Microsoft Dynamics® AX 2012

Windows Event Tracing in Microsoft Dynamics AX 2012

White Paper

This paper describes how to use the Windows Event Tracing infrastructure in Microsoft Dynamics AX 2012.

Date: September 2011

www.microsoft.com/dynamics/ax

Author: Nikolay Muzykin, Sustained Engineering

Microsoft Dynamics

Table of Contents

Introduction

In Microsoft Dynamics® AX 2012, functionality to support tracing throughout the product has been added. This functionality makes it easier for administrators and developers to investigate, analyze, and address potential problems by providing insight into what is happening in the lower layers of the product when user scenarios are running.

Microsoft Dynamics AX 2012 uses the Windows event tracing framework for instrumentation. In a basic usage scenario, you would record a trace into a file and then open it in a viewer to analyze it. This document explains how this can be done by using standard tools. For more advanced scenarios, please see the Windows event tracing framework reference in the Relevant links section later in this document.

Instrumented feature areas

The following areas in Microsoft Dynamics AX 2012 are instrumented out of the box:

- Data access
- Security
- Time zone operations
- X++ interpreter
- Enterprise portal for Microsoft Dynamics AX
- Windows form manipulation
- Setup
- Master resource planning

Also, several methods are exposed to X++ to allow basic tracing in customizations or adding more tracing to existing areas.

Recording traces

Because a standard Windows framework is used in Microsoft Dynamics AX 2012, standard Windows tools can be used to control tracing. The following sections describe how some of these tools can be used to save traces of Microsoft Dynamics AX 2012 events.

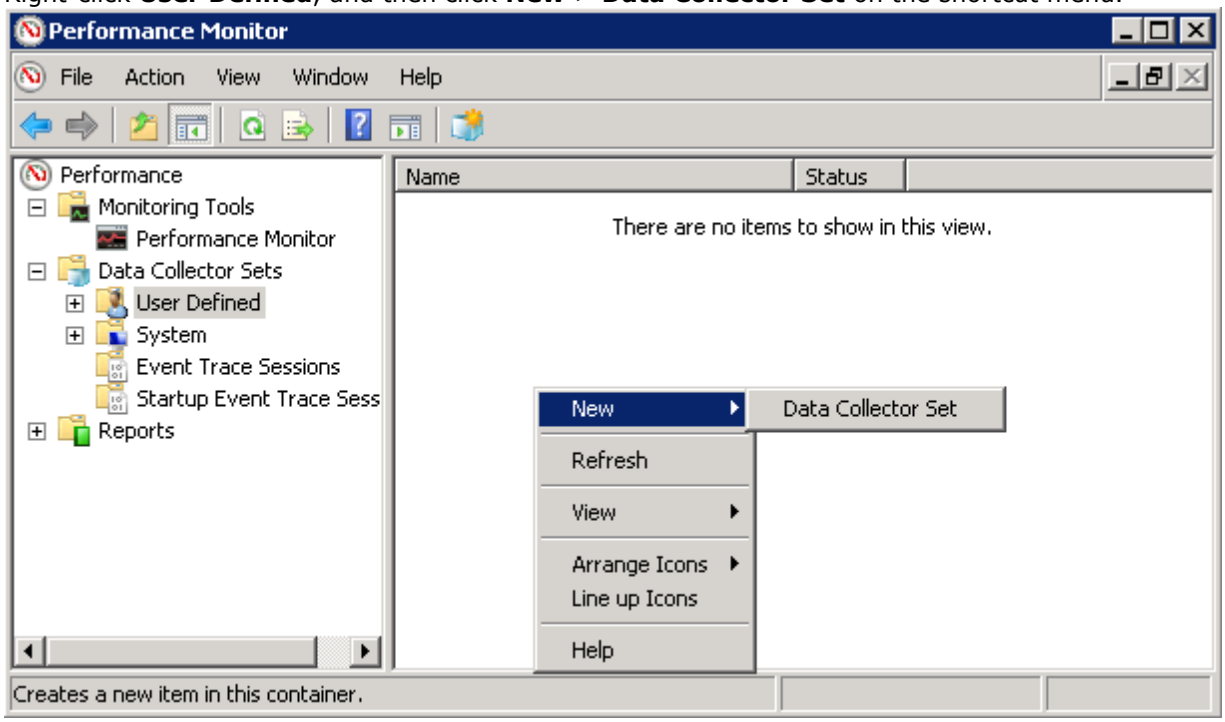
Using Performance Monitor to record traces

Complete the following procedures to record events by using the Performance Monitor console that is available in Windows.

Start the Create New Data Collector Set Wizard

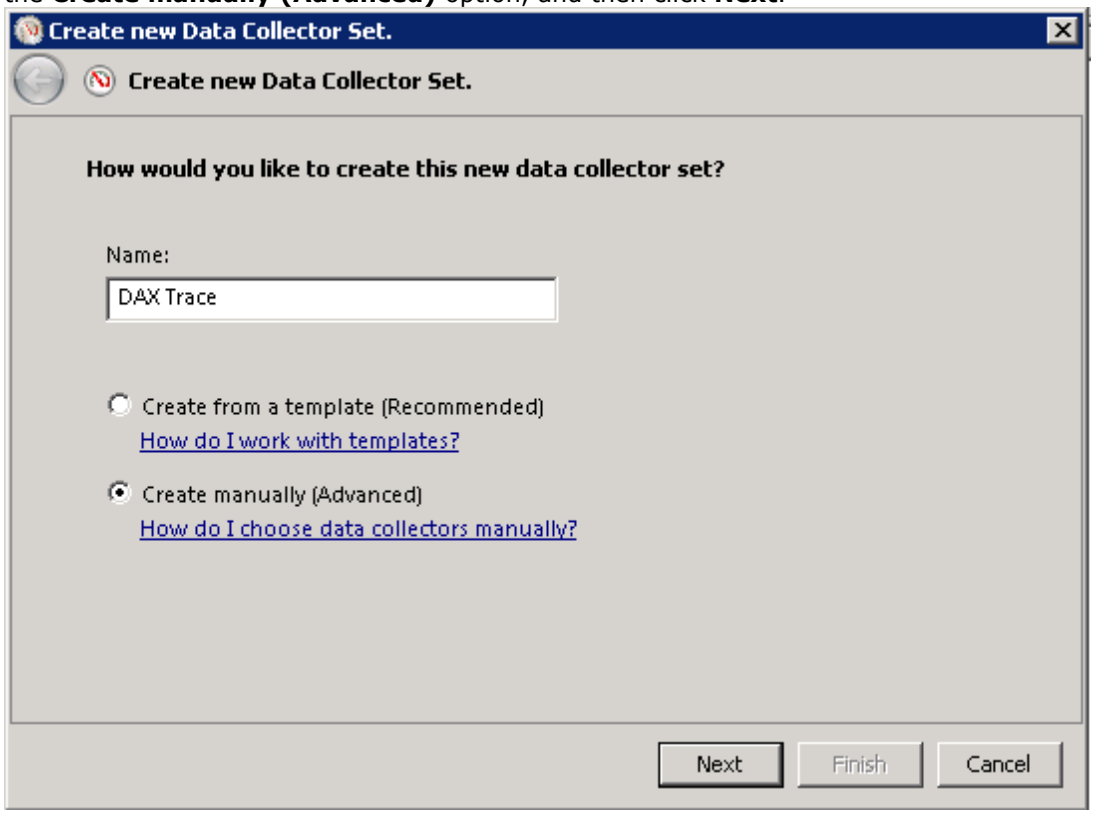
1. Open Performance Monitor by running perfmon.msc.
2. Go to Data Collector Sets\User Defined node.

3. Right-click **User Defined**, and then click **New** > **Data Collector Set** on the shortcut menu.

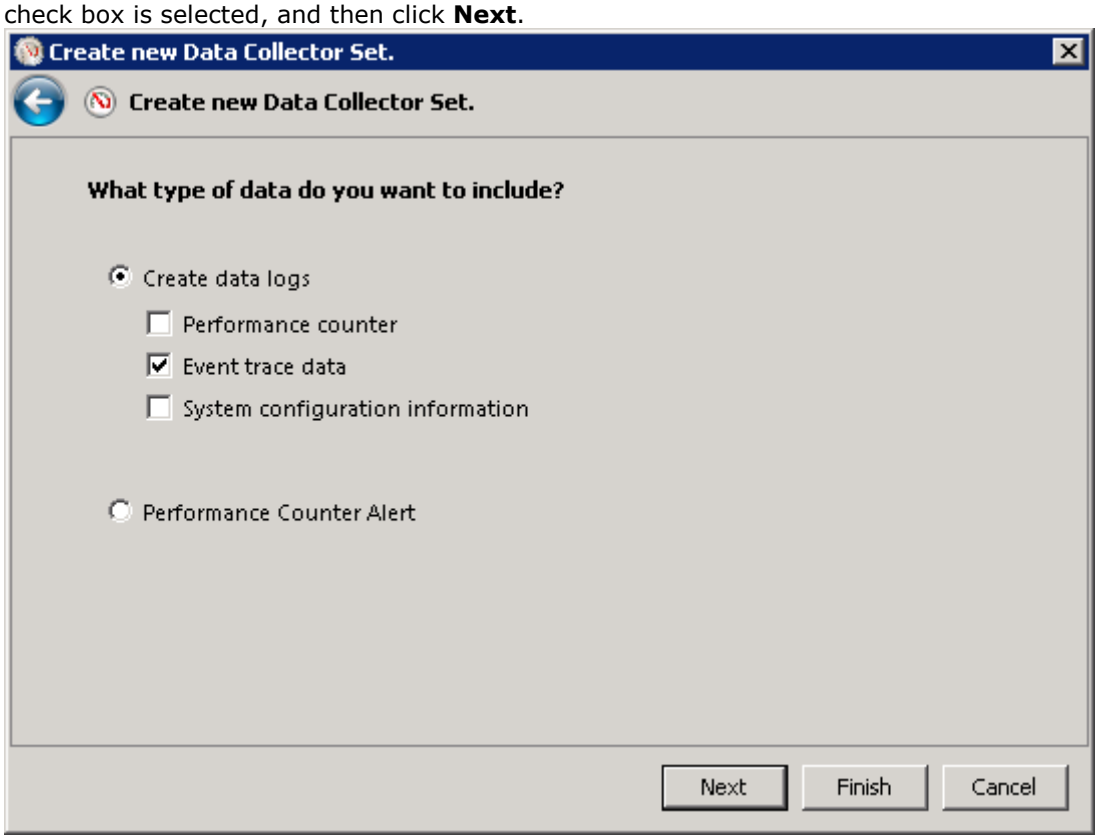


Complete the Create New Data Collector Set Wizard

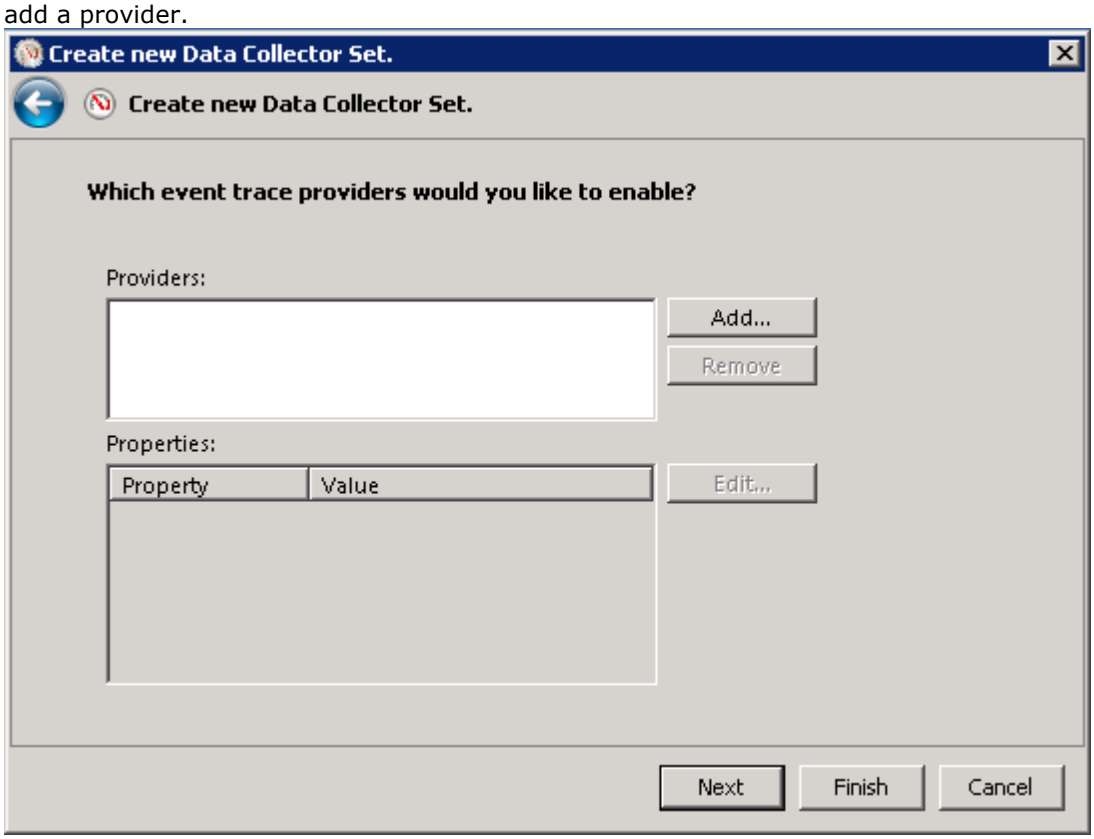
1. On the **How would you like to create this new data collector set** page, enter a name, select the **Create manually (Advanced)** option, and then click **Next**.



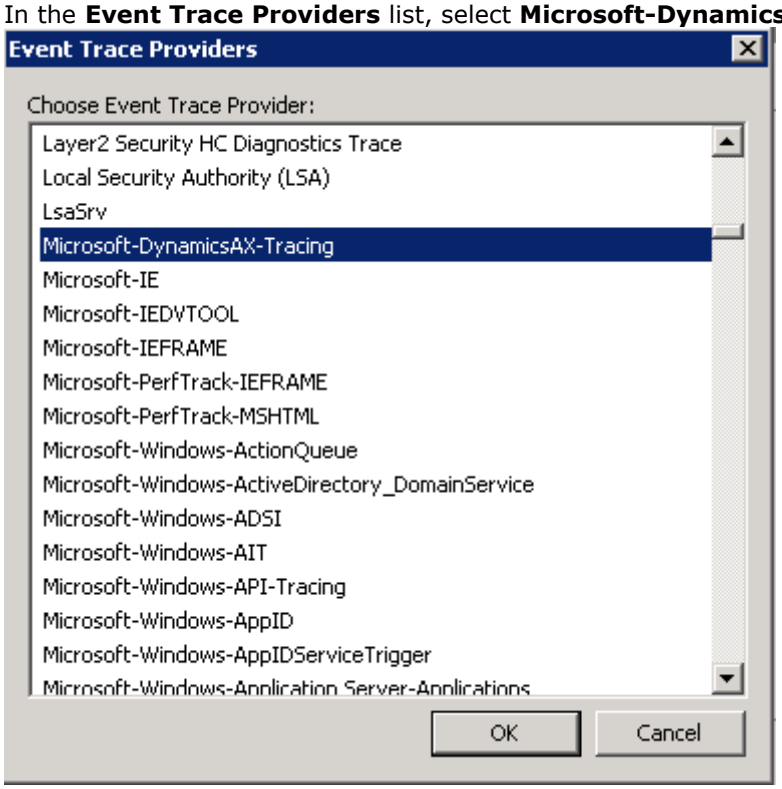
2. On the **What type of data do you want to include** page, verify that the **Event trace data** check box is selected, and then click **Next**.



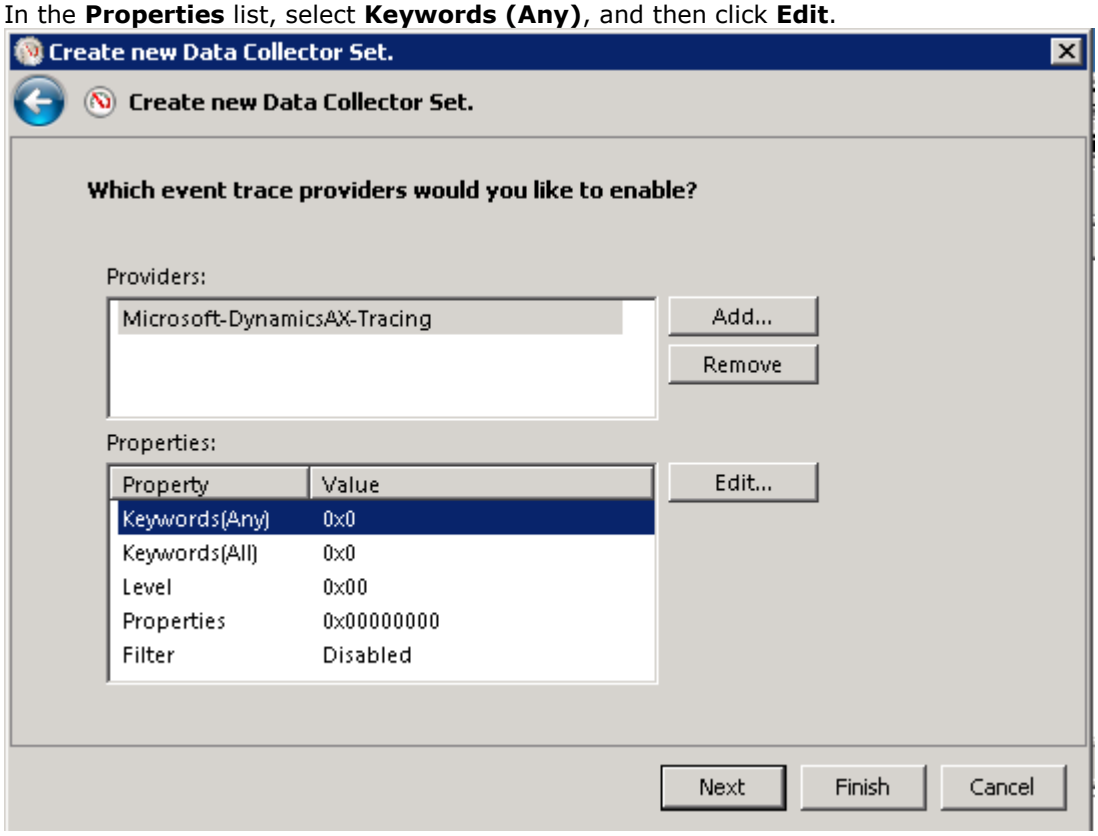
3. On the **Which event trace providers would you like to enable** page, click the **Add** button to add a provider.



4. In the **Event Trace Providers** list, select **Microsoft-DynamicsAX-Tracing**, and then click **OK**.

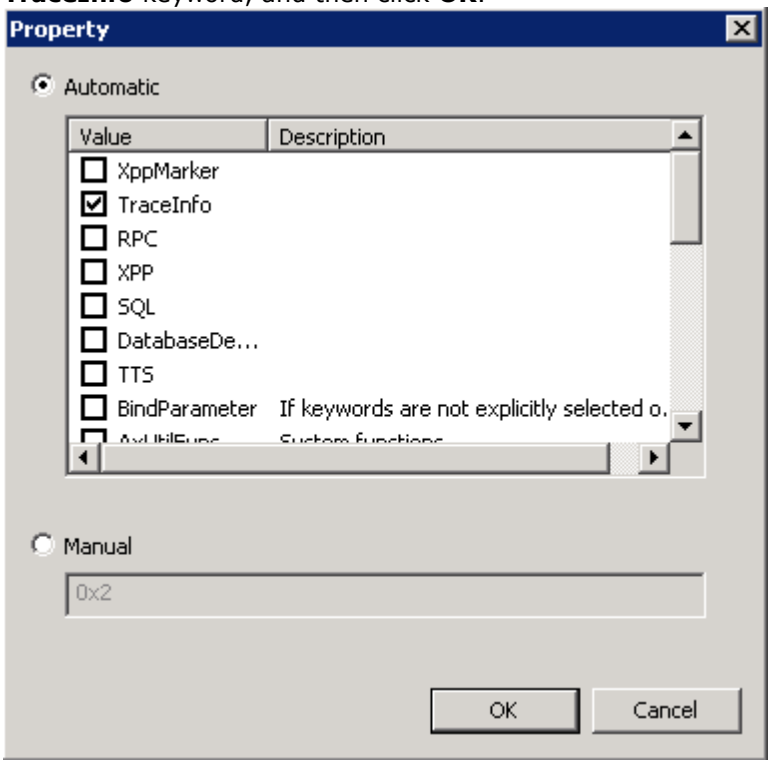


5. In the **Properties** list, select **Keywords (Any)**, and then click **Edit**.



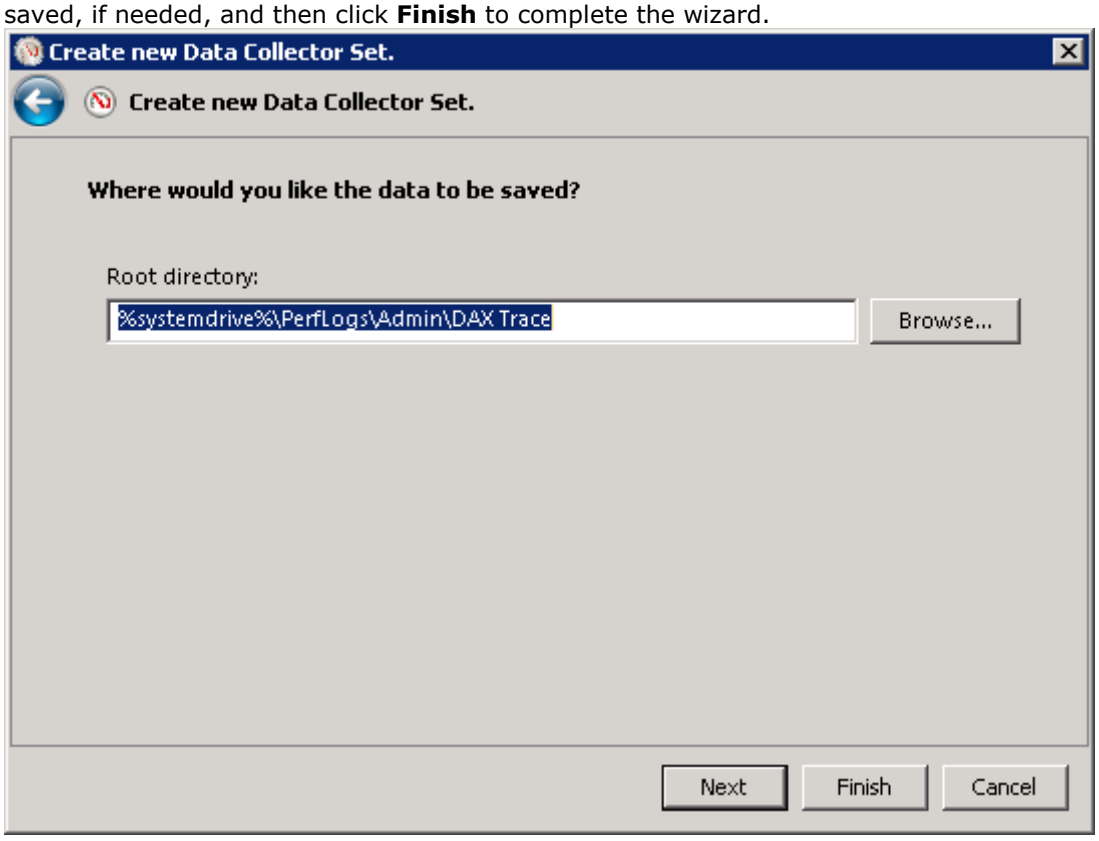
6. In the **Property** list, select the check boxes for the keywords that you want to trace. For example, to trace events generated by calling the xClassTrace::logComponent method, select the

TraceInfo keyword, and then click **OK**.



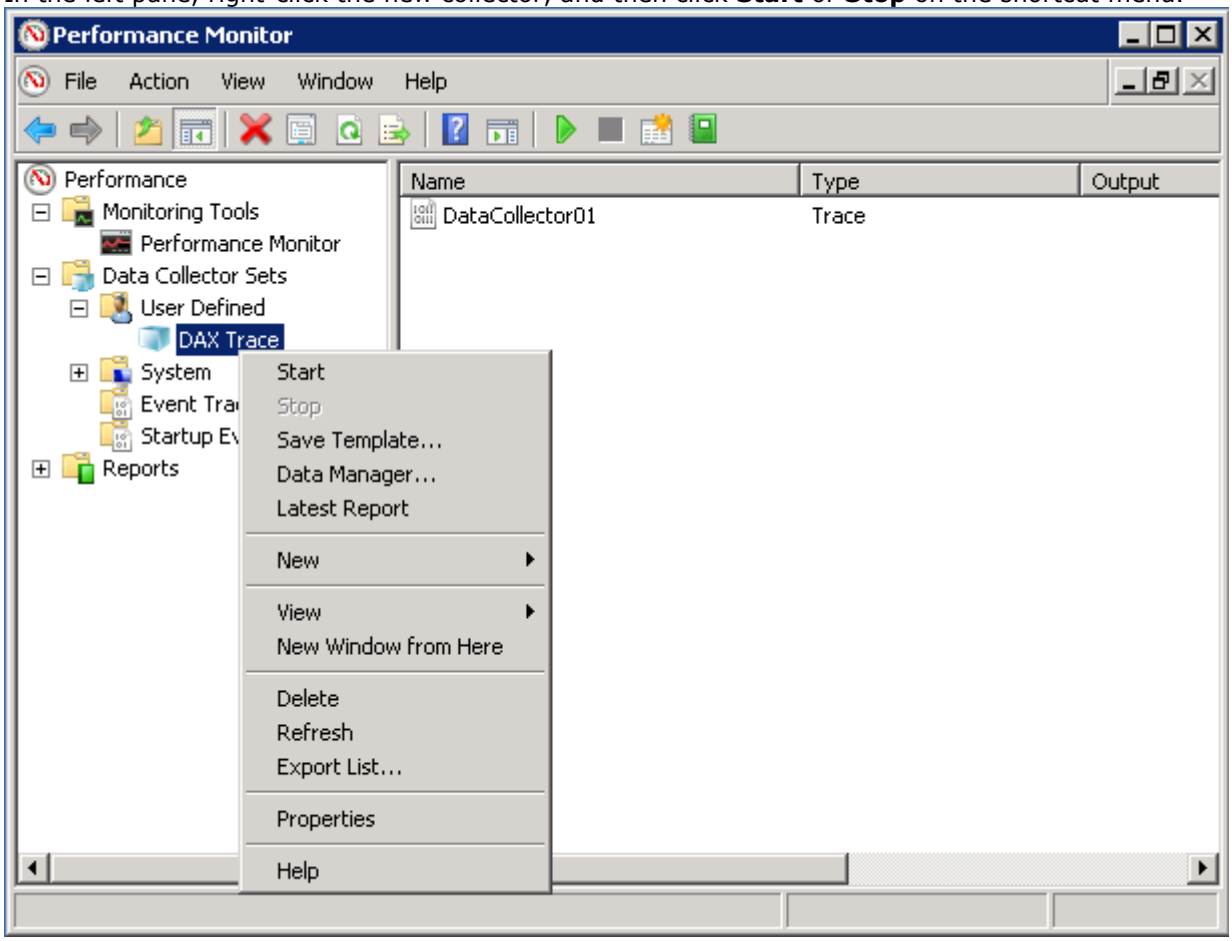
Note For a list of available keywords and the events that are associated with them, see "Appendix 1: Events reference," later in this document.

7. On the **Where would you like the data to be stored** page, change the folder where the data is saved, if needed, and then click **Finish** to complete the wizard.



After the collector is created, it can be started to write events to the trace file and stopped afterwards from the new tree node created under the **Data Collector Sets > User Defined** item:

- In the left pane, right-click the new collector, and then click **Start** or **Stop** on the shortcut menu.

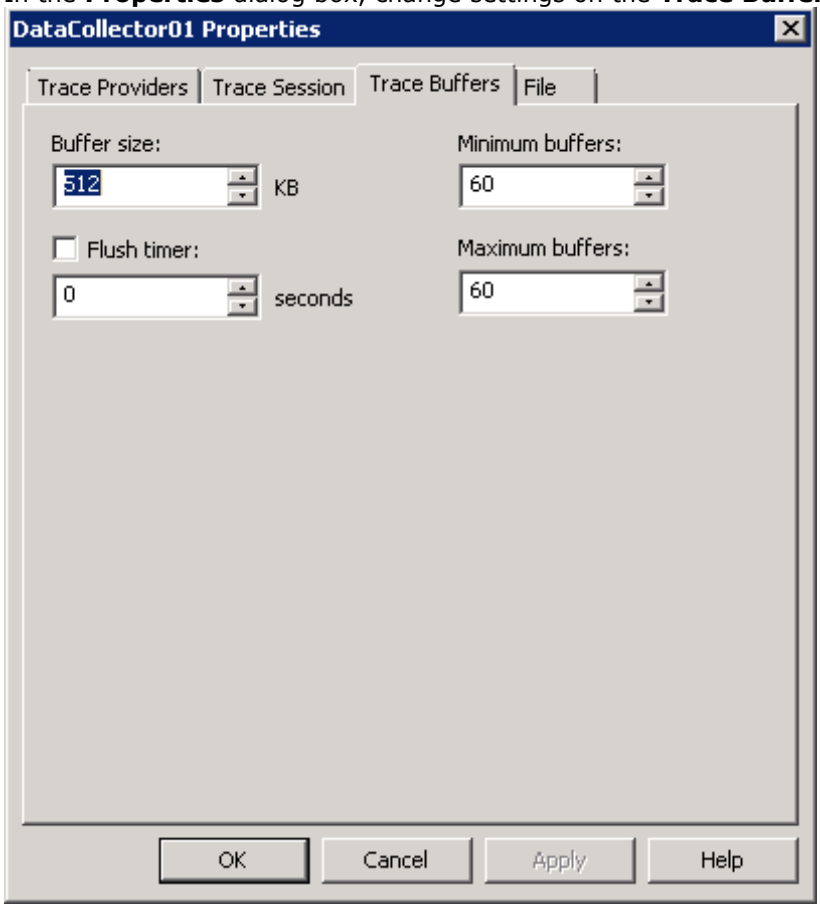


Data collector properties can be changed to gather other kinds of information such as performance counters, other Microsoft Dynamics AX events, or events from other applications and system components.

Also, you may need to modify trace settings to increase buffer size or count to make sure there are no dropped events when recording traces of operations that produce large amount of data:

1. In the left pane, right-click the new collector, and then click **Properties** on the shortcut menu.

2. In the **Properties** dialog box, change settings on the **Trace Buffers** tab.



These settings can be adjusted to balance the memory and storage performance required to record traces without dropping events during specific scenarios. In general, a higher volume of events requires more and larger buffers. For example, the following settings are recommended for data collectors that include X++ interpreter events or data access events.

Parameter	Value
Buffer size	512
Minimum buffers	60
Maximum buffers	60

Using the LogMan tool to record traces

The logman.exe command line tool that is distributed with Windows can be used to create and control trace collection:

1. Run the following command to create a data collector set. In the command, "DAX Trace" is the name to be given to collector, and the other parameters specify which keywords to enable in which provider:

```
logman.exe create trace "DAX Trace" -p Microsoft-DynamicsAX-Tracing 2
```

Parameter	Description
"Dax Trace"	The name of the trace collector to create
Microsoft-DynamicsAX-Tracing	The trace provider name
2	The keyword to enable. See the list of available keyword values and events that are associated with them in "Appendix 1: Events reference," later in this document.

2. Run the following command to start tracing:

```
logman.exe start "DAX Trace"
```

3. Run the following command to stop tracing after executing actions that you want to analyze:

```
logman.exe stop "DAX Trace"
```

Note The data collector sets that are created by using the Performance Monitor console can be manipulated by using the logman.exe tool and vice versa.

Using the XPerf tool to record traces

XPerf.exe is a command-line tool that is distributed as part of the Windows Performance Analysis toolkit in the Windows SDK. It can be used to control and process trace data. This tool is especially useful for recording information about Windows kernel activity along with Microsoft Dynamic AX events. To record a trace:

1. Run the following command to create and start a logger:

```
Xperf.exe -start "DAX Trace" -on Microsoft-DynamicsAX-Tracing:2 -f DAXTrace.etl
```

Parameter	Description
"DAX Trace"	The name of the logger
Microsoft-DynamicsAX-Tracing	The trace provider name
2	The keyword to enable. See the list of available keyword values and events that are associated with them in "Appendix 1: Events reference," later in this document.
DAXTrace.etl	The name of the file that the log is stored in

2. While the logger is active, its status can be queried by using the XPerf –loggers parameter:

```
Xperf.exe –loggers "DAX Trace"
```

3. To stop recording, execute the following command:

```
XPerf.exe –stop "DAX Trace"
```

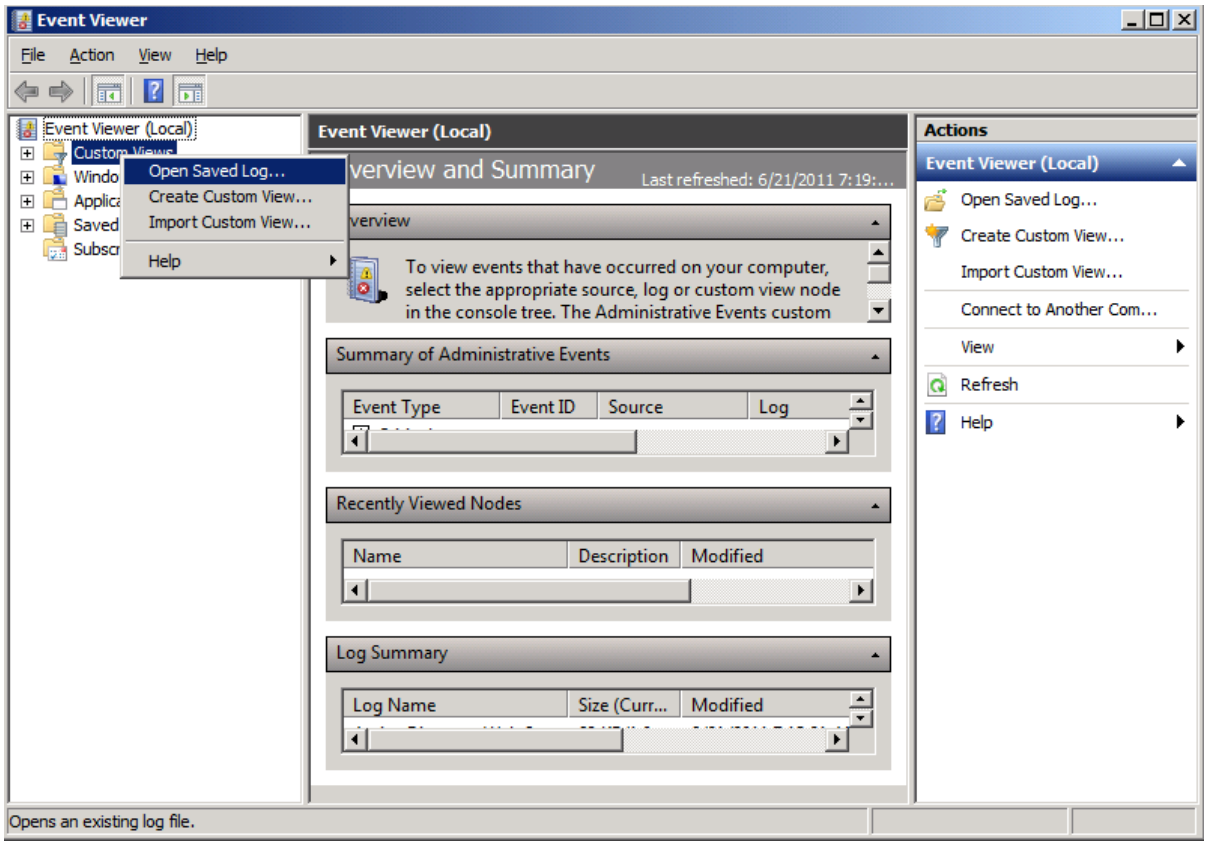
Viewing traces

Any tool that can process events created by using the Windows Events framework can be used to view Microsoft Dynamics AX event traces.

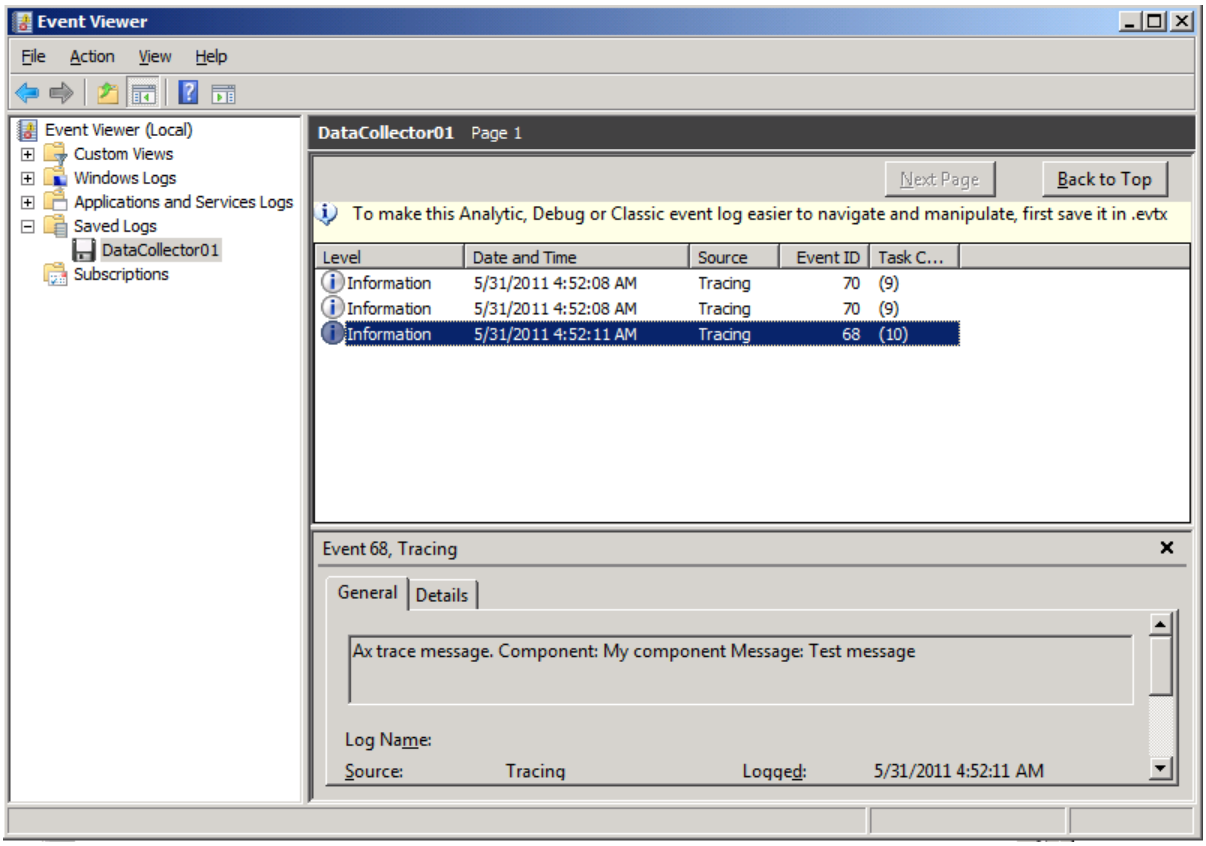
Using Event Viewer to view traces

The Event Viewer console that is available in Windows can be used to view the log files:

1. Open the Event Viewer by running eventvwr.msc.
2. In the left pane, right click **Custom Views**, click **Open Saved Log** on the shortcut menu, and then select the log file that was specified when the data collector was created.



The log can then be viewed in the same way as standard windows event logs.



Using the TraceRpt and XPerf tools to view traces

The tracerpt.exe tool that is available with Windows can be used to convert .etl files into other formats, such as .csv or .xml files. For example, the following command will convert an .etl trace into a trace.xml file:

```
tracerpt "c:\PerfLogs\Admin\DAX Trace_000001.etl" -o trace.xml -of XML
```

The contents of the resulting .xml files are similar to the following:

```
- <Events>
  + <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  - <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
    + <System>
    + <EventData>
    - <RenderingInfo Culture="en-US">
        <Level>Information</Level>
        <Opcode>Info</Opcode>
      - <Keywords>
          <Keyword>TraceInfo</Keyword>
        </Keywords>
        <Task>AxAppTask</Task>
        <Message>Ax trace message. Component: infolog Message: Test message</Message>
        <Channel>Analytic</Channel>
      </RenderingInfo>
    </Event>
  </Events>
-
```

The XPerf.exe tool can be used to manipulate or analyze .etl files as well. For example, the following command will convert an .etl trace into .csv format:

```
Xperf.exe -i DAXTrace.etl -o DAXTrace.csv
```

Using the XPerfView tool to view traces

The XPerfView utility can be used to view traces that include Windows kernel events in way that is easy to understand, and can show Microsoft Dynamics AX events as well.

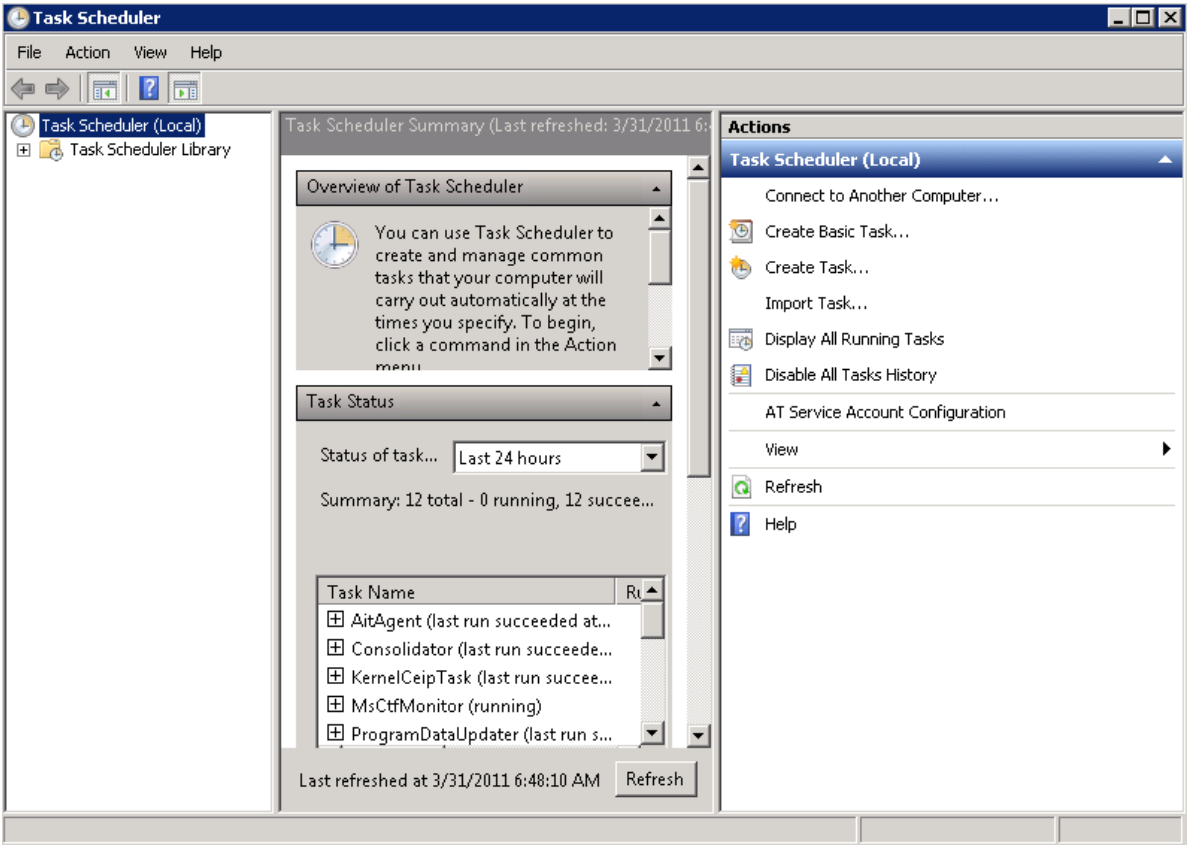
See the "Relevant links" section later in this document for more information about this tool.

Triggering tasks on events

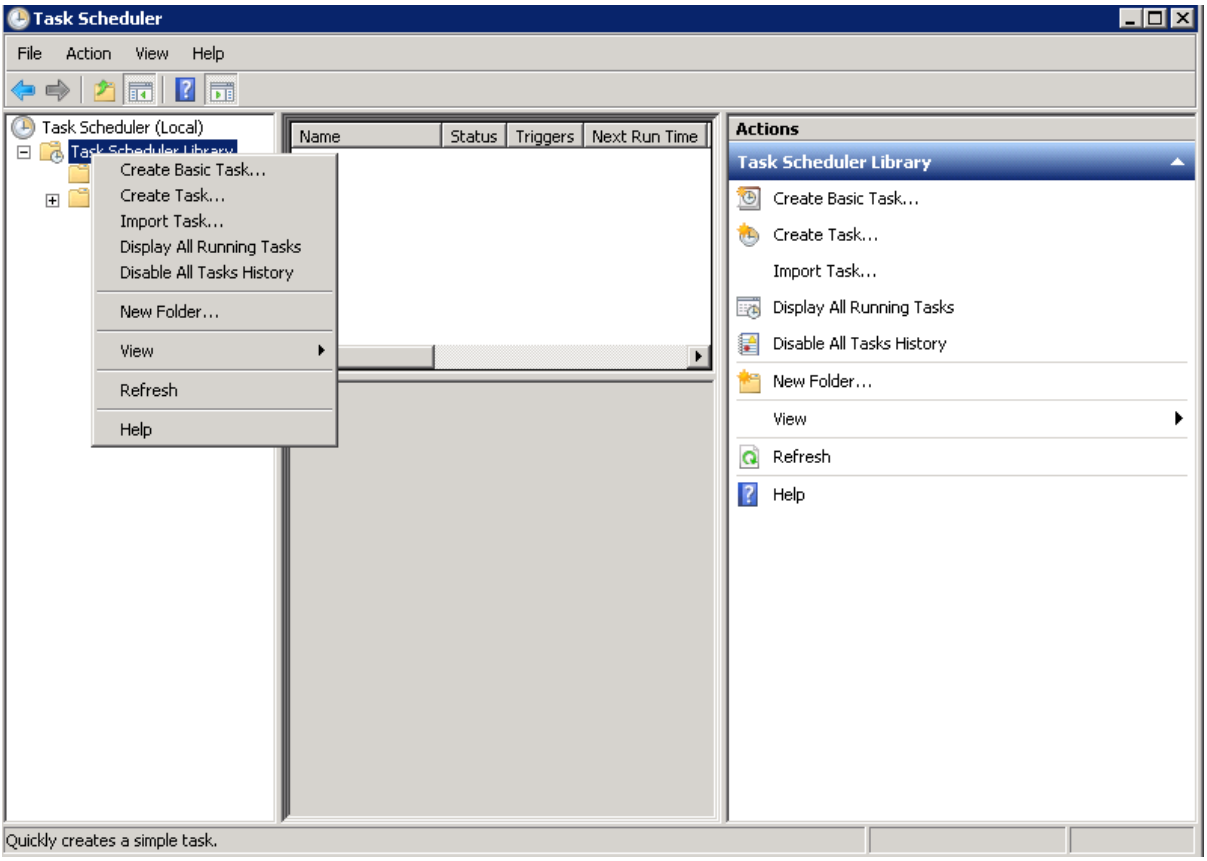
Events that are generated by Microsoft Dynamics AX can be used to trigger tasks in Windows Task Scheduler. For example, this functionality can be used to inform the administrator when a certain event happens, to start detailed diagnostic tracing, or to restart failing services.

It can be configured by using the Task Scheduler console that is available in Windows:

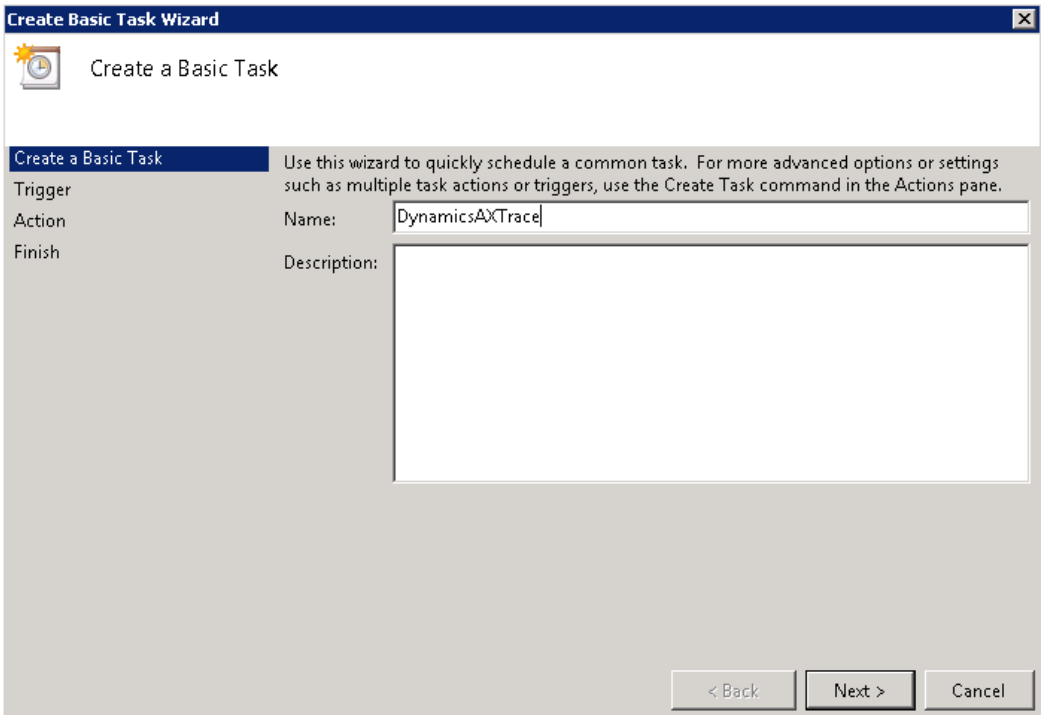
1. Open the Task Scheduler console by running taskschd.msc.



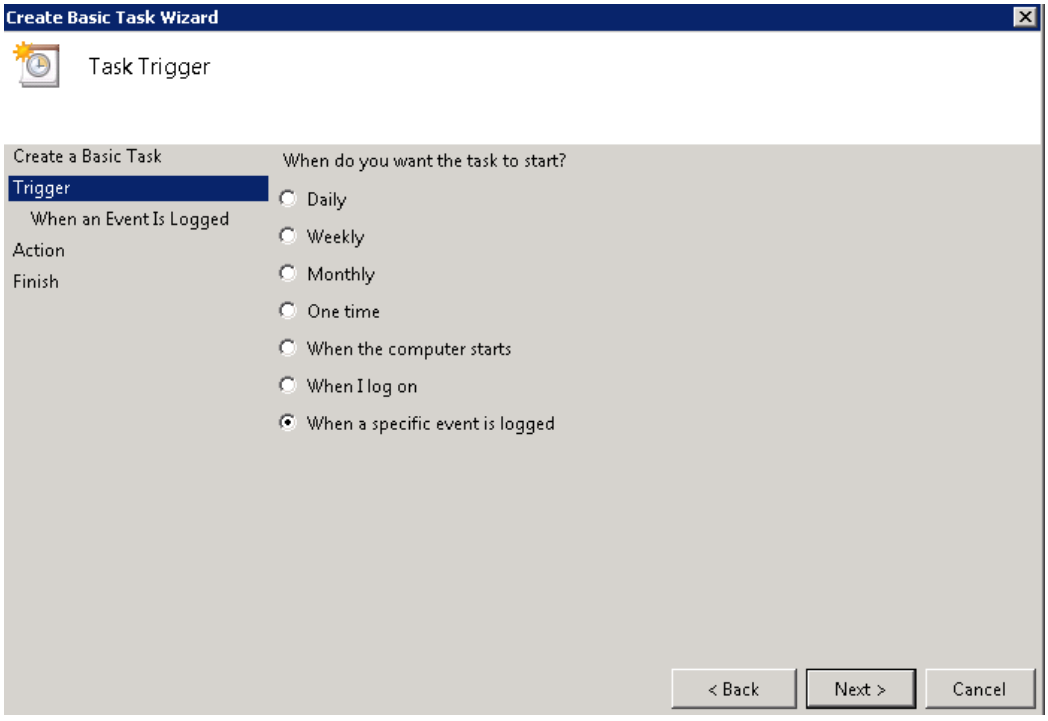
2. In the left pane, right-click **Task Scheduler Library**, and then click **Create Basic Task** on the shortcut menu to start the Create Basic Task Wizard.



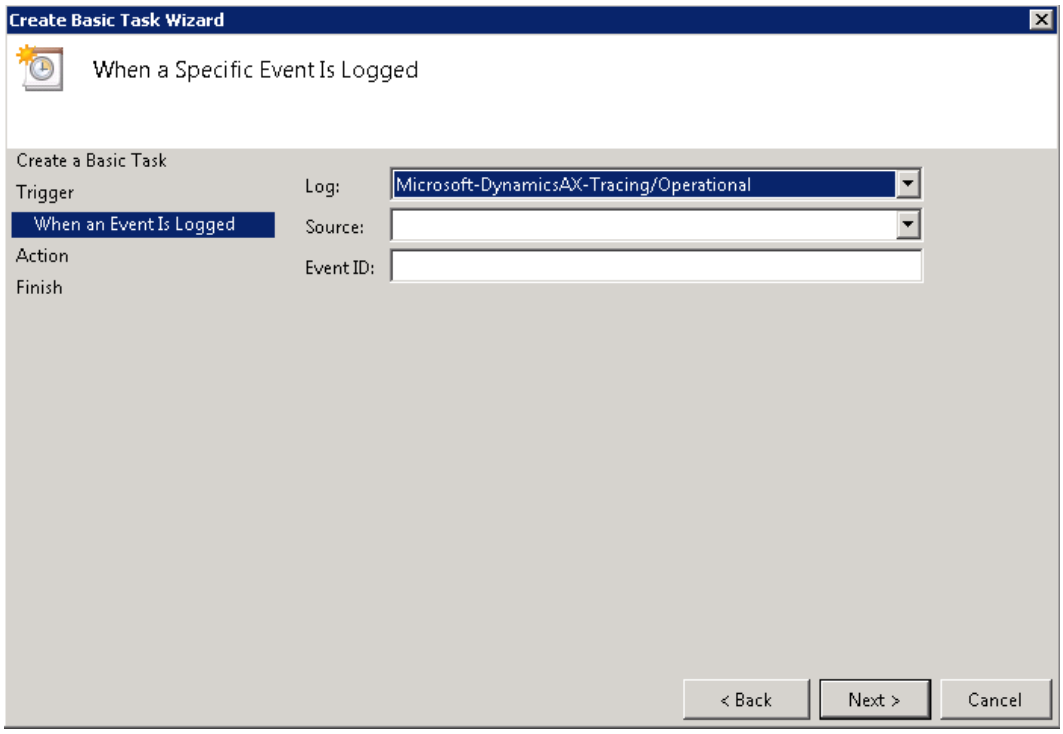
3. On the **Create a Basic Task** page, enter a task name, and then click **Next**.



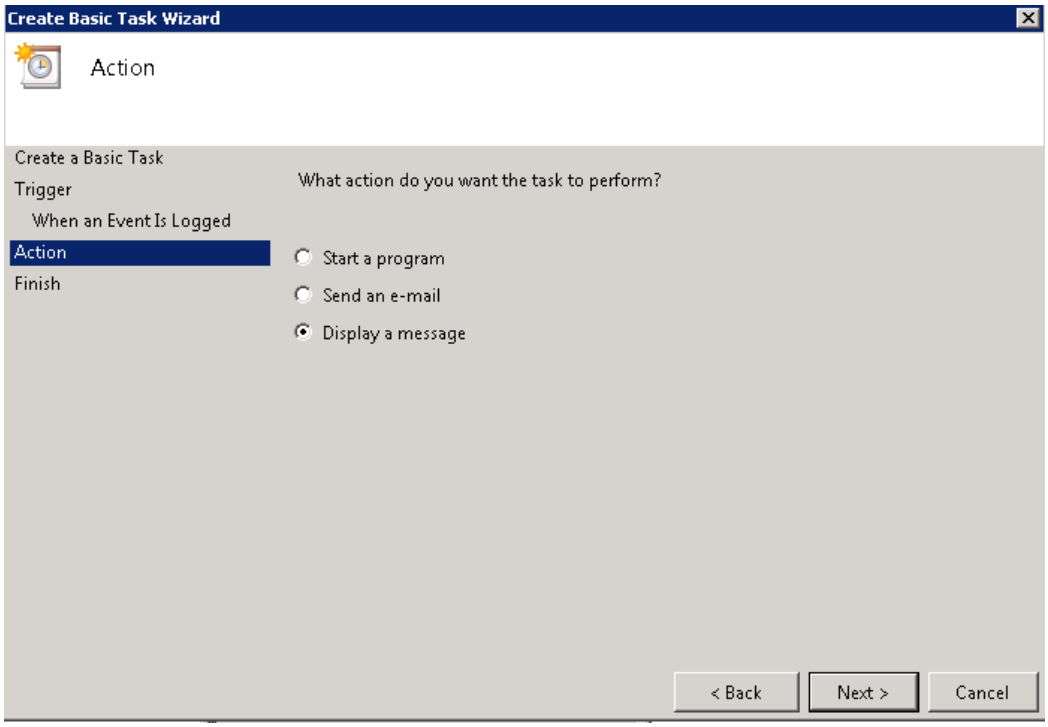
4. On the **Task Trigger** page, select the **When a specific event is logged** option button to specify the task trigger, and then click **Next**.



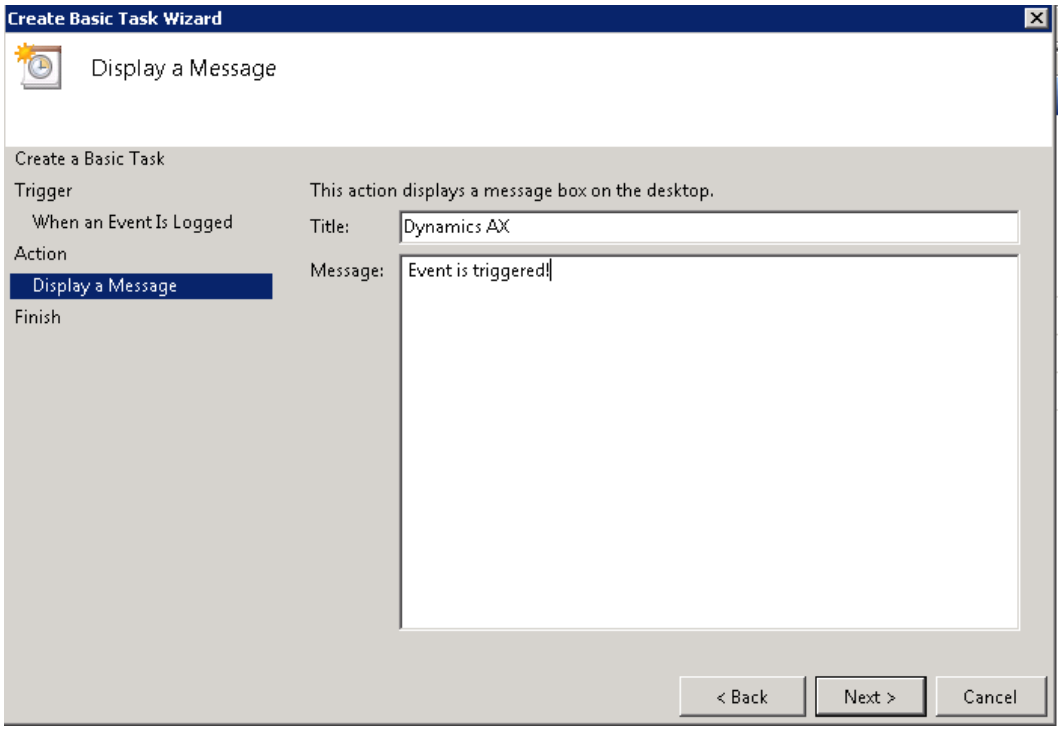
5. On the **When a Specific Event Is Logged** page, in the **Log** field, select the Microsoft Dynamics AX tracing channel, and then click **Next**.



6. On the **Action** page, select an option button for an appropriate action, for example, **Display a message**.



7. On the next page, configure the selected action and click **Next**. On the following page, click **Finish** to complete the wizard.



Now any events that are logged into the "Operational" channel cause a message box to be displayed. Make sure that this channel is enabled by using the Event Viewer console, because this is required to trigger tasks.

A created task can be edited by using the Task Scheduler console to provide a more granular filter that specifies which events must trigger the task and other parameters.

Instrumenting application code

In addition to built-in instrumentation, Microsoft Dynamics AX exposes several methods that allow solutions to be built on top of it to create basic events. The methods can be found in the xClassTrace class exposed to X++.

To instrument custom application code, you only need to add method calls that log messages you are interested in. After that, tracing can be configured by using standard tools as described earlier with no additional coding.

xClassTrace::logComponentMessage

Calling this method from your code triggers an event, putting the provided message into the log if tracing is enabled as an AxTraceComponentMessageEvent event. If tracing is disabled for the channel and keywords that this event belongs to, the xClassTrace::logComponentMessage method returns immediately and has negligible performance impact.

Parameter	Description
component	A string that identifies the component that triggered the event. Use it to distinguish different solutions or feature areas within a single solution.
message	A message string that details what happened.

Example:

```
xClassTrace::logComponentMessage('MyCompanySolution', strfmt('Processing item id %1', itemId));
```

xClassTrace::isTracingEnabled

This method returns true if tracing is enabled. It can be used to eliminate any performance impact of instrumentation by making sure that an event message is only generated when it is going to be used.

Note This method only needs to be used if there is significant overhead in collecting the information that is required to create event message or other parameters. If there is no such overhead, it is enough to call the xClassTrace::logComponentMessage method, because calling it has no performance impact when tracing is disabled.

Parameter	Description
level	Determines the status of what event level (information messages, warnings, or errors) must be enabled. Returns true if any level is enabled by default.
keyword	Determines which tracing keywords must be checked. Returns true if tracing is enabled for any keyword by default. See the list of available keyword values and events that are associated with them in "Appendix 1: Events reference," later in this document.

Example:

```
if (xClassTrace::isTracingEnabled())
{
    eventMessage = collectEventData(); // generate event message
    xClassTrace::logComponentMessage('MyCompanySolution', eventMessage);
}
```

xClassTrace::start

Calling this method enables tracing and starts saving events into a log file. It can be used to automate tracing parts of the application that you are interested in during development.

Parameter	Description
logFileName	The name of the file to save the log into. The log file is created under the log folder on the client or AOS system, depending on where this method is executed.
logBufferSize	The buffer size in kilobytes to use when recording events. Larger buffer sizes help make sure that events are not dropped during recording. The default value is 512.
minBuffers	The minimum number of buffers to allocate. This value specifies how many buffers are allocated at all times when the recording session is active. Higher numbers of buffers help make sure that events are not dropped during recording. The default value is 60.
maxBuffers	The maximum number of buffers to allocate. This value specifies how many buffers can be allocated when the minimum number is not enough to handle all incoming events. Higher numbers of buffers help make sure that events are not dropped during recording. The default value is 60.
Keywords	What tracing keywords to enable. See the list of available keyword values and events that are associated with them in "Appendix 1: Events reference," later in this document. All keywords are enabled by default.
maxFileSize	The maximum size in megabytes of the log file. Specifying higher values here allows tracing to run longer before the log file is filled. The default value is 1000.

Example:

```
xClassTrace::start('MyLogFile.etl');
execute(); // run the code being debugged
xClassTrace::stop();
```

xClassTrace::stop

Calling this method stops a tracing session that was started by calling the xClassTrace::start method.

Relevant links

Refer to the links below for more information about the Windows Events tracing functionality:

- Windows events reference: http://msdn.microsoft.com/en-us/library/aa964766.aspx
- XPerf and other trace analysis tools: http://msdn.microsoft.com/en-us/performance/default.aspx

Appendix 1: Events reference

The following tables list events that are generated by Microsoft Dynamics AX, and specify the keywords and channels that they are associated with.

1. Microsoft-DynamicsAX-Admin channel:

Event	ID	Keyword
ComponentInstallationStart	75	Setup
ComponentInstallationSuccess	76	Setup
ComponentInstallationWarning	77	Setup
ComponentInstallationError	78	Setup
InstallationStart	79	Setup
InstallationSuccess	80	Setup
InstallationError	81	Setup

2. Microsoft-DynamicsAX-Analytic channel:

Event	ID	Keyword
OutOfMemoryError	0	ServerHealth
SecurityTPFAuthz	11	Database, Security
TableLoading	12	Caching, AOT
ClassLoading	13	Caching, AOT
UnitOfWorkAddClone	14	Database
UnitOfWorkConflict	15	Database
UnitOfWorkSaveChangesBegin	16	Database
UnitOfWorkSaveChangesEnd	17	Database
UnitOfWorkStateChange	18	Database
UnitOfWorkPack	19	Database
UnitOfWorkUnpack	20	Database
UnitOfWorkTopologicalOrder	21	Database
UnitOfWorkDbOperation	22	Database
ViewDictViewComputedColumnMethodCalled	23	Database
ViewDictViewComputedColumnMethodFailed	24	Database
ViewDictViewComputedColumnMethodSucceeded	25	Database
ViewComputedColumnMethodCalled	26	Database
ViewComputedColumnMethodFailed	27	Database
ViewComputedColumnMethodSucceeded	28	Database
ViewDDLGenerationStarted	29	Database
ViewDDLGenerationFinished	30	Database
ViewFieldListGenerationStarted	31	Database
ViewFieldListGenerationFinished	32	Database
DataAccessCaching	33	Caching, Database
DataAccessSCSCCaching	34	Caching, Database
QueryFilterOperationEvent	35	Database
SecurityAPIInvoked	36	Security

SecurityEntryPointAuthz	37	Security
SecurityLoadRolePermissions	38	Database, Security
SecurityLoadUserRolesDatabase	39	Security
SecurityMenuItemInvoked	40	AOT, Security
TimeZoneConversion	43	
SecurityWebEntryPointAuthorization	44	EP, AOT, Security
SecurityWebUserRoles	45	EP, AOT, Security
WebFrameworkException	46	EP
SCscInsertRecordSetOrder	50	
AxTransactionBeginEvent	51	XppMarker
AxTransactionEndEvent	52	XppMarker
AxTraceMessageEvent	53	TraceInfo
CSRoundTripBegin	54	RPC
CSRoundTripEnd	55	RPC
XppMethodBegin	56	XPP
XppMethodEnd	57	XPP
XppUtilFunc	58	AxUtilFunc
AxSqlStmtEvent	59	SQL
DBConnect	60	DatabaseDetailed
DBDisconnect	61	DatabaseDetailed
TTSBegin	62	TTS
TTSCommit	63	TTS
TTSAbort	64	TTS
SqlInputBind	65	BindParameter
SqlRowFetch	66	DatabaseDetailed
SqlRowFetchCumu	67	DatabaseDetailed
AxTraceComponentMessageEvent	68	TraceInfo
SecurityAccessDenied	69	Security
TraceInformation	70	BindParameter, TTS, DatabaseDetailed, SQL, XPP, RPC, TraceInfo, XppMarker
SetupLogMessage	71	
SetupLogError	72	
ServerStartup	73	Database
ModelStoredProc	74	SQL
ModelSqlInputBind	94	BindParameter
AssemblyCacheElementBegin	98	Caching
AssemblyCacheElement	99	Caching

AssemblyCacheElementEnd	100	Caching
SecurityScScFilteredAllTables	101	Security
SecurityEmptyCrossCompanyFilter	102	Security
SecurityLoadRolePermission	103	Security
SecurityLoadUserRolesCache	104	Security
SecurityEntryPointAuthzFailed	105	Security
XppParameterInfo	106	XPPParm

3. Microsoft-DynamicsAX-ClientAccess channel:

Event	ID	Keyword
ClientEventFormOpen	95	
ClientEventFormClose	96	
ClientEventControlClicked	97	

4. Microsoft-DynamicsAX-Operational channel:

Event	ID	Keyword
MRPWrkCtrSchedulerLoadOrder	82	MRP
MRPWrkCtrSchedulerSaveJobs	83	MRP
MRPError	84	MRP
MRPReqProcessStatus	85	MRP
MRPPartitionOrders	86	MRP
MRPBOMLevel	87	MRP
MRPWrkCtrSchedulerSaveDerived	88	MRP
MRPReqTaskController	89	MRP
MRPActionMessage	90	MRP
MRPNetResourceScheduler	91	MRP
MRPItem	92	MRP
MRPReqTask	93	MRP

The following values can be used when specifying keywords. Most of the tools allow more than one keyword to be specified by adding them together:

Keyword	Value
XppMarker	0x0000000000000001
TraceInfo	0x0000000000000002
RPC	0x0000000000000004
XPP	0x0000000000000008
SQL	0x0000000000000010
DatabaseDetailed	0x0000000000000020
TTS	0x0000000000000040

BindParameter	0x0000000000000080
AxUtilFunc	0x0000000000000100
XPPParm	0x0000000000000200
Setup	0x0000010000000000
MRP	0x0000020000000000
EP	0x0000040000000000
ServerHealth	0x0000080000000000
Caching	0x0000100000000000
AOT	0x0000200000000000
Database	0x0000400000000000
Security	0x0000800000000000

More details can be found by examining the TraceProviderCrimson.man manifest file that is installed with Microsoft Dynamics AX. The ECManGen tool available in Windows SDK can be used to view the manifest file.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics

Microsoft