AlwaysOn Availability Groups – Enable or Disable Jobs on Failover:

<https://www.sqlrx.com/alwayson-availability-groups-enable-or-disable-jobs-on-failover/>

After reading up on AlwaysOn Monitoring and Alerting (<http://blog.sqlrx.com/2015/08/27/alwayson-monitoring-and-alerting/>), I was asked to come up with a way to automatically enable jobs that may only need to run on the primary replica of an availability group. Requirements were that on failover of an AlwaysOn Availability Group some jobs should be enabled on the primary replica and some jobs should be disabled on the secondary replica. After some thinking and research on the sys.dm\_hadr\_availability\_group\_states dynamic management view, I came up with a solution that uses a job category, a failover alert setup for error 1480 that is notated in the AlwaysOn Monitoring and Alerting post and some code in a job that is only run if the failover alert fires.

The sys.dm\_hadr\_availability\_group\_states dmv gives information on the state of availability groups that are running on the instance including the name of the instance that is the primary. <https://msdn.microsoft.com/en-us/library/ff878537.aspx>.

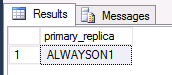
Here is my query:

SELECT s.primary\_replica

FROM sys.dm\_hadr\_availability\_group\_states s

JOIN sys.availability\_groups ag ON ag.group\_id = s.group\_id

WHERE ag.name = ‘AGroup\_Dev’

[](https://www.sqlrx.com/wp-content/uploads/2015/10/blog_20151008_1.png)

Since I can get at information showing if an instance is primary or secondary, all I have to do is find the jobs that should be enabled. The best way I could think of to do this is to assign a specific job category to jobs that need some type of action taken on them and query for jobs with the correct category on them.

SELECT j.name

FROM msdb.dbo.sysjobs j

WHERE EXISTS (SELECT c.category\_id

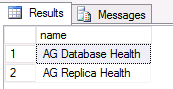
FROM msdb.dbo.syscategories c

WHERE j.category\_id = c.category\_id

AND c.name = ‘SomeCategory’)

AND j.enabled = 0  — disabled but should be enabled on failover

ORDER BY j.name

[](https://www.sqlrx.com/wp-content/uploads/2015/10/blog_20151008_2.png)

All I would need to do is add some logic that would roll through (yes, sadly that means a cursor) each job returned by the query above and run sp\_update\_job to enable or disable it.

FETCH NEXT FROM agjobs INTO @jobname

— Enable AG jobs if primary

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @SQLString = ‘EXEC msdb.dbo.sp\_update\_job @job\_name = ”’+@jobname+”’ , @enabled = 0′

EXEC sp\_executesql @SQLString

— Get the next record

FETCH NEXT FROM agjobs INTO @jobname

END

— Close and deallocate the cursor

CLOSE agjobs

DEALLOCATE agjobs

So, now let’s put it all together and fill in the rest of the code…..

In our development server we are using the VM’s and instances that were used in the AlwaysOn Monitoring and Alerting post and the AGroup\_Dev availability group that was set up. There are 3 instances: AlwaysOn1 is set as the primary, AlwaysOn2 is a secondary in synchronous mode, and AlwaysOn3 is a secondary in asynchronous mode.

On all 3 instances we have the jobs AG Database Health and AG Replica Health. These jobs should only be enabled to run if the instance is the primary replica for AGroup\_Dev.

First, create a new job category that will be assigned to any jobs that must have a specific action applied to it if a failover occurs. In my example below, I have created the job category AG1 on the AlwaysOn1 instance.

USE [msdb]

GO

— Create AG job category

DECLARE @ReturnCode INT

SELECT @ReturnCode = 0

IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE name = N’AG1′ AND category\_class = 1)

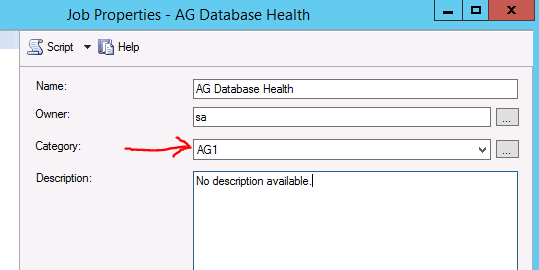
BEGIN

EXEC @ReturnCode = msdb.dbo.sp\_add\_category @class = N’JOB’, @type = N’LOCAL’, @name = N’AG1′

END

GO

I also created job categories (AG2 & AG3) on AlwaysOn2 and AlwaysOn3 respectively. I did this so that I could keep track of what instance the category applies to. Once my job category is created, I then assigned it to the AG Database Health and AG Replica Health jobs.

[](https://www.sqlrx.com/wp-content/uploads/2015/10/blog_20151008_3.png)

Next, make sure that Error 1480 is set to generate an alert.

EXEC msdb.dbo.sp\_add\_alert

@name = N’AG Role Change (failover)’,

@message\_id = 1480,

@severity = 0,

@enabled = 1,

@delay\_between\_responses = 0,

@include\_event\_description\_in = 1;

GO

Now we can make the job that needs to query the dmv and enable or disable jobs depending on the value in the primary\_replica column. I created the AlwaysOn Job Management job on each instance using the code below and simply changed the places where the job category is used to have the correct category name for the instance it is on.

— create job to run if alert 1480 fires

/\*\*\*\*\*\* Object:  Job [AlwaysOn Job Management]    \*\*\*\*\*\*/

BEGIN TRANSACTION

DECLARE @ReturnCode INT

SELECT @ReturnCode = 0

DECLARE @jobId BINARY(16)

EXEC @ReturnCode = msdb.dbo.sp\_add\_job @job\_name=N’AlwaysOn Job Management’,

@enabled=1,

@notify\_level\_eventlog=0,

@notify\_level\_email=0,

@notify\_level\_netsend=0,

@notify\_level\_page=0,

@delete\_level=0,

@description=N’No description available.’,

@category\_name=N'[Uncategorized (Local)]’,

@owner\_login\_name=N’sa’,

@job\_id = @jobId OUTPUT

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

/\*\*\*\*\*\* Object:  Step [Enable or Disable]    \*\*\*\*\*\*/

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobstep @job\_id=@jobId, @step\_name=N’Enable or Disable’,

@step\_id=1,

@cmdexec\_success\_code=0,

@on\_success\_action=1,

@on\_success\_step\_id=0,

@on\_fail\_action=2,

@on\_fail\_step\_id=0,

@retry\_attempts=0,

@retry\_interval=0,

@os\_run\_priority=0,

@subsystem=N’TSQL’,

@command=N’DECLARE @primreplica VARCHAR(100)

DECLARE @AGName VARCHAR(100)

DECLARE @jobname VARCHAR(128)

DECLARE @SQLString NVARCHAR(2000)         — Dynamic SQL string

SET @AGName = ”AGroup\_Dev”  — must know ag name

SELECT @primreplica = s.primary\_replica

FROM sys.dm\_hadr\_availability\_group\_states s

JOIN sys.availability\_groups ag ON ag.group\_id = s.group\_id

WHERE ag.name = @AGName

IF UPPER(@primreplica) =  UPPER(@@SERVERNAME)

BEGIN

— PRIMARY

— Find all disabled AG jobs.

DECLARE agjobs CURSOR FOR

SELECT j.name

FROM msdb.dbo.sysjobs j

WHERE EXISTS (SELECT c.category\_id

FROM msdb.dbo.syscategories c

WHERE j.category\_id = c.category\_id

AND name = ”AG1”)

AND j.enabled = 0  — disabled but should be enabled on failover

ORDER BY j.name

— Open the cursor

OPEN agjobs

FETCH NEXT FROM agjobs INTO @jobname

— Enable AG jobs if primary

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @SQLString = ”EXEC msdb.dbo.sp\_update\_job @job\_name = ”””+@jobname+””” , @enabled = 1”

EXEC sp\_executesql @SQLString

–PRINT @SQLString

— Get the next record

FETCH NEXT FROM agjobs INTO @jobname

END

— Close and deallocate the cursor

CLOSE agjobs

DEALLOCATE agjobs

END

ELSE

BEGIN

— SECONDARY

— Find all enabled AG jobs.

DECLARE agjobs CURSOR FOR

SELECT j.name

FROM msdb.dbo.sysjobs j

WHERE EXISTS (SELECT c.category\_id

FROM msdb.dbo.syscategories c

WHERE j.category\_id = c.category\_id

AND name = ”AG1”)

AND j.enabled = 1  — enabled but should be disabled on failover

ORDER BY j.name

— Open the cursor

OPEN agjobs

FETCH NEXT FROM agjobs INTO @jobname

— Disable all AG1 jobs if secondary

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @SQLString = ”EXEC msdb.dbo.sp\_update\_job @job\_name = ”””+@jobname+””” , @enabled = 0”

EXEC sp\_executesql @SQLString

–PRINT @SQLString

— Get the next record

FETCH NEXT FROM agjobs INTO @jobname

END

— Close and deallocate the cursor

CLOSE agjobs

DEALLOCATE agjobs

END’,

@database\_name=N’master’,

@flags=0

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_update\_job @job\_id = @jobId, @start\_step\_id = 1

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobserver @job\_id = @jobId, @server\_name = N'(local)’

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

COMMIT TRANSACTION

GOTO EndSave

QuitWithRollback:

IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION

EndSave:

GO

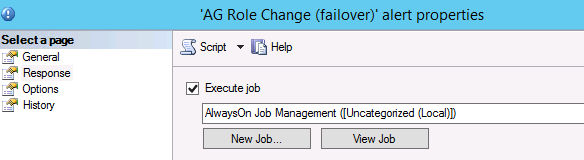
I don’t have a schedule set up for the AlwaysOn Job Management job because I only want this to run if a failover alert is sent. All that is left to do is to make the 1480 failover alert run the AlwaysOn Job Management job.

— Make the alert run the Job management job

EXEC msdb.dbo.sp\_update\_alert @name=N’AG Role Change (failover)’,

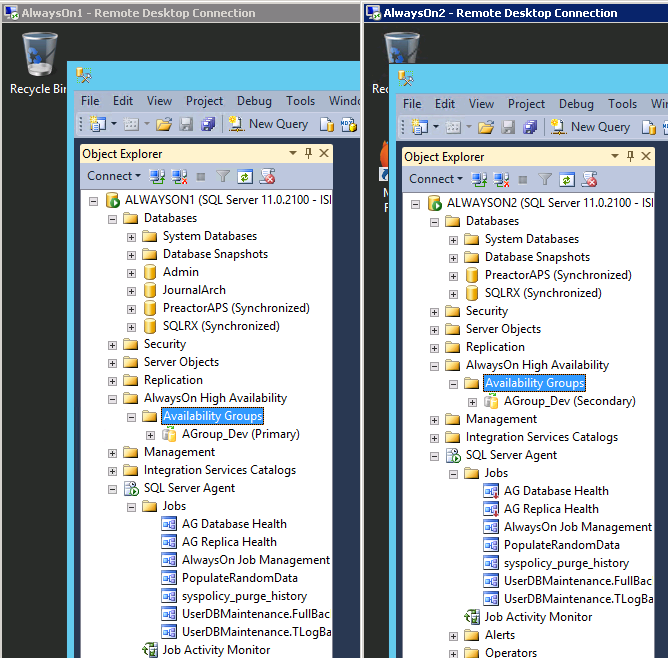
@job\_name=N’AlwaysOn Job Management’

GO

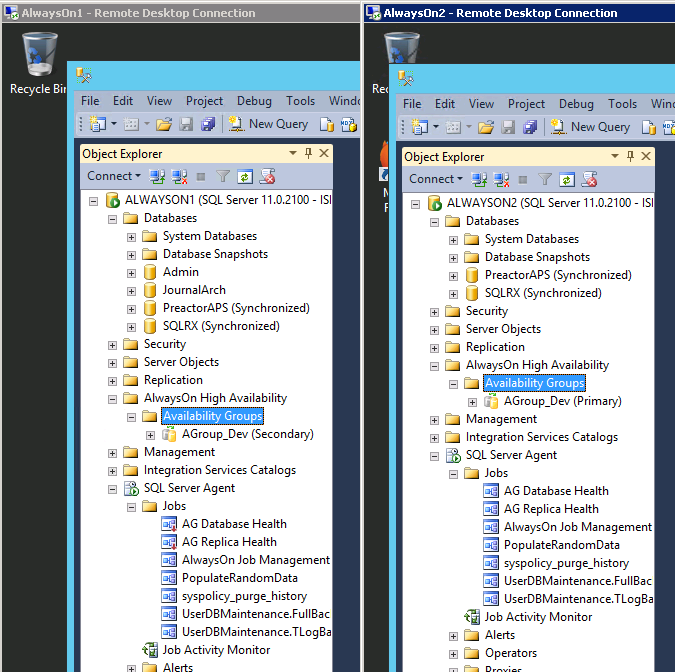
[](https://www.sqlrx.com/wp-content/uploads/2015/10/blog_20151008_4.png)

I did this for all instances that are in the AlwaysOn group AGroup\_Dev, so now we need to test by failing over to each node and making sure that the jobs with the special category assigned are enabled or disabled.

Here are AlwaysOn1 and AlwaysOn2 before failover:

[](https://www.sqlrx.com/wp-content/uploads/2015/10/blog_20151008_5.png)

And here is how they look after failover:

[](https://www.sqlrx.com/wp-content/uploads/2015/10/blog_20151008_6.png)

As you can see the AG Database Health and AG Replica Health jobs are now disabled on AlwaysOn1 and are enabled on AlwaysOn2.

Of course, the job code can be changed to accommodate more than 1 availability group and more than 1 job category. Be creative and have fun with it! Hope this helps anyone who needs this type of functionality for their AlwaysOn replicas.

For more information about blog posts, concepts and definitions, further explanations, or questions you may have…please contact us at [SQLRX@sqlrx.com](mailto:SQLRX@sqlrx.com). We will be happy to help! Leave a comment and feel free to track back to us. We love to talk tech with anyone in our SQL family!