SQL Sentry Event Architecture

Last Modified on 06 July 2021

The concepts and terminology covered in this section are important for a full understanding of SQL Sentry software.

Earlier versions of SQL Sentry focused primarily on SQL Server agent jobs; however, this is no longer the case. Event manager's new Event Connector Architecture now enables monitoring and management of different types of scheduling systems and various types of jobs, tasks, and status events.

This necessitated some dramatic changes to both our system architecture and terminology. As a result, throughout this guide more generic terms such as event store connection and event object (or simply connection and object) are used where before references to SQL Servers or jobs were used. The following topics cover these architectural and terminology changes in detail.

Event Connector Architecture

SQL Sentry's Event Connector System encompasses the entire process by which SQL Sentry collects historical and active status information from various event stores, and translates the data into the calendars, lists, and graphs in the SQL Sentry client. The data collected is also used to trigger the various actions carried out by the SQL Sentry monitoring service, such as sending email notifications.

Execution agents trigger execution engines that write data to event stores. There are many different types of agents, engines, and stores, and it's important to remember that this process isn't new, and isn't something we invented. However, SQL Sentry's primary focus is collecting data from the respective stores, translating it, correlating it, rendering it in a meaningful fashion to the DBA, and enabling automated actions based upon it. We've had to define some terms that abstract these components to make the process by which SQL Sentry collects and uses this data more understandable. Creating new terms isn't something we do for fun, but rather something we've had to do here simply because all other existing enterprise schedulers utilize their own proprietary execution agents and engines. Because they don't wrap other agents and engines as we do, there probably wasn't much of a need for these terms before now.

We believe our agent-less approach is superior to the proprietary scheduling agent approach employed by enterprise schedulers in many ways, including significantly reduced installation, configuration, and maintenance overhead, as well as the ability to integrate with systems such as the SQL Server agent in a much more native fashion. For example, we don't work with SQL agent jobs from the command line using *osql.exe* as other schedulers do, instead we talk directly to SQL Server in its native languages (DMO, SMO, ADO.NET, etc.). In general, we let existing scheduling agents do what they do best, schedule jobs and tasks, and we don't try to replace them.

The part of the process that's proprietary to SQL Sentry is how it uses event providers to access and translate the event store data over event store connections into the event objects used by the SQL Sentry client and Server. We aren't just dealing with SQL agent jobs any more, so we've had to come up with a few terms to abstract these components to make them more understandable and manageable. It's important to note that SQL Sentry is a 100% .NET-based application, so this object-oriented approach is very much in line with how we've designed the software, and in many ways these concepts and terms are descended from it.

Execution Agents

Execution agents represent any process responsible for the execution of one or more execution engines.

There are two types of execution agents: schedulers and ad hoc.

Schedulers

Schedulers trigger execution engines on an automated basis at predetermined times and/or intervals. There are many different types of scheduling agents; there are those like SQL agent and task scheduler that can be used by anyone to schedule jobs and tasks. There are also proprietary schedulers. These are schedulers that exist as part of some application but don’t operate as standalone schedulers, and therefore can’t be used to schedule tasks unrelated to the application itself. Many anti-virus systems, disk defragmentation systems, and network backup systems fall into this category.

The schedulers that SQL Sentry is most interested in are those executing tasks that can impact SQL Server performance and have dependency relationships with other SQL Server-related jobs or tasks.

Schedulers always have job (or task) history logs that record data such as start time, end time, duration, and job output. If the Scheduler allows ad hoc (unscheduled) execution of its jobs, this is also normally logged. Examples of this are manually executed SQL agent jobs and Windows task scheduler tasks that are written to the respective history logs just as scheduled executions are.

The SQL Server agent isn't only a scheduling agent, it has job subsystems that manage job steps and schedules, but it also contains an alert engine that detects various conditions of the SQL Server process and performance counters, and it can auto-restart a failed SQL Server agent or SQL monitoring service.

Schedulers sometimes have their own separate logs that record general information, errors, or warning conditions not necessarily related to jobs. A good example of this is the SQL Server agent log. Unlike the SQL Server agent, the Windows task scheduler doesn't have its own log, only a task history log. In the case of the SQL Server agent, SQL Sentry collects data from both the SQL Server agent log itself as well as its job history log.

Ad Hoc

Non-scheduled executions are triggered by ad hoc agents, which include users and response-based systems. When a user executes a job manually, it's considered an ad hoc execution. Likewise, when some other active process detects a transient condition in the environment and triggers an execution engine, it's considered a response-based ad hoc execution.

SQL Sentry is a great example of a response-based system; it's able to detect various conditions related to event sources and trigger a variety of pre-defined actions in response. For more information, see the Alerting and Responses Overview topic.

In the case where a user executes an engine directly, there are no scheduler logs, and SQL Sentry is dependent on the engine’s own execution logs if they exist. If the engine doesn't log or support active status queries from an API or other interface, then SQL Sentry can’t do anything with it; we can’t show the event instances on a calendar, trigger conditions, and actions, and/or monitor performance for the event. An example of this is an SSIS package that is run from within SSMS without package logging enabled, or a manually executed VBScript (.vbs) file that doesn’t perform any logging.

The most important point to remember regarding execution agents is that if the scheduler and/or the execution engine logs event data, and/or either allows active status queries, then SQL Sentry can work with it.

Execution Engines

An execution engine is a name for any application or program that does some type of work, typically background processing. Examples include the following:

- **SQL Agent Jobs**
- **sqlmaint.exe**—The executable responsible for SQL agent maintenance plan functions.
- **ReportServer service**—A Windows service that processes **Reporting Services** reports and produces associated output.
- **Task Scheduler**—A Windows service responsible for running tasks.
- **Disk defragmentation software**—An executable that defragments logical disk drives.
- **Anti-virus software**—An executable that scans a system for viruses.

One of the primary differences between execution engines and other programs is that they usually run as background processes, meaning they quietly go about their work without any significant user interface or user interaction. Execution engines aren't like UI-driven productivity applications like Microsoft® Word® or Excel®.

Most execution engines throw off status information, typically in the form of an execution log. This element is key for many of the functions of SQL Sentry, including notifications, chaining, queuing, etc. If an execution engine doesn't log status information to its own execution log, then SQL Sentry defaults to use the logs of the associated scheduler. The fact that it doesn't have an execution log doesn't mean that it's not an execution engine, just that for SQL Sentry’s purposes there isn't much that can be done with it; we need some type of execution log data with an associated provider to show the event instance on the calendar, trigger conditions and actions, monitor performance, etc.

Event Stores

An event store is a collection of like event instances kept in a common location, typically a database or file-based storage. Some examples are shown in the following table:

Event Provider	Provider Type	Event Type	Event Store
SQL Server Agent	Scheduler	Status	File Storage
SQL Server Agent Job	Execution Engine	Active	SQL Server Database (MSDB)
Windows Task Scheduler	Scheduler	Active	Windows API + FIle Storage
Windows Event Log	Event Log	Status	Windows API

Event Logs

Event logs are a type of event store that hold only status event instances. The Windows event logs, SQL Server logs, and SQL Server agent logs are examples of event logs.

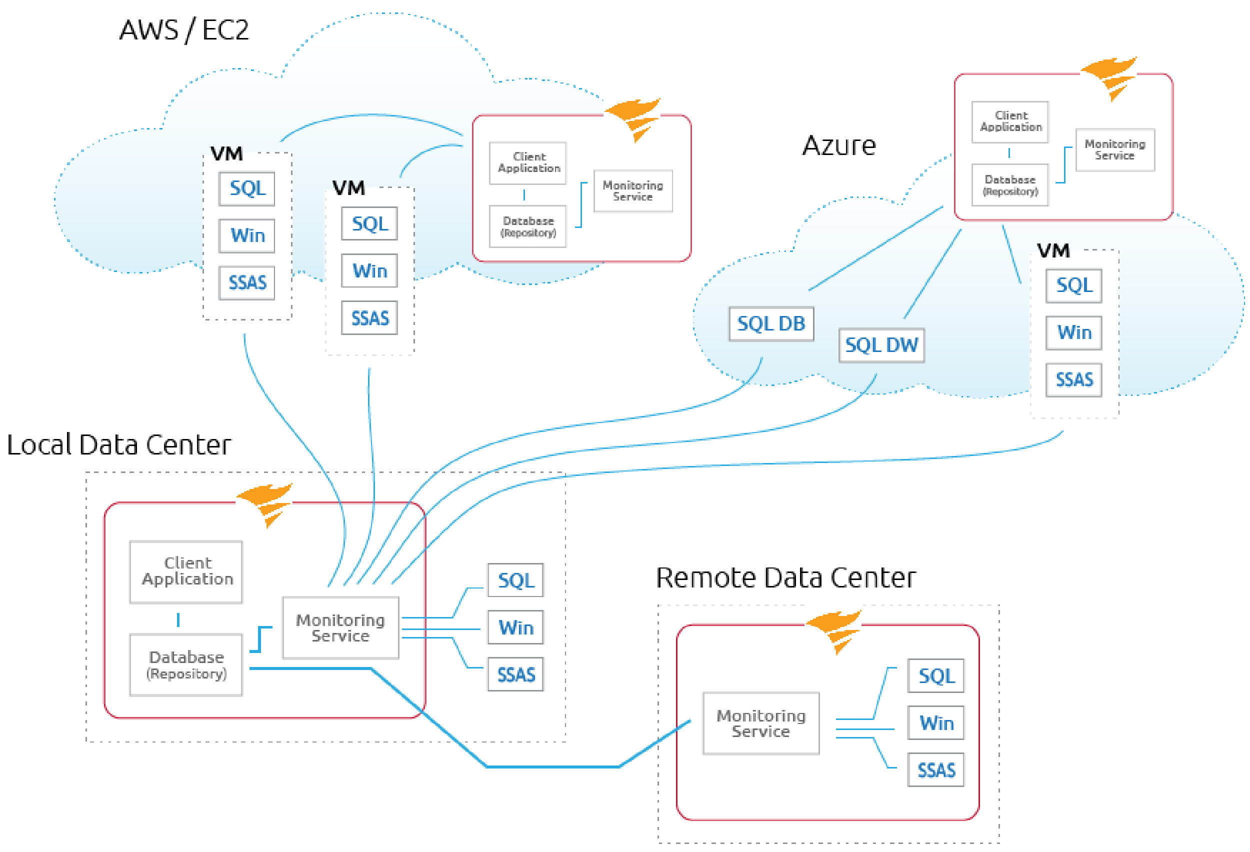
Event Store Connections

An event store connection, usually referred to as a connection, is used by event sources to access data in event stores from some network transport mechanism.

In the SQL Sentry client, each **SQL Server** and **Windows Instance** node represents a single connection.

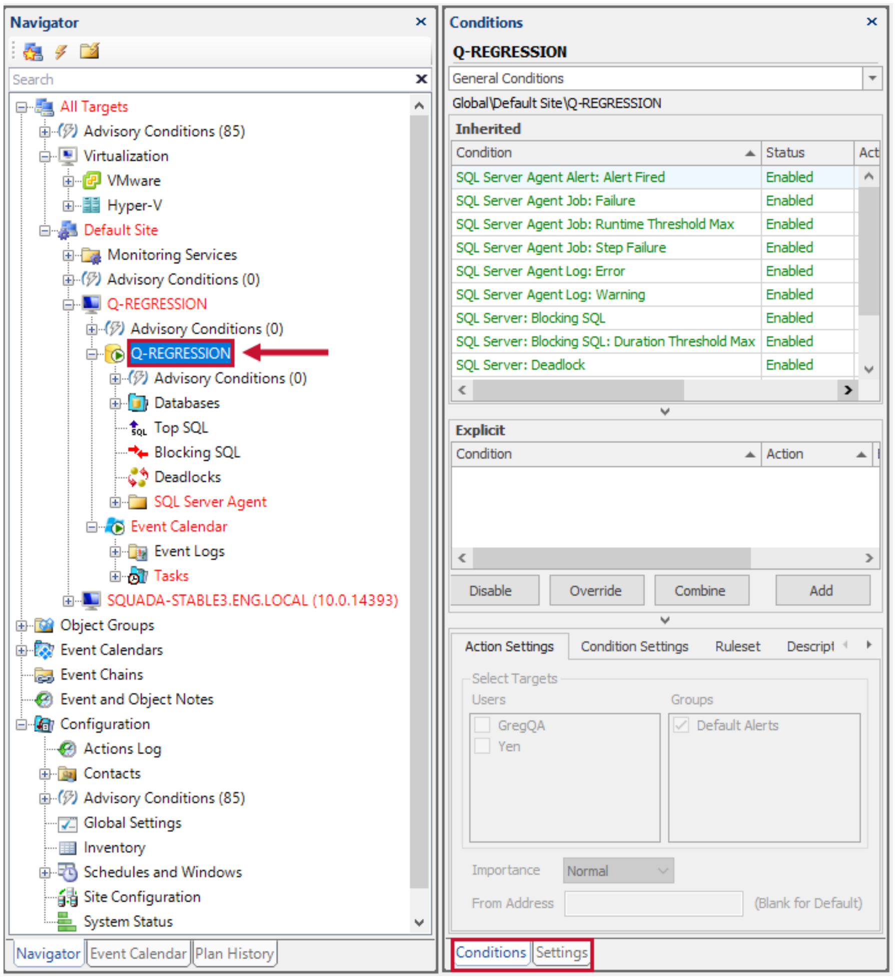
For example, the previous jobs event source utilizes the SQL Server agent job event provider over an *ADO.NET* connection to the SQL Server to access event data stored in the MSDB database.

For maximum efficiency and performance, SQL Sentry allows a connection to be shared by multiple event sources simultaneously; all event sources under the **SQL Server Instance** node above (jobs, maintenance plans, alerts, reporting services, and SQL agent log) utilize the same physical network connection. This concept is illustrated in more detail in the following Enterprise Architecture Diagram.



Configuring Instance Level Settings

Whenever you select an **Instance** node in the **Navigator** pane, the **Conditions** and **Settings** pane automatically refreshes to display all conditions, actions, and settings for the selected instance. Any changes made here apply to the selected instance only, overriding any globally configured settings.



Conditions

Q-REGRESSION

General Conditions

Global\Default Site\Q-REGRESSION

Inherited

Condition	Status	Act
SQL Server Agent Alert: Alert Fired	Enabled	
SQL Server Agent Job: Failure	Enabled	
SQL Server Agent Job: Runtime Threshold Max	Enabled	
SQL Server Agent Job: Step Failure	Enabled	
SQL Server Agent Log: Error	Enabled	
SQL Server Agent Log: Warning	Enabled	
SQL Server: Blocking SQL	Enabled	
SQL Server: Blocking SQL: Duration Threshold Max	Enabled	
SQL Server: Deadlock	Enabled	

Explicit

Condition	Action

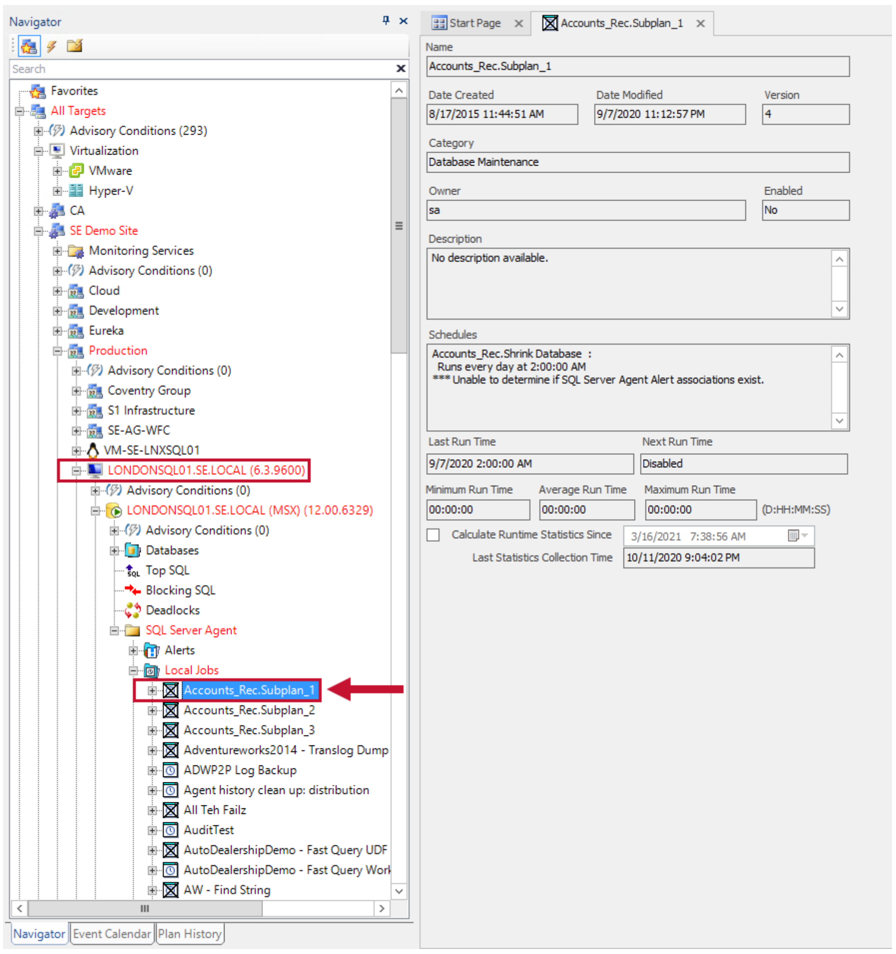
Select Targets

Users	Groups
☐ GregQA	☑ Default Alerts
☐ Yen	

Importance: Normal

From Address: (Blank for Default)

Instance level settings can be overridden at the object level by selecting an **Object** node in the **Navigator** pane.



Name: Accounts_Rec.Subplan_1

Date Created: 8/17/2015 11:44:51 AM
Date Modified: 9/7/2020 11:12:57 PM
Version: 4

Category: Database Maintenance

Owner: sa
Enabled: No

Description: No description available.

Schedules:
Accounts_Rec.Shrink Database :
Runs every day at 2:00:00 AM
*** Unable to determine if SQL Server Agent Alert associations exist.

Last Run Time: 9/7/2020 2:00:00 AM
Next Run Time: Disabled

Minimum Run Time: 00:00:00
Average Run Time: 00:00:00
Maximum Run Time: 00:00:00 (D:HH:MM:SS)

Calculate Runtime Statistics Since: 3/16/2021 7:38:56 AM
Last Statistics Collection Time: 10/11/2020 9:04:02 PM

Event Providers

An event provider is a module that enables SQL Sentry to communicate with and translate data from an event store over an event store connection into a common format that can be used by the various parts of the SQL Sentry system, including calendars, graphs, and notifications. A SQL Sentry event provider is analogous to an OLEDB provider or ODBC driver. There are three types of event providers: schedulers, execution engines, and event logs.

Event Provider	**Description**
Scheduler	Enables interfacing with the native scheduler logs, such as for SQL Server agent or Windows task scheduler.
Execution Engine	Connects with the native stores for execution engines for collecting historical information and active status information if supported by the provider. Not all providers support active status queries.
Event Log	Enables translation of status event data contained in log-type stores, such as for the SQL agent alert log or the Windows event log.

SQL Sentry Providers

Event Provider	Provider Type
SQL Server Agent	Scheduler
SQL Server Agent Job	Execution Engine
SQL Server Agent Alert	Event Log
Maintenance Plan	Execution Engine
Reporting Services	Execution Engine
Window Task Scheduler	Scheduler

SQL Sentry uses a plug-in architecture that allows future modules to be added on, typically without requiring the installation of a new version of the SQL Sentry software.