Performance Analyzer for Microsoft Dynamics

Deployment and User Guide

Microsoft Corp.

Contents

CHANGE RECORD

Date	Author	Version	Change Reference
March 1, 2012	Microsoft	1.0	Created document

INTRODUCTION

Performance Analyzer for Microsoft Dynamics (Performance Analyzer) is the tool used by Microsoft Dynamics support, Premier Field Engineers, and product team members to diagnose performance issues with Dynamics products.

***IMPORTANT**: The purpose of the Performance Analyzer is to be used on a continual basis so it is important for administrators to understand the components that make up the tool to ensure all jobs and collectors are running.*

Performance Analyzer collects a variety of pertinent information from the database server, application object server (AOS), and application server. This information is captured from a number of collectors provided by Performance Analyzer that includes query statistics, query plans, index statistics, database and AOS server configurations, AOS event logs, and AOT metadata. In addition, blocking and deadlocking events are collected through SQL tracing events while performance counter data is collected from the database and AOS servers.

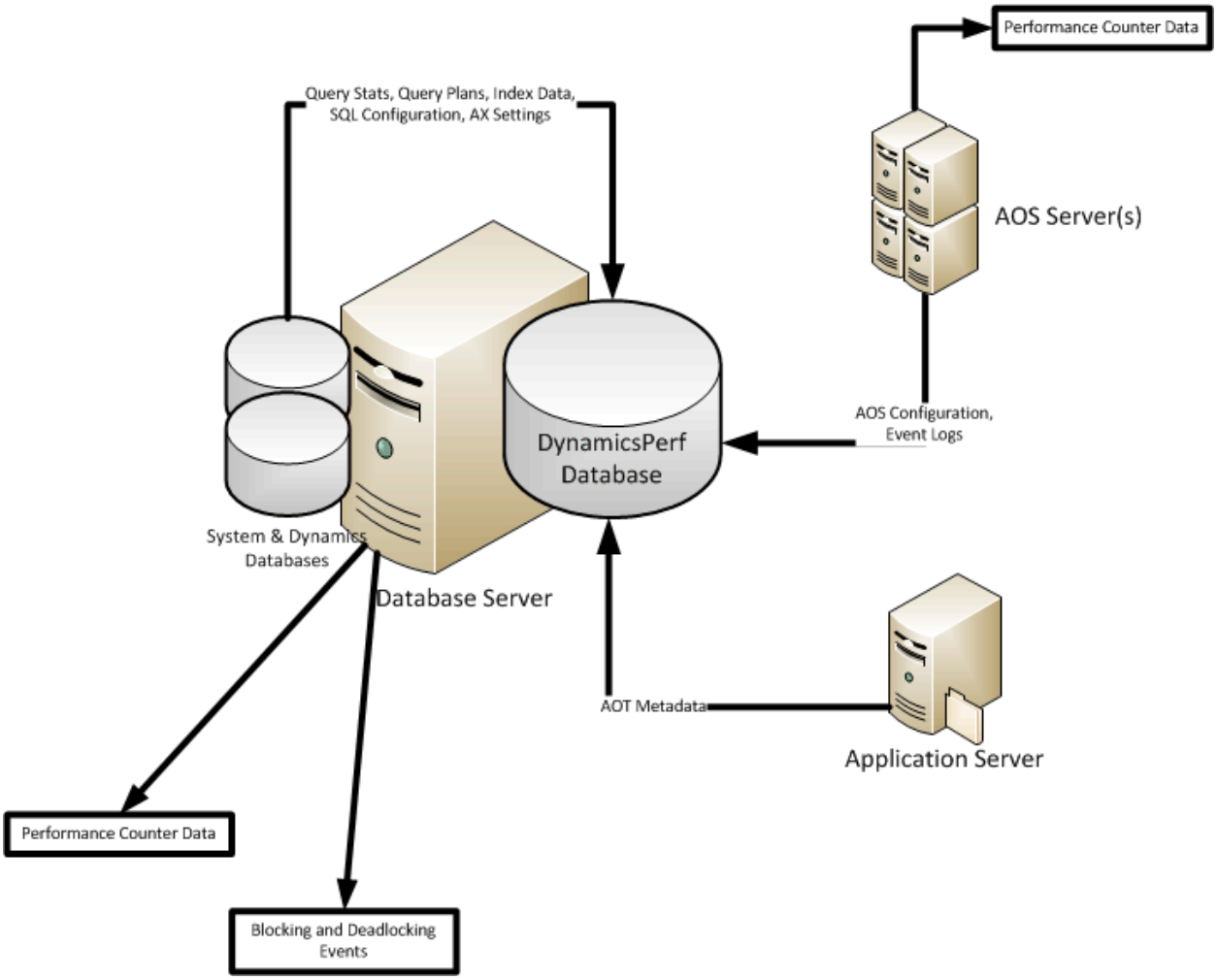


FIGURE 1 PERFORMANCE ANALYZER FOR DYNAMICS AX

As you can see from the illustration above, the **DynamicsPerf** database is the central repository for most of the data collected for Performance Analyzer.

Performance Analyzer is delivered as a SQL Server solution file and includes a set of SQL jobs, X++ classes, VB scripts, and performance counters to initiate the collection process. It also includes a set of sample SQL scripts that can be used to query and analyze the populated tables and views in the DynamicsPerf database.

The Performance Analyzer is delivered as a SQL Server solution and consists of a number of collectors as SQL jobs, X++ classes, VB scripts, and performance counters. These collectors that make up the Performance Analyzer are categorized within this document as the following:

- Capture Performance Data
- Capture AOT Metadata
- Capture AOS Settings and Event Logs
- Capture Database Blocking
- Capture Performance Counter Data

We will discuss each one of the collectors in the following sections and the process for deploying and maintaining Performance Analyzer in later sections.

CAPTURE PERFORMANCE DATA

The Capture Performance Data collector is initiated through the **DYNPERF_Capture_Stats** SQL job. When executed, this job captures query statistics, query plans, index statistics, SQL database information, and SQL configuration. For Dynamics AX, it will also collect valuable AX set up information as well as long running query data from AX.

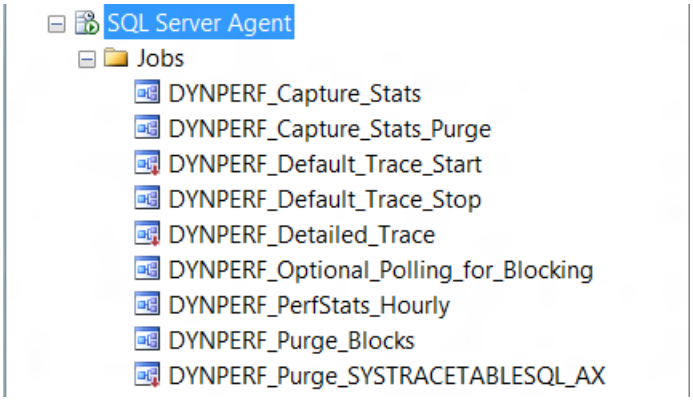


FIGURE 2 DYNPERF_Capture_Stats Job

The **DYNPERF_Capture_Stats** job can be configured to capture performance data for a single database or multiple databases on the server. The **DYNPERF_Capture_Stats** job should be run on a daily basis! By default, the schedule is set to run at 5:00 PM each day but you can change it. To help illustrate uses of this job, below are sample use cases.

Use Case – Verify SQL Configuration

Simon, the Systems Implementer, is experiencing some performance issues and wants to use Performance Analyzer to determine if his database is configured correctly for Dynamics AX. Performance Analyzer collects valuable database server information used to troubleshoot Dynamics AX.

Simon verifies that the “DYNPERF_Capture_Stats” job executed successfully the previous night and that it populated the tables so he executes a sample SQL configuration query included with Performance Analyzer to view server information, configuration, and the setup of the database files.

Simon notices that the Max Degree of Parallelism is set to 0 and confirms with Microsoft that this option should be set to 1 for Dynamics AX.

As a DBA, Simon could have used any number of SQL DMV’s that are available to get this information but prefers to use the SQL data captured in the **DynamicsPerf** databases because he knows it is relevant to Dynamics AX and organized and grouped in a way that makes sense.

Use Case – Analyze Expensive Queries

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze the most expensive queries. Performance Analyzer collects the database server’s query statistics and plans.

Simon verifies that the **DYNPERF_Capture_Stats** job executed successfully the previous night and that it populated the tables so he executes some sample queries included with Performance Analyzer to view the top 100 queries by total elapsed time. When he runs this query he is able to see elapsed times, execution counts, query parameter values, number of reads/writes, and the query plan among other useful information.

Simon clicks on the query plan for the most expensive query to analyze the plan and notices that the optimizer is recommended a new non-clustered index which will have a large impact.

Simon also runs one of the sample queries included with Performance Analyzer to get a complete list of all queries that the optimizer suspects can be optimized by new or changed indexes.

As a DBA, Simon could have used any number of SQL DMV’s that are available to get this information but prefers to use the query data captured in the **DynamicsPerf** databases because he knows it is relevant to Dynamics AX and organized and grouped in a way that makes sense.

Use Case – Analyze Indexes

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze which queries may be scanning tables so he can review the indexes for them.

Simon verifies that the **DYNPERF_Capture_Stats** job executed successfully the previous night and that it populated the tables so he executes some sample queries included with Performance Analyzer to view the top 100 queries that scans tables. He notices that the table could benefit by a new index and makes the changes.

Simon also wants to take a more general look at the indexes and runs other sample queries included with Performance Analyzer to find any Heaps or Clustered indexes that should be changed.

As a DBA, Simon could have used any number of SQL DMV’s that are available to get this information but prefers to use the index data captured in the **DynamicsPerf** databases because he knows it is relevant to Dynamics AX and organized and grouped in a way that makes sense.

Collect AX Long Running Queries

For Dynamics AX, you can capture long running SQL statements generated by the Dynamics AX application to include; SQL text, Duration, Call stack, and AX user ID. This information is useful in identifying areas of potential optimization from within AX code.

This information is pulled into the **DynamicsPerf** database through the "DYNPERF_Capture_Stats" job but only if:

1) Client tracing is enabled on the AOS instance
2) Tracing is enabled on the user.

Use Case – Analyze AX Long Running Queries

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze the longest running queries through AX code. Simon enabled client tracing on the AOS servers and runs a script to enable tracing for every AX user within the AX database. Simon validates that this information has been captured so he executes the sample queries included with Performance Analyzer to view the top 100 longest running queries from AX source code.

He analyzes the longest running query and notes the SQL text, AX classes, and user ID. He discusses this information with the user to determine the exact processes for this user when running this query. He then works with the developer of this code to determine if any optimization opportunities.

Collect Hourly Performance Data

You may also optionally capture performance data hourly and there is a separate job for this, **DYNPERF_PerfStats_Hourly**. Four Data Management Views are used to capture this information:

- Sys.indexes
- Sys.dm_db_index_usage_stats
- Sys.dm_io_virtual_file_stats
- Sys.dm_os_wait_stats

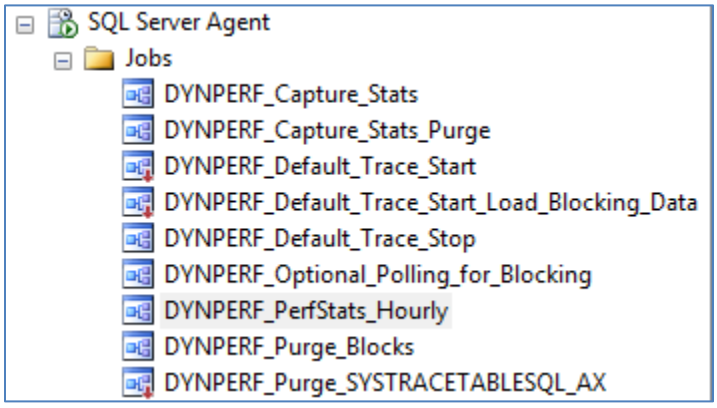


FIGURE 3 DYNPERF_PerfStats_Hourly Job

CAPTURE AOT METADATA

The Capture AOS Metadata collector is specific to Dynamics AX and used to capture AOT property data to include table properties, index properties, and cache settings. This collector is initiated through the AOTExport X++ Class.

When the X++ AOTExport Class is imported into AX, it will create the following objects:

- AOTTABLEPROPERTIES table
- AOTINDEXPROPERTIES table
- AOTINDEXFIELDS table
- AOTExport Class

The AOTExport class is both runnable and batchable. When executed, it will populate the tables listed above. Subsequent executions of **DYNPERF_Capture_Stats** will pull data from those tables into related tables in the **DynamicsPerf** database.

***NOTE**: when populated, the table listed above will remain small in size, typically occupying no more than 2-3 MB space within the AX database. Likewise, the execution of AOTExport will be of short duration and minimal impact to system performance. It is still recommended, however, to schedule the execution of AOTExport during none peak periods.*

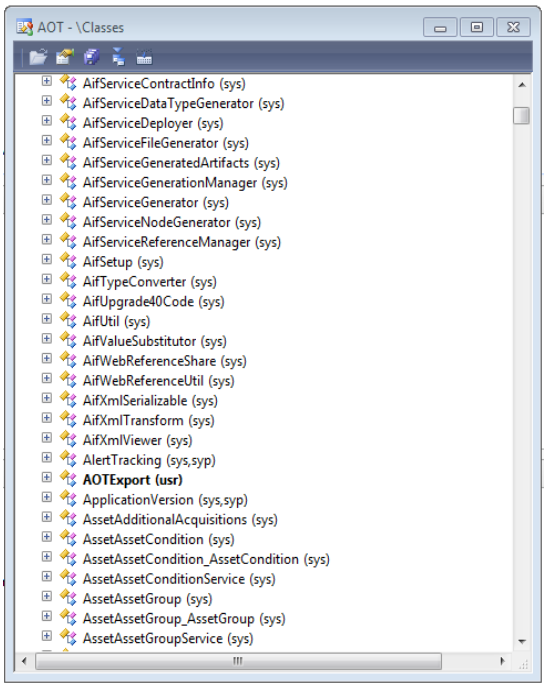


FIGURE 4 AOTExport Class

The AOTExport X++ Class should be scheduled on a periodic basis to ensure all changes into the AOT are added to the DynamicsPerf database.

Use Case – Verify Cache Settings

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze the cache settings for his AX tables. Performance Analyzer collects table and index properties from the AOT.

Simon validates that the **DYNPERF_Capture_Stats** job ran the previous night and that it populated the required tables with the cache settings. He executes a script to determine which tables in AX have the "EntireTableCache" property enabled and the table is over 128K in size.

Simon realizes that any results mean that the cache settings should be changed to another AX cache setting as tables this large should not be set to "EntireTableCache".

CAPTURE AOS SETTINGS AND EVENT LOGS

The Capture AOS Settings and Event Logs collector will capture AOS configuration and event logs from each active AOS Server in the environment. This collector is initiated through the AOSANALYSIS.VBS vb script. It works in conjunction with the AOSANLAYSIS.CMD batch script to populate tables in the **DynamicsPerf** database with registry settings and two weeks of event logs for each AOS instance currently connected to the AX database.

- AOSANALYSIS. VBS – This script will populate tables in the DynamicsPerf database with registry settings and two weeks of event logs for each AOS instance currently connected to the AX database.
- AOSANALYSIS.CMD – This is the batch script used to execute AOSANALYSIS.VBS. Two arguments must be passed"
    - Database server and instance name
    - AX database name

The Capture AOS Settings and Event Logs can be executed manually when needed or scheduled to run periodically.

Use Case – Analyze AOS Configuration Settings

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze the configuration settings for all of the AOS servers. Performance Analyzer collects AOS configuration information from all active AOS servers.

Before he executes the AOSANALYSIS vb script, he modifies the batch script with his server name and AX database name. Simon then runs the vb script. The information is now pulled into the **DynamicsPerf** database where he can analyze the settings.

Simon runs scripts against the tables that were populated in the **DynamicPerf** database and notices that the debugging settings are enabled on all of his AOS servers. Microsoft has confirmed with Simon that it is not best practice to run with the debug settings enabled as it could cause performance issues. Simon makes the necessary changes to his configuration.

CAPTURE DATABASE BLOCKING

The Capture Database Blocking collector is used to track blocking events over a long period of time. It records blocking information into a trace file. This collector is initiated through the **DYNPERF_Default_Trace_Start** job. This job will run for 25 hours and it is recommended to schedule this job to run at 12:00 AM every day. If you wish to stop the trace you execute the **DYNPERF_Default_Trace_Stop** job.

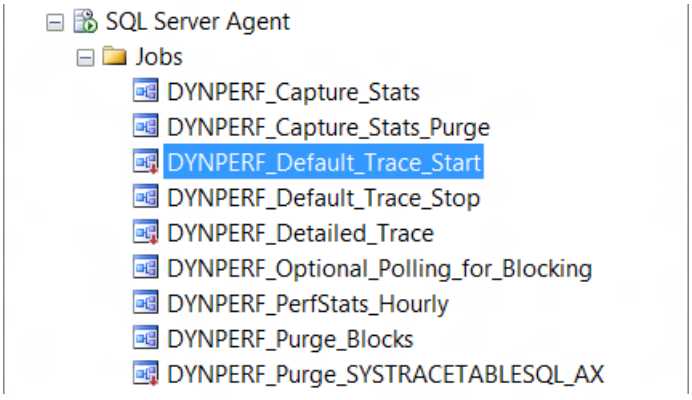


FIGURE 5 START TRACING JOB

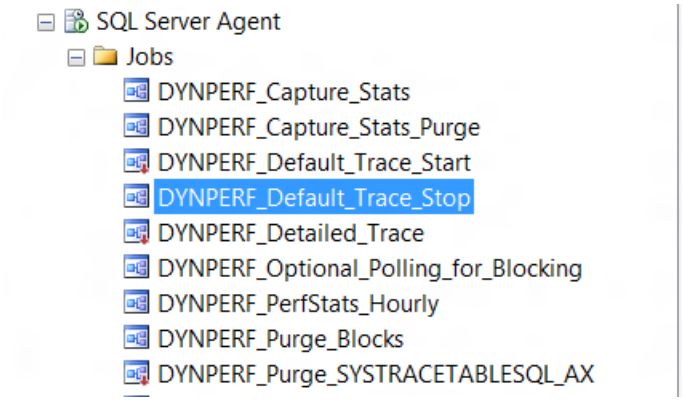


FIGURE 6 STOP TRACING JOB

Use Case – Analyze Blocking

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze if any blocking is occurring during a process. Performance Analyzer collects blocking information into a trace file. Simon ensures that the tracing is executing and that it has successfully created the trace files.

He uses the sample queries included with Performance Analyzer to view trace files and notices several blocking events on the InventSumLogTTS table with long durations of time. He queries his users and finds out that this will occur if Master planning is executing at the same time users are querying sales orders and instructs that Master planning be a scheduled process after hours.

CAPTURE PERFORMANCE COUNTER DATA

The Capture Performance Counter data collectors are used to collect performance counter logs on the database server and AOS servers. This is in the form of an .xml file that can be imported into your database or AOS servers as a template. Once this template is loaded it includes a set of counters that are useful for analyzing performance bottlenecks for a Dynamics system. This information is then logged to performance counter log files (.blg).

These templates should be deployed and scheduled to run on a continual basis.

Use Case – Analyze Disk Subsystem Performance

Simon, the Systems Implementer, is experiencing performance issues and wants to use Performance Analyzer to analyze if his disk subsystem is performing optimally during busy times. Performance Analyzer collects performance counter data which can be used to analyze several system-wide areas to include disk performance, memory usage, cpu usage, network usage, etc. Simon ensures that the performance counters are executing and that it is successfully creating the performance counter logs.

He imports the performance counter log files into his Performance Monitor and adds the counters to monitory disk subsystem read/write latency. When reviewing these counters he notices that the average read/write times are way above recommended thresholds which will have an enormous impact on overall system performance. Simon, contacts his SAN and SQL support to look for ways to improve disk latency.

DEPLOYING PERFORMANCE ANALYZER

There are several steps that need to be completed in order to successfully deploy Performance Analyzer for Microsoft Dynamics. The Performance Analyzer is meant to be deployed and set up for data collection on a continual basis throughout the life of your AX system. This ensures that if performance issues arise, you are able to quickly identify the bottleneck as well as use for comparison purposes.

Deployment Setup Checklist

The following is a summarized checklist of the steps to deploy Performance Analyzer. See the steps below for detailed information.

Step #	Task
1	Create Database, Objects, and Jobs
2	Configure and Schedule Performance Data Capture
3	Configure and Schedule Database Blocking Capture
4	Enable Long Running Query Capture for AX (AX)
5	Configure and Schedule AOT Metadata Capture (AX)
6	Configure and Schedule AOS Configuration and Event Logs Capture
7	Configure and Schedule Performance Counter Logging on Database Server
8	Configure and Schedule Performance Counter Logging on AOS Server(s)

Before you begin

Before you deploy Performance Analyzer, you must complete the following:

1. Extract the DynamicsPerfxxx.zip file to a location to where you can browse from the database and AOS servers
2. Ensure you have rights to create new databases on the database server
3. Ensure you have read access to the AX database
4. Ensure you have write access to the DynamicsPerf database (this database gets created as part of Performance Analyzer)
5. Ensure you have Admin permissions to each of the AOS servers connected to the AX database
6. Ensure you have created a local folder on the database server called \SQLTRACE to store the trace files that get generated
7. Ensure that every active AOS server in the AX instance has been started with the *'Allow client tracing on Application Object Server instance'* checkbox enabled

Create Database, Objects, and Jobs

In order to use Performance Analyzer, you must first create the **DynamicsPerf** database, it's objects, and jobs. In the following steps you will create the **DynamicsPerf** database, its objects, and jobs.

1. On the database server, open SQL Server Management Studio (SSMS)
2. Click File>Open, Project/Solution
3. Browse to the location for where you extracted the DynamicsPerf1.15 for SQL2008+.zip
4. Select the *Performance Analyzer 1.15 for Microsoft Dynamics.ssmssln* file
5. In Solution Explorer, open the *1-Create_Core_Objects.sql* script
6. Execute the script. [This will create the **DynamicsPerf** database and SQL jobs.]

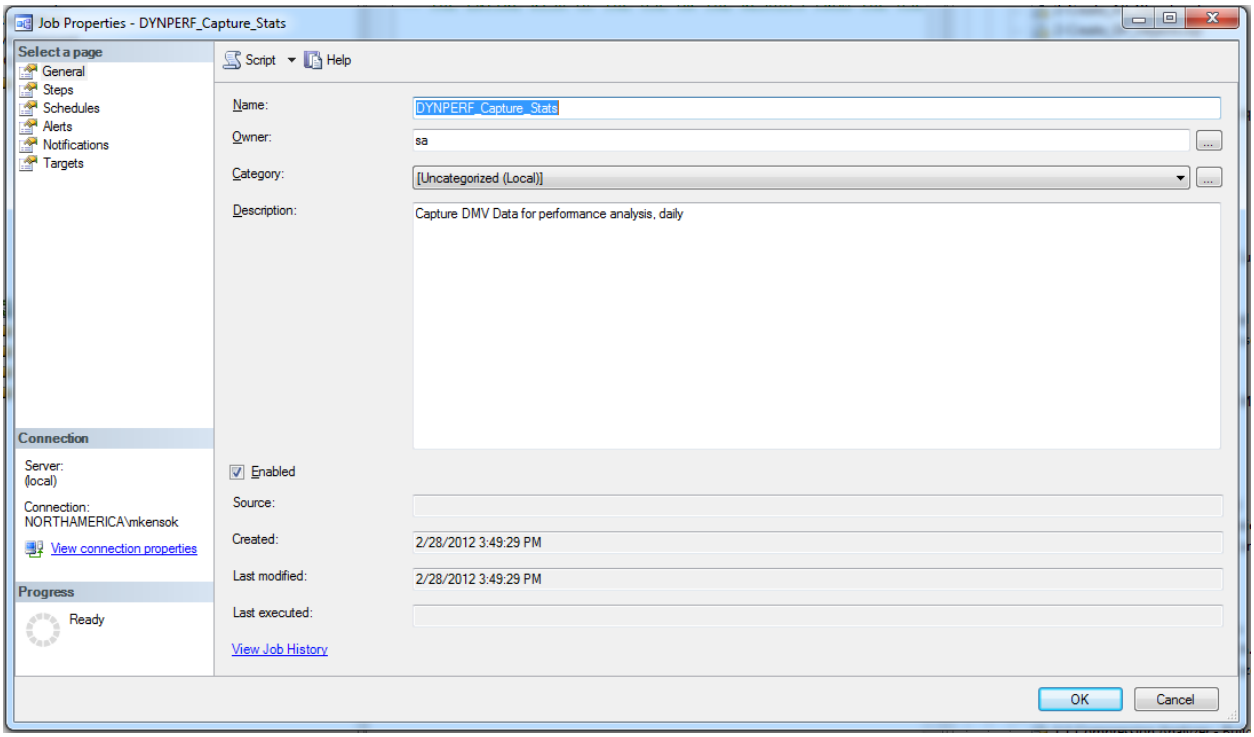
***NOTE**: Ensure that you read the notes in the script if you wish to path the **DynamicsPerf** files to a location other than the C drive*

7. In Solution Explorer, open the *2-Create_AX_Objects.sql* script from the Solution Explorer
8. Execute the script. [This will create the necessary objects in the DynamicsPerf database]

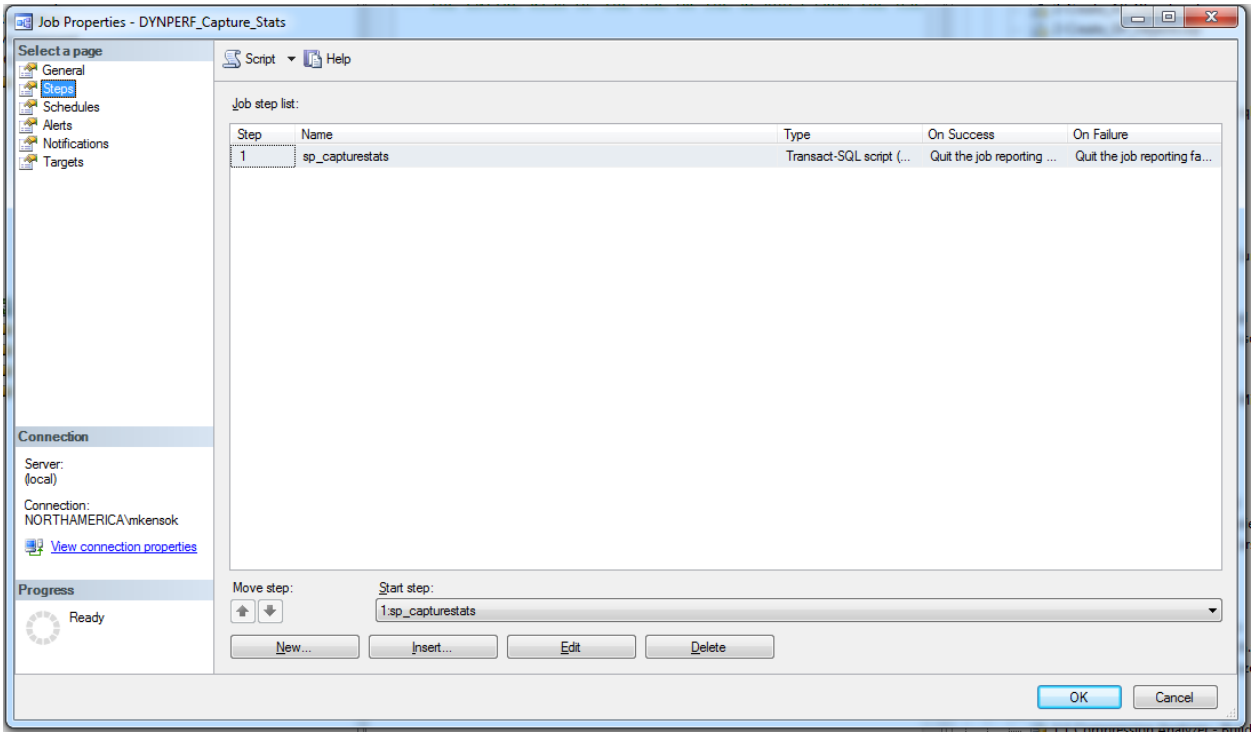
Configure and Schedule Performance Data Capture

After you have created the **DynamicsPerf** database, objects, and jobs it is time to configure the database and jobs to capture the performance data. In the following steps you will configure and schedule the capture of the performance data. The performance data collected will be stored in the **DynamicsPerf** database:

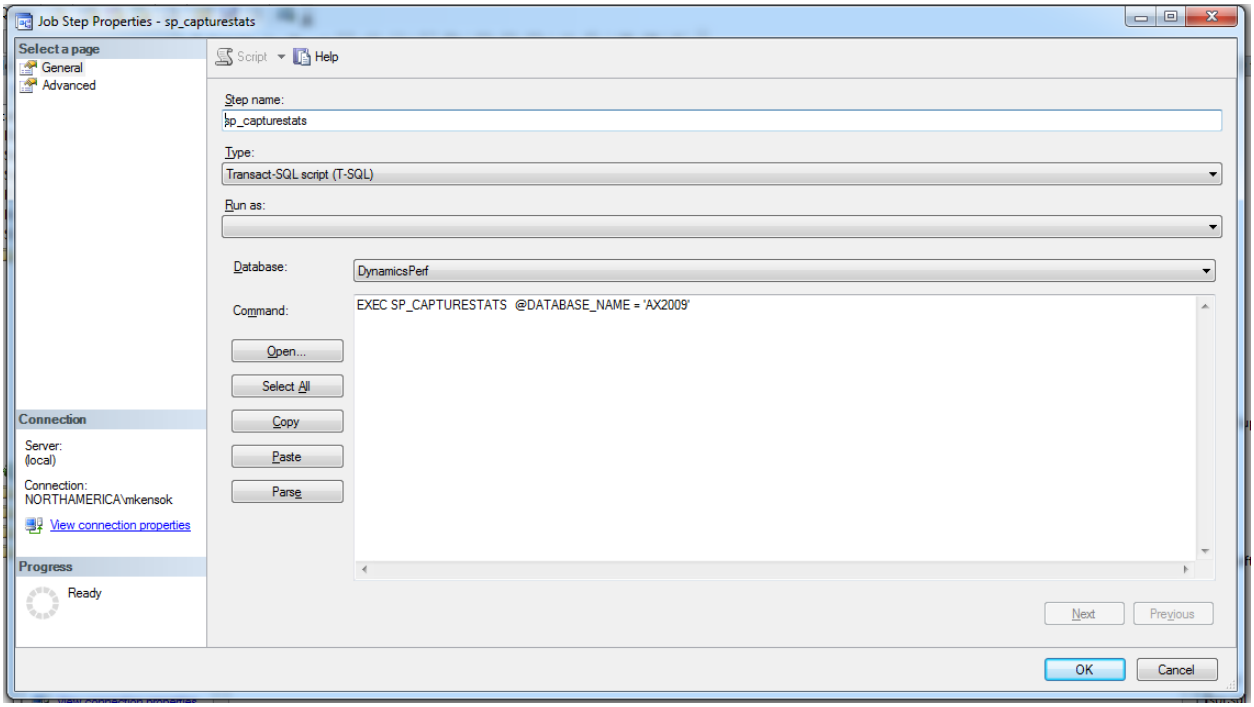
1. On the database server, open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_Capture_Stats** job



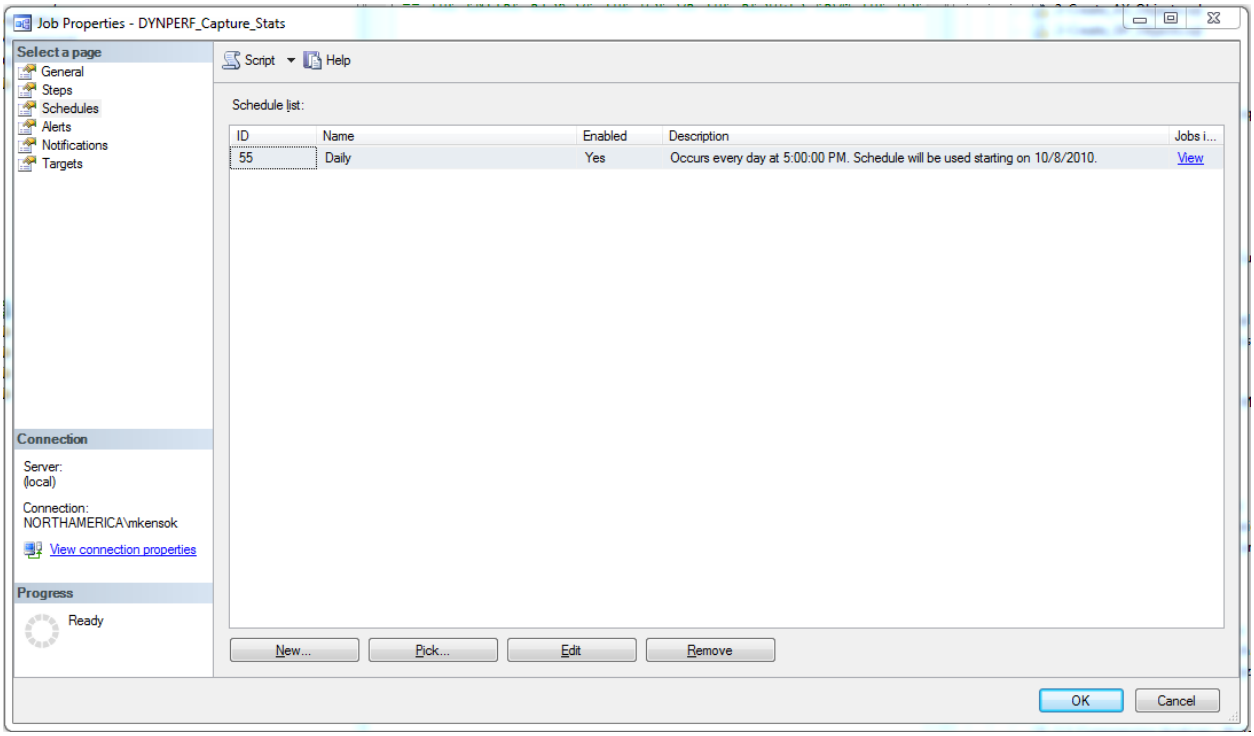
4. In the Select a page pane, select Steps



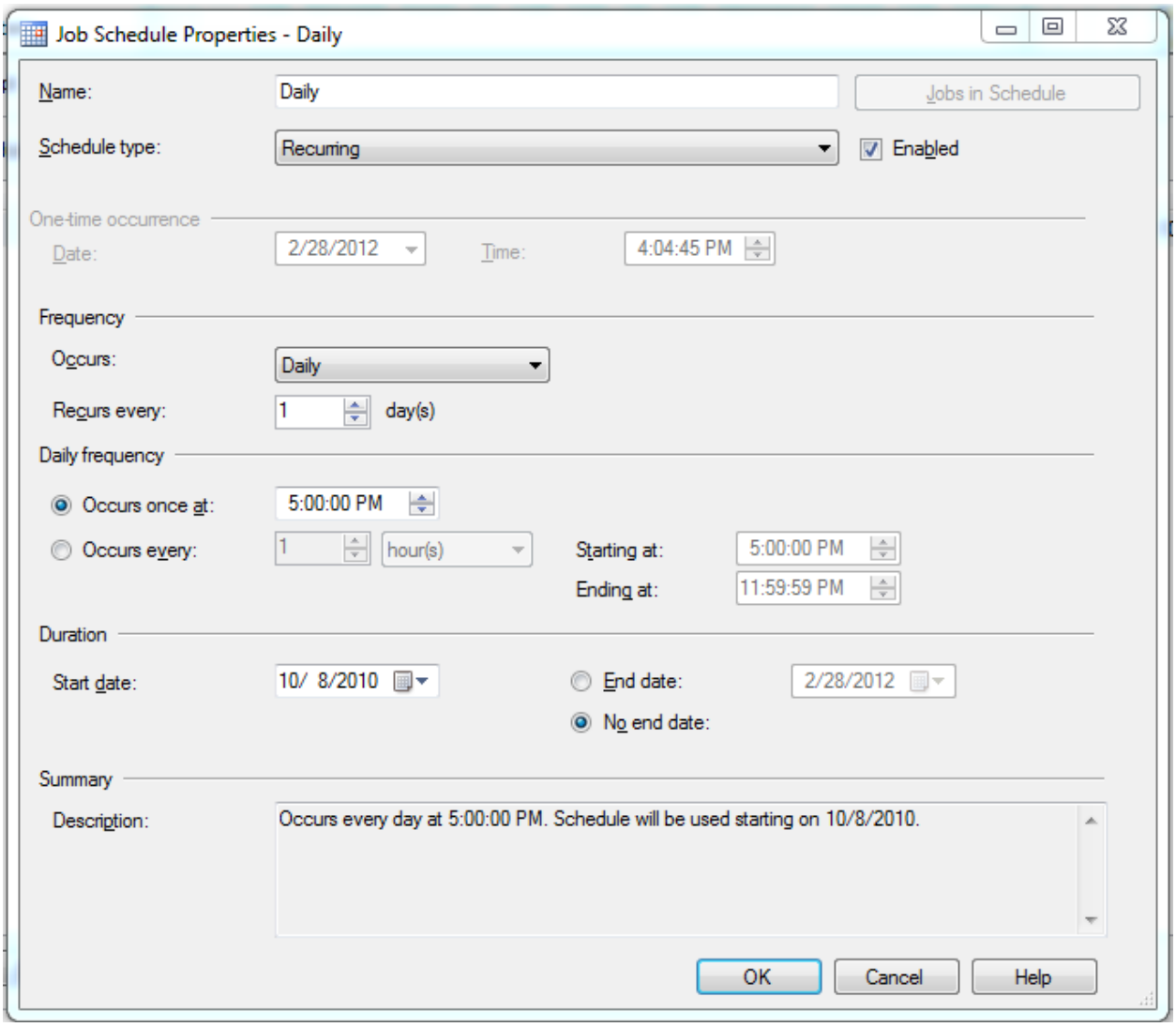
5. Double click Step 1 sp_capturestats to open it.



6. Change the @DATABASE_NAME to the name of the AX database name (Example: ‘AX2009’)
7. Select OK to close the window
8. In the Select a page pane select Schedules



9. By default, this task is enabled and will run every day at 5:00 PM. If you wish to change the time for which it runs daily:
    a. Double click the Daily schedule to open it



b. Make your changes in this window

c. Click OK to close the window

10. Click OK to close the **DYNPERF_Capture_Stats** job window

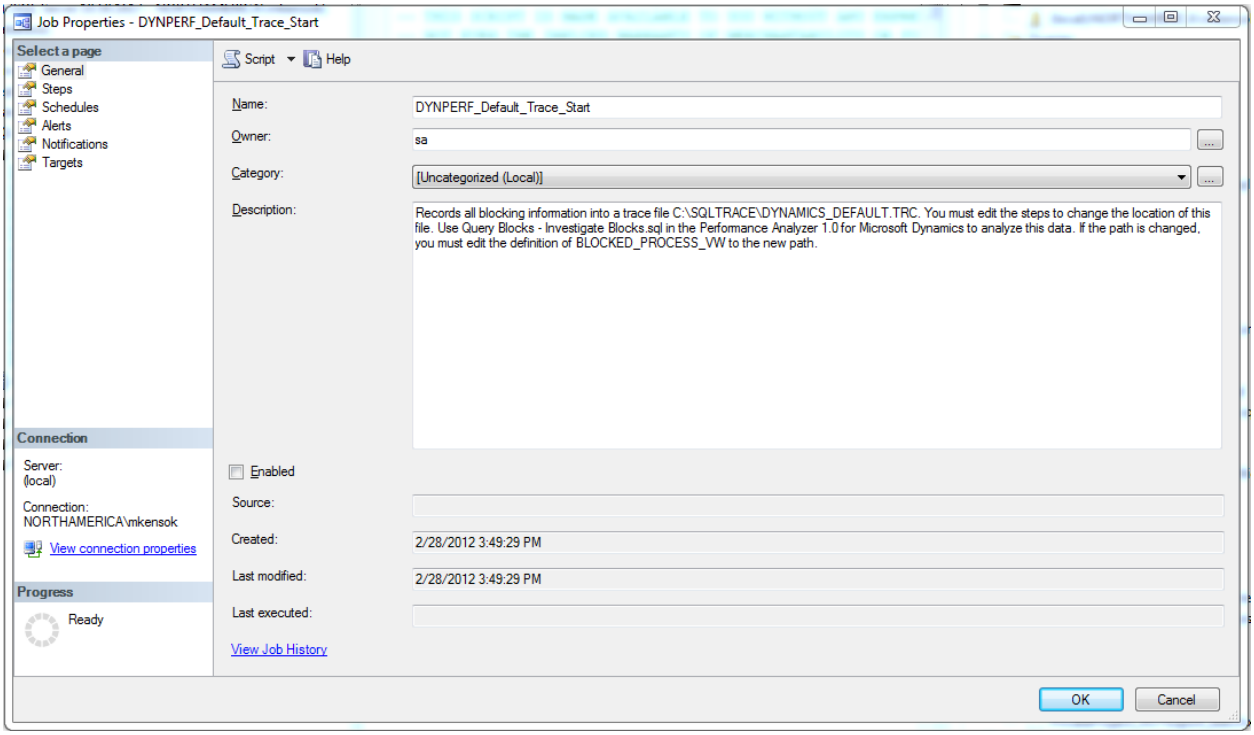
Configure and Schedule Database Blocking Capture

In order to collect database blocking events, it is necessary to configure and schedule the database blocking capture. In the following steps you will configure and schedule the capture of database blocking events. The events collected will be contained in SQL trace files.

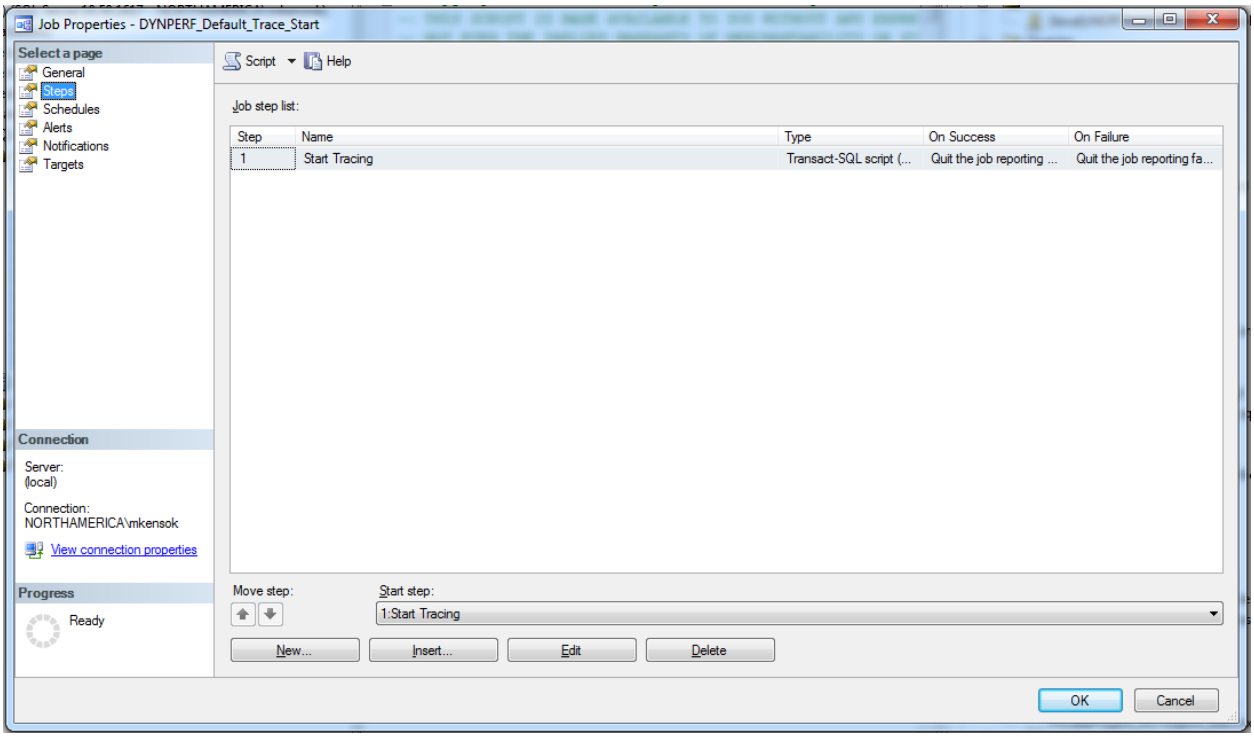
***NOTE**: This option should be used in all cases for tracking blocking events and is recommended option if you are tracking blocking over a long period of time.*

---

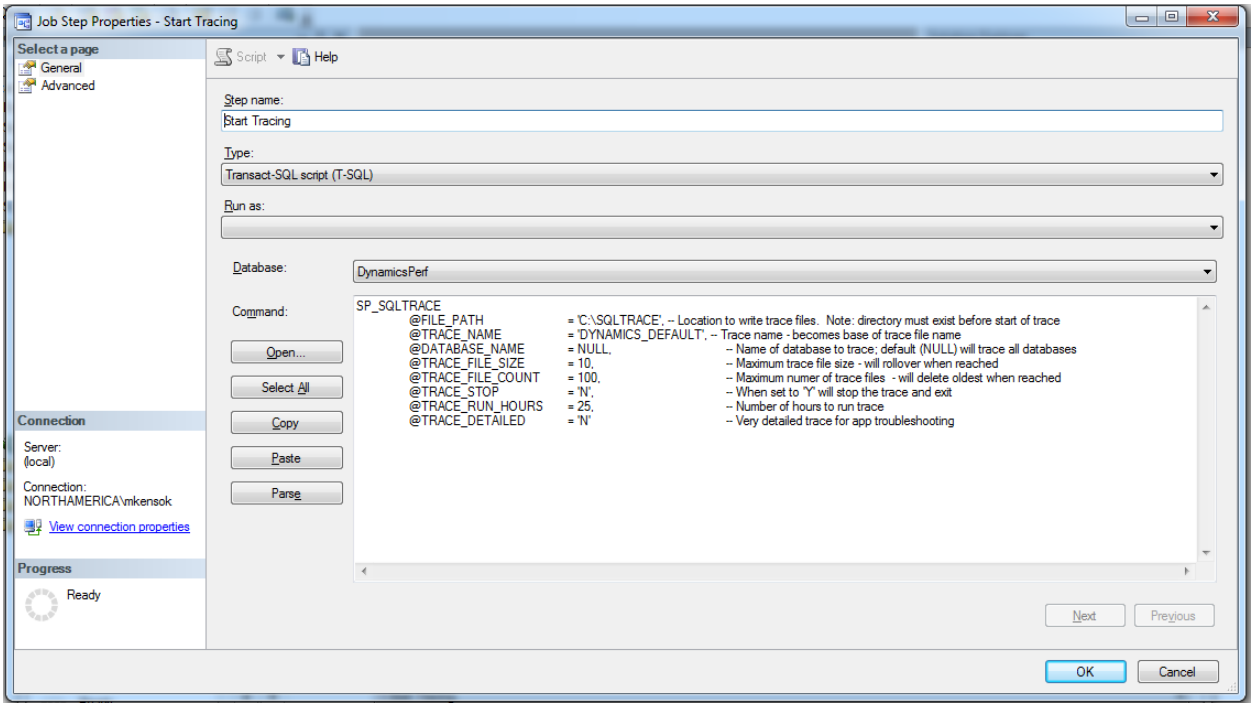
1. On the database server, open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_Default_Trace_Start** job



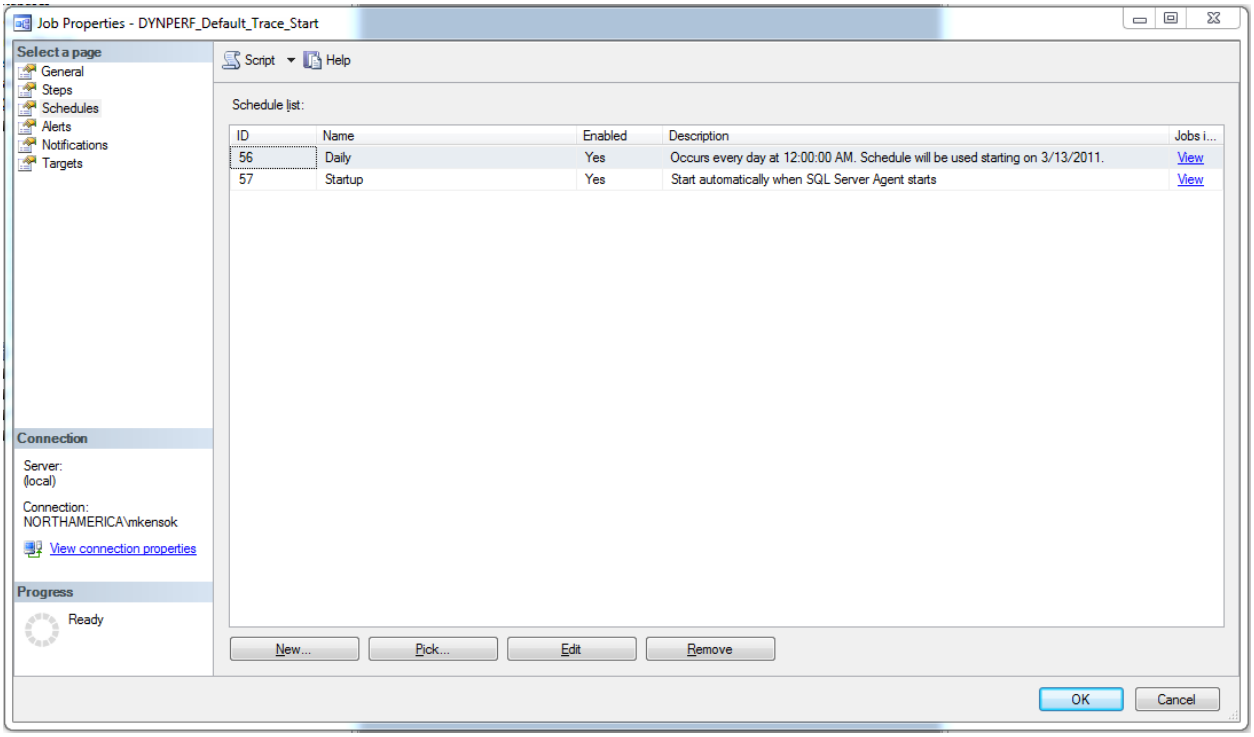
4. Check the Enable checkbox
5. In the Select a page pane, select Steps



6. Double click Step 1 Start Tracing to open it.



7. If the @FILE_PATH is the incorrect path to the \SQLTRACE folder that you created in the “Before you begin” steps, change it here
8. Click OK to close the window
9. In the Select a page pane select Schedules



10. Verify that each task is enabled and scheduled every day at 12:00 AM and that it starts automatically when SQL Server agents starts
11. Click OK to close the window

Enable Long Running Query Capture for AX

If using Dynamics AX, you can set thresholds which capture long running queries from AX source code. In the following steps you will configure the system to capture long running queries from AX source code. The data collected will be stored in the **DynamicsPerf** database.

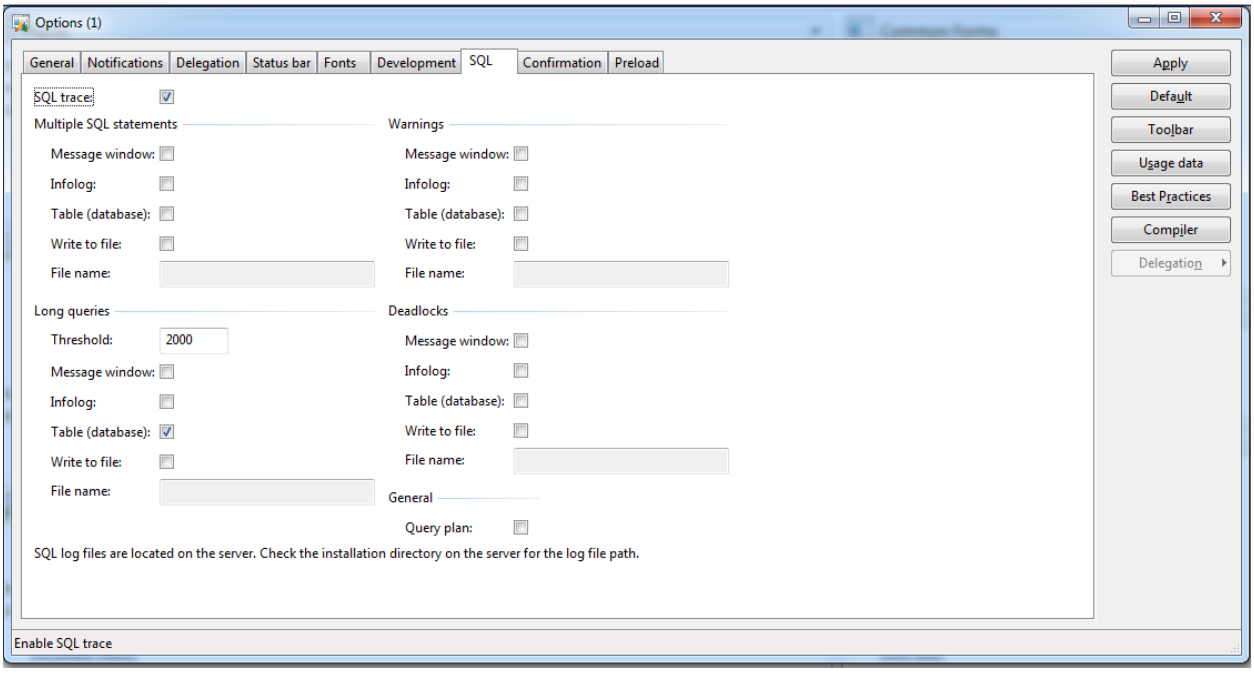
***NOTE**: This enables long duration tracing for all AX users by updating the USERINFO table and sets the long running query threshold to 5000ms. The 'Allow client tracing on Application Object Service instance' checkbox on the AOS Server Configuration Utility for each AOS Server must be marked before executing this stored procedure.*

---

1. Please ensure step 7 in the "Before you begin" section has been completed
2. On the database server, open SQL Server Management Studio (SSMS)
3. Click File>Open, Project/Solution
4. Browse to the location for where you extracted the DynamicsPerf1.15 for SQL2008+.zip
5. Select the *Performance Analyzer 1.15 for Microsoft Dynamics.ssmssln* file
6. In Solution Explorer, open the *DynamicsAX Client Tracing.sql* script
7. Change <dbname> to the name of your AX database
8. Execute only the part listed below from the script against the DynamicsPerf database to enable client tracing for all AX users

```
/******************  Set AX Client tracing *************/
/* NOTE: must enable AX client tracing on the AOS servers */
USE DynamicsPerf

GO

EXEC SET_AX_SQLTRACE
  @DATABASE_NAME = '<dbname>',
  @QUERY_TIME_LIMIT = 5000
```

9. To view the results of a user within AX:
    a. Open Dynamics AX
    b. Go to Tools>Options
    c. Select the SQL tab
    d. Notice the SQL checkbox is marked, the long query threshold is 5000, and the Table (database) checkbox is enabled



Configure and Schedule AOT Metadata Capture

To be able to review the table and index property settings from within the AOT for AX tables, you will configure and schedule the AOT metadata capture. In the following steps you will configure and schedule the AOT metadata capture. The data collected will be stored in the **DynamicsPerf** database.

1. Launch an AX client
2. Open the Application Object Tree (AOT) in Dynamics AX
3. Click the Import icon
4. Browse to the *PrivateProject_AOTExport_Batch.xpo* file found where you extracted the files from in step 1 of the “Before you begin” section
5. Click OK to import
6. Open Basic>Inquiries>Batch jobs
7. The Batch jobs window opens
8. Create new batch job with Job description name of AOTExport
9. Click Save
10. Click View tasks button
11. Create new Batch task
    a. Task description = AOTExport
    b. Company = DAT
    c. Class name = AOTExport
12. Click Save
13. Close Batch tasks window

14. On the Batch jobs window, select the Recurrence button
    a. Select Recurring pattern of Days and Every weekday
    b. Click OK
15. Click OK to close Recurrence window
16. On the Batch jobs window, select the AOTExport batch job
17. Select Functions>Change status
18. Change status to *Waiting*

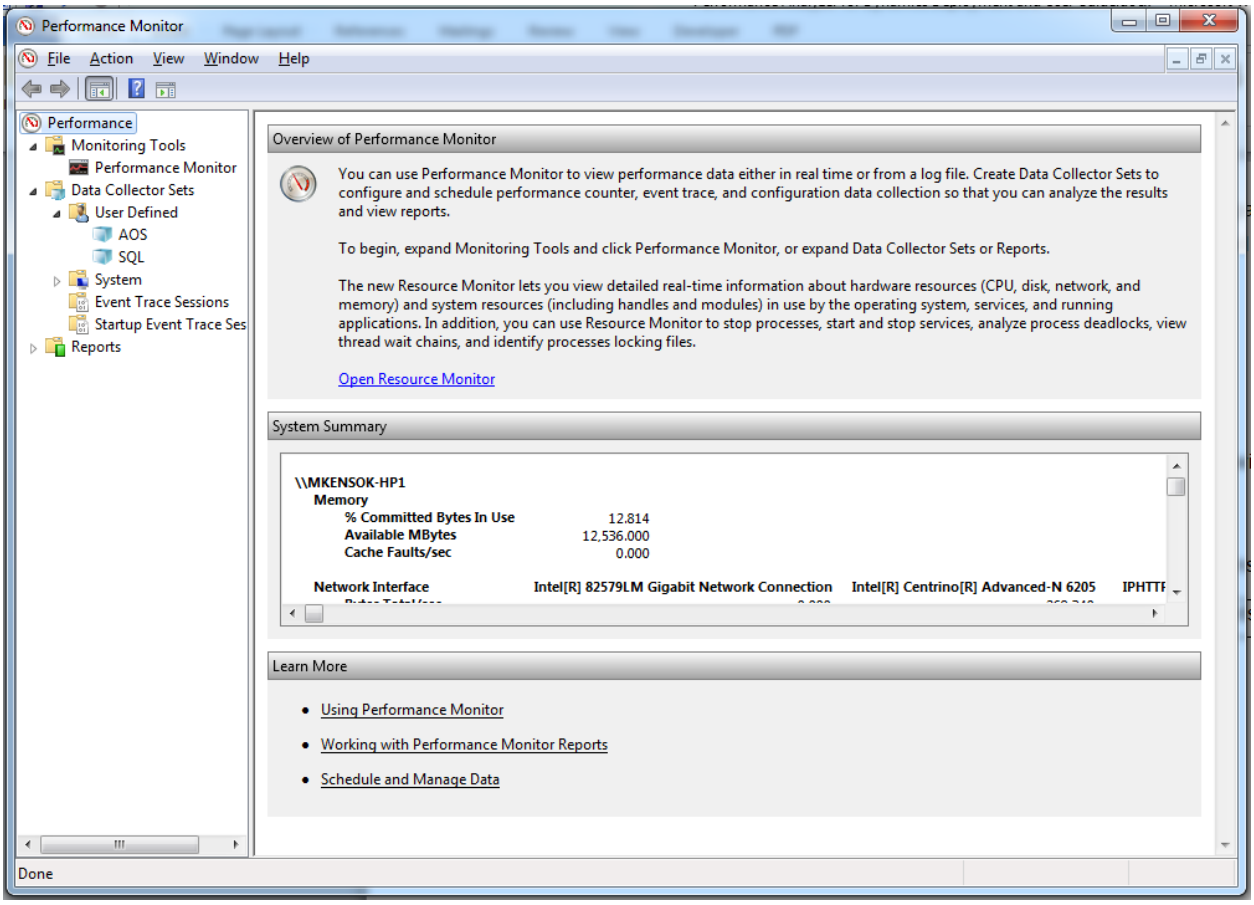
Configure and Schedule Performance Counter Logging on Database Server

To log valuable information about your database server such as disk, cpu, memory, etc., it is important to configure and schedule the performance counter logging. In the following steps, you will configure the database server for performance counter logging. This information will be logged to performance counter log files.

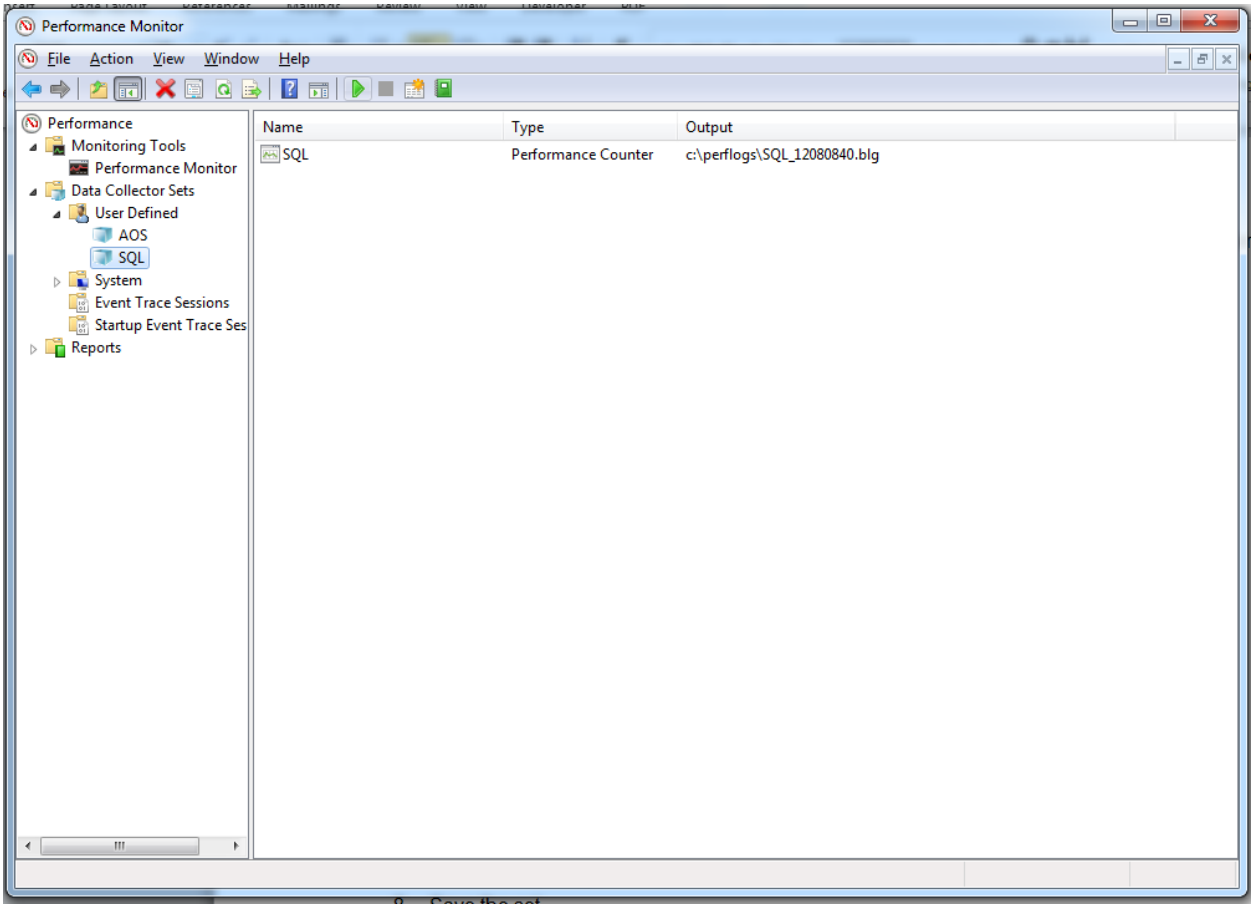
1. If you are using a Named instance of SQL Server follow these steps, otherwise go to step 2:
    a. Browse to the Server2008_SQL_Named_Instance.xml file
    b. Open file in Notepad
    c. Replace InstanceName with the actual name of your named SQL instance

```
<Counter>\MSSQL$InstanceName:Access Methods\Forwarded Records/sec</Counter>
```

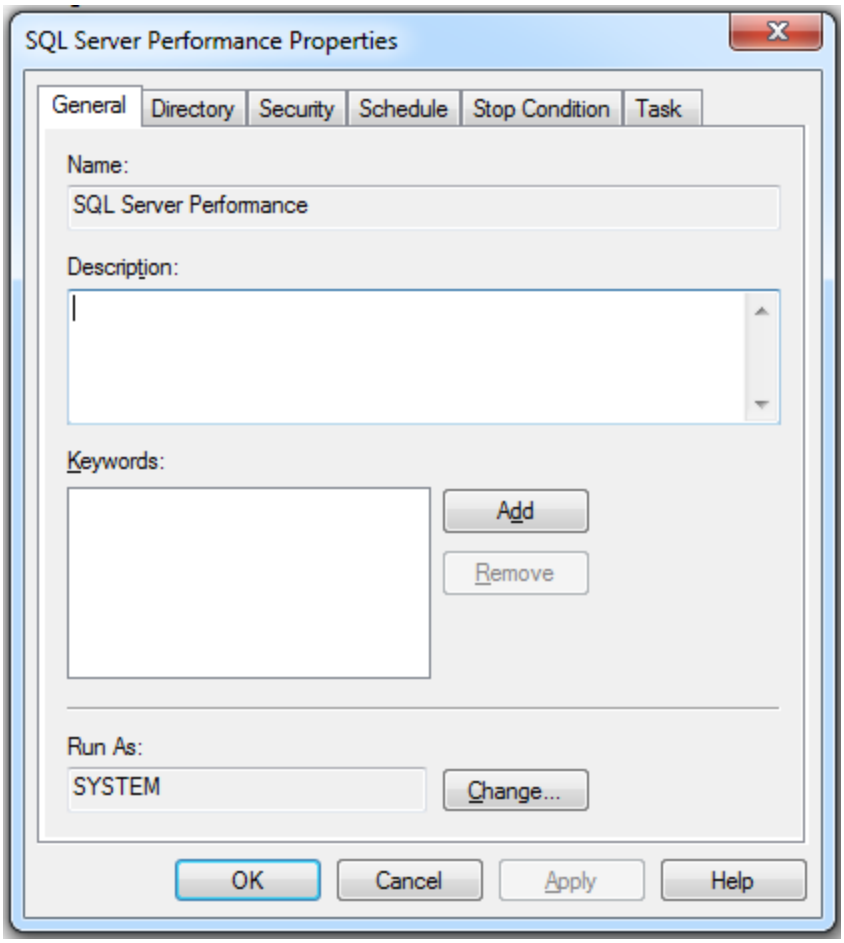
    d. Save the file
2. Start > Run > Perfmon
3. Expand Data Collector Sets



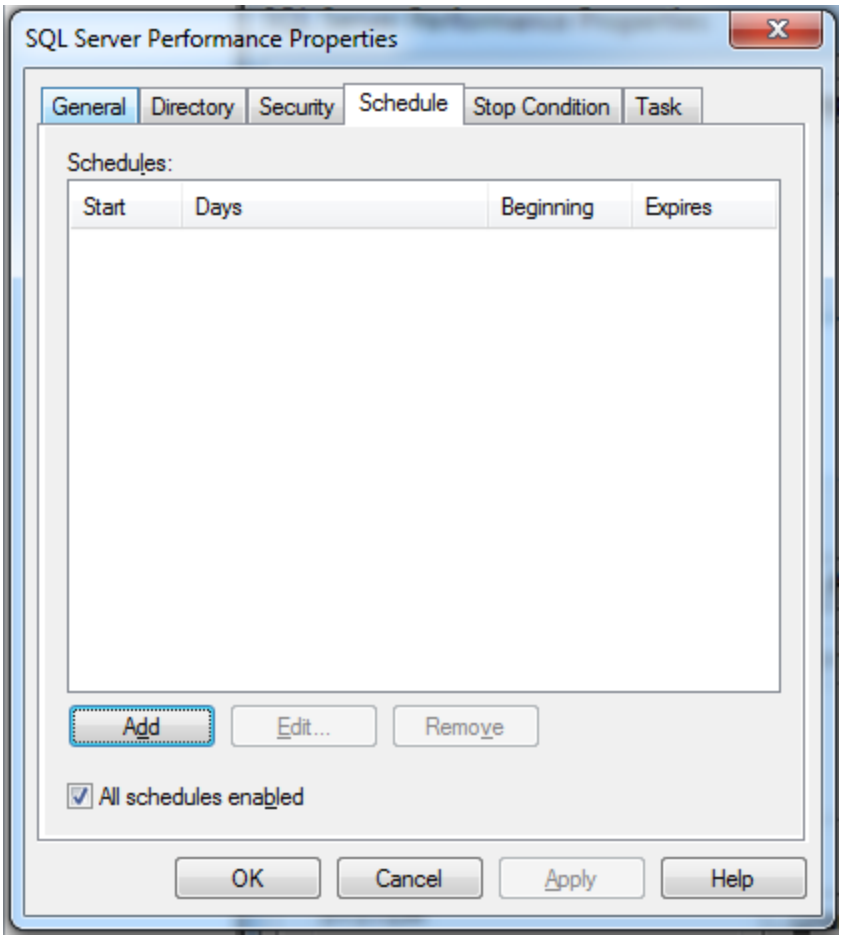
4. Right click User Defined and select New > Data Collector Set.
5. Name it “SQL Server Performance”



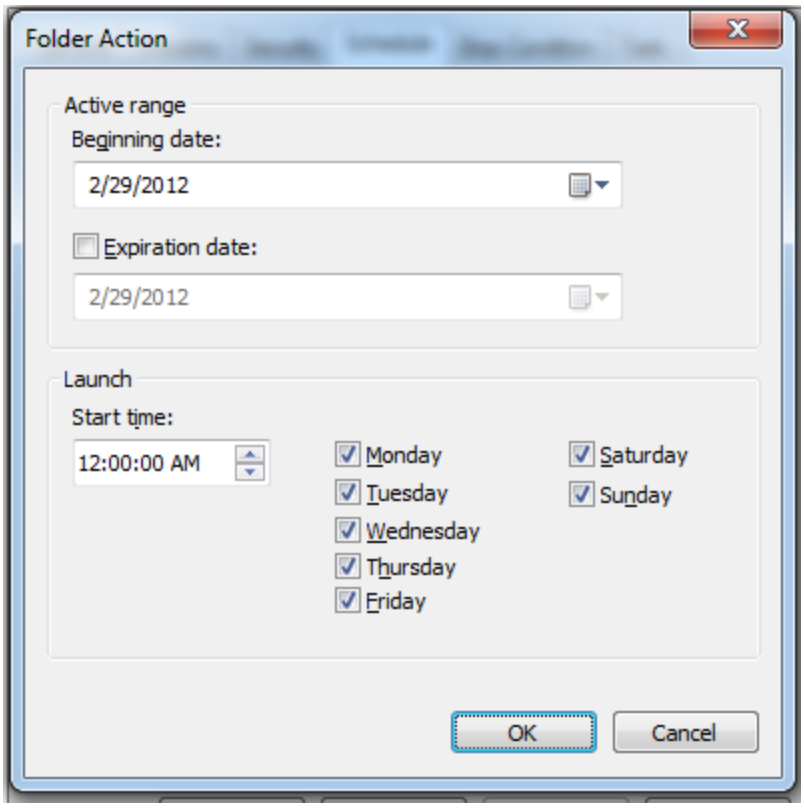
6. Select the “Create from a template” option.
7. Click Next
8. Select “System Performance” and click Browse to browse to the Server2008_SQL_Default_Instance .xml file if you are using a Default SQL Server or the Server2008_SQL_Named_Instance.xml file if you are using a Named SQL Server
9. Click Finish
10. Right-click on the SQL Server Performance data collector set and click Properties



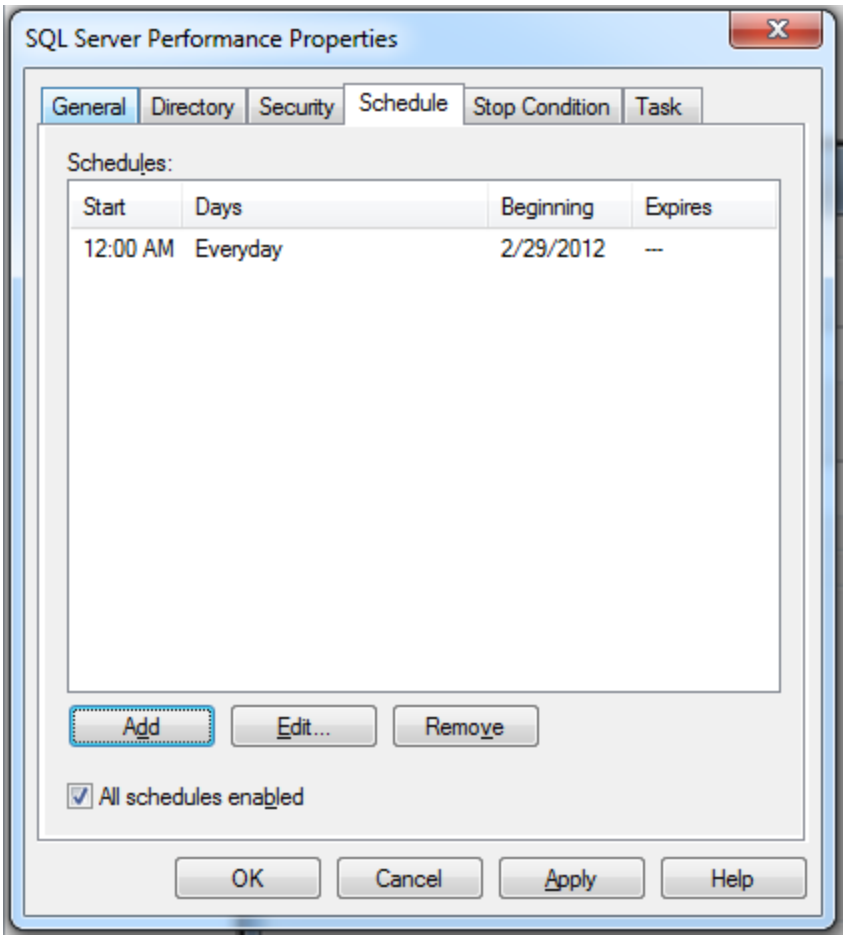
11. NOTE: By default, the performance counter logs will log to C:\perflogs. If you wish to change this path follow these steps. Otherwise, go to step 12:
    a. Select the Directory tab
    b. Browse to a local root directory to where you want to store the performance counter logs.
12. Select the Schedule tab



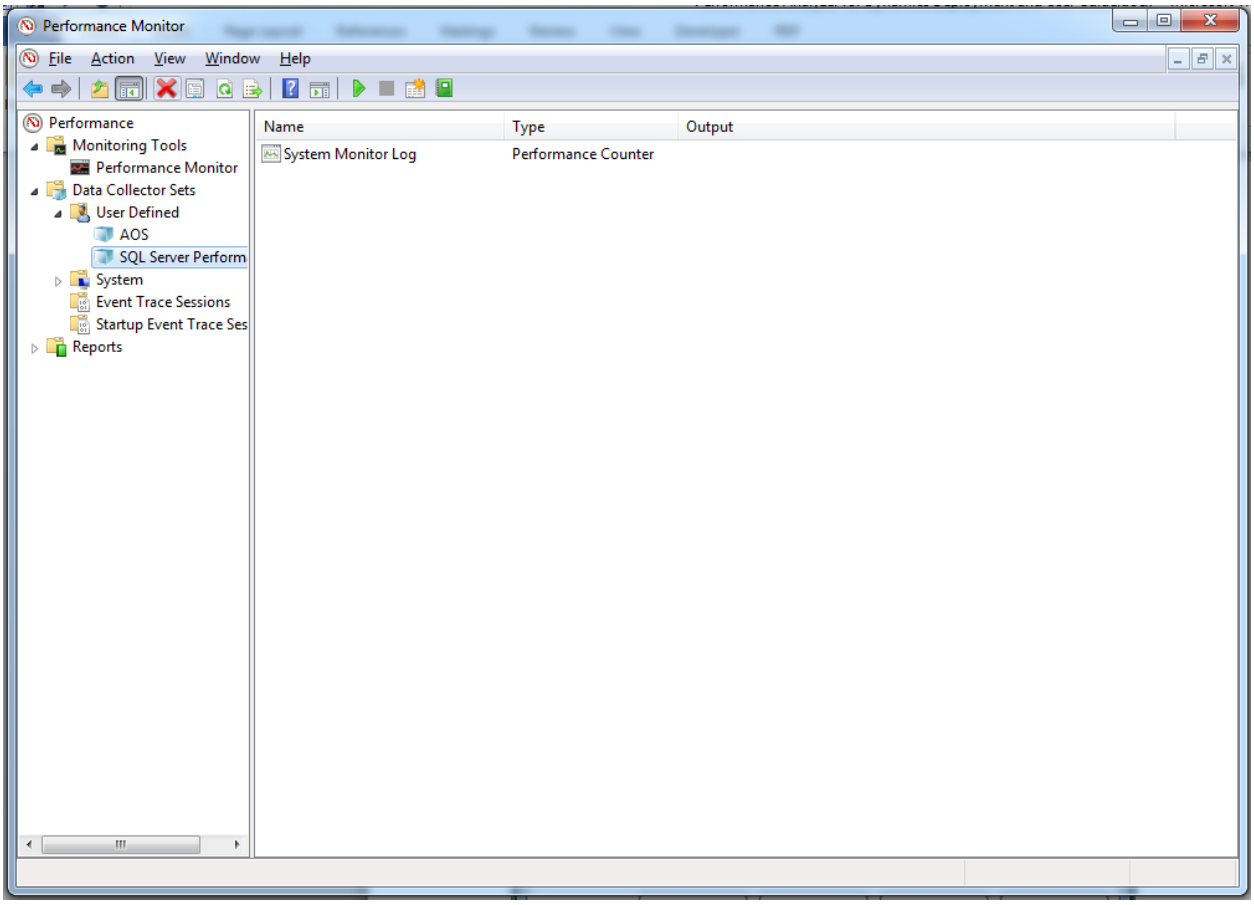
13. Click the Add button to create new schedule



14. Select beginning date as of today and leave the rest as default so it will run continuously without an end date.
15. Click OK to close Folder action window
16. Ensure All schedules enabled checkbox is checked



17. Click OK to close window



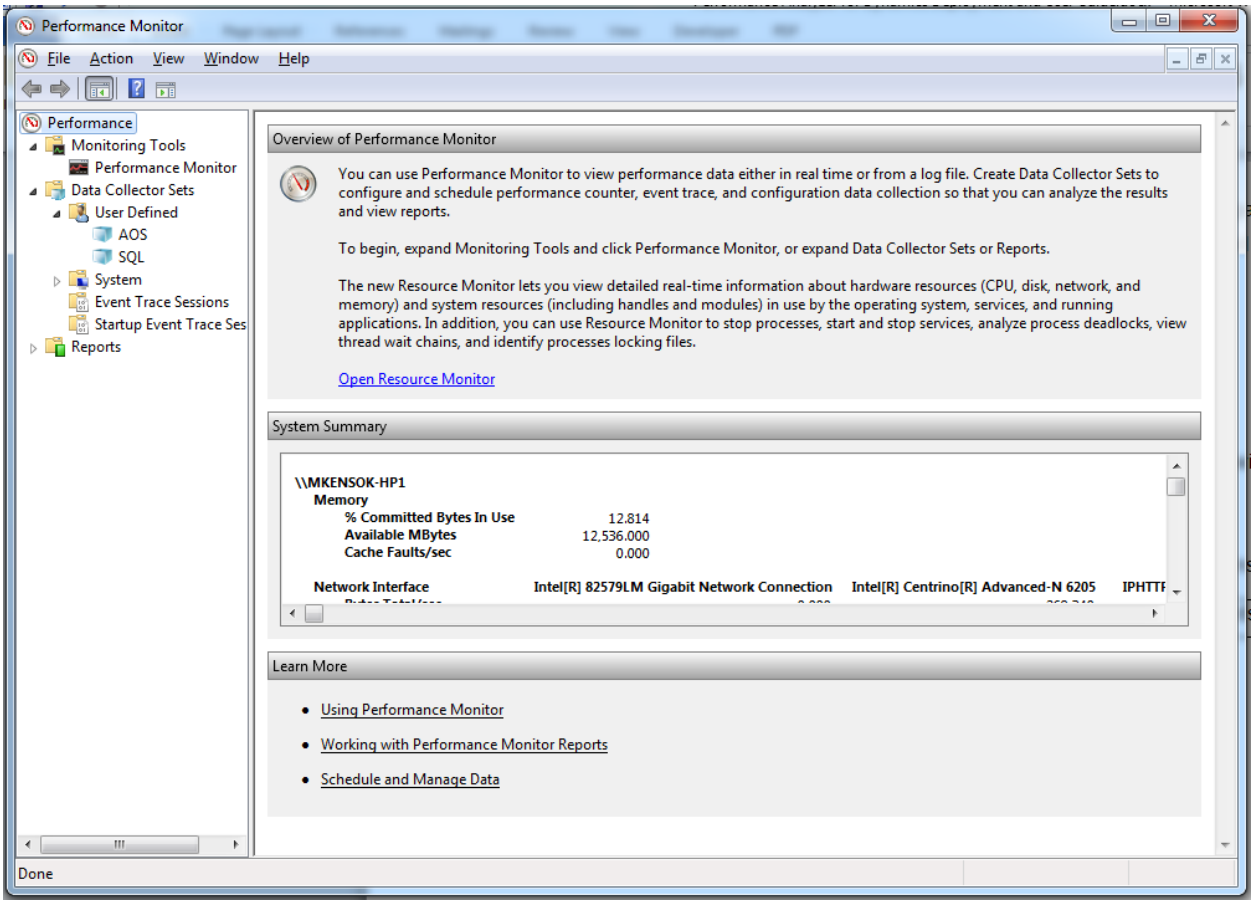
Configure and Schedule Performance Counter Logging on AOS Server(s)

To log valuable information about your AOS servers such as cpu, memory, etc., it is important to configure and schedule the performance counter logging. In the following steps, you will configure the database server for performance counter logging. This information will be logged to performance counter log files.

***NOTE**: You will repeat this step on every AOS Server*

---

1. Start > Run > Perfmon
2. Expand Data Collector Sets
3. Expand Data Collector Sets



4. Right click User Defined and select New > Data Collector Set.
5. Name it “AOS Server Performance”
6. Select the “Create from a template” option.
7. Click Next
8. Select “System Performance” and click Browse to browse to the Server2008_AOS.xml file
9. Click Finish
10. Right-click on the AOS Server Performance data collector set and click Properties
11. NOTE: By default, the performance counter logs will log to C:\perflogs. If you wish to change this path follow these steps. Otherwise, go to step 12:
    a. Select the Directory tab
    b. Browse to a local root directory to where you want to store the performance counter logs.
12. Select the Schedule tab
13. Click the Add button to create new schedule
14. Select beginning date as of today and leave the rest as default so it will run continuously without an end date.
15. Click OK to close Folder action window
16. Ensure All schedules enabled checkbox is checked

17. Click OK to close window

Deployment Verification Checklist

The following is a list of items that should be checked periodically to ensure Performance Analyzer is running and collecting the data.

#	Verification	Where?
1	Is the performance data being collected on a regular schedule?	Check the history of the DYNPERF_Capture_Stats job to ensure it is running every day
2	Is the performance data being purged on a regular schedule?	Check the history of the DYNPER_Stats_Purge job to ensure it is running every day
3	Is the AOS configuration and event logs being collected on a regular schedule?	Check the history of the AOSANALYSIS job to ensure it is running on regular schedule.
4	Is the AOT metadata being collected on a regular schedule?	Check the AOTExport batch job within AX to ensure it is running on regular schedule
5	Is the Database blocking being collected on a regular schedule?	Check the history of the DYNPERF_Default_Trace_Start job to ensure it is running every day
6	Is AX client tracing enabled on every AOS server?	Check if the “Allow client tracing on Application Object Server instance” is enabled on every AOS server
7	Is AX tracing enabled for every user?	Run the DynamicsAX Client Tracing.sql (enable portion of the script only) periodically to ensure set for all users
8	Are the AX tracing tables being periodically purged?	Check the history of the DYNPERF_Purge_SYSTRACETABLESQL job to ensure it is running on a regular schedule
9	Are the performance counter logs running on the database server?	Check if the templates that you imported are running in the Performance Monitor on the database server
10	Are the performance counter logs running on the AOS servers?	Check if the templates that you imported are running in the Performance Monitor on every AOS server
11	If you are using the DYNPERF_Optional_Polling_for_Blocking job is the data being periodically purged?	Check the history of the DYNPERF_Optional_Polling_for_Blocking job to ensure it is running periodically

PERFORMANCE ANALYZER MAINTENANCE

There are several tasks that need to be scheduled to properly maintain the **DynamicsPerf** database such as purging unneeded data.

Maintenance Checklist

The following is a summarized checklist of the steps to maintain Performance Analyzer. See the steps below for detailed information.

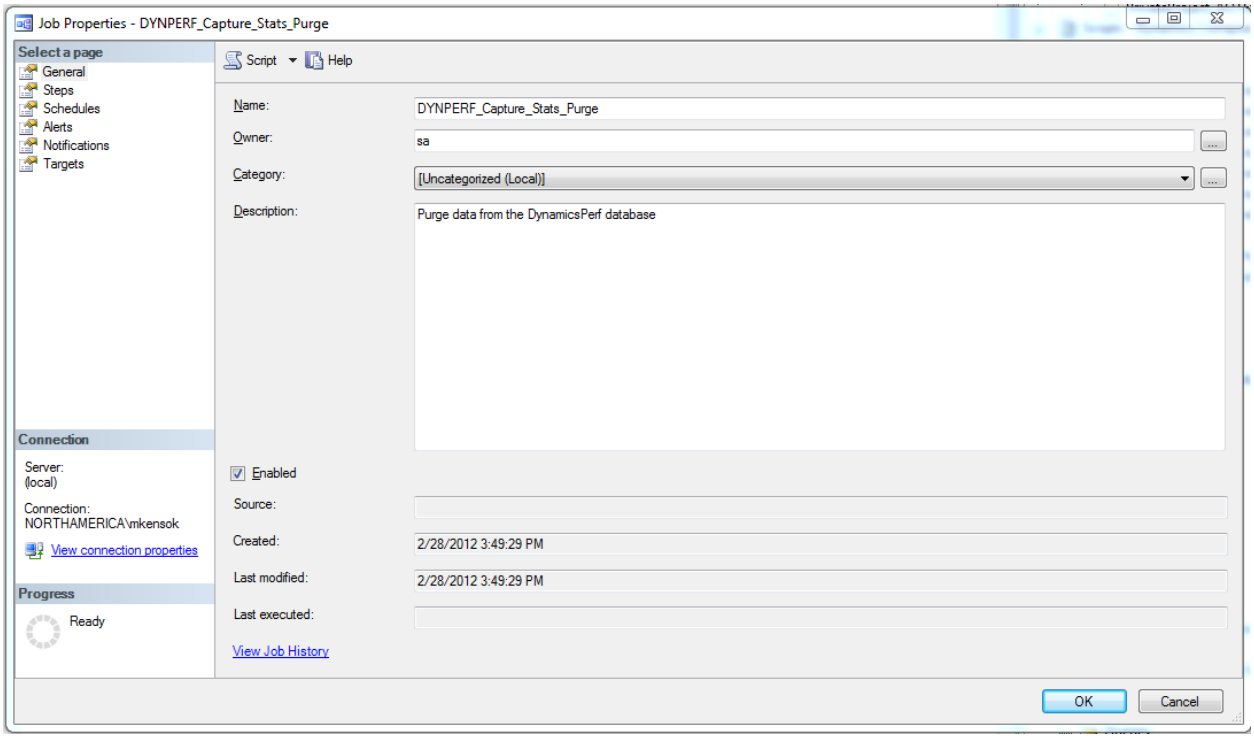
Step #	Task
1	Configure and Schedule Performance Data Purge
2	Configure and Schedule AX Long Running Query Collection Purge
3	Configure and Schedule Blocking Data Polling Purge

Configure and Schedule Performance Data Purge

In the following steps, you will configure and schedule the process for purging unneeded data from the **DynamicsPerf** database. This is important for several reasons but mainly to rid old performance data and maintain optimal database size. To do this you will use the **DYNPERF_Capture_Stats_Purge** job.

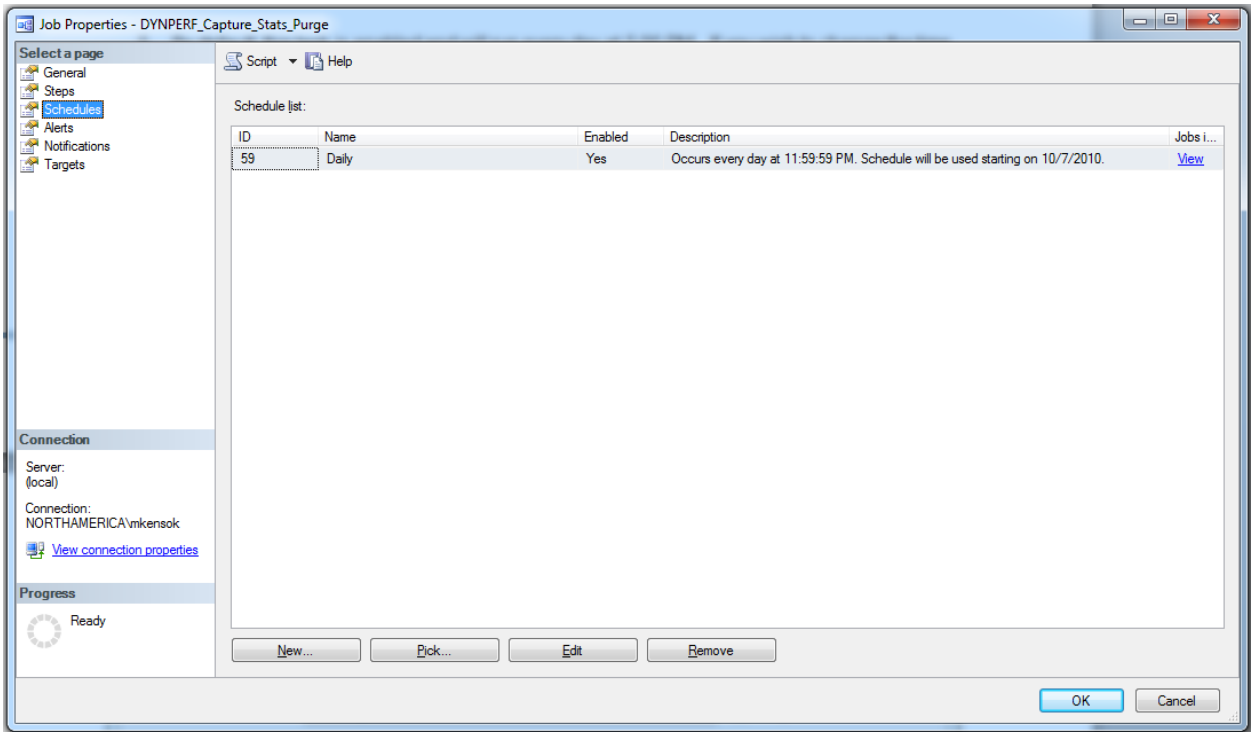
This is a job designed to remove any DMV performance data captured that is older than 14 days. A schedule has been added to this job so that it runs daily to prevent the **DynamicsPerf** database from consuming all disk space.

1. Open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_Capture_Stats_Purge** job

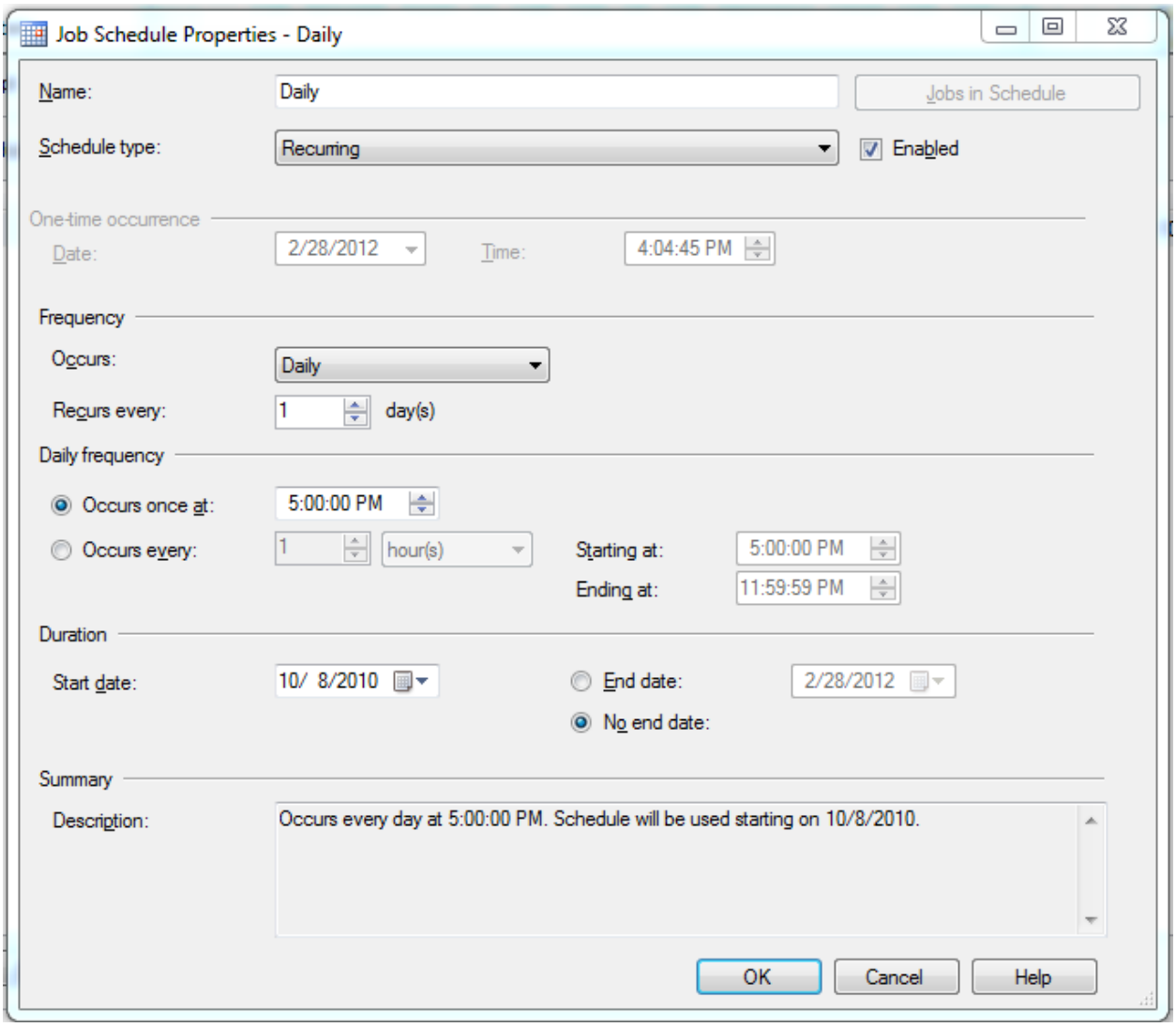


***NOTE**: By default this job will purge data from the DynamicsPerf database that are 14 days and older.*

4. In the Select a page pane select Schedules



5. By default, this task is enabled and will run every day at 11:59:59 PM. If you wish to change the time for which it runs daily:
    a. Double click the Daily schedule to open it



b. Make your changes in this window

c. Click OK to close the window

6. Click OK to close the **DYNPERF_Capture_Stats_Purge** job window

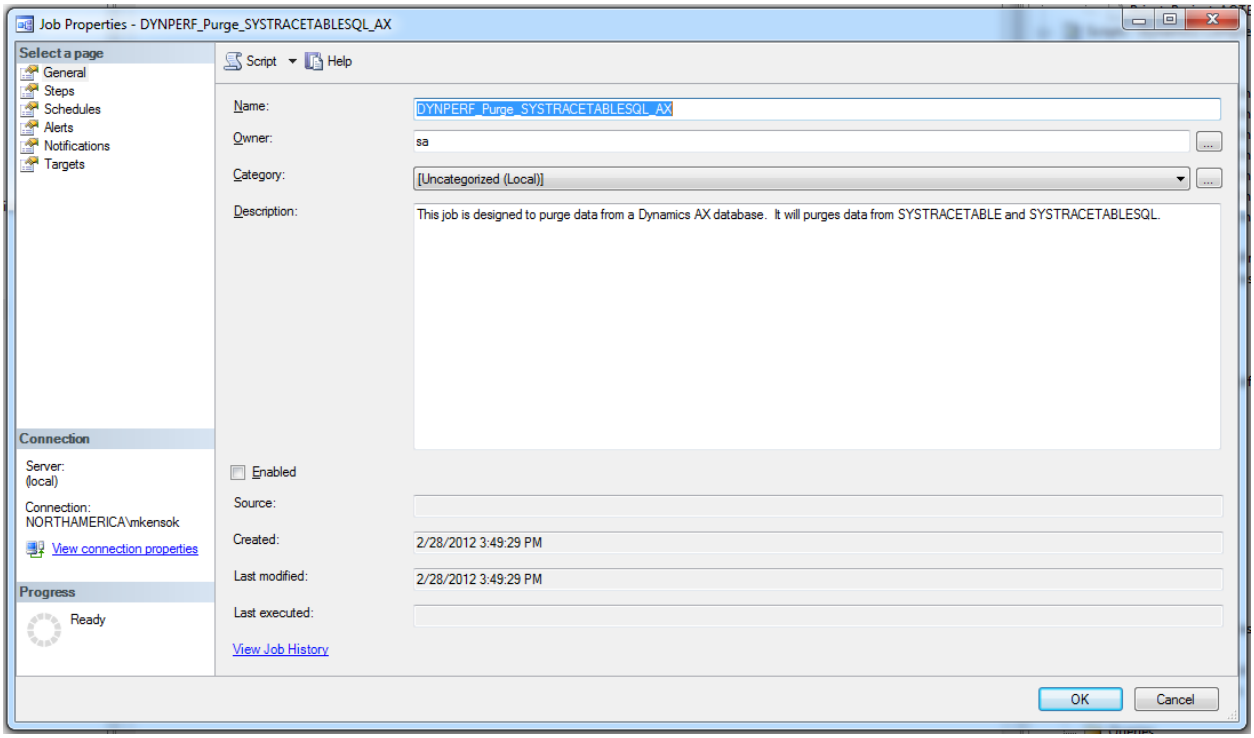
Configure and Schedule AX Long Running Query Collection Purge

In the following steps, you will configure and schedule the process for purging the contents of the tracing tables within the Dynamics AX database that are used to capture long running query traces from Dynamics AX. After a period of time, this data is no longer useful so it is good practice to schedule this deletion to maintain optimal Dynamics AX database size.

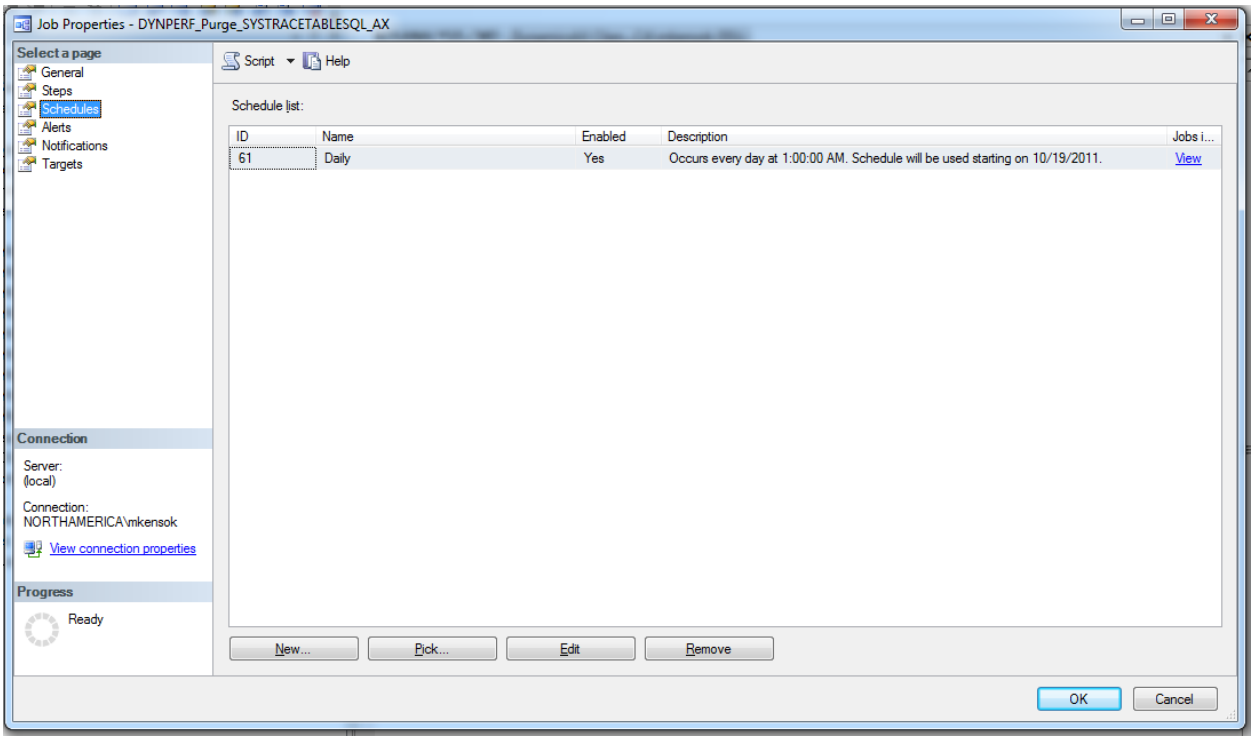
NOTE: *The overhead of enabling this is directly proportional to the number of events that exceed the threshold since a record is written to the SYSTRACETABLESQL table each time that happens. When the duration threshold is set to a reasonably high value (1000ms or greater), the overhead is very small and should not be noticeable to end users. If you're interested in how much logging is happening, just refer to the SYSTRACETABLESQL table. A timestamp exists on each row indicating when it was logged.*

These options can be programmatically turned on or off at any time via a stored procedure in the Health Check database. This helps to avoid the tedious work of managing the settings via the user options screens within the application.

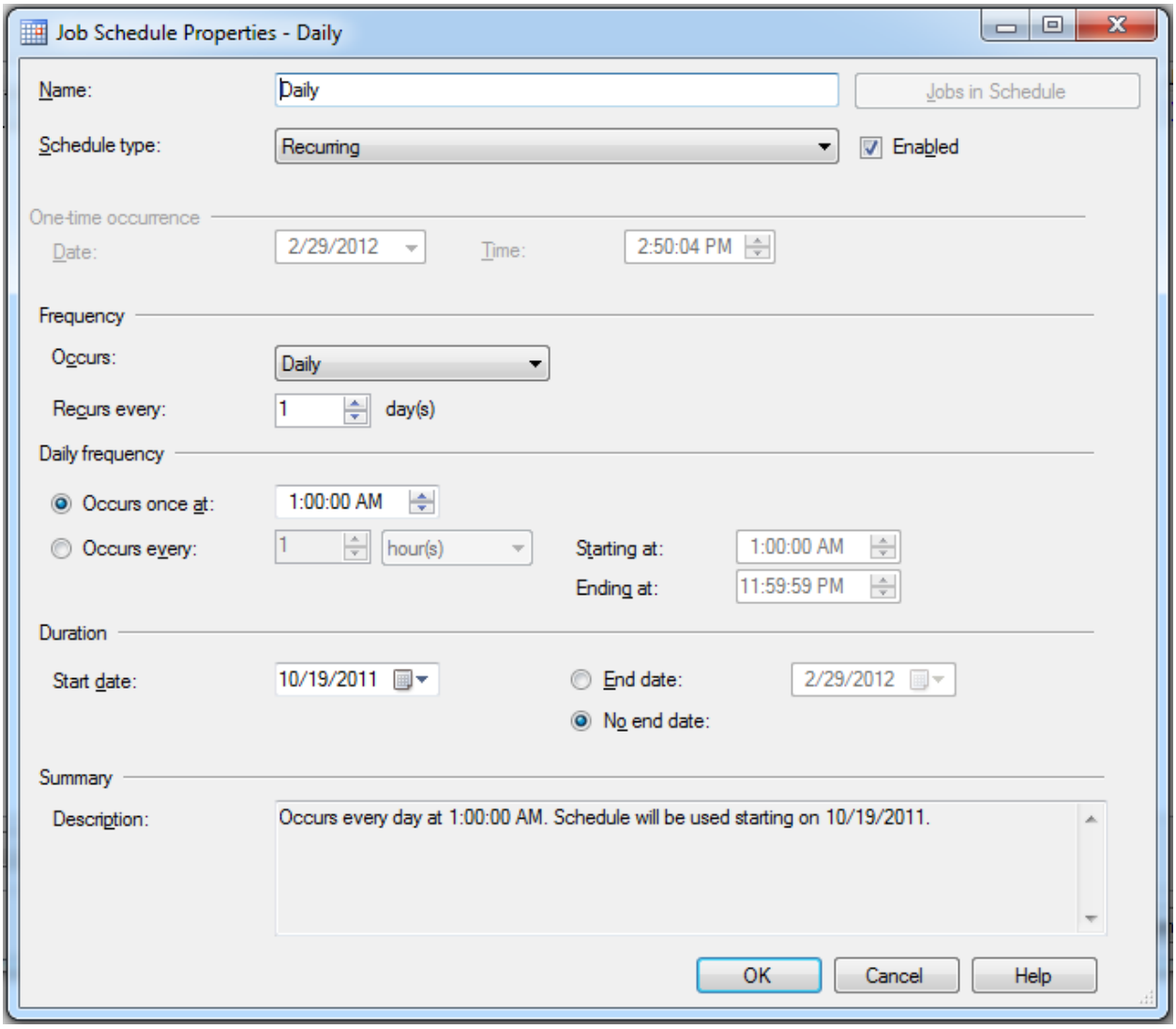
1. Open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_Purge_SYSTRACETABLESQL_AX** job



4. Check the Enabled checkbox
5. In the Select a page pane select Schedules



6. Double click the schedule task

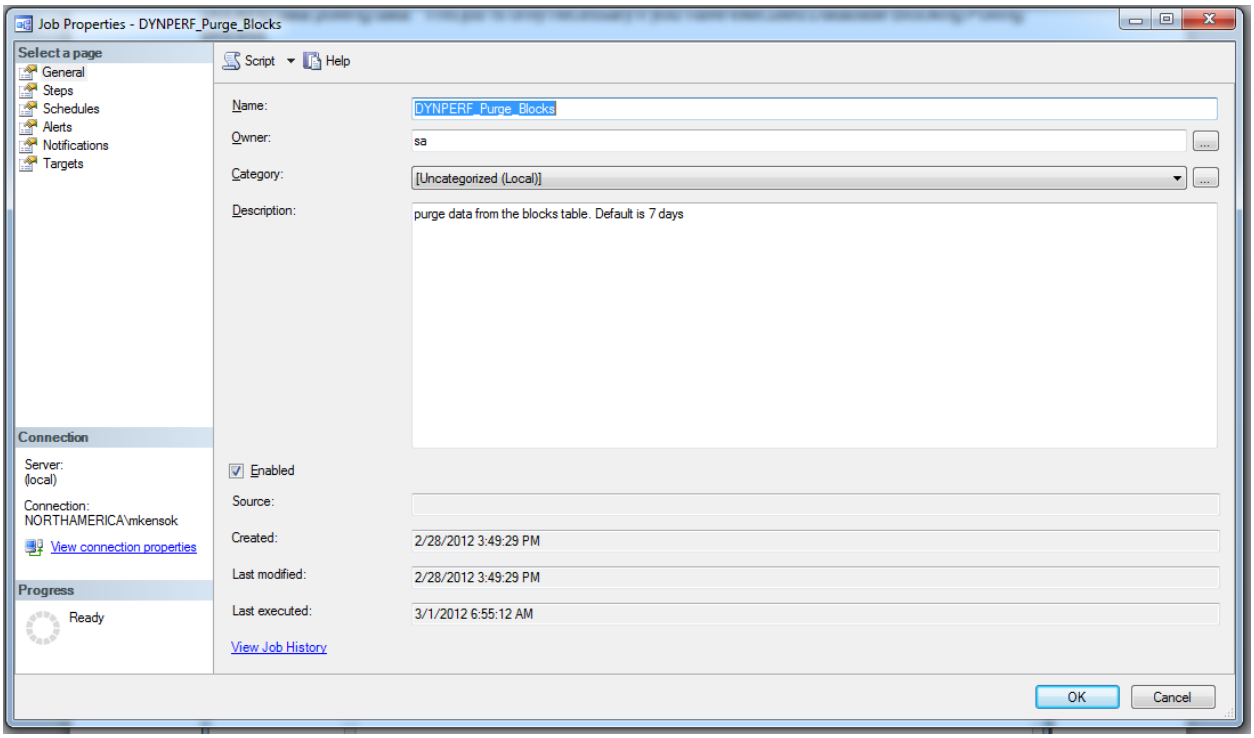


7. Change the schedule to how often you want to purge this data. A monthly recurring schedule may be sufficient to begin with.
8. Click OK to close the window
9. Click OK to close the job properties window

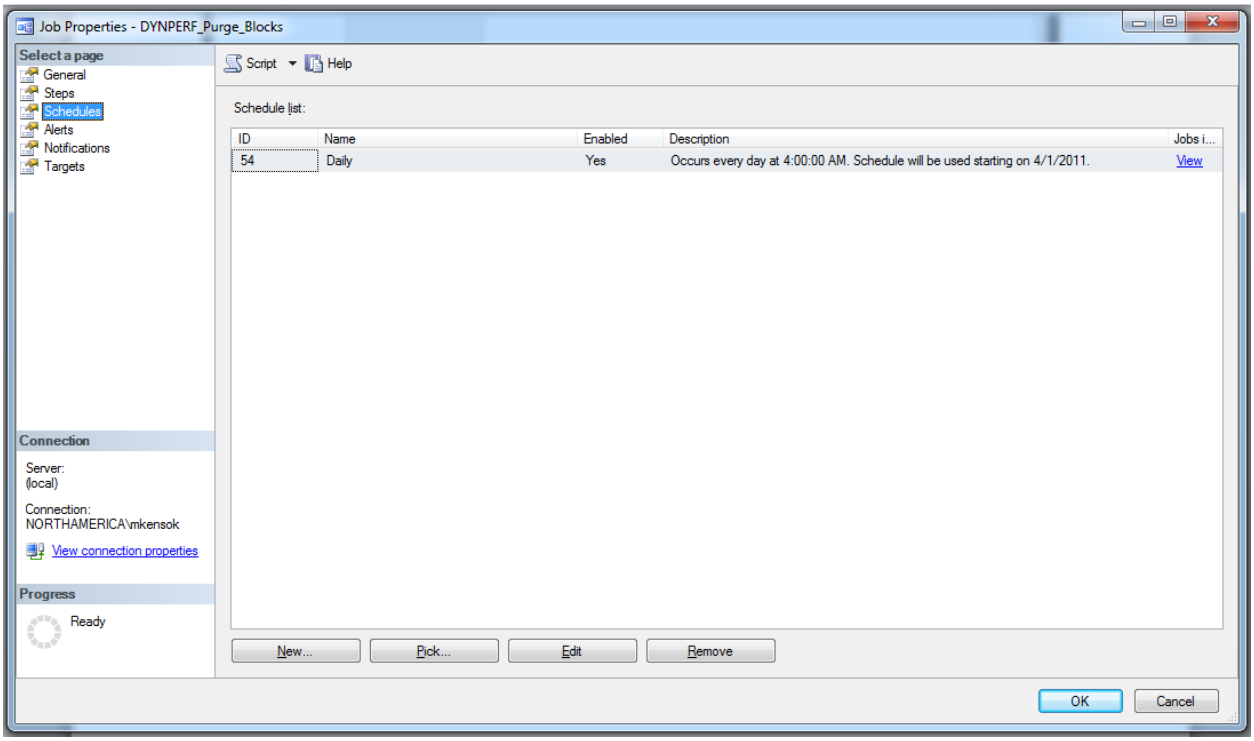
Configure and Schedule Blocking Data Polling Purge

In the following steps, you will configure and schedule the process for purging the tables which hold the blocking data polling data. You will use the **DYNPERF_Purge_Blocks** job. This is a job designed to remove any blocking data that was captured using the **DYNPERF_Optional_Polling_for_Blocking** job.

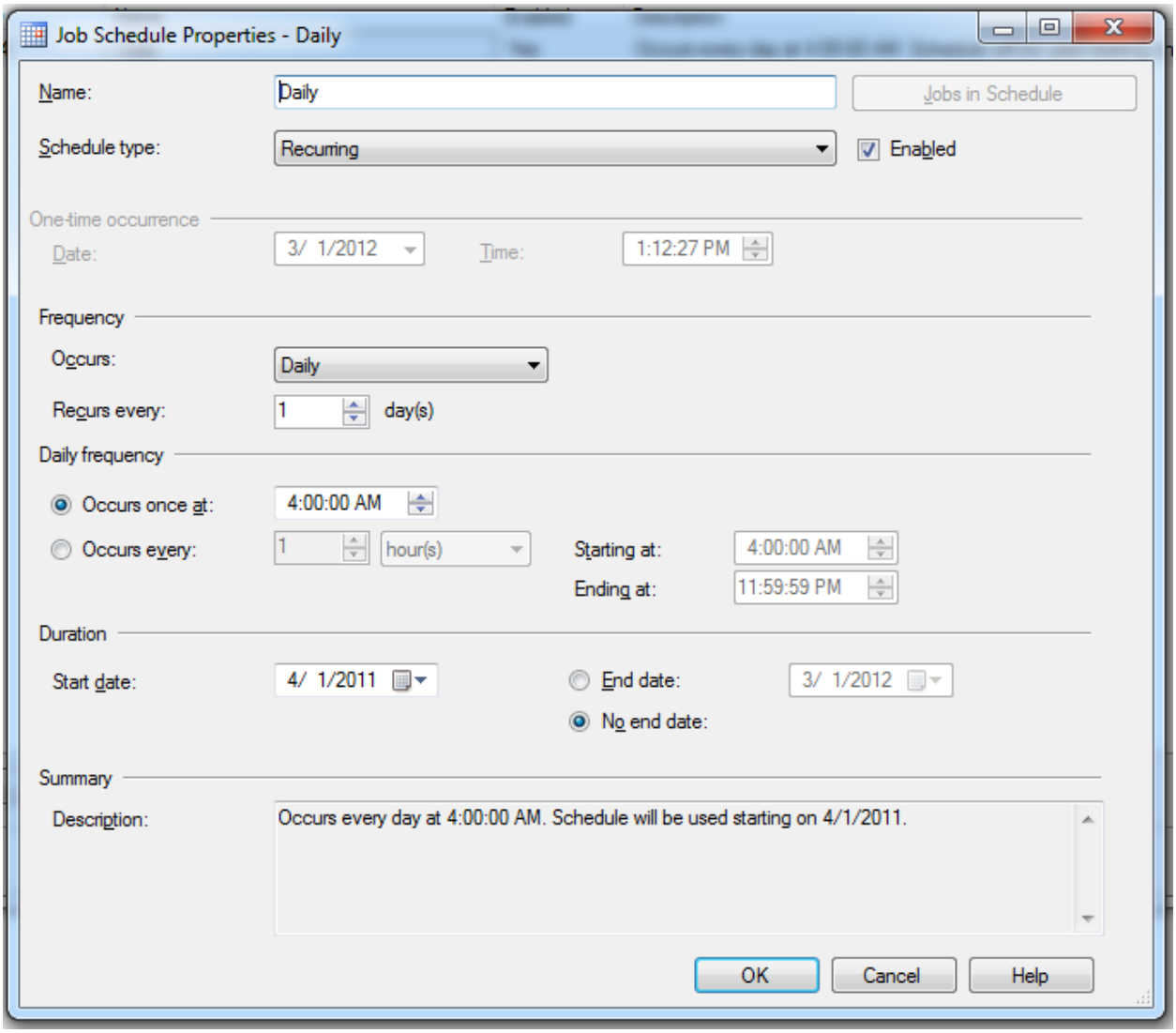
1. Open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_Purge_Blocks** job



4. Check the Enabled checkbox
5. In the Select a page pane select Schedules



6. Double click the schedule task



7. Change the schedule to how often you want to purge this data. It is recommended to run this daily.
8. Click OK to close the window
9. Click OK to close the job properties window

OTHER COMMANDS AND PROCEDURES

This section describes other commands and processes that can be used with the Performance Analyzer.

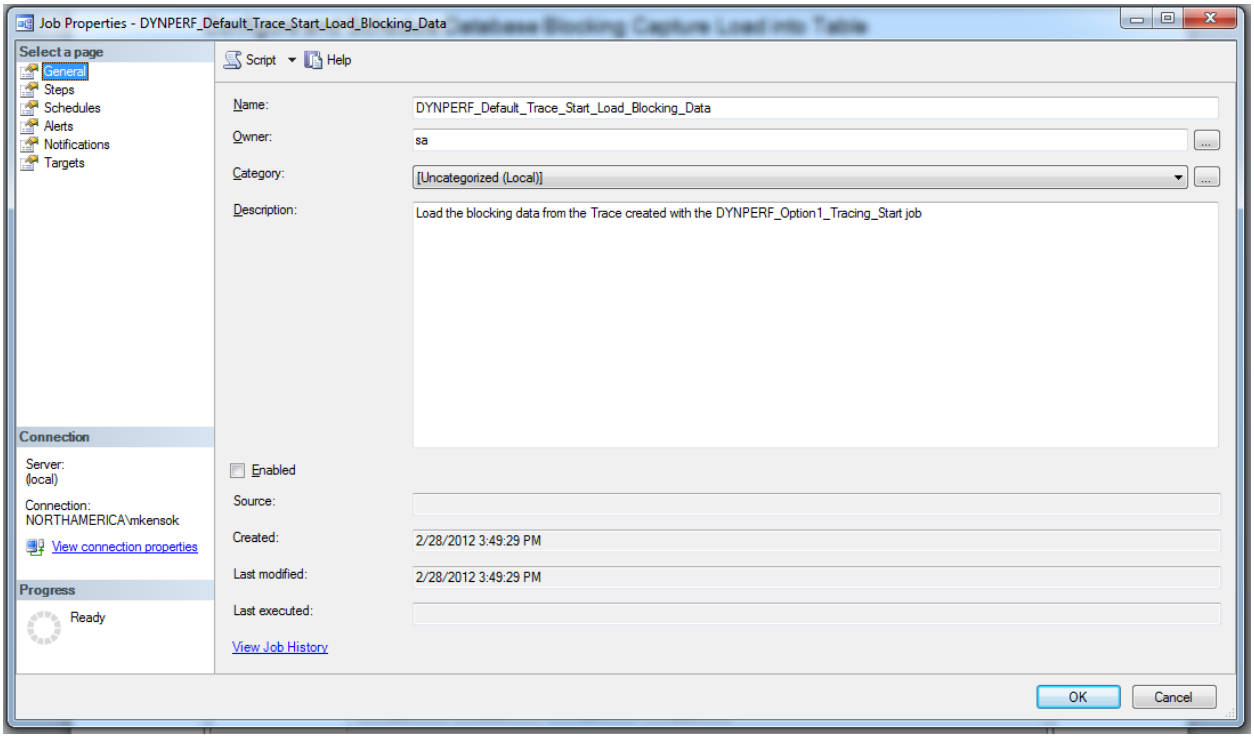
(Optional) Configure and Schedule Database Blocking Capture Load into Table

This process will load the data that is collected from the *"Configure and Schedule Database Blocking Capture"* process into a table in the **DynamicsPerf** database and can be used to view blocking events in a SQL table as opposed to viewing the raw SQL trace files. In the following steps you will configure and schedule the load of this data from the trace files and into SQL tables.

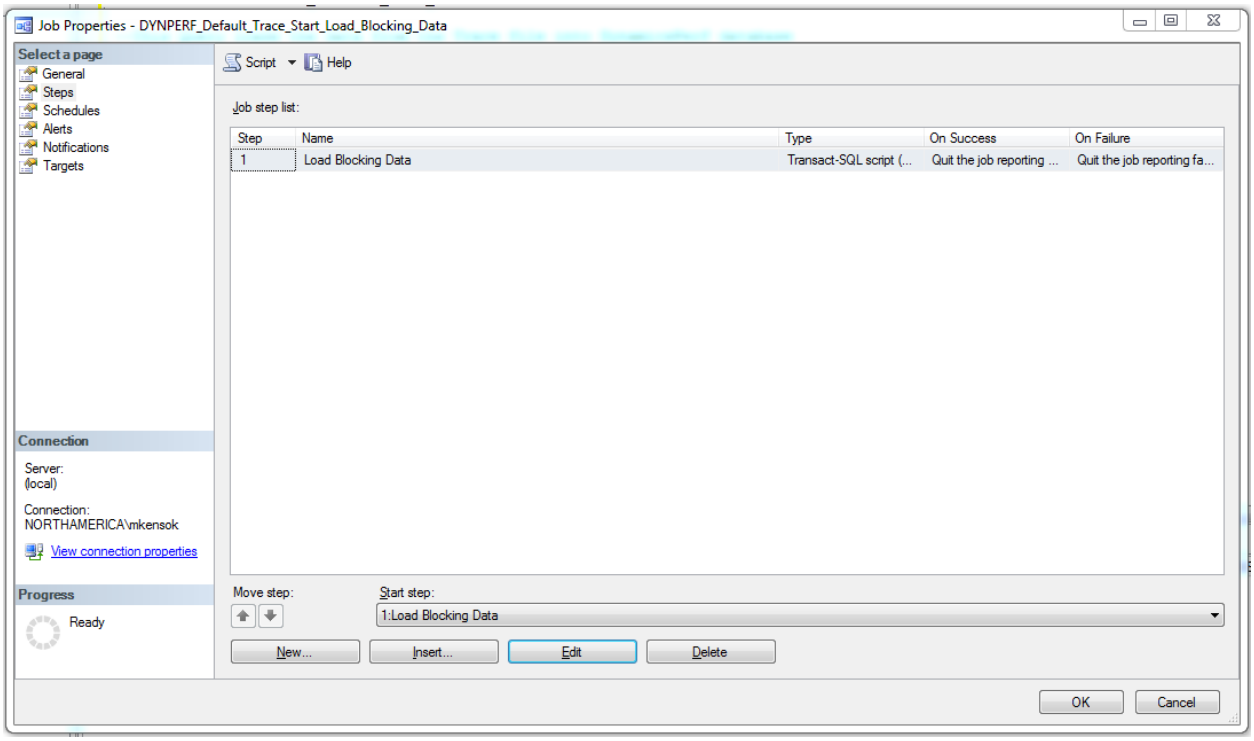
***NOTE**: Use the BLOCKED_PROCESS_INFO_VW table in the **DynamicsPerf** database to view this information.*

---

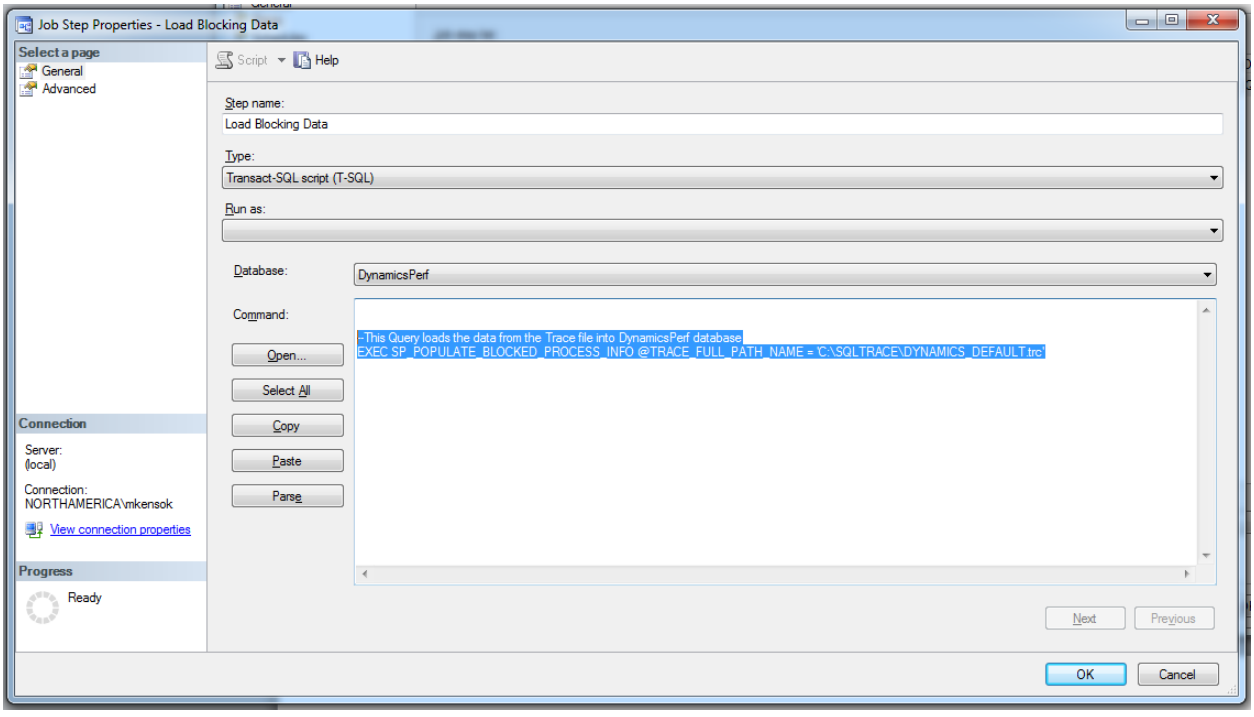
1. On the database server, open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_Default_Trace_Start_Load_Blocking_Data** job



4. Check the Enable checkbox
5. In the Select a page pane, select Steps

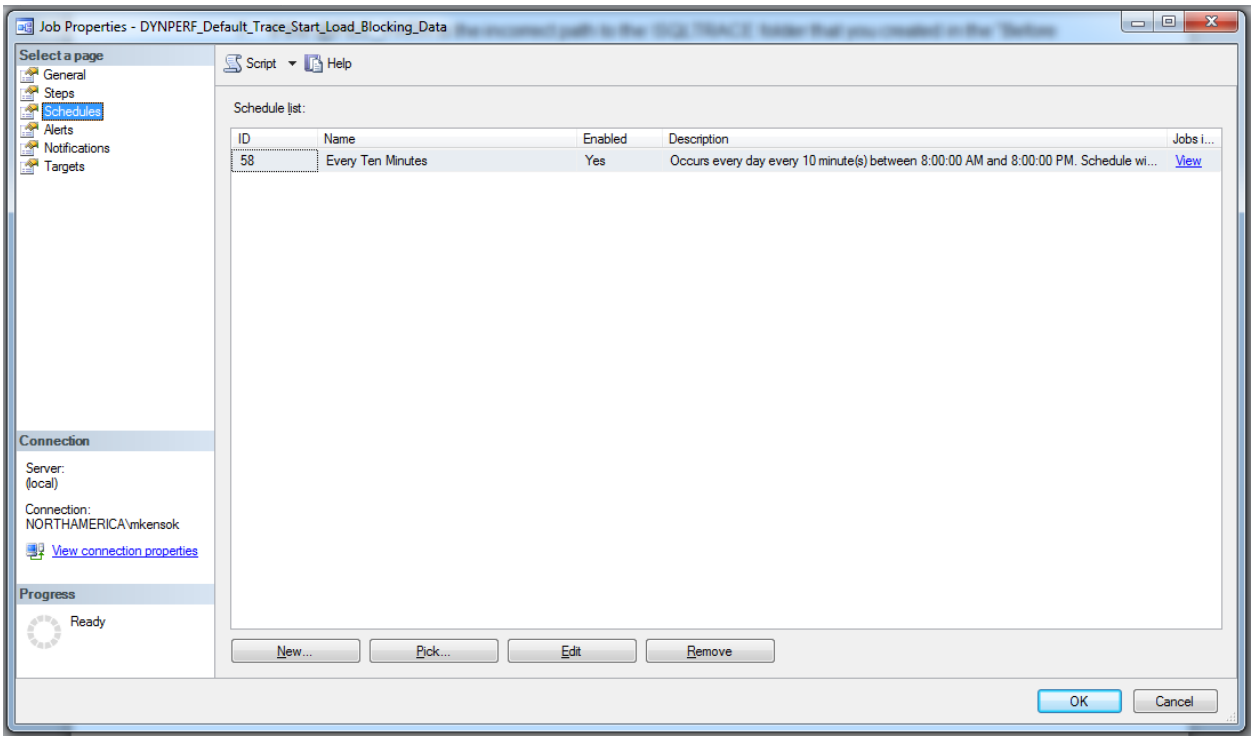


6. Double click Step 1 Load Blocking Data to open it.



7. If the @FILE_PATH is the incorrect path to the \SQLTRACE folder that you created in the “Before you begin” steps, change it here
8. Click OK to close the window

9. In the Select a page pane select Schedules

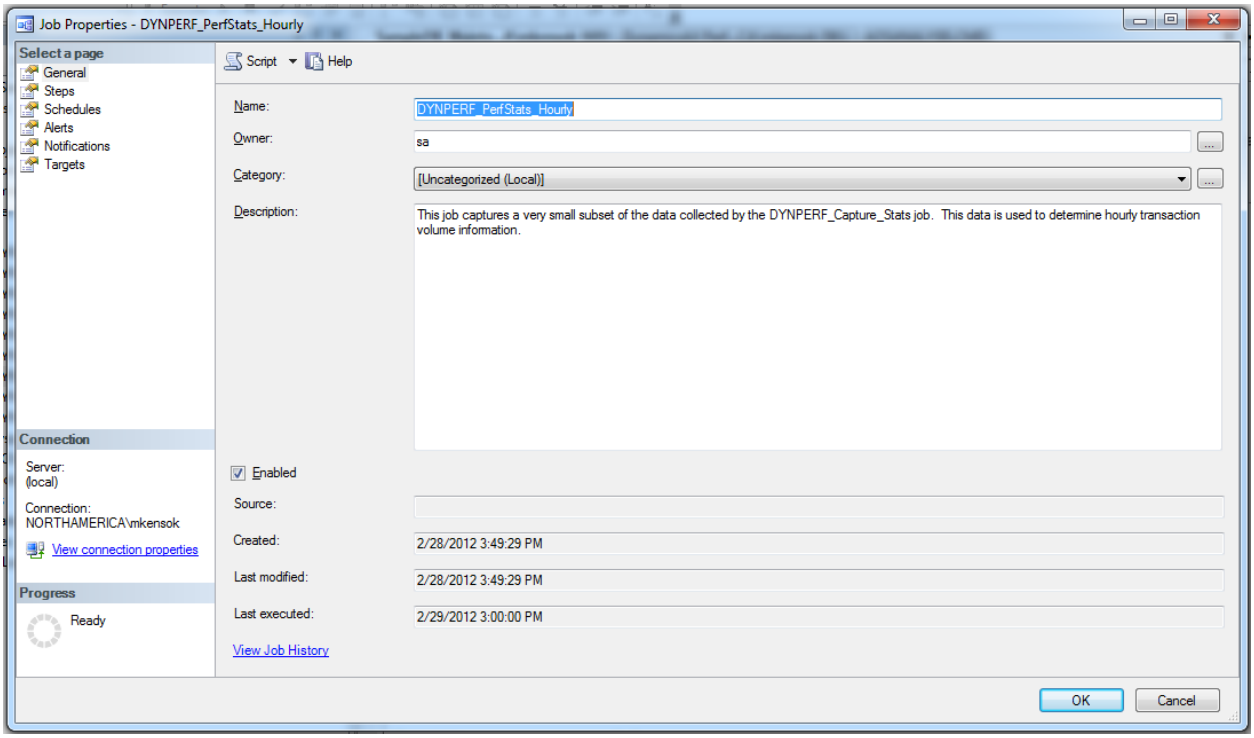


10. By default, this job will run every ten minutes. Change it in here if you wish to adjust the load schedule time.
11. Click OK to close the window

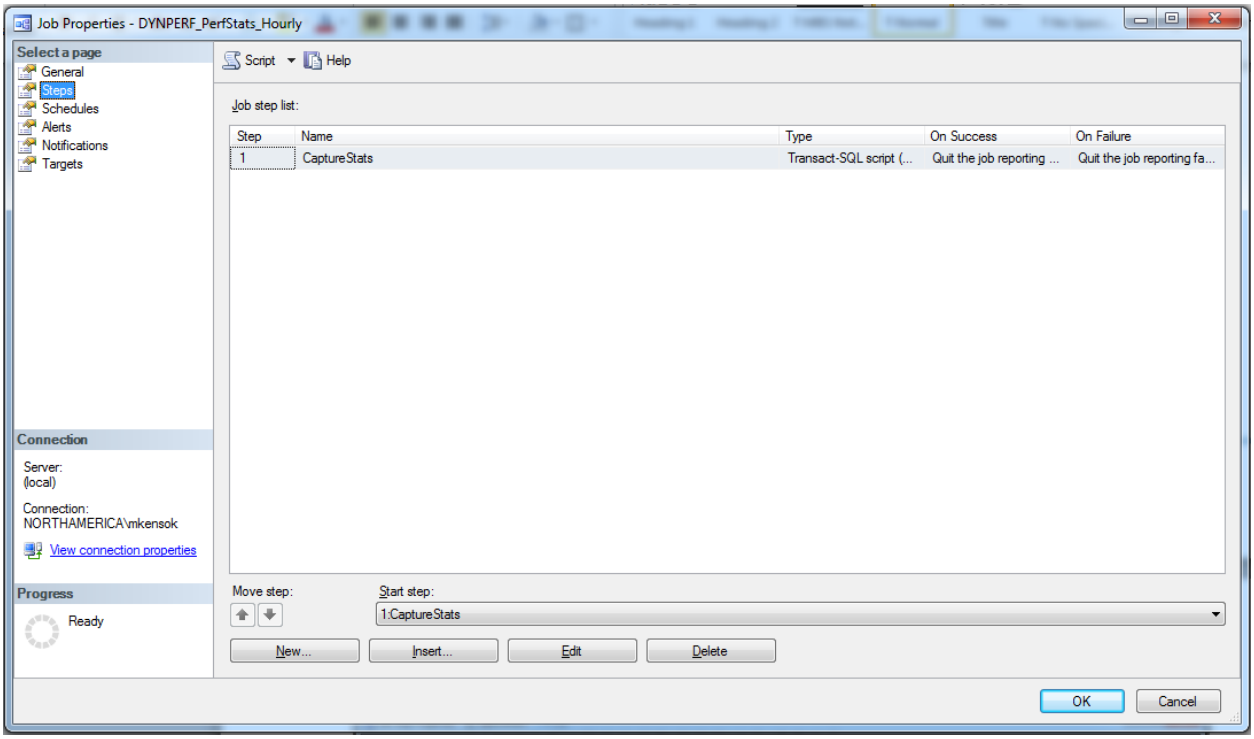
(Optional) Configure and Schedule Hourly Performance Data Capture

It is optional if you want to capture performance data hourly. In the following steps, you will configure and schedule to capture lightweight performance data hourly. This information will be captured in the **DynamicsPerf** database.

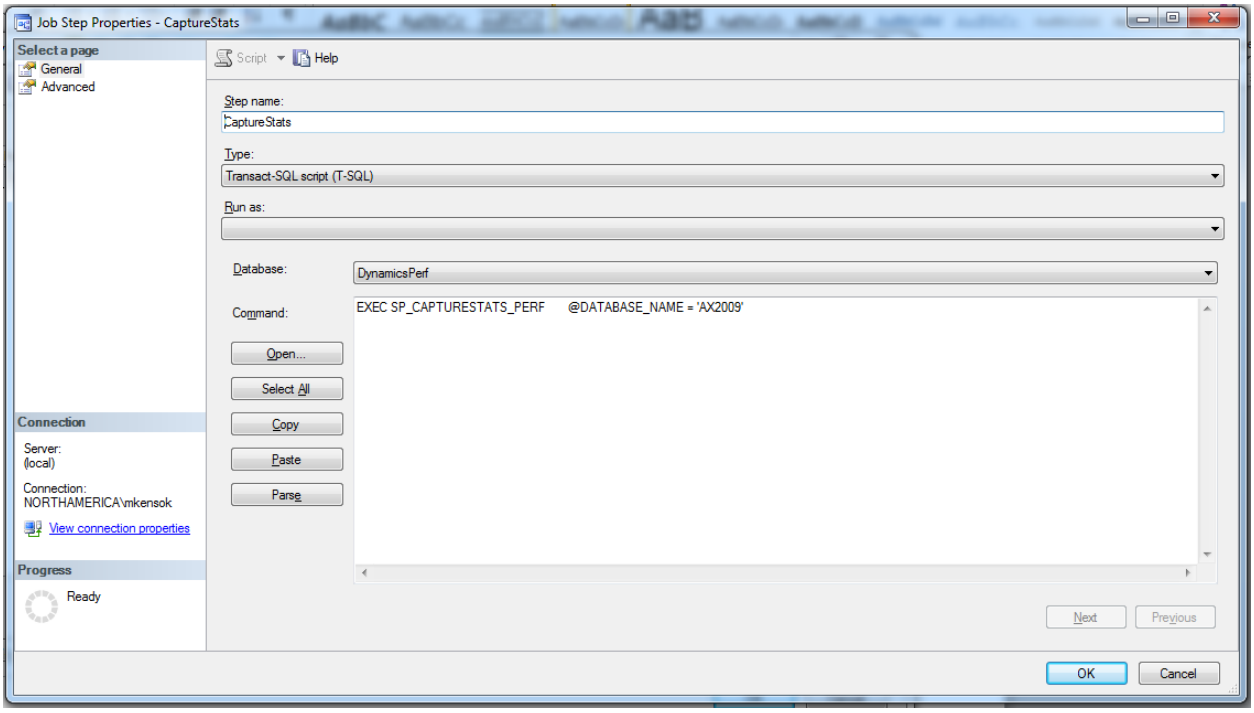
1. Open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the **DYNPERF_PerfStats_Hourly** job



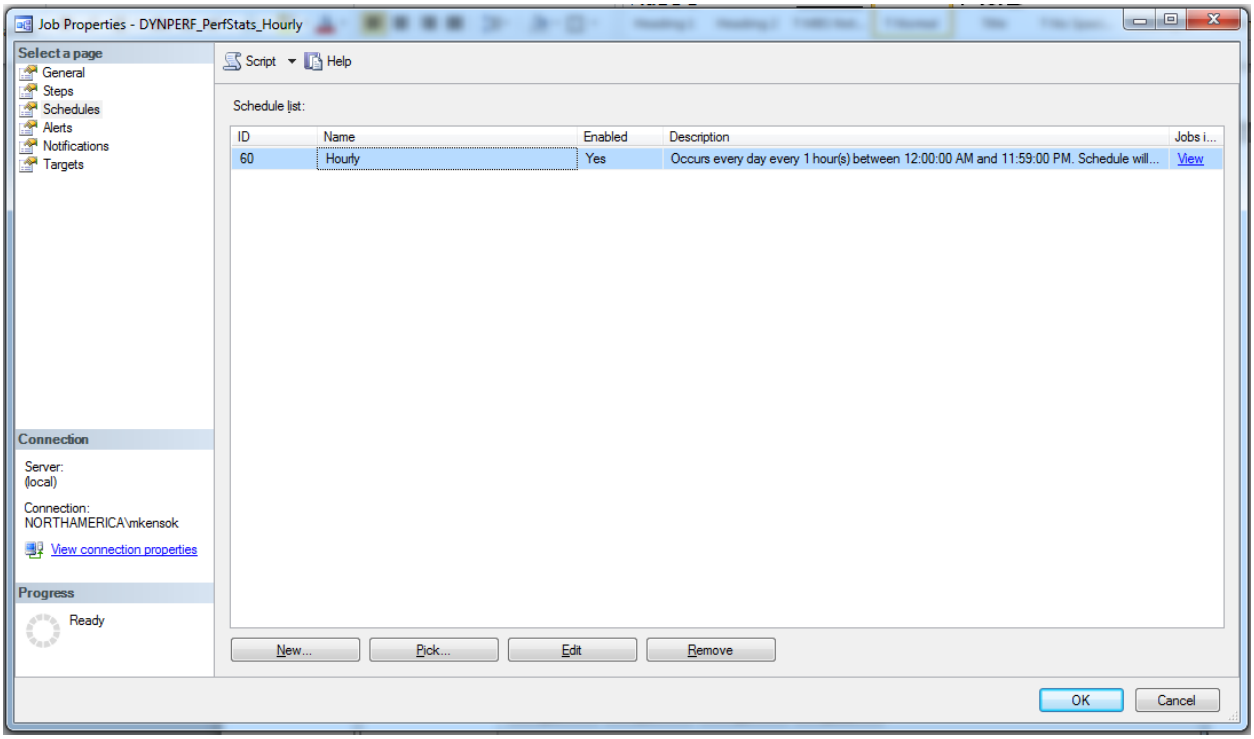
4. In the Select a page pane, select Steps



5. Double click Step 1 capturestats to open it.



6. Change the @DATABASE_NAME to the name of the AX database name (Example: ‘AX2009’)
7. Select OK to close the window
8. In the Select a page pane select Schedules



9. By default, this task is enabled and will run every hour on the day.

10. Click OK to close the DYNPERF_PerfStats_Hourly job window

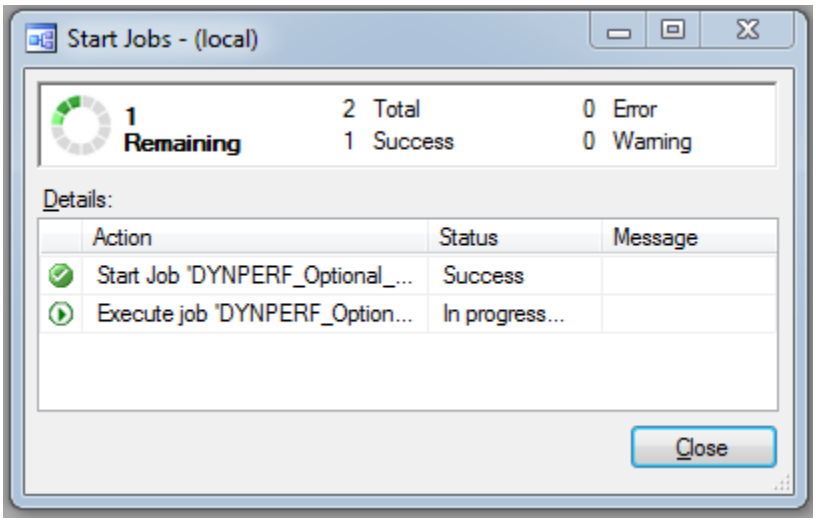
How to Execute Database Blocking Data Polling

This records all blocking into a table called Blocks in the **DynamicsPerf** database via polling. This method can put stress on SQL Server if there are many processes getting blocked, but works well for a fast check of blocking or when there is a limited amount of blocking. Follow the steps below to execute.

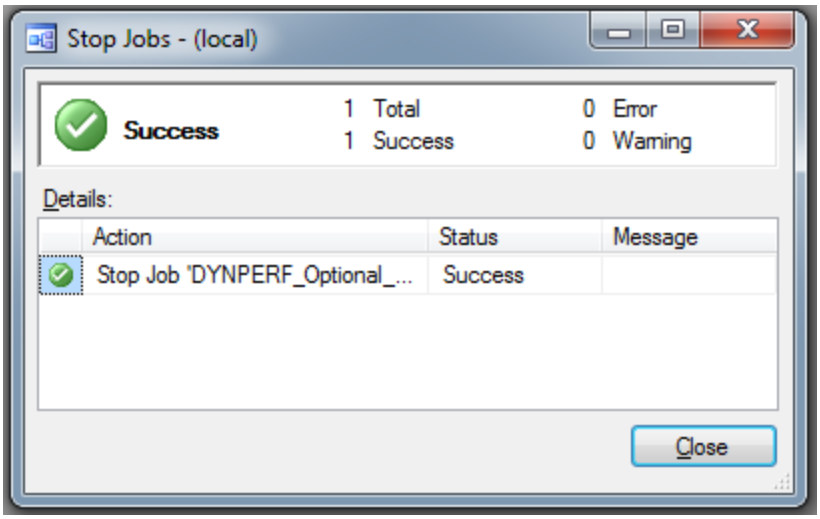
***IMPORTANT**: This option is provided for cases where SQL Tracing cannot be done. This process works fine for short term tracking of blocks, under 1 hour, or in cases where you know you have blocking occurring at that point in time. This job shouldn't be run for long periods of time.*

---

1. On the database server, open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Select the **DYNPERF_Optional_Polling_for_Blocking** job
4. Right click>Start Job at Step…



5. When you are done collecting blocking data
6. Select the **DYNPERF_Optional_Polling_for_Blocking** job
7. Right click>Stop job



How to Capture Performance Data Manually

There may be times when you want to capture the performance data manually instead of waiting for it to run at its scheduled time. In the following steps, you will capture performance data manually.

1. Open SQL Server Management Studio (SSMS)
2. Click File>Open, Project/Solution
3. Browse to the location for where you extracted the DynamicsPerf1.15 for SQL2008+.zip
4. Select the *Performance Analyzer 1.15 for Microsoft Dynamics.ssmssln* file
5. In Solution Explorer, open the *Manual –CaptureStats.sql* script

```
USE DynamicsPerf
EXEC SP_CAPTURESTATS     @DATABASE_NAME = 'XXXXXXXXX'
                                        --, @DEBUG = 'Y'
```

6. Change the @DATABASE_NAME to the name of the AX database name (Example: ‘AX2009’)
7. Execute the script against the DynamicsPerf database

How to Capture Performance Data from Multiple Databases

It is possible to capture performance data from multiple databases. This may be necessary if you wish to verify the performance of databases in addition to your AX database. In the following steps, you will configure the performance data capture for all databases.

1. Open SQL Server Management Studio (SSMS)
2. Click File>Open, Project/Solution
3. Browse to the location for where you extracted the DynamicsPerf1.15 for SQL2008+.zip
4. Select the *Performance Analyzer 1.15 for Microsoft Dynamics.ssmssln* file
5. If you wish to do this on a recurring basis, modify the DYNPERF_Capture_Stats job with the following information or manually execute the script below one time against the DynamicsPerf database:

- Use the following SQL script

```
USE DynamicsPerf
EXEC SP_CAPTURESTATS     @DATABASE_NAME = NULL
```

- Insert the name of the other databases that you wish to capture performance data for into the DATABSES_2_COLLECT table

Disable Long Running Query Capture for AX

To disable the long running query capture for AX, follow these steps"

1. On the database server, open SQL Server Management Studio (SSMS)
2. Click File>Open, Project/Solution
3. Browse to the location for where you extracted the DynamicsPerf1.15 for SQL2008+.zip
4. Select the *Performance Analyzer 1.15 for Microsoft Dynamics.ssmssln* file
5. In Solution Explorer, open the *DynamicsAX Client Tracing.sql* script
6. Change <dbname> to the name of your AX database
7. Execute only the part listed below from the script against the DynamicsPerf database to enable client tracing for all AX users

```
/******************  Set AX Client tracing *************/
/* NOTE: must enable AX client tracing on the AOS servers */
USE DynamicsPerf

GO

EXEC SET_AX_SQLTRACE
  @DATABASE_NAME = '<dbname>',
  @QUERY_TIME_LIMIT = 5000
```

8. To view the results of a user within AX:
    a. Open Dynamics AX
    b. Go to Tools>Options
    c. Select the SQL tab
    d. Notice the SQL checkbox is unmarked, the long query threshold is blank, and the Table (database) checkbox is disabled

How to Manually Execute the AOT Metadata Capture

If you want to manually execute the process for capturing AOT metadata, follow these steps:

1. Launch an AX client
2. Open the Application Object Tree (AOT) in Dynamics AX

3. Click the Import icon
4. Browse to the *PrivateProject_AOTExport_Batch.xpo* file found where you extracted the files from in step 1 of the “Before you begin” section
5. Click OK to import
6. Within the AOT, browse to Classes and find the AOTExport Class
7. Right click>Open
8. The cursor will appear for a few seconds indicating that it is running

How to Manually Run the AOS Configuration and Event Logs Capture

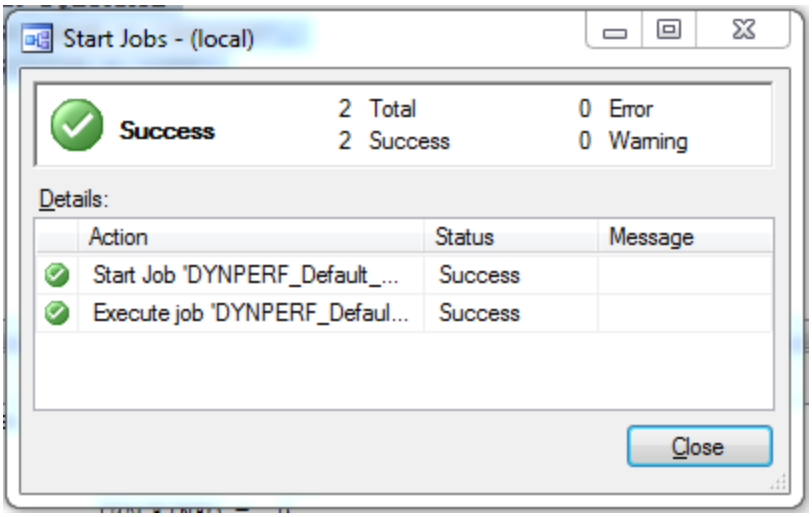
To manually execute the AOS configuration and event logs capture, follow these steps:

1. Browse to the location to where you extracted the files in step 1 of the “Before you begin” section
2. Edit the AOSANALYSIS.CMD file
3. Change the first parameter e.g., DYNAMICSVM to the name of your database server
4. Change the second parameter e.g., DYNAMICSDB to the name of your AX database name
5. Save the edited AOSANALYSIS.CMD file
6. Open a Command prompt
7. Browse to the location of the AOSANALYSIS.CMD and AOSANALYSIS.VBS files
8. From the Command prompt, enter AOSANALYSIS
9. Wait until the name of all AOS servers are listed and completed before you exit

How to Manually Stop Database Blocking Capture

To manually stop the capturing of database blocking events, follow these steps:

1. On the database server, open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Select the DYNPERF_Default_Trace_Stop job
4. Right click>Start job at Step…



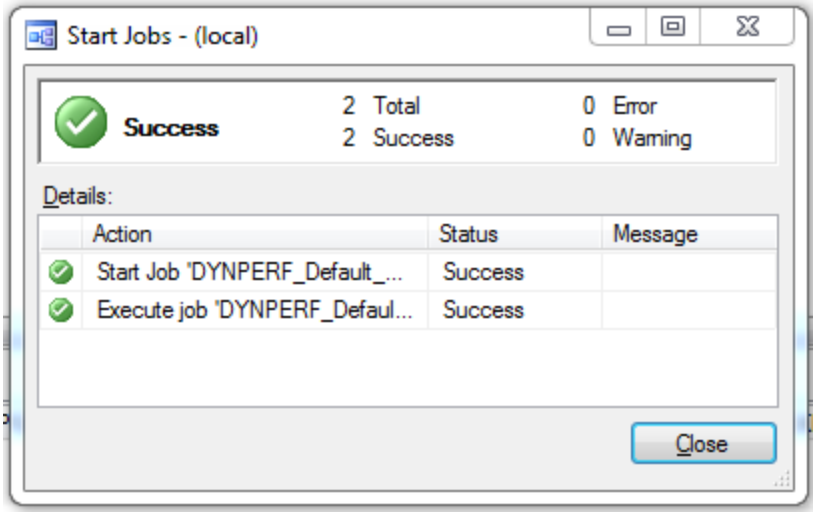
5. This will Stop the trace
6. Or, an alternative way is to run the following script against the DynamicsPerf database

```
/**************  Stop the Trace ****************************/

EXEC SP_SQLTRACE @TRACE_NAME = 'DYNAMICS_DEFAULT', -- Trace name - becomes base of trace file name

        @TRACE_STOP = 'Y' -- When set to 'Y' will stop the trace and exit
```

How to Manually Run Database Blocking Capture Load into Table

If you wish to manually run the database blocking capture load into table process, follow these steps:

1. On the database server, open SQL Server Management Studio (SSMS)
2. In Object Explorer, expand SQL Server Agent>Jobs
3. Open the DYNPERF_Default_Trace_Start_Load_Blocking_Data job
4. Right click> Start Job at Step…



5. Or, an alternative way is to run the following script against the DynamicsPerf database

```
--This Query loads the data from the Trace file into DynamicsPerf database

EXEC SP_POPULATE_BLOCKED_PROCESS_INFO @TRACE_FULL_PATH_NAME =
'C:\SQLTRACE\DYNAMICS_DEFAULT.trc'
```

Alternative Parameters for SP_CAPTURESTATS

There are several different ways to run SP_CAPTURESTATS to collect the data.

1. RECOMMENDED - Run SP_CAPTURESTATS with only the required parameter (e.g., @DATABASE_NAME) and do not limit any rows it collects. Suggestion is to schedule this to run every 8 hours for 7 days.

```
EXEC SP_CAPTURESTATS
@DATABASE_NAME= ‘dbname’
```

2. CAN USE THIS ONE IF COLLECTING FOR LARGE AMOUNTS OF DATA - Specify @TOP_ROWS to SP_CAPTURESTATS in order to limit the volume of data collected, and schedule periodically through the day. If running this way we suggest 2-3 times per day, possibly around shift change, or prior to and following your batch processing window.

```
EXEC SP_CAPTURESTATS
@DATABASE_NAME= 'dbname',
@TOP_ROWS=1000
```

APPENDIX A – STORED PROCEDURES

This section provides detailed information about the stored procedures as part of Performance Analyzer.

SET_AX_SQLTRACE

This stored procedure enables long duration tracing for all AX users by updating the USERINFO table. The 'Allow client tracing on Application Object Service instance' checkbox on the AOS Server Configuration Utility for each AOS Server must be marked before executing this stored procedure.

SP_CAPTURESTATS

This stored procedure is used to capture index and query statistics. It takes several parameters:

- @DATABASE_NAME – specify your database name here. Only one database can be specified.
- @TOP_ROWS and @TOP_COLUMN are used together to limit our collection of query statistics to the most expensive. We can tell how many rows (@TOP_ROWS) and by what criteria (@TOP_COLUMN). Use of these parameters results in a SELECT TOP and ORDER BY DESC to be added into the query that retrieves the query statistics. @TOP_COLUMN defaults to 0 which is all statistics. @TOP_COLUMN defaults to 'total_elapsed_time' which is the most common criteria, although this parameter is not evaluated unless @TOP_ROWS is > 0. Virtually any column from sys.dm_exec_query_stats can be used for @TOP_COLUMN – execution_count, total_worker_time(CPU), etc. If you open the procedure you should find the list easily. Note these two parameters only apply to query-related collection; index statistics are always collected for all tables and indexes in the database specified.
- @RUN_NAME is used to supply run name, and it defaults to a friendly date/time format. The value for RUN_NAME is usually supplied as predicate whenever we query either of the above mentioned views. You can see what runs are present by querying table STATS_COLLECTION_SUMMARY.
- @INDEX_PHYSICAL_STATS defaults to 'N'. When 'Y' is specified, we collect from the sys.dm_index_physical_stats management view. This is relatively expensive (it does scans on indexes at the level just above leaf), and the data it provides is really not needed often. Primary use is tracking fragmentation.
- @SKIP_STATS defaults to 'N'. When 'Y' is specified , then the capturing of database statistics will not occur.

SP_CAPTURESTATS_PERF

This stored procedure is used to capture a very small subset of the data captured by SP_CAPTURESTATS. It takes one parameter:

- @DATABASE_NAME – specify the AX or other database name here. Only one database can be specified.

SP_PURGESTATS

This stored procedure can be used to delete data from the DYNAMICSPERF objects. It accepts an optional parameter @PURGE_DAYS which defaults to 14 days, meaning data older than 14 days will be deleted.

- @PURGE_DAYS – Number of days to keep data, all data prior to this will be deleted from DynamicsPerf

If a running sp_capturestats on a recurring basis then a weekly execution to purge old query statistics is adequate unless you are constrained for space and/or increase the volume of query statistics gathered by increasing or eliminating the @TOP_ROWS parameter from the execution of SP_CAPTURESTATS. Example:

```
EXEC SP_PURGESTATS
@PURGE_DAYS= 14
```

SP_PURGEBLOCKS

This stored procedure can be used to delete blocking data from the DYNAMICSPERF database. It accepts an optional parameter @PURGE_DAYS which defaults to 14 days, meaning data older than 14 days will be deleted.

- @PURGE_DAYS – Number of days to keep data, all data prior to this will be deleted from DynamicsPerf

```
EXEC SP_PURGEBLOCKS
@PURGE_DAYS= 30
```

SP_SQLTRACE

This stored procedure is used to capture several important but lightweight events via SQL Server trace. There are 2 SQL configuration changes that are changed when running the CreateDynamicsPerfObjects.sql:

- sp_configure 'blocked process threshold',5 reconfigure - This will enable the trace to collect blocking data for blocking events 5 seconds or longer. This is a low cost event, and may remain enabled after completion of the health check or performance engagement. In the event the customer wishes to disable, set the threshold back to 0.
- sp_configure 'default trace enabled',0 reconfigure

The SP_SQLTRACE stored procedure accepts the following parameters:

- @FILE_PATH (required)– The location for where the trace files will be stored. This directory MUST exist before you start the trace and you MUST have at least 1 GB of space free
- @DATABASE_NAME (required)– The name of the production AX database that you want to trace. NULL means it will trace ALL databases
- @TRACE_FILE_SIZE – The maximum size of the trace files that will be generated. It will rollover onto a new file once this size has been reached
- @TRACE_FILE_COUNT – The maximum number of trace files to be generated. It will delete the oldest one first when this is reached
- @TRACE_STOP – This will stop or start the trace
- @TRACE_RUN_HOURS – The number of hours that you will run the trace
- @HOSTNAME - Hostname filter for trace (optional)
- @DURATION_SECS - enables statement, rpc, batch trace by specified duration

Below is an example execution if I want to run the trace file for 7 days with a trace location of C:\SQLTRACE and a database name of DynamicsPerf.

```
EXEC SP_SQLTRACE
@FILE_PATH = ‘C:\SQLTRACE’, REQUIRED
@DATABASE_NAME= 'AXDB', REQUIRED
@TRACE_FILE_SIZE = 10,
@TRACE_FILE_COUNT = 100
@TRACE_STOP = ‘N’
@TRACE_RUN_HOURS = 168
```

If the trace needs to be stopped earlier, rerun SP_SQLTRACE and specify @TRACE_STOP = 'Y'

```
EXEC SP_SQLTRACE
@TRACE_STOP = 'Y'
```

SP_POPULATE_BLOCKED_PROCESS_INFO

This is a stored procedure for loading blocking data that is captured using SP_SQLTRACE. This procedure can be used to load the blocking data into the DynamicsPerf. This simplifies analysis of the data and also allows a backup of the database to be sent to others for analysis without also having to send the Trace file separately.
The SP_POPULATE_BLOCKED_PROCESS_INFO stored procedure accepts the following parameters:

- @TRACE_FULL_PATH_NAME (required)– The location for where the trace files will be loaded from.

SP_LOCKS_MS

This stored procedure is used to capture database blocking. This procedure is called via the DynamicsPerf_Logblocks job.
The SP_LOCKS_MS stored procedure accepts the following parameters:

- @delay (required)– This is the wait time in milliseconds that the procedure uses to capture blocking data. For example, 2000 means that the procedure looks for blocking every 2 seconds.

SP_LOGBLOCKS_MS

This procedure is called by SP_LOCKS_MS procedure. This procedure captures the blocking data into the BLOCKS table in the DynamicsPerf database.

APPENDIX B – VIEWS

This section provides detailed information about the views as part of Performance Analyzer.

AX_INDEX_DETAIL_CURR_VW

This view displays index detail extracted from the AOT. It can be used in conjunction with **INDEX_STATS_CURR_VW** to determine if there are any inconsistencies between what is defined in the AOT and what is defined in SQL Server. All views that end with **_CURR_VW** query only the most recently captured data. There is no need to specify a RUN_NAME in the WHERE clause.

AX_INDEX_DETAIL_VW

This view displays the same data as **AX_INDEX_DETAIL_CURR_VW.** The only difference is that this view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

AX_SQLTRACE_VW

This view displays query detail and statistics for queries that meet the long running query duration threshold set in AX. The source for this data is the AX table SYSTRACETABLESQL. Information available in this view includes:

- AX User ID.
- Query text (TSQL statement).
- Call stack.
- Query statistics (duration, rows affected, etc).

AX_TABLE_DETAIL_CURR_VW

This view displays table properties extracted from the AOT. Information available in this view includes:

- Application layer (SYS, VAR, USR, etc)
- OCC settings
- Cache settings
- Database logging settings

All views that end with **_CURR_VW** query only the most recently captured data. There is no need to specify a RUN_NAME in the WHERE clause.

AX_TABLE_DETAIL_VW

This view displays the same data as **AX_TABLE_DETAIL_CURR_VW.** The only difference is that this view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

BLOCKED_PROCESS_VW

This view queries the set of trace files created by the **SP_SQLTRACE** stored procedure. The purpose of this view is to display any blocking information (blocked process report events) recorded in the trace files.

INDEX_STATS_CURR_VW

This view displays index detail and usage statistics such as:

- Index properties (name, clustered or non-clustered, primary key, unique, etc).
- Index keys (column names and order).
- Index usage statistics (seek count, scan count, etc).

All views that end with **_CURR_VW** query only the most recently captured data. There is no need to specify a RUN_NAME in the WHERE clause.

INDEX_STATS_VW

This view displays the same data as **INDEX_STATS_CURR_VW.** The only difference is that this view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

QUERY_STATS_CURR_VW

This view displays query detail and statistics such as:

- Query text (TSQL statement).
- Execution plan (XML).
- Query statistics (execution count, duration, read count, etc).

All views that end with **_CURR_VW** query only the most recently captured data. There is no need to specify a RUN_NAME in the WHERE clause.

QUERY_STATS_VW

This view displays the same data as **QUERY_STATS_CURR_VW.** The only difference is that this view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

BLOCKED_PROCESSES_INFO_VW

This view displays blocking information. You must run the SP_POPULATE_BLOCKED_PROCESS_INFO procedure to populate this view.
All views that end with **_CURR_VW** query only the most recently captured data. There is no need to specify a RUN_NAME in the WHERE clause.

BUFFER_DETAIL_VW

This view displays all details about the SQL Server data cache buffer. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

BUFFER_DETAIL_CURR_VW

This view displays the same data as **BUFFER_DETAIL_VW.** The only difference is that this view restricts the dataset to the most recently captured data.

MISSING_INDEXES_VW

This view displays all details about Queries that SQL Server is suggesting an index be added. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

MISSING_INDEXES_CURR_VW

This view displays the same data as **MISSING_INDEXES_VW.** The only difference is that this view restricts the dataset to the most recently captured data.

QUERY_STATS_HASH_VW

This view contains all query information aggregated by QUERY_HASH. This view contains useful information on SQL Server 2008 and above. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

QUERY_STATS_HASH_CURR_VW

This view displays the same data as **QUERY_STATS_HASH.** The only difference is that this view restricts the dataset to the most recently captured data.

SQL_CONFIGURATION_VW

This view contains all information from sp_configure. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

SQL_CONFIGURATION_CURR_VW

This view displays the same data as **SQL_CONFIGURATION_VW.** The only difference is that this view restricts the dataset to the most recently captured data.

SQL_DATABASEFILES_VW

This view contains all information database files for all databases. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

SQL_DATABASEFILES_CURR_VW

This view displays the same data as **SQL_DATABASEFILES_VW.** The only difference is that this view restricts the dataset to the most recently captured data.

SQL_DATABASES_VW

This view contains database information for all databases. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

SQL_DATABASES_CURR_VW

This view displays the same data as **SQL_DATABASES_VW.** The only difference is that this view restricts the dataset to the most recently captured data.

SQL_JOBS_VW

This view contains information for all SQL jobs. This view does not automatically restrict the dataset to the most recently captured data. You need to manually specify the **RUN_NAME** you wish to query. Use this view when you want the flexibility to query data from current and past captures.

SQL_JOBS_CURR_VW

This view displays the same data as **SQL_JOBS_VW.** The only difference is that this view restricts the dataset to the most recently captured data.

AX_NUM_SEQUENCES_VW

This view displays the numbersequence table from Dynamics AX. This view is used to compare two data captures to investigate the consumption rate of numbers per number sequence.

AX_DATABASELOGGING_VW

This view displays the how database logging is setup in Dynamics AX. This view requires that the AOTExport.XPO be run.

AX_SERVER_CONFIGURATION_VW

This view displays the Dynamics AX Cluster Configurations. This view is used to determine which AOS instances are running on which physical machines.

AX_BATCH_CONFIGURATION_VW

This view displays the batch configuration setup in Dynamics AX. This includes, which batch groups are running on which AOS instances. This also includes which batch jobs are setup on each batch group. This view is used for analyzing the optimal batch job configurations.

PERF_HOURLY_ROWDATA_VW

This view displays the data about changes in the number of rows per table per hour. This data is populated by the DYNPERF_PerfStats_Hourly SQL job. If this view is not populated then that SQL job is not running.

PERF_HOURLY_IOSTATS_VW

This view displays the data about changes in the number of disk waits per database file per hour. This data is populated by the DYNPERF_PerfStats_Hourly SQL job. If this view is not populated then that SQL job is not running.

PERF_HOURLY_WAITSTATS_VW

This view displays the data about changes in the number of SQL Server waits per hour. This data is populated by the DYNPERF_PerfStats_Hourly SQL job. If this view is not populated then that SQL job is not running.

SERVER_OS_VERSION_VW

This view displays information about the Windows operating system that SQL Server is installed on. This view will only have data if SQL Server 2008 R2 with Service Pack 1 or higher is installed.