting to the actual system we'll use to backup our objects. It's pretty basic now but you can expand on it greatly by adding extra columns or events to the trigger.

The change log system is using a database scoped trigger. That means every database you're going to use it on is going to need the trigger defined there. It's up to you whether you want to put the changelog table in each database or use a central database. I'm going to use a central database model, therefore all of the my inserts will be qualified with eventslogdb.dbo.changelog. We're going to capture 9 events: create, alter and drop table; create, alter and drop procedure; create, alter and drop function.

The changelog table will be defined as:

CREATE TABLE [dbo].[ChangeLog](

[LogId] [int] IDENTITY(1,1) NOT NULL,

[DatabaseName] [varchar] (256) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL,

[EventType] [varchar](50) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL,

[ObjectName] [varchar](256) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL,

[ObjectType] [varchar](25) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL,

[SqlCommand] [varchar](max) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL,

[EventDate] [datetime] NOT NULL CONSTRAINT [DF\_EventsLog\_EventDate] DEFAULT (getdate()),

[LoginName] [varchar](256) COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS NOT NULL

) ON [PRIMARY]

Our DDL trigger is defined as:

create trigger backup\_objects

on database

for create\_procedure, alter\_procedure, drop\_procedure,

create\_table, alter\_table, drop\_table,

create\_function, alter\_function, drop\_function

as

set nocount on

declare @data xml

set @data = EVENTDATA()

insert into eventslogdb.dbo.changelog(databasename, eventtype,

objectname, objecttype, sqlcommand, loginname)

values(

@data.value('(/EVENT\_INSTANCE/DatabaseName)[1]', 'varchar(256)'),

@data.value('(/EVENT\_INSTANCE/EventType)[1]', 'varchar(50)'),

@data.value('(/EVENT\_INSTANCE/ObjectName)[1]', 'varchar(256)'),

@data.value('(/EVENT\_INSTANCE/ObjectType)[1]', 'varchar(25)'),

@data.value('(/EVENT\_INSTANCE/TSQLCommand)[1]', 'varchar(max)'),

@data.value('(/EVENT\_INSTANCE/LoginName)[1]', 'varchar(256)')

)

go

Just for the sake of the example I put this trigger in the same database where my changelog table resides.

**Let's breakdown that trigger.**

create trigger backup\_objects

on database

We're creating a trigger named backup\_objects that is scoped for the database.

for create\_procedure, alter\_procedure, drop\_procedure,

create\_table, alter\_table, drop\_table,

create\_function, alter\_function, drop\_function

The trigger will fire when a create procedure, alter procedure, drop procedure, create table, alter table, drop table, create function, alter function or drop function statement is executed.

set nocount on

If nocount is set to off then you'll get a row count when executing any DDL statements for the above events. No big deal, but I don't getting a message like "1 row(s) affected" when altering a proc.

declare @data xml

set @data = EVENTDATA()

Remember that EVENTDATA will get the details of the executed DDL statement. We're simply assigning it to a local xml variable called @data.

insert into eventslogdb.dbo.changelog(databasename, eventtype,

objectname, objecttype, sqlcommand, loginname)

values(

@data.value('(/EVENT\_INSTANCE/DatabaseName)[1]', 'varchar(256)'),

@data.value('(/EVENT\_INSTANCE/EventType)[1]', 'varchar(50)'),

@data.value('(/EVENT\_INSTANCE/ObjectName)[1]', 'varchar(256)'),

@data.value('(/EVENT\_INSTANCE/ObjectType)[1]', 'varchar(25)'),

@data.value('(/EVENT\_INSTANCE/TSQLCommand)[1]', 'varchar(max)'),

@data.value('(/EVENT\_INSTANCE/LoginName)[1]', 'varchar(256)')

)

Insert the event data into our changelog table. @data.value is calling the xquery function "value". The value function is defined as ["value(Xquery, SQLType)"](http://msdn2.microsoft.com/en-us/library/ms178030.aspx).

@data.value('(/EVENT\_INSTANCE/DatabaseName)[1]', 'varchar(256)') is going to return the name of the database where the DDL trigger is being fired as a varchar(256). The rest of the @data.value calls are similar to this one. You should be able to figure out what each of those functions return.

I used the LoginName element from EVENTDATA() because it returns the login associated with the SID (security identification number). It's similar to using suser\_name(). If you use the UserName element from the EVENTDATA function you will not get the true identification of the user executing the statement. If it's always you making the change, then this is not a problem. However, if you have more than one DBA (or... gasp... developers!) making changes, it's a good idea to keep track of who is making the change.

**Causing the triggers to fire**

If you've paid attention you now know that the DDL trigger we created will fire when we create, alter or drop a table, proc or function. I'm going to create this proc in my database:

use eventslogdb

go

CREATE proc [dbo].[usp\_GetDBVersion]

as

select @@version

Yes, it's simplistic but it works. This will fire the backup\_objects trigger and insert a row into the changelog table. Now let's take a look at the row in the changelog table:

SELECT [LogId]

,[DatabaseName]

,[EventType]

,[ObjectName]

,[ObjectType]

,[SqlCommand]

,[EventDate]

,[LoginName]

FROM [EventsLogDb].[dbo].[ChangeLog]

Results:

LogId DatabaseName EventType ObjectName ObjectType

----------- ------------ ---------------- ------------------ -------------

1 EventsLogDB CREATE\_PROCEDURE usp\_GetDBVersion PROCEDURE

SqlCommand EventDate LoginName

------------------------------------- ------------------------ --------------

CREATE proc [dbo].[usp\_GetDBVersion] 2007-07-23 20:14:29.780 ROCK-XP\chris

as

select @@version

The data in the table is pretty self-explanitory. I'm sure you can see how helpful this would be if you need to track down a change made to one of your objects.

One last bit of information. If you alter or drop the changelog table and do not update/drop the DDL trigger to reflect the change, any future DDL statements could fail because of this. It's happened to me and I hope it doesn't happen to you!

So that's it. It's a simple yet powerful way to keep track of changes to your database. Remember, just "set it and forget it". My apologies to Mr. Popeil.