***The very first question: when do the wait stats occur?***

Here is the simple answer. When SQL Server is executing any task, and if for any reason it has to wait for resources to execute the task, this wait is recorded by SQL Server with the reason for the delay. Later on we can analyze these wait stats to understand the reason the task was delayed and maybe we can eliminate the wait for SQL Server. It is not always possible to remove the wait type 100%, but there are few suggestions that can help.

Before we continue learning about wait types and wait stats, we need to understand three important milestones of the query life-cycle.

**Running** - a query which is being executed on a CPU is called a running query. This query is responsible for CPU time.

**Runnable** – a query which is ready to execute and waiting for its turn to run is called a runnable query. This query is responsible for Signal Wait time. (In other words, the query is ready to run but CPU is servicing another query).

**Suspended** – a query which is waiting due to any reason (to know the reason, we are learning wait stats) to be converted to runnable is suspended query. This query is responsible for wait time. (In other words, this is the time we are trying to reduce).

In simple words, query execution time is a summation of the query Executing CPU Time (Running) + Query Wait Time (Suspended) + Query Signal Wait Time (Runnable). Again, it may be possible a query goes to all these stats multiple times.

Let us try to understand the whole thing with a simple analogy of a taxi and a passenger.

Two friends, Tom and Danny, go to the mall together. When they leave the mall, they decide to take a taxi. Tom and Danny both stand in the line waiting for their turn to get into the taxi. This is the Signal Wait Time as they are ready to get into the taxi but the taxis are currently serving other customer and they have to wait for their turn. In other word they are in a runnable state.

Now when it is their turn to get into the taxi, the taxi driver informs them he does not take credit cards and only cash is accepted. Neither Tom nor Danny have enough cash, they both cannot get into the vehicle. Tom waits outside in the queue and Danny goes to ATM to fetch the cash. During this time the taxi cannot wait, they have to let other passengers get into the taxi. As Tom and Danny both are outside in the queue, this is the Query Wait Time and they are in the suspended state. They cannot do anything till they get the cash.

Once Danny gets the cash, they are both standing in the line again, creating one more Signal Wait Time. This time when their turn comes they can pay the taxi driver in cash and reach their destination. The time taken for the taxi to get from the mall to the destination is running time (CPU time) and the taxi is running.

I hope this analogy is bit clear with the wait stats. You can check the Signalwait stats using following query of [Glenn Berry](http://sqlserverperformance.wordpress.com/).

[view source](http://beyondrelational.com/modules/2/blogs/121/posts/14775/sql-server-dmv-sysdmoswaitstats-explanation-wait-type.aspx#viewSource)



[print](http://beyondrelational.com/modules/2/blogs/121/posts/14775/sql-server-dmv-sysdmoswaitstats-explanation-wait-type.aspx#printSource)[?](http://beyondrelational.com/modules/2/blogs/121/posts/14775/sql-server-dmv-sysdmoswaitstats-explanation-wait-type.aspx#about)

1.-- Signal Waits for instance

2.SELECT CAST(100.0 \* SUM(signal\_wait\_time\_ms) / SUM (wait\_time\_ms) AS NUMERIC(20,2))

3.AS [%signal (cpu) waits],

4.CAST(100.0 \* SUM(wait\_time\_ms - signal\_wait\_time\_ms) / SUM (wait\_time\_ms) AS NUMERIC(20,2))

5.AS [%resource waits]

6.FROM sys.dm\_os\_wait\_stats OPTION (RECOMPILE);

Higher the Signal wait stats are not good for the system. Very high value indicates CPU pressure. In my experience, when systems are running smooth and without any glitch the Signal wait stat is lower than 20%. Again, this number can be debated (and it is from my experience and is not documented anywhere). In other words, lower is better and higher is not good for the system.

|  |  |
| --- | --- |
| **Wait type** | **Description** |
| ABR | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| ASSEMBLY\_LOAD | Occurs during exclusive access to assembly loading. |
| ASYNC\_DISKPOOL\_LOCK | Occurs when there is an attempt to synchronize parallel threads that are performing tasks such as creating or initializing a file. |
| ASYNC\_IO\_COMPLETION | Occurs when a task is waiting for I/Os to finish. |
| ASYNC\_NETWORK\_IO | Occurs on network writes when the task is blocked behind the network. Verify that the client is processing data from the server. |
| AUDIT\_GROUPCACHE\_LOCK | Occurs when there is a wait on a lock that controls access to a special cache. The cache contains information about which audits are being used to audit each audit action group. |
| AUDIT\_LOGINCACHE\_LOCK | Occurs when there is a wait on a lock that controls access to a special cache. The cache contains information about which audits are being used to audit login audit action groups. |
| AUDIT\_ON\_DEMAND\_TARGET\_LOCK | Occurs when there is a wait on a lock that is used to ensure single initialization of audit related Extended Event targets. |
| AUDIT\_XE\_SESSION\_MGR | Occurs when there is a wait on a lock that is used to synchronize the starting and stopping of audit related Extended Events sessions. |
| BACKUP | Occurs when a task is blocked as part of backup processing. |
| BACKUP\_OPERATOR | Occurs when a task is waiting for a tape mount. To view the tape status, query [sys.dm\_io\_backup\_tapes](http://msdn.microsoft.com/en-us/library/ms176071.aspx). If a mount operation is not pending, this wait type may indicate a hardware problem with the tape drive. |
| BACKUPBUFFER | Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data. This type is not typical, except when a task is waiting for a tape mount. |
| BACKUPIO | Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data. This type is not typical, except when a task is waiting for a tape mount. |
| BACKUPTHREAD | Occurs when a task is waiting for a backup task to finish. Wait times may be long, from several minutes to several hours. If the task that is being waited on is in an I/O process, this type does not indicate a problem. |
| BAD\_PAGE\_PROCESS | Occurs when the background suspect page logger is trying to avoid running more than every five seconds. Excessive suspect pages cause the logger to run frequently. |
| BROKER\_CONNECTION\_RECEIVE\_TASK | Occurs when waiting for access to receive a message on a connection endpoint. Receive access to the endpoint is serialized. |
| BROKER\_ENDPOINT\_STATE\_MUTEX | Occurs when there is contention to access the state of a Service Broker connection endpoint. Access to the state for changes is serialized. |
| BROKER\_EVENTHANDLER | Occurs when a task is waiting in the primary event handler of the Service Broker. This should occur very briefly. |
| BROKER\_INIT | Occurs when initializing Service Broker in each active database. This should occur infrequently. |
| BROKER\_MASTERSTART | Occurs when a task is waiting for the primary event handler of the Service Broker to start. This should occur very briefly. |
| BROKER\_RECEIVE\_WAITFOR | Occurs when the RECEIVE WAITFOR is waiting. This is typical if no messages are ready to be received. |
| BROKER\_REGISTERALLENDPOINTS | Occurs during the initialization of a Service Broker connection endpoint. This should occur very briefly. |
| BROKER\_SERVICE | Occurs when the Service Broker destination list that is associated with a target service is updated or re-prioritized. |
| BROKER\_SHUTDOWN | Occurs when there is a planned shutdown of Service Broker. This should occur very briefly, if at all. |
| BROKER\_TASK\_STOP | Occurs when the Service Broker queue task handler tries to shut down the task. The state check is serialized and must be in a running state beforehand. |
| BROKER\_TO\_FLUSH | Occurs when the Service Broker lazy flusher flushes the in-memory transmission objects to a work table. |
| BROKER\_TRANSMITTER | Occurs when the Service Broker transmitter is waiting for work. |
| BUILTIN\_HASHKEY\_MUTEX | May occur after startup of instance, while internal data structures are initializing. Will not recur once data structures have initialized. |
| CHECK\_PRINT\_RECORD | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| CHECKPOINT\_QUEUE | Occurs while the checkpoint task is waiting for the next checkpoint request. |
| CHKPT | Occurs at server startup to tell the checkpoint thread that it can start. |
| CLEAR\_DB | Occurs during operations that change the state of a database, such as opening or closing a database. |
| CLR\_AUTO\_EVENT | Occurs when a task is currently performing common language runtime (CLR) execution and is waiting for a particular autoevent to be initiated. Long waits are typical, and do not indicate a problem. |
| CLR\_CRST | Occurs when a task is currently performing CLR execution and is waiting to enter a critical section of the task that is currently being used by another task. |
| CLR\_JOIN | Occurs when a task is currently performing CLR execution and waiting for another task to end. This wait state occurs when there is a join between tasks. |
| CLR\_MANUAL\_EVENT | Occurs when a task is currently performing CLR execution and is waiting for a specific manual event to be initiated. |
| CLR\_MEMORY\_SPY | Occurs during a wait on lock acquisition for a data structure that is used to record all virtual memory allocations that come from CLR. The data structure is locked to maintain its integrity if there is parallel access. |
| CLR\_MONITOR | Occurs when a task is currently performing CLR execution and is waiting to obtain a lock on the monitor. |
| CLR\_RWLOCK\_READER | Occurs when a task is currently performing CLR execution and is waiting for a reader lock. |
| CLR\_RWLOCK\_WRITER | Occurs when a task is currently performing CLR execution and is waiting for a writer lock. |
| CLR\_SEMAPHORE | Occurs when a task is currently performing CLR execution and is waiting for a semaphore. |
| CLR\_TASK\_START | Occurs while waiting for a CLR task to complete startup. |
| CLRHOST\_STATE\_ACCESS | Occurs where there is a wait to acquire exclusive access to the CLR-hosting data structures. This wait type occurs while setting up or tearing down the CLR runtime. |
| CMEMTHREAD | Occurs when a task is waiting on a thread-safe memory object. The wait time might increase when there is contention caused by multiple tasks trying to allocate memory from the same memory object. |
| CXPACKET | Occurs when trying to synchronize the query processor exchange iterator. You may consider lowering the degree of parallelism if contention on this wait type becomes a problem. |
| CXROWSET\_SYNC | Occurs during a parallel range scan. |
| DAC\_INIT | Occurs while the dedicated administrator connection is initializing. |
| DBMIRROR\_DBM\_EVENT | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| DBMIRROR\_DBM\_MUTEX | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| DBMIRROR\_EVENTS\_QUEUE | Occurs when database mirroring waits for events to process. |
| DBMIRROR\_SEND | Occurs when a task is waiting for a communications backlog at the network layer to clear to be able to send messages. Indicates that the communications layer is starting to become overloaded and affect the database mirroring data throughput. |
| DBMIRROR\_WORKER\_QUEUE | Indicates that the database mirroring worker task is waiting for more work. |
| DBMIRRORING\_CMD | Occurs when a task is waiting for log records to be flushed to disk. This wait state is expected to be held for long periods of time. |
| DEADLOCK\_ENUM\_MUTEX | Occurs when the deadlock monitor and sys.dm\_os\_waiting\_tasks try to make sure that SQL Server is not running multiple deadlock searches at the same time. |
| DEADLOCK\_TASK\_SEARCH | Large waiting time on this resource indicates that the server is executing queries on top of sys.dm\_os\_waiting\_tasks, and these queries are blocking deadlock monitor from running deadlock search. This wait type is used by deadlock monitor only. Queries on top of sys.dm\_os\_waiting\_tasks use DEADLOCK\_ENUM\_MUTEX. |
| DEBUG | Occurs during Transact-SQL and CLR debugging for internal synchronization. |
| DISABLE\_VERSIONING | Occurs when SQL Server polls the version transaction manager to see whether the timestamp of the earliest active transaction is later than the timestamp of when the state started changing. If this is this case, all the snapshot transactions that were started before the ALTER DATABASE statement was run have finished. This wait state is used when SQL Server disables versioning by using the ALTER DATABASE statement. |
| DISKIO\_SUSPEND | Occurs when a task is waiting to access a file when an external backup is active. This is reported for each waiting user process. A count larger than five per user process may indicate that the external backup is taking too much time to finish. |
| DISPATCHER\_QUEUE\_SEMAPHORE | Occurs when a thread from the dispatcher pool is waiting for more work to process. The wait time for this wait type is expected to increase when the dispatcher is idle. |
| DLL\_LOADING\_MUTEX | Occurs once while waiting for the XML parser DLL to load. |
| DROPTEMP | Occurs between attempts to drop a temporary object if the previous attempt failed. The wait duration grows exponentially with each failed drop attempt. |
| DTC | Occurs when a task is waiting on an event that is used to manage state transition. This state controls when the recovery of Microsoft Distributed Transaction Coordinator (MS DTC) transactions occurs after SQL Server receives notification that the MS DTC service has become unavailable.  This state also describes a task that is waiting when a commit of a MS DTC transaction is initiated by SQL Server and SQL Server is waiting for the MS DTC commit to finish. |
| DTC\_ABORT\_REQUEST | Occurs in a MS DTC worker session when the session is waiting to take ownership of a MS DTC transaction. After MS DTC owns the transaction, the session can roll back the transaction. Generally, the session will wait for another session that is using the transaction. |
| DTC\_RESOLVE | Occurs when a recovery task is waiting for the master database in a cross-database transaction so that the task can query the outcome of the transaction. |
| DTC\_STATE | Occurs when a task is waiting on an event that protects changes to the internal MS DTC global state object. This state should be held for very short periods of time. |
| DTC\_TMDOWN\_REQUEST | Occurs in a MS DTC worker session when SQL Server receives notification that the MS DTC service is not available. First, the worker will wait for the MS DTC recovery process to start. Then, the worker waits to obtain the outcome of the distributed transaction that the worker is working on. This may continue until the connection with the MS DTC service has been reestablished. |
| DTC\_WAITFOR\_OUTCOME | Occurs when recovery tasks wait for MS DTC to become active to enable the resolution of prepared transactions. |
| DUMP\_LOG\_COORDINATOR | Occurs when a main task is waiting for a subtask to generate data. Ordinarily, this state does not occur. A long wait indicates an unexpected blockage. The subtask should be investigated. |
| DUMPTRIGGER | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| EC | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| EE\_PMOLOCK | Occurs during synchronization of certain types of memory allocations during statement execution. |
| EE\_SPECPROC\_MAP\_INIT | Occurs during synchronization of internal procedure hash table creation. This wait can only occur during the initial accessing of the hash table after the SQL Server instance starts. |
| ENABLE\_VERSIONING | Occurs when SQL Server waits for all update transactions in this database to finish before declaring the database ready to transition to snapshot isolation allowed state. This state is used when SQL Server enables snapshot isolation by using the ALTER DATABASE statement. |
| ERROR\_REPORTING\_MANAGER | Occurs during synchronization of multiple concurrent error log initializations. |
| EXCHANGE | Occurs during synchronization in the query processor exchange iterator during parallel queries. |
| EXECSYNC | Occurs during parallel queries while synchronizing in query processor in areas not related to the exchange iterator. Examples of such areas are bitmaps, large binary objects (LOBs), and the spool iterator. LOBs may frequently use this wait state. |
| EXECUTION\_PIPE\_EVENT\_INTERNAL | Occurs during synchronization between producer and consumer parts of batch execution that are submitted through the connection context. |
| FAILPOINT | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FCB\_REPLICA\_READ | Occurs when the reads of a snapshot (or a temporary snapshot created by DBCC) sparse file are synchronized. |
| FCB\_REPLICA\_WRITE | Occurs when the pushing or pulling of a page to a snapshot (or a temporary snapshot created by DBCC) sparse file is synchronized. |
| FS\_FC\_RWLOCK | Occurs when there is a wait by the FILESTREAM garbage collector to do either of the following:   * Disable garbage collection (used by backup and restore). * Execute one cycle of the FILESTREAM garbage collector. |
| FS\_GARBAGE\_COLLECTOR\_SHUTDOWN | Occurs when the FILESTREAM garbage collector is waiting for cleanup tasks to be completed. |
| FS\_HEADER\_RWLOCK | Occurs when there is a wait to acquire access to the FILESTREAM header of a FILESTREAM data container to either read or update contents in the FILESTREAM header file (Filestream.hdr). |
| FS\_LOGTRUNC\_RWLOCK | Occurs when there is a wait to acquire access to FILESTREAM log truncation to do either of the following:   * Temporarily disable FILESTREAM log (FSLOG) truncation (used by backup and restore). * Execute one cycle of FSLOG truncation. |
| FSA\_FORCE\_OWN\_XACT | Occurs when a FILESTREAM file I/O operation needs to bind to the associated transaction, but the transaction is currently owned by another session. |
| FSAGENT | Occurs when a FILESTREAM file I/O operation is waiting for a FILESTREAM agent resource that is being used by another file I/O operation. |
| FSTR\_CONFIG\_MUTEX | Occurs when there is a wait for another FILESTREAM feature reconfiguration to be completed. |
| FSTR\_CONFIG\_RWLOCK | Occurs when there is a wait to serialize access to the FILESTREAM configuration parameters. |
| FT\_METADATA\_MUTEX | Documented for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FT\_RESTART\_CRAWL | Occurs when a full-text crawl needs to restart from a last known good point to recover from a transient failure. The wait lets the worker tasks currently working on that population to complete or exit the current step. |
| FULLTEXT GATHERER | Occurs during synchronization of full-text operations. |
| GUARDIAN | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| HADR\_AG\_MUTEX | Occurs when an AlwaysOn DDL statement or Windows Server Failover Clustering command is waiting for exclusive read/write access to the configuration of an availability group. |
| HADR\_AR\_CRITICAL\_SECTION\_ENTRY | Occurs when an AlwaysOn DDL statement or Windows Server Failover Clustering command is waiting for exclusive read/write access to the runtime state of the local replica of the associated availability group. |
| HADR\_AR\_MANAGER\_MUTEX | Occurs when an availability replica shutdown is waiting for startup to complete or an availability replica startup is waiting for shutdown to complete. Internal use only.   |  | | --- | | **NoteNote** | | Availability replica shutdown is initiated either by SQL Server shutdown or by SQL Server handling the loss of quorum by the Windows Server Failover Clustering node. Availability replica startup is initiated either by SQL Server startup or by SQL Server recovering from the loss of quorum by the Windows Server Failover Clustering node. | |
| HADR\_ARCONTROLLER\_NOTIFICATIONS\_SUBSCRIBER\_LIST | The publisher for an availability replica event (such as a state change or configuration change) is waiting for exclusive read/write access to the list of event subscribers. Internal use only. |
| HADR\_BACKUP\_BULK\_LOCK | The AlwaysOn primary database received a backup request from a secondary database and is waiting for the background thread to finish processing the request on acquiring or releasing the BulkOp lock. |
| HADR\_BACKUP\_QUEUE | The backup background thread of the AlwaysOn primary database is waiting for a new work request from the secondary database. (typically, this occurs when the primary database is holding the BulkOp log and is waiting for the secondary database to indicate that the primary database can release the lock). |
| HADR\_CLUSAPI\_CALL | A SQL Server thread is waiting to switch from non-preemptive mode (scheduled by SQL Server) to preemptive mode (scheduled by the operating system) in order to invoke Windows Server Failover Clustering APIs. |
| HADR\_COMPRESSED\_CACHE\_SYNC | Waiting for access to the cache of compressed log blocks that is used to avoid redundant compression of the log blocks sent to multiple secondary databases. |
| HADR\_DATABASE\_FLOW\_CONTROL | Waiting for messages to be sent to the partner when the maximum number of queued messages has been reached. Indicates that the log scans are running faster than the network sends. This is an issue only if network sends are slower than expected. |
| HADR\_DATABASE\_VERSIONING\_STATE | Occurs on the versioning state change of an AlwaysOn secondary database. This wait is for internal data structures and is usually is very short with no direct effect on data access. |
| HADR\_DATABASE\_WAIT\_FOR\_RESTART | Waiting for the database to restart under AlwaysOn Availability Groups control. Under normal conditions, this is not a customer issue because waits are expected here. |
| HADR\_DATABASE\_WAIT\_FOR\_TRANSITION\_TO\_VERSIONING | A query on object(s) in a readable secondary database of an AlwaysOn availability group is blocked on row versioning while waiting for commit or rollback of all transactions that were in-flight when the secondary replica was enabled for read workloads. This wait type guarantees that row versions are available before execution of a query under snapshot isolation. |
| HADR\_DB\_COMMAND | Waiting for responses to conversational messages (which require an explicit response from the other side, using the AlwaysOn conversational message infrastructure). A number of different message types use this wait type. |
| HADR\_DB\_OP\_COMPLETION\_SYNC | Waiting for responses to conversational messages (which require an explicit response from the other side, using the AlwaysOn conversational message infrastructure). A number of different message types use this wait type. |
| HADR\_DB\_OP\_START\_SYNC | An AlwaysOn DDL statement or a Windows Server Failover Clustering command is waiting for serialized access to an availability database and its runtime state. |
| HADR\_DBR\_SUBSCRIBER | The publisher for an availability replica event (such as a state change or configuration change) is waiting for exclusive read/write access to the runtime state of an event subscriber that corresponds to an availability database. Internal use only. |
| HADR\_DBR\_SUBSCRIBER\_FILTER\_LIST | The publisher for an availability replica event (such as a state change or configuration change) is waiting for exclusive read/write access to the list of event subscribers that correspond to availability databases. Internal use only. |
| HADR\_DBSTATECHANGE\_SYNC | Concurrency control wait for updating the internal state of the database replica. |
| HADR\_FILESTREAM\_BLOCK\_FLUSH | The FILESTREAM AlwaysOn transport manager is waiting until processing of a log block is finished. |
| HADR\_FILESTREAM\_FILE\_CLOSE | The FILESTREAM AlwaysOn transport manager is waiting until the next FILESTREAM file gets processed and its handle gets closed. |
| HADR\_FILESTREAM\_FILE\_REQUEST | An AlwaysOn secondary replica is waiting for the primary replica to send all requested FILESTREAM files during UNDO. |
| HADR\_FILESTREAM\_IOMGR | The FILESTREAM AlwaysOn transport manager is waiting for R/W lock that protects the FILESTREAM AlwaysOn I/O manager during startup or shutdown. |
| HADR\_FILESTREAM\_IOMGR\_IOCOMPLETION | The FILESTREAM AlwaysOn I/O manager is waiting for I/O completion. |
| HADR\_FILESTREAM\_MANAGER | The FILESTREAM AlwaysOn transport manager is waiting for the R/W lock that protects the FILESTREAM AlwaysOn transport manager during startup or shutdown. |
| HADR\_GROUP\_COMMIT | Transaction commit processing is waiting to allow a group commit so that multiple commit log records can be put into a single log block. This wait is an expected condition that optimizes the log I/O, capture, and send operations. |
| HADR\_LOGCAPTURE\_SYNC | Concurrency control around the log capture or apply object when creating or destroying scans. This is an expected wait when partners change state or connection status. |
| HADR\_LOGCAPTURE\_WAIT | Waiting for log records to become available. Can occur either when waiting for new log records to be generated by connections or for I/O completion when reading log not in the cache. This is an expected wait if the log scan is caught up to the end of log or is reading from disk. |
| HADR\_LOGPROGRESS\_SYNC | Concurrency control wait when updating the log progress status of database replicas. |
| HADR\_NOTIFICATION\_DEQUEUE | A background task that processes Windows Server Failover Clustering notifications is waiting for the next notification. Internal use only. |
| HADR\_NOTIFICATION\_WORKER\_EXCLUSIVE\_ACCESS | The AlwaysOn availability replica manager is waiting for serialized access to the runtime state of a background task that processes Windows Server Failover Clustering notifications. Internal use only. |
| HADR\_NOTIFICATION\_WORKER\_STARTUP\_SYNC | A background task is waiting for the completion of the startup of a background task that processes Windows Server Failover Clustering notifications. Internal use only. |
| HADR\_NOTIFICATION\_WORKER\_TERMINATION\_SYNC | A background task is waiting for the termination of a background task that processes Windows Server Failover Clustering notifications. Internal use only. |
| HADR\_PARTNER\_SYNC | Concurrency control wait on the partner list. |
| HADR\_READ\_ALL\_NETWORKS | Waiting to get read or write access to the list of WSFC networks. Internal use only.   |  | | --- | | **NoteNote** | | The engine keeps a list of WSFC networks that is used in dynamic management views (such as [sys.dm\_hadr\_cluster\_networks](http://msdn.microsoft.com/en-us/library/hh213657.aspx)) or to validate AlwaysOn Transact-SQL statements that reference WSFC network information. This list is updated upon engine startup, WSFC related notifications, and internal AlwaysOn restart (for example, losing and regaining of WSFC quorum). Tasks will usually be blocked when an update in that list is in progress. | |
| HADR\_RECOVERY\_WAIT\_FOR\_CONNECTION | Waiting for the secondary database to connect to the primary database before running recovery. This is an expected wait, which can lengthen if the connection to the primary is slow to establish. |
| HADR\_RECOVERY\_WAIT\_FOR\_UNDO | Database recovery is waiting for the secondary database to finish the reverting and initializing phase to bring it back to the common log point with the primary database. This is an expected wait after failovers.Undo progress can be tracked through the Windows System Monitor (perfmon.exe) and dynamic management views. |
| HADR\_REPLICAINFO\_SYNC | Waiting for concurrency control to update the current replica state. |
| HADR\_SYNC\_COMMIT | Waiting for transaction commit processing for the synchronized secondary databases to harden the log. This wait is also reflected by the Transaction Delay performance counter. This wait type is expected for synchronized availability groups and indicates the time to send, write, and acknowledge log to the secondary databases. |
| HADR\_SYNCHRONIZING\_THROTTLE | Waiting for transaction commit processing to allow a synchronizing secondary database to catch up to the primary end of log in order to transition to the synchronized state. This is an expected wait when a secondary database is catching up. |
| HADR\_TDS\_LISTENER\_SYNC | Either the internal AlwaysOn system or the WSFC cluster will request that listeners are started or stopped. The processing of this request is always asynchronous, and there is a mechanism to remove redundant requests. There are also moments that this process is suspended because of configuration changes. All waits related with this listener synchronization mechanism use this wait type. Internal use only. |
| HADR\_TDS\_LISTENER\_SYNC\_PROCESSING | Used at the end of an AlwaysOn Transact-SQL statement that requires starting and/or stopping an [availability group listener](http://msdn.microsoft.com/en-us/library/hh213417.aspx). Since the start/stop operation is done asynchronously, the user thread will block using this wait type until the situation of the listener is known. |
| HADR\_TIMER\_TASK | Waiting to get the lock on the timer task object and is also used for the actual waits between times that work is being performed. For example, for a task that runs every 10 seconds, after one execution, AlwaysOn Availability Groups waits about 10 seconds to reschedule the task, and the wait is included here. |
| HADR\_TRANSPORT\_DBRLIST | Waiting for access to the transport layer's database replica list. Used for the spinlock that grants access to it. |
| HADR\_TRANSPORT\_FLOW\_CONTROL | Waiting when the number of outstanding unacknowledged AlwaysOn messages is over the out flow control threshold. This is on an availability replica-to-replica basis (not on a database-to-database basis). |
| HADR\_TRANSPORT\_SESSION | AlwaysOn Availability Groups is waiting while changing or accessing the underlying transport state. |
| HADR\_WORK\_POOL | Concurrency control wait on the AlwaysOn Availability Groups background work task object. |
| HADR\_WORK\_QUEUE | AlwaysOn Availability Groups background worker thread waiting for new work to be assigned. This is an expected wait when there are ready workers waiting for new work, which is the normal state. |
| HADR\_XRF\_STACK\_ACCESS | Accessing (look up, add, and delete) the extended recovery fork stack for an AlwaysOn availability database. |
| HTTP\_ENUMERATION | Occurs at startup to enumerate the HTTP endpoints to start HTTP. |
| HTTP\_START | Occurs when a connection is waiting for HTTP to complete initialization. |
| IMPPROV\_IOWAIT | Occurs when SQL Server waits for a bulkload I/O to finish. |
| INTERNAL\_TESTING | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| IO\_AUDIT\_MUTEX | Occurs during synchronization of trace event buffers. |
| IO\_COMPLETION | Occurs while waiting for I/O operations to complete. This wait type generally represents non-data page I/Os. Data page I/O completion waits appear as PAGEIOLATCH\_\* waits. |
| IO\_RETRY | Occurs when an I/O operation such as a read or a write to disk fails because of insufficient resources, and is then retried. |
| IOAFF\_RANGE\_QUEUE | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| KSOURCE\_WAKEUP | Used by the service control task while waiting for requests from the Service Control Manager. Long waits are expected and do not indicate a problem. |
| KTM\_ENLISTMENT | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| KTM\_RECOVERY\_MANAGER | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| KTM\_RECOVERY\_RESOLUTION | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| LATCH\_DT | Occurs when waiting for a DT (destroy) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH\_\* waits is available in sys.dm\_os\_latch\_stats. Note that sys.dm\_os\_latch\_stats groups LATCH\_NL, LATCH\_SH, LATCH\_UP, LATCH\_EX, and LATCH\_DT waits together. |
| LATCH\_EX | Occurs when waiting for an EX (exclusive) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH\_\* waits is available in sys.dm\_os\_latch\_stats. Note that sys.dm\_os\_latch\_stats groups LATCH\_NL, LATCH\_SH, LATCH\_UP, LATCH\_EX, and LATCH\_DT waits together. |
| LATCH\_KP | Occurs when waiting for a KP (keep) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH\_\* waits is available in sys.dm\_os\_latch\_stats. Note that sys.dm\_os\_latch\_stats groups LATCH\_NL, LATCH\_SH, LATCH\_UP, LATCH\_EX, and LATCH\_DT waits together. |
| LATCH\_NL | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| LATCH\_SH | Occurs when waiting for an SH (share) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH\_\* waits is available in sys.dm\_os\_latch\_stats. Note that sys.dm\_os\_latch\_stats groups LATCH\_NL, LATCH\_SH, LATCH\_UP, LATCH\_EX, and LATCH\_DT waits together. |
| LATCH\_UP | Occurs when waiting for an UP (update) latch. This does not include buffer latches or transaction mark latches. A listing of LATCH\_\* waits is available in sys.dm\_os\_latch\_stats. Note that sys.dm\_os\_latch\_stats groups LATCH\_NL, LATCH\_SH, LATCH\_UP, LATCH\_EX, and LATCH\_DT waits together. |
| LAZYWRITER\_SLEEP | Occurs when lazywriter tasks are suspended. This is a measure of the time spent by background tasks that are waiting. Do not consider this state when you are looking for user stalls. |
| LCK\_M\_BU | Occurs when a task is waiting to acquire a Bulk Update (BU) lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_IS | Occurs when a task is waiting to acquire an Intent Shared (IS) lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_IU | Occurs when a task is waiting to acquire an Intent Update (IU) lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_IX | Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RIn\_NL | Occurs when a task is waiting to acquire a NULL lock on the current key value, and an Insert Range lock between the current and previous key. A NULL lock on the key is an instant release lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RIn\_S | Occurs when a task is waiting to acquire a shared lock on the current key value, and an Insert Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RIn\_U | Task is waiting to acquire an Update lock on the current key value, and an Insert Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RIn\_X | Occurs when a task is waiting to acquire an Exclusive lock on the current key value, and an Insert Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RS\_S | Occurs when a task is waiting to acquire a Shared lock on the current key value, and a Shared Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RS\_U | Occurs when a task is waiting to acquire an Update lock on the current key value, and an Update Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RX\_S | Occurs when a task is waiting to acquire a Shared lock on the current key value, and an Exclusive Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RX\_U | Occurs when a task is waiting to acquire an Update lock on the current key value, and an Exclusive range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_RX\_X | Occurs when a task is waiting to acquire an Exclusive lock on the current key value, and an Exclusive Range lock between the current and previous key. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_S | Occurs when a task is waiting to acquire a Shared lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_SCH\_M | Occurs when a task is waiting to acquire a Schema Modify lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_SCH\_S | Occurs when a task is waiting to acquire a Schema Share lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_SIU | Occurs when a task is waiting to acquire a Shared With Intent Update lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_SIX | Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_U | Occurs when a task is waiting to acquire an Update lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_UIX | Occurs when a task is waiting to acquire an Update With Intent Exclusive lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LCK\_M\_X | Occurs when a task is waiting to acquire an Exclusive lock. For a lock compatibility matrix, see [sys.dm\_tran\_locks (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190345.aspx). |
| LOGBUFFER | Occurs when a task is waiting for space in the log buffer to store a log record. Consistently high values may indicate that the log devices cannot keep up with the amount of log being generated by the server. |
| LOGGENERATION | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| LOGMGR | Occurs when a task is waiting for any outstanding log I/Os to finish before shutting down the log while closing the database. |
| LOGMGR\_FLUSH | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| LOGMGR\_QUEUE | Occurs while the log writer task waits for work requests. |
| LOGMGR\_RESERVE\_APPEND | Occurs when a task is waiting to see whether log truncation frees up log space to enable the task to write a new log record. Consider increasing the size of the log file(s) for the affected database to reduce this wait. |
| LOWFAIL\_MEMMGR\_QUEUE | Occurs while waiting for memory to be available for use. |
| MISCELLANEOUS | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| MSQL\_DQ | Occurs when a task is waiting for a distributed query operation to finish. This is used to detect potential Multiple Active Result Set (MARS) application deadlocks. The wait ends when the distributed query call finishes. |
| MSQL\_XACT\_MGR\_MUTEX | Occurs when a task is waiting to obtain ownership of the session transaction manager to perform a session level transaction operation. |
| MSQL\_XACT\_MUTEX | Occurs during synchronization of transaction usage. A request must acquire the mutex before it can use the transaction. |
| MSQL\_XP | Occurs when a task is waiting for an extended stored procedure to end. SQL Server uses this wait state to detect potential MARS application deadlocks. The wait stops when the extended stored procedure call ends. |
| MSSEARCH | Occurs during Full-Text Search calls. This wait ends when the full-text operation completes. It does not indicate contention, but rather the duration of full-text operations. |
| NET\_WAITFOR\_PACKET | Occurs when a connection is waiting for a network packet during a network read. |
| OLEDB | Occurs when SQL Server calls the SQL Server Native Client OLE DB Provider. This wait type is not used for synchronization. Instead, it indicates the duration of calls to the OLE DB provider. |
| ONDEMAND\_TASK\_QUEUE | Occurs while a background task waits for high priority system task requests. Long wait times indicate that there have been no high priority requests to process, and should not cause concern. |
| PAGEIOLATCH\_DT | Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Destroy mode. Long waits may indicate problems with the disk subsystem. |
| PAGEIOLATCH\_EX | Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Exclusive mode. Long waits may indicate problems with the disk subsystem. |
| PAGEIOLATCH\_KP | Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Keep mode. Long waits may indicate problems with the disk subsystem. |
| PAGEIOLATCH\_NL | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PAGEIOLATCH\_SH | Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Shared mode. Long waits may indicate problems with the disk subsystem. |
| PAGEIOLATCH\_UP | Occurs when a task is waiting on a latch for a buffer that is in an I/O request. The latch request is in Update mode. Long waits may indicate problems with the disk subsystem. |
| PAGELATCH\_DT | Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Destroy mode. |
| PAGELATCH\_EX | Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Exclusive mode. |
| PAGELATCH\_KP | Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Keep mode. |
| PAGELATCH\_NL | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PAGELATCH\_SH | Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Shared mode. |
| PAGELATCH\_UP | Occurs when a task is waiting on a latch for a buffer that is not in an I/O request. The latch request is in Update mode. |
| PARALLEL\_BACKUP\_QUEUE | Occurs when serializing output produced by RESTORE HEADERONLY, RESTORE FILELISTONLY, or RESTORE LABELONLY. |
| PREEMPTIVE\_ABR | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PREEMPTIVE\_AUDIT\_ACCESS\_EVENTLOG | Occurs when the SQL Server Operating System (SQLOS) scheduler switches to preemptive mode to write an audit event to the Windows event log. |
| PREEMPTIVE\_AUDIT\_ACCESS\_SECLOG | Occurs when the SQLOS scheduler switches to preemptive mode to write an audit event to the Windows Security log. |
| PREEMPTIVE\_CLOSEBACKUPMEDIA | Occurs when the SQLOS scheduler switches to preemptive mode to close backup media. |
| PREEMPTIVE\_CLOSEBACKUPTAPE | Occurs when the SQLOS scheduler switches to preemptive mode to close a tape backup device. |
| PREEMPTIVE\_CLOSEBACKUPVDIDEVICE | Occurs when the SQLOS scheduler switches to preemptive mode to close a virtual backup device. |
| PREEMPTIVE\_CLUSAPI\_CLUSTERRESOURCECONTROL | Occurs when the SQLOS scheduler switches to preemptive mode to perform Windows failover cluster operations. |
| PREEMPTIVE\_COM\_COCREATEINSTANCE | Occurs when the SQLOS scheduler switches to preemptive mode to create a COM object. |
| PREEMPTIVE\_HADR\_LEASE\_MECHANISM | AlwaysOn Availability Groups lease manager scheduling for CSS diagnostics. |
| PREEMPTIVE\_SOSTESTING | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PREEMPTIVE\_STRESSDRIVER | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PREEMPTIVE\_TESTING | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PREEMPTIVE\_XETESTING | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| PRINT\_ROLLBACK\_PROGRESS | Used to wait while user processes are ended in a database that has been transitioned by using the ALTER DATABASE termination clause. For more information, see [ALTER DATABASE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms174269.aspx). |
| PWAIT\_HADR\_CHANGE\_NOTIFIER\_TERMINATION\_SYNC | Occurs when a background task is waiting for the termination of the background task that receives (via polling) Windows Server Failover Clustering notifications. Internal use only. |
| PWAIT\_HADR\_CLUSTER\_INTEGRATION | An append, replace, and/or remove operation is waiting to grab a write lock on an AlwaysOn internal list (such as a list of networks, network addresses, or availability group listeners). Internal use only. |
| PWAIT\_HADR\_OFFLINE\_COMPLETED | An AlwaysOn drop availability group operation is waiting for the target availability group to go offline before destroying Windows Server Failover Clustering objects. |
| PWAIT\_HADR\_ONLINE\_COMPLETED | An AlwaysOn create or failover availability group operation is waiting for the target availability group to come online. |
| PWAIT\_HADR\_POST\_ONLINE\_COMPLETED | An AlwaysOn drop availability group operation is waiting for the termination of any background task that was scheduled as part of a previous command. For example, there may be a background task that is transitioning availability databases to the primary role. The DROP AVAILABILITY GROUP DDL must wait for this background task to terminate in order to avoid race conditions. |
| PWAIT\_HADR\_WORKITEM\_COMPLETED | Internal wait by a thread waiting for an async work task to complete. This is an expected wait and is for CSS use. |
| QPJOB\_KILL | Indicates that an asynchronous automatic statistics update was canceled by a call to KILL as the update was starting to run. The terminating thread is suspended, waiting for it to start listening for KILL commands. A good value is less than one second. |
| QPJOB\_WAITFOR\_ABORT | Indicates that an asynchronous automatic statistics update was canceled by a call to KILL when it was running. The update has now completed but is suspended until the terminating thread message coordination is complete. This is an ordinary but rare state, and should be very short. A good value is less than one second. |
| QRY\_MEM\_GRANT\_INFO\_MUTEX | Occurs when Query Execution memory management tries to control access to static grant information list. This state lists information about the current granted and waiting memory requests. This state is a simple access control state. There should never be a long wait on this state. If this mutex is not released, all new memory-using queries will stop responding. |
| QUERY\_ERRHDL\_SERVICE\_DONE | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| QUERY\_EXECUTION\_INDEX\_SORT\_EVENT\_OPEN | Occurs in certain cases when offline create index build is run in parallel, and the different worker threads that are sorting synchronize access to the sort files. |
| QUERY\_NOTIFICATION\_MGR\_MUTEX | Occurs during synchronization of the garbage collection queue in the Query Notification Manager. |
| QUERY\_NOTIFICATION\_SUBSCRIPTION\_MUTEX | Occurs during state synchronization for transactions in Query Notifications. |
| QUERY\_NOTIFICATION\_TABLE\_MGR\_MUTEX | Occurs during internal synchronization within the Query Notification Manager. |
| QUERY\_NOTIFICATION\_UNITTEST\_MUTEX | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| QUERY\_OPTIMIZER\_PRINT\_MUTEX | Occurs during synchronization of query optimizer diagnostic output production. This wait type only occurs if diagnostic settings have been enabled under direction of Microsoft Product Support. |
| QUERY\_TRACEOUT | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| QUERY\_WAIT\_ERRHDL\_SERVICE | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| RECOVER\_CHANGEDB | Occurs during synchronization of database status in warm standby database. |
| REPL\_CACHE\_ACCESS | Occurs during synchronization on a replication article cache. During these waits, the replication log reader stalls, and data definition language (DDL) statements on a published table are blocked. |
| REPL\_SCHEMA\_ACCESS | Occurs during synchronization of replication schema version information. This state exists when DDL statements are executed on the replicated object, and when the log reader builds or consumes versioned schema based on DDL occurrence. |
| REPLICA\_WRITES | Occurs while a task waits for completion of page writes to database snapshots or DBCC replicas. |
| REQUEST\_DISPENSER\_PAUSE | Occurs when a task is waiting for all outstanding I/O to complete, so that I/O to a file can be frozen for snapshot backup. |
| REQUEST\_FOR\_DEADLOCK\_SEARCH | Occurs while the deadlock monitor waits to start the next deadlock search. This wait is expected between deadlock detections, and lengthy total waiting time on this resource does not indicate a problem. |
| RESMGR\_THROTTLED | Occurs when a new request comes in and is throttled based on the GROUP\_MAX\_REQUESTS setting. |
| RESOURCE\_QUEUE | Occurs during synchronization of various internal resource queues. |
| RESOURCE\_SEMAPHORE | Occurs when a query memory request cannot be granted immediately due to other concurrent queries. High waits and wait times may indicate excessive number of concurrent queries, or excessive memory request amounts. |
| RESOURCE\_SEMAPHORE\_MUTEX | Occurs while a query waits for its request for a thread reservation to be fulfilled. It also occurs when synchronizing query compile and memory grant requests. |
| RESOURCE\_SEMAPHORE\_QUERY\_COMPILE | Occurs when the number of concurrent query compilations reaches a throttling limit. High waits and wait times may indicate excessive compilations, recompiles, or uncachable plans. |
| RESOURCE\_SEMAPHORE\_SMALL\_QUERY | Occurs when memory request by a small query cannot be granted immediately due to other concurrent queries. Wait time should not exceed more than a few seconds, because the server transfers the request to the main query memory pool if it fails to grant the requested memory within a few seconds. High waits may indicate an excessive number of concurrent small queries while the main memory pool is blocked by waiting queries. |
| SEC\_DROP\_TEMP\_KEY | Occurs after a failed attempt to drop a temporary security key before a retry attempt. |
| SECURITY\_MUTEX | Occurs when there is a wait for mutexes that control access to the global list of Extensible Key Management (EKM) cryptographic providers and the session-scoped list of EKM sessions. |
| SEQUENTIAL\_GUID | Occurs while a new sequential GUID is being obtained. |
| SERVER\_IDLE\_CHECK | Occurs during synchronization of SQL Server instance idle status when a resource monitor is attempting to declare a SQL Server instance as idle or trying to wake up. |
| SHUTDOWN | Occurs while a shutdown statement waits for active connections to exit. |
| SLEEP\_BPOOL\_FLUSH | Occurs when a checkpoint is throttling the issuance of new I/Os in order to avoid flooding the disk subsystem. |
| SLEEP\_DBSTARTUP | Occurs during database startup while waiting for all databases to recover. |
| SLEEP\_DCOMSTARTUP | Occurs once at most during SQL Server instance startup while waiting for DCOM initialization to complete. |
| SLEEP\_MSDBSTARTUP | Occurs when SQL Trace waits for the msdb database to complete startup. |
| SLEEP\_SYSTEMTASK | Occurs during the start of a background task while waiting for tempdb to complete startup. |
| SLEEP\_TASK | Occurs when a task sleeps while waiting for a generic event to occur. |
| SLEEP\_TEMPDBSTARTUP | Occurs while a task waits for tempdb to complete startup. |
| SNI\_CRITICAL\_SECTION | Occurs during internal synchronization within SQL Server networking components. |
| SNI\_HTTP\_WAITFOR\_0\_DISCON | Occurs during SQL Server shutdown, while waiting for outstanding HTTP connections to exit. |
| SNI\_LISTENER\_ACCESS | Occurs while waiting for non-uniform memory access (NUMA) nodes to update state change. Access to state change is serialized. |
| SNI\_TASK\_COMPLETION | Occurs when there is a wait for all tasks to finish during a NUMA node state change. |
| SOAP\_READ | Occurs while waiting for an HTTP network read to complete. |
| SOAP\_WRITE | Occurs while waiting for an HTTP network write to complete. |
| SOS\_CALLBACK\_REMOVAL | Occurs while performing synchronization on a callback list in order to remove a callback. It is not expected for this counter to change after server initialization is completed. |
| SOS\_DISPATCHER\_MUTEX | Occurs during internal synchronization of the dispatcher pool. This includes when the pool is being adjusted. |
| SOS\_LOCALALLOCATORLIST | Occurs during internal synchronization in the SQL Server memory manager. |
| SOS\_MEMORY\_USAGE\_ADJUSTMENT | Occurs when memory usage is being adjusted among pools. |
| SOS\_OBJECT\_STORE\_DESTROY\_MUTEX | Occurs during internal synchronization in memory pools when destroying objects from the pool. |
| SOS\_PROCESS\_AFFINITY\_MUTEX | Occurs during synchronizing of access to process affinity settings. |
| SOS\_RESERVEDMEMBLOCKLIST | Occurs during internal synchronization in the SQL Server memory manager. |
| SOS\_SCHEDULER\_YIELD | Occurs when a task voluntarily yields the scheduler for other tasks to execute. During this wait the task is waiting for its quantum to be renewed. |
| SOS\_SMALL\_PAGE\_ALLOC | Occurs during the allocation and freeing of memory that is managed by some memory objects. |
| SOS\_STACKSTORE\_INIT\_MUTEX | Occurs during synchronization of internal store initialization. |
| SOS\_SYNC\_TASK\_ENQUEUE\_EVENT | Occurs when a task is started in a synchronous manner. Most tasks in SQL Server are started in an asynchronous manner, in which control returns to the starter immediately after the task request has been placed on the work queue. |
| SOS\_VIRTUALMEMORY\_LOW | Occurs when a memory allocation waits for a resource manager to free up virtual memory. |
| SOSHOST\_EVENT | Occurs when a hosted component, such as CLR, waits on a SQL Server event synchronization object. |
| SOSHOST\_INTERNAL | Occurs during synchronization of memory manager callbacks used by hosted components, such as CLR. |
| SOSHOST\_MUTEX | Occurs when a hosted component, such as CLR, waits on a SQL Server mutex synchronization object. |
| SOSHOST\_RWLOCK | Occurs when a hosted component, such as CLR, waits on a SQL Server reader-writer synchronization object. |
| SOSHOST\_SEMAPHORE | Occurs when a hosted component, such as CLR, waits on a SQL Server semaphore synchronization object. |
| SOSHOST\_SLEEP | Occurs when a hosted task sleeps while waiting for a generic event to occur. Hosted tasks are used by hosted components such as CLR. |
| SOSHOST\_TRACELOCK | Occurs during synchronization of access to trace streams. |
| SOSHOST\_WAITFORDONE | Occurs when a hosted component, such as CLR, waits for a task to complete. |
| SQLCLR\_APPDOMAIN | Occurs while CLR waits for an application domain to complete startup. |
| SQLCLR\_ASSEMBLY | Occurs while waiting for access to the loaded assembly list in the appdomain. |
| SQLCLR\_DEADLOCK\_DETECTION | Occurs while CLR waits for deadlock detection to complete. |
| SQLCLR\_QUANTUM\_PUNISHMENT | Occurs when a CLR task is throttled because it has exceeded its execution quantum. This throttling is done in order to reduce the effect of this resource-intensive task on other tasks. |
| SQLSORT\_NORMMUTEX | Occurs during internal synchronization, while initializing internal sorting structures. |
| SQLSORT\_SORTMUTEX | Occurs during internal synchronization, while initializing internal sorting structures. |
| SQLTRACE\_BUFFER\_FLUSH | Occurs when a task is waiting for a background task to flush trace buffers to disk every four seconds. |
| SQLTRACE\_LOCK | Occurs during synchronization on trace buffers during a file trace. |
| SQLTRACE\_SHUTDOWN | Occurs while trace shutdown waits for outstanding trace events to complete. |
| SQLTRACE\_WAIT\_ENTRIES | Occurs while a SQL Trace event queue waits for packets to arrive on the queue. |
| SRVPROC\_SHUTDOWN | Occurs while the shutdown process waits for internal resources to be released to shutdown cleanly. |
| TEMPOBJ | Occurs when temporary object drops are synchronized. This wait is rare, and only occurs if a task has requested exclusive access for temp table drops. |
| THREADPOOL | Occurs when a task is waiting for a worker to run on. This can indicate that the maximum worker setting is too low, or that batch executions are taking unusually long, thus reducing the number of workers available to satisfy other batches. |
| TIMEPRIV\_TIMEPERIOD | Occurs during internal synchronization of the Extended Events timer. |
| TRACEWRITE | Occurs when the SQL Trace rowset trace provider waits for either a free buffer or a buffer with events to process. |
| TRAN\_MARKLATCH\_DT | Occurs when waiting for a destroy mode latch on a transaction mark latch. Transaction mark latches are used for synchronization of commits with marked transactions. |
| TRAN\_MARKLATCH\_EX | Occurs when waiting for an exclusive mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions. |
| TRAN\_MARKLATCH\_KP | Occurs when waiting for a keep mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions. |
| TRAN\_MARKLATCH\_NL | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| TRAN\_MARKLATCH\_SH | Occurs when waiting for a shared mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions. |
| TRAN\_MARKLATCH\_UP | Occurs when waiting for an update mode latch on a marked transaction. Transaction mark latches are used for synchronization of commits with marked transactions. |
| TRANSACTION\_MUTEX | Occurs during synchronization of access to a transaction by multiple batches. |
| UTIL\_PAGE\_ALLOC | Occurs when transaction log scans wait for memory to be available during memory pressure. |
| VIA\_ACCEPT | Occurs when a Virtual Interface Adapter (VIA) provider connection is completed during startup. |
| VIEW\_DEFINITION\_MUTEX | Occurs during synchronization on access to cached view definitions. |
| WAIT\_FOR\_RESULTS | Occurs when waiting for a query notification to be triggered. |
| WAITFOR | Occurs as a result of a WAITFOR Transact-SQL statement. The duration of the wait is determined by the parameters to the statement. This is a user-initiated wait. |
| WAITFOR\_TASKSHUTDOWN | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| WAITSTAT\_MUTEX | Occurs during synchronization of access to the collection of statistics used to populate sys.dm\_os\_wait\_stats. |
| WCC | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| WORKTBL\_DROP | Occurs while pausing before retrying, after a failed worktable drop. |
| WRITE\_COMPLETION | Occurs when a write operation is in progress. |
| WRITELOG | Occurs while waiting for a log flush to complete. Common operations that cause log flushes are checkpoints and transaction commits. |
| XACT\_OWN\_TRANSACTION | Occurs while waiting to acquire ownership of a transaction. |
| XACT\_RECLAIM\_SESSION | Occurs while waiting for the current owner of a session to release ownership of the session. |
| XACTLOCKINFO | Occurs during synchronization of access to the list of locks for a transaction. In addition to the transaction itself, the list of locks is accessed by operations such as deadlock detection and lock migration during page splits. |
| XACTWORKSPACE\_MUTEX | Occurs during synchronization of defections from a transaction, as well as the number of database locks between enlist members of a transaction. |
| XE\_BUFFERMGR\_ALLPROCESSED\_EVENT | Occurs when Extended Events session buffers are flushed to targets. This wait occurs on a background thread. |
| XE\_BUFFERMGR\_FREEBUF\_EVENT | Occurs when either of the following conditions is true:   * An Extended Events session is configured for no event loss, and all buffers in the session are currently full. This can indicate that the buffers for an Extended Events session are too small, or should be partitioned. * Audits experience a delay. This can indicate a disk bottleneck on the drive where the audits are written. |
| XE\_DISPATCHER\_CONFIG\_SESSION\_LIST | Occurs when an Extended Events session that is using asynchronous targets is started or stopped. This wait indicates either of the following:   * An Extended Events session is registering with a background thread pool. * The background thread pool is calculating the required number of threads based on current load. |
| XE\_DISPATCHER\_JOIN | Occurs when a background thread that is used for Extended Events sessions is terminating. |
| XE\_DISPATCHER\_WAIT | Occurs when a background thread that is used for Extended Events sessions is waiting for event buffers to process. |
| XE\_MODULEMGR\_SYNC | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| XE\_OLS\_LOCK | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| XE\_PACKAGE\_LOCK\_BACKOFF | Identified for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FT\_COMPROWSET\_RWLOCK | Full-text is waiting on fragment metadata operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FT\_IFTS\_RWLOCK | Full-text is waiting on internal synchronization. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FT\_IFTS\_SCHEDULER\_IDLE\_WAIT | Full-text scheduler sleep wait type. The scheduler is idle. |
| FT\_IFTSHC\_MUTEX | Full-text is waiting on an fdhost control operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FT\_IFTSISM\_MUTEX | Full-text is waiting on communication operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed. |
| FT\_MASTER\_MERGE | Full-text is waiting on master merge operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed. |