SQL Server Wait Types – Ready Reference

Objective: In which version of SQL Server does a particular Wait Type exist?

For any errors, suggestions & feedback, write to amit.bansal@sqlmaestros.com

Count	Wait_Type	Description	2014	2012	2008 R2	2008
1	ABR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
2	AM_INDBUILD_ALLOCATION		☑	☑	☒	☒
3	AM_SCHEMAMGR_UNSHARED_CACHE		☑	☑	☒	☒
4	ASSEMBLY_LOAD	Occurs during exclusive access to assembly loading.	☑	☑	☑	☑
5	ASYNC_DISKPOOL_LOCK	Occurs when there is an attempt to synchronize parallel threads that are performing tasks such as creating or initializing a file.	☑	☑	☑	☑
6	ASYNC_IO_COMPLETION	Occurs when a task is waiting for I/Os to finish.	☑	☑	☑	☑
7	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.	☑	☑	☑	☑
8	ASYNC_OP_COMPLETION		☑	☒	☒	☒
9	ASYNC_OP_CONTEXT_READ		☑	☒	☒	☒
10	ASYNC_OP_CONTEXT_WRITE		☑	☒	☒	☒
11	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.	☑	☑	☑	☑
12	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.	☑	☑	☑	☑
13	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.	☑	☑	☑	☑
14	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.	☑	☑	☑	☑
15	BACKUP	Occurs when a task is blocked as part of backup processing.	☑	☑	☑	☑
16	BACKUP_OPERATOR	Occurs when a task is waiting for a tape mount. To view the tape status, query sys.dm_io_backup_tapes. If a mount operation is not pending, this wait type may indicate a hardware problem with the tape drive.	☑	☑	☑	☑
17	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.	☑	☑	☑	☑
18	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.	☑	☑	☑	☑
19	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.	☑	☑	☑	☑
20	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.	☑	☑	☑	☑
21	BMPALLOCATION		☑	☒	☒	☒
22	BMPBUILD		☑	☒	☒	☒
23	BMPREPARTITION		☑	☒	☒	☒
24	BMPREPLICATION		☑	☒	☒	☒
25	BROKER_CONNECTION_RECEIVE_TASK	Occurs when waiting for access to receive a message on a connection endpoint. Receive access to the endpoint is serialized.	☑	☑	☑	☑
26	BROKER_DISPATCHER		☑	☑	☑	☒
27	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.	☑	☑	☑	☑
28	BROKER_EVENTHANDLER	Occurs when a task is waiting in the primary event handler of the Service Broker. This should occur very briefly.	☑	☑	☑	☑
29	BROKER_FORWARDER		☑	☑	☒	☒
30	BROKER_INIT	Occurs when initializing Service Broker in each active database. This should occur infrequently.	☑	☑	☑	☑
31	BROKER_MASTERSTART	Occurs when a task is waiting for the primary event handler of the Service Broker to start. This should occur very briefly.	☑	☑	☑	☑

32	BROKER_RECEIVE_WAITFOR	Occurs when the RECEIVE WAITFOR is waiting. This is typical if no messages are ready to be received.	☑	☑	☑	☑
33	BROKER_REGISTERALLENDPOINTS	Occurs during the initialization of a Service Broker connection endpoint. This should occur very briefly.	☑	☑	☑	☑
34	BROKER_SERVICE	Occurs when the Service Broker destination list that is associated with a target service is updated or re-prioritized.	☑	☑	☑	☑
35	BROKER_SHUTDOWN	Occurs when there is a planned shutdown of Service Broker. This should occur very briefly, if at all.	☑	☑	☑	☑
36	BROKER_TASK_SHUTDOWN		☑	☒	☒	☒
37	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.	☑	☑	☑	☑
38	BROKER_TASK_SUBMIT		☑	☒	☒	☒
39	BROKER_TO_FLUSH	Occurs when the Service Broker lazy flusher flushes the in-memory transmission objects to a work table.	☑	☑	☑	☑
40	BROKER_TRANSMISSION_OBJECT		☑	☑	☒	☒
41	BROKER_TRANSMISSION_TABLE		☑	☑	☒	☒
42	BROKER_TRANSMISSION_WORK		☑	☑	☒	☒
43	BROKER_TRANSMITTER	Occurs when the Service Broker transmitter is waiting for work.	☑	☑	☑	☑
44	BUILTIN_HASHKEY_MUTEX	May occur after startup of instance, while internal data structures are initializing. Will not recur once data structures have initialized.	☑	☑	☑	☑
45	CHANGE_TRACKING_WAITFORCHANGES		☑	☑	☑	☒
46	CHECK_PRINT_RECORD	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
47	CHECKPOINT_QUEUE	Occurs while the checkpoint task is waiting for the next checkpoint request.	☑	☑	☑	☑
48	CHKPT	Occurs at server startup to tell the checkpoint thread that it can start.	☑	☑	☑	☑
49	CLEAR_DB	Occurs during operations that change the state of a database, such as opening or closing a database.	☑	☑	☑	☑
50	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.	☑	☑	☑	☑
51	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.	☑	☑	☑	☑
52	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.	☑	☑	☑	☑
53	CLR_MANUAL_EVENT	Occurs when a task is currently performing CLR execution and is waiting for a specific manual event to be initiated.	☑	☑	☑	☑
54	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.	☑	☑	☑	☑
55	CLR_MONITOR	Occurs when a task is currently performing CLR execution and is waiting to obtain a lock on the monitor.	☑	☑	☑	☑
56	CLR_RWLOCK_READER	Occurs when a task is currently performing CLR execution and is waiting for a reader lock.	☑	☑	☑	☑
57	CLR_RWLOCK_WRITER	Occurs when a task is currently performing CLR execution and is waiting for a writer lock.	☑	☑	☑	☑
58	CLR_SEMAPHORE	Occurs when a task is currently performing CLR execution and is waiting for a semaphore.	☑	☑	☑	☑
59	CLR_TASK_START	Occurs while waiting for a CLR task to complete startup.	☑	☑	☑	☑
60	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.	☑	☑	☑	☑
61	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.	☑	☑	☑	☑
62	COLUMNSTORE_BUILD_THROTTLE		☑	☒	☒	☒
63	COMMIT_TABLE		☑	☑	☑	☑
64	COUNTRECOVERYMGR		☑	☑	☒	☒
65	CREATE_DATINISERVICE		☑	☑	☒	☒
66	CXPACKET	Occurs with parallel query plans when trying to synchronize the query processor exchange iterator. If waiting is excessive and cannot be reduced by tuning the query (such as adding indexes), consider adjusting the cost threshold for parallelism or lowering the degree of parallelism.	☑	☑	☑	☑
67	CXROWSET_SYNC	Occurs during a parallel range scan.	☑	☑	☑	☑

68	DAC_INIT	Occurs while the dedicated administrator connection is initializing.	☑	☑	☑	☑
69	DBCC_SCALE_OUT_EXPR_CACHE		☑	☑	☒	☒
70	DBMIRROR_DBM_EVENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
71	DBMIRROR_DBM_MUTEX	Identified for informational purposes only. Not supported. Futur	☑	☑	☑	☑
72	DBMIRROR_EVENTS_QUEUE	Occurs when database mirroring waits for events to process.	☑	☑	☑	☑
73	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.	☑	☑	☑	☑
74	DBMIRROR_WORKER_QUEUE	Indicates that the database mirroring worker task is waiting for more work.	☑	☑	☑	☑
75	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.	☑	☑	☑	☑
76	DBSEEDING_FLOWCONTROL		☑	☒	☒	☒
77	DBSEEDING_OPERATION		☑	☒	☒	☒
78	DBSTATE		☑	☑	☒	☒
79	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.	☑	☑	☑	☑
80	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.	☑	☑	☑	☑
81	DEBUG	Occurs during Transact-SQL and CLR debugging for internal synchronization.	☑	☑	☑	☑
82	DIRTY_PAGE_POLL		☑	☑	☒	☒
83	DIRTY_PAGE_SYNC		☑	☑	☒	☒
84	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.	☑	☑	☑	☑
85	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.	☑	☑	☑	☑
86	DISPATCHER_PRIORITY_QUEUE_SEMAPHORE		☑	☑	☒	☒
87	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.	☑	☑	☑	☑
88	DLL_LOADING_MUTEX	Occurs once while waiting for the XML parser DLL to load.	☑	☑	☑	☑
89	DROP_DATABASE_TIMER_TASK		☑	☒	☒	☒
90	DROPTEMP	Occurs between attempts to drop a temporary object if the previous attempt failed. The wait duration grows exponentially with each failed drop attempt.	☑	☑	☑	☑
91	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.	☑	☑	☑	☑
92	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.	☑	☑	☑	☑
93	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.	☑	☑	☑	☑
94	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.	☑	☑	☑	☑

95	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.	☑	☑	☑	☑
96	DTC_WAITFOR_OUTCOME	Occurs when recovery tasks wait for MS DTC to become active to enable the resolution of prepared transactions.	☑	☑	☑	☑
97	DTCPNTSYNC		☑	☑	☒	☒
98	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.	☑	☑	☑	☑
99	DUMP_LOG_COORDINATOR_QUEUE		☑	☑	☑	☑
100	DUMPTRIGGER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
101	EC	Identified for informational purposes only. Not supported. Futur	☑	☑	☑	☑
102	EE_PMOLOCK	Occurs during synchronization of certain types of memory allocations during statement execution.	☑	☑	☑	☑
103	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.	☑	☑	☑	☑
104	ENABLE_EMPTY_VERSIONING		☑	☑	☒	☒
105	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.	☑	☑	☑	☑
106	ERROR_REPORTING_MANAGER	Occurs during synchronization of multiple concurrent error log initializations.	☑	☑	☑	☑
107	EXCHANGE	Occurs during synchronization in the query processor exchange iterator during parallel queries.	☑	☑	☑	☑
108	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.	☑	☑	☑	☑
109	EXECUTION_PIPE_EVENT_INTERNAL	Occurs during synchronization between producer and consumer parts of batch execution that are submitted through the connection context.	☑	☑	☑	☑
110	FABRIC_HADR_TRANSPORT_CONNECTION		☑	☒	☒	☒
111	FABRIC_REPLICA_CONTROLLER_LIST		☑	☒	☒	☒
112	FABRIC_REPLICA_CONTROLLER_STATE_AND_CONFIG		☑	☒	☒	☒
113	FABRIC_REPLICA_PUBLISHER_EVENT_PUBLISH		☑	☒	☒	☒
114	FABRIC_REPLICA_PUBLISHER_SUBSCRIBER_LIST		☑	☒	☒	☒
115	FABRIC_WAIT_FOR_BUILD_REPLICA_EVENT_PROCESSING		☑	☒	☒	☒
116	FAILPOINT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
117	FCB_REPLICA_READ	Occurs when the reads of a snapshot (or a temporary snapshot created by DBCC) sparse file are synchronized.	☑	☑	☑	☑
118	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.	☑	☑	☑	☑
119	FEATURE_SWITCHES_UPDATE		☑	☒	☒	☒
120	FFT_NSO_DB_KILL_FLAG		☑	☑	☒	☒
121	FFT_NSO_DB_LIST		☑	☑	☒	☒
122	FFT_NSO_FCB		☑	☑	☒	☒
123	FFT_NSO_FCB_FIND		☑	☑	☒	☒
124	FFT_NSO_FCB_PARENT		☑	☑	☒	☒
125	FFT_NSO_FCB_RELEASE_CACHED_ENTRIES		☑	☑	☒	☒
126	FFT_NSO_FILEOBJECT		☑	☑	☒	☒
127	FFT_NSO_TABLE_LIST		☑	☑	☒	☒
128	FFT_NTFS_STORE		☑	☑	☒	☒

129	FFT_RECOVERY		☑	☑	☒	☒
130	FFT_RSFX_COMM		☑	☑	☒	☒
131	FFT_RSFX_WAIT_FOR_MEMORY		☑	☑	☒	☒
132	FFT_STARTUP_SHUTDOWN		☑	☑	☒	☒
133	FFT_STORE_DB		☑	☑	☒	☒
134	FFT_STORE_ROWSET_LIST		☑	☑	☒	☒
135	FFT_STORE_TABLE		☑	☑	☒	☒
136	FILESTREAM_CACHE		☑	☑	☒	☒
137	FILESTREAM_CHUNKER		☑	☑	☒	☒
138	FILESTREAM_CHUNKER_INIT		☑	☑	☒	☒
139	FILESTREAM_FCB		☑	☑	☒	☒
140	FILESTREAM_FILE_OBJECT		☑	☑	☒	☒
141	FILESTREAM_WORKITEM_QUEUE		☑	☑	☒	☒
142	FILETABLE_SHUTDOWN		☑	☑	☒	☒
143	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.	☑	☑	☑	☑
144	FS_GARBAGE_COLLECTOR_SHUTDOWN	Occurs when the FILESTREAM garbage collector is waiting for cleanup tasks to be completed.	☑	☑	☑	☑
145	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).	☑	☑	☑	☑
146	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.	☑	☑	☑	☑
147	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.	☑	☑	☑	☑
148	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.	☑	☑	☑	☑
149	FSTR_CONFIG_MUTEX	Occurs when there is a wait for another FILESTREAM feature reconfiguration to be completed.	☑	☑	☑	☑
150	FSTR_CONFIG_RWLOCK	Occurs when there is a wait to serialize access to the FILESTREAM configuration parameters.	☑	☑	☑	☑
151	FT_COMPROWSET_RWLOCK	Full-text is waiting on fragment metadata operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
152	FT_IFTS_RWLOCK	Full-text is waiting on internal synchronization. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
153	FT_IFTS_SCHEDULER_IDLE_WAIT	Full-text scheduler sleep wait type. The scheduler is idle.	☑	☑	☑	☑
154	FT_IFTSHC_MUTEX	Full-text is waiting on an fdhost control operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
155	FT_IFTSISM_MUTEX	Full-text is waiting on communication operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
156	FT_MASTER_MERGE	Full-text is waiting on master merge operation. Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
157	FT_MASTER_MERGE_COORDINATOR		☑	☑	☒	☒
158	FT_METADATA_MUTEX	Documented for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
159	FT_PROPERTYLIST_CACHE		☑	☑	☒	☒
160	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.	☑	☑	☑	☑
161	FULLTEXT GATHERER	Occurs during synchronization of full-text operations.	☑	☑	☑	☑
162	GDMA_GET_RESOURCE_OWNER		☑	☑	☒	☒
163	GHOSTCLEANUPSYNCMGR		☑	☑	☒	☒

164	GUARDIAN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
165	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.	☑	☑	☒	☒
166	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.	☑	☑	☒	☒
167	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.	☑	☑	☒	☒
168	HADR_AR_UNLOAD_COMPLETED		☑	☑	☒	☒
169	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.	☑	☑	☒	☒
170	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.	☑	☑	☒	☒
171	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).	☑	☑	☒	☒
172	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.	☑	☑	☒	☒
173	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.	☑	☑	☒	☒
174	HADR_CONNECTIVITY_INFO		☑	☑	☒	☒
175	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.	☑	☑	☒	☒
176	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.	☑	☑	☒	☒
177	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.	☑	☑	☒	☒
178	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.	☑	☑	☒	☒
179	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.	☑	☑	☒	☒
180	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.	☑	☑	☒	☒
181	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.	☑	☑	☒	☒
182	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.	☑	☑	☒	☒
183	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.	☑	☑	☒	☒
184	HADR_DBSEEDING		☑	☒	☒	☒
185	HADR_DBSEEDING_LIST		☑	☒	☒	☒
186	HADR_DBSTATECHANGE_SYNC	Concurrency control wait for updating the internal state of the database replica.	☑	☑	☒	☒

187	HADR_FABRIC_CALLBACK		☑	☒	☒	☒
188	HADR_FILESTREAM_BLOCK_FLUSH	The FILESTREAM AlwaysOn transport manager is waiting until processing of a log block is finished.	☑	☑	☒	☒
189	HADR_FILESTREAM_FILE_CLOSE	The FILESTREAM AlwaysOn transport manager is waiting until the next FILESTREAM file gets processed and its handle gets closed.	☑	☑	☒	☒
190	HADR_FILESTREAM_FILE_REQUEST	An AlwaysOn secondary replica is waiting for the primary replica to send all requested FILESTREAM files during UNDO.	☑	☑	☒	☒
191	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.	☑	☑	☒	☒
192	HADR_FILESTREAM_IOMGR_IOCOMPLETION	The FILESTREAM AlwaysOn I/O manager is waiting for I/O completion.	☑	☑	☒	☒
193	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.	☑	☑	☒	☒
194	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.	☑	☑	☒	☒
195	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.	☑	☑	☒	☒
196	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.	☑	☑	☒	☒
197	HADR_LOGPROGRESS_SYNC	Concurrency control wait when updating the log progress status of database replicas.	☑	☑	☒	☒
198	HADR_NOTIFICATION_DEQUEUE	A background task that processes Windows Server Failover Clustering notifications is waiting for the next notification. Internal use only.	☑	☑	☒	☒
199	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.	☑	☑	☒	☒
200	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.	☑	☑	☒	☒
201	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.	☑	☑	☒	☒
202	HADR_PARTNER_SYNC	Concurrency control wait on the partner list.	☑	☑	☒	☒
203	HADR_READ_ALL_NETWORKS	Waiting to get read or write access to the list of WSFC networks. Internal use only.	☑	☑	☒	☒
204	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.	☑	☑	☒	☒
205	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.	☑	☑	☒	☒
206	HADR_REPLICAINFO_SYNC	Waiting for concurrency control to update the current replica state.	☑	☑	☒	☒
207	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.	☑	☑	☒	☒
208	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.	☑	☑	☒	☒
209	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.	☑	☑	☒	☒

210	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. 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.	☑	☑	☒	☒
211	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.	☑	☑	☒	☒
212	HADR_TRANSPORT_DBRLIST	Waiting for access to the transport layer's database replica list. Used for the spinlock that grants access to it.	☑	☑	☒	☒
213	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).	☑	☑	☒	☒
214	HADR_TRANSPORT_SESSION	AlwaysOn Availability Groups is waiting while changing or accessing the underlying transport state.	☑	☑	☒	☒
215	HADR_WORK_POOL	Concurrency control wait on the AlwaysOn Availability Groups background work task object.	☑	☑	☒	☒
216	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.	☑	☑	☒	☒
217	HADR_XRF_STACK_ACCESS	Accessing (look up, add, and delete) the extended recovery fork stack for an AlwaysOn availability database.	☑	☑	☒	☒
218	HTBUILD		☑	☑	☒	☒
219	HTDELETE		☑	☒	☒	☒
220	HTMEMO		☑	☒	☒	☒
221	HTREINIT		☑	☒	☒	☒
222	HTREPARTITION		☑	☑	☒	☒
223	HTTP_ENUMERATION	Occurs at startup to enumerate the HTTP endpoints to start HTTP.	☑	☑	☑	☑
224	HTTP_START	Occurs when a connection is waiting for HTTP to complete initialization.	☑	☑	☑	☑
225	HTTP_STORAGE_CONNECTION		☑	☒	☒	☒
226	IMPPROV_IOWAIT	Occurs when SQL Server waits for a bulkload I/O to finish.	☑	☑	☑	☑
227	INTERNAL_TESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
228	IO_AUDIT_MUTEX	Occurs during synchronization of trace event buffers.	☑	☑	☑	☑
229	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.	☑	☑	☑	☑
230	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.	☑	☑	☑	☑
231	IOAFF_RANGE_QUEUE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
232	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.	☑	☑	☑	☑
233	KTM_ENLISTMENT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
234	KTM_RECOVERY_MANAGER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
235	KTM_RECOVERY_RESOLUTION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
236	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.	☑	☑	☑	☑
237	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.	☑	☑	☑	☑
238	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.	☑	☑	☑	☑

239	LATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
240	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.	☑	☑	☑	☑
241	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.	☑	☑	☑	☑
242	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.	☑	☑	☑	☑
243	LCK_M_BU	Occurs when a task is waiting to acquire a Bulk Update (BU) lock.	☑	☑	☑	☑
244	LCK_M_BU_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Bulk Update (BU) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
245	LCK_M_BU_LOW_PRIORITY	Occurs when a task is waiting to acquire a Bulk Update (BU) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
246	LCK_M_IS	Occurs when a task is waiting to acquire an Intent Shared (IS) lock.	☑	☑	☑	☑
247	LCK_M_IS_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Intent Shared (IS) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
248	LCK_M_IS_LOW_PRIORITY	Occurs when a task is waiting to acquire an Intent Shared (IS) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
249	LCK_M_IU	Occurs when a task is waiting to acquire an Intent Update (IU) lock.	☑	☑	☑	☑
250	LCK_M_IU_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Intent Update (IU) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
251	LCK_M_IU_LOW_PRIORITY	Occurs when a task is waiting to acquire an Intent Update (IU) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
252	LCK_M_IX	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock.	☑	☑	☑	☑
253	LCK_M_IX_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
254	LCK_M_IX_LOW_PRIORITY	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
255	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.	☑	☑	☑	☑
256	LCK_M_RIn_NL_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a NULL lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. A NULL lock on the key is an instant release lock. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
257	LCK_M_RIn_NL_LOW_PRIORITY	Occurs when a task is waiting to acquire a NULL lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. A NULL lock on the key is an instant release lock. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
258	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.	☑	☑	☑	☑
259	LCK_M_RIn_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a shared lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
260	LCK_M_RIn_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a shared lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
261	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.	☑	☑	☑	☑

262	LCK_M_RIn_U_ABORT_BLOCKERS	Task is waiting to acquire an Update lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
263	LCK_M_RIn_U_LOW_PRIORITY	Task is waiting to acquire an Update lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
264	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.	☑	☑	☑	☑
265	LCK_M_RIn_X_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Exclusive lock with Abort Blockers on the current key value, and an Insert Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
266	LCK_M_RIn_X_LOW_PRIORITY	Occurs when a task is waiting to acquire an Exclusive lock with Low Priority on the current key value, and an Insert Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
267	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.	☑	☑	☑	☑
268	LCK_M_RS_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared lock with Abort Blockers on the current key value, and a Shared Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
269	LCK_M_RS_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared lock with Low Priority on the current key value, and a Shared Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
270	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.	☑	☑	☑	☑
271	LCK_M_RS_U_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update lock with Abort Blockers on the current key value, and an Update Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
272	LCK_M_RS_U_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update lock with Low Priority on the current key value, and an Update Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
273	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.	☑	☑	☑	☑
274	LCK_M_RX_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared lock with Abort Blockers on the current key value, and an Exclusive Range with Abort Blockers lock between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
275	LCK_M_RX_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared lock with Low Priority on the current key value, and an Exclusive Range with Low Priority lock between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
276	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.	☑	☑	☑	☑
277	LCK_M_RX_U_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update lock with Abort Blockers on the current key value, and an Exclusive range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
278	LCK_M_RX_U_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update lock with Low Priority on the current key value, and an Exclusive range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
279	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.	☑	☑	☑	☑
280	LCK_M_RX_X_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Exclusive lock with Abort Blockers on the current key value, and an Exclusive Range lock with Abort Blockers between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
281	LCK_M_RX_X_LOW_PRIORITY	Occurs when a task is waiting to acquire an Exclusive lock with Low Priority on the current key value, and an Exclusive Range lock with Low Priority between the current and previous key. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒

282	LCK_M_S	Occurs when a task is waiting to acquire a Shared lock.	☑	☑	☑	☑
283	LCK_M_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
284	LCK_M_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
285	LCK_M_SCH_M	Occurs when a task is waiting to acquire a Schema Modify lock.	☑	☑	☑	☑
286	LCK_M_SCH_M_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Schema Modify lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
287	LCK_M_SCH_M_LOW_PRIORITY	Occurs when a task is waiting to acquire a Schema Modify lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
288	LCK_M_SCH_S	Occurs when a task is waiting to acquire a Schema Share lock.	☑	☑	☑	☑
289	LCK_M_SCH_S_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Schema Share lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
290	LCK_M_SCH_S_LOW_PRIORITY	Occurs when a task is waiting to acquire a Schema Share lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
291	LCK_M_SIU	Occurs when a task is waiting to acquire a Shared With Intent Update lock.	☑	☑	☑	☑
292	LCK_M_SIU_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared With Intent Update lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
293	LCK_M_SIU_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared With Intent Update lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
294	LCK_M_SIX	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock.	☑	☑	☑	☑
295	LCK_M_SIX_ABORT_BLOCKERS	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
296	LCK_M_SIX_LOW_PRIORITY	Occurs when a task is waiting to acquire a Shared With Intent Exclusive lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
297	LCK_M_U	Occurs when a task is waiting to acquire an Update lock.	☑	☑	☑	☑
298	LCK_M_U_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
299	LCK_M_U_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
300	LCK_M_UIX	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock.	☑	☑	☑	☑
301	LCK_M_UIX_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
302	LCK_M_UIX_LOW_PRIORITY	Occurs when a task is waiting to acquire an Update With Intent Exclusive lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
303	LCK_M_X	Occurs when a task is waiting to acquire an Exclusive lock.	☑	☑	☑	☑
304	LCK_M_X_ABORT_BLOCKERS	Occurs when a task is waiting to acquire an Exclusive lock with Abort Blockers. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
305	LCK_M_X_LOW_PRIORITY	Occurs when a task is waiting to acquire an Exclusive lock with Low Priority. (Related to the low priority wait option of ALTER TABLE and ALTER INDEX.)	☑	☒	☒	☒
306	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.	☑	☑	☑	☑
307	LOGCAPTURE_LOGPOOLTRUNCPOINT		☑	☑	☒	☑
308	LOGGENERATION	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
309	LOGMGR	Occurs when a task is waiting for any outstanding log I/Os to finish before shutting down the log while closing the database.	☑	☑	☑	☑
310	LOGMGR_FLUSH	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
311	LOGMGR_QUEUE	Occurs while the log writer task waits for work requests.	☑	☑	☑	☑

312	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.	☑	☑	☑	☑
313	LOGPOOL_CACHESIZE		☑	☑	☒	☒
314	LOGPOOL_CONSUMER		☑	☑	☒	☒
315	LOGPOOL_CONSUMERSET		☑	☑	☒	☒
316	LOGPOOL_FREEPOOLS		☑	☑	☒	☒
317	LOGPOOL_MGRSET		☑	☑	☒	☒
318	LOGPOOL_REPLACEMENTSET		☑	☑	☒	☒
319	LOGPOOLREFCOUNTEDOBJECT_REFDONE		☑	☑	☒	☒
320	LOWFAIL_MEMMGR_QUEUE	Occurs while waiting for memory to be available for use.	☑	☑	☒	☒
321	MD_AGENT_YIELD		☑	☑	☒	☒
322	MD_LAZYCACHE_RWLOCK		☑	☑	☑	☑
323	MISCELLANEOUS	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
324	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.	☑	☑	☑	☑
325	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.	☑	☑	☑	☑
326	MSQL_XACT_MUTEX	Occurs during synchronization of transaction usage. A request must acquire the mutex before it can use the transaction.	☑	☑	☑	☑
327	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.	☑	☑	☑	☑
328	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.	☑	☑	☑	☑
329	NET_WAITFOR_PACKET	Occurs when a connection is waiting for a network packet during a network read.	☑	☑	☑	☑
330	NODE_CACHE_MUTEX		☑	☑	☑	☑
331	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.	☑	☑	☑	☑
332	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.	☑	☑	☑	☑
333	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.	☑	☑	☑	☑
334	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.	☑	☑	☑	☑
335	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.	☑	☑	☑	☑
336	PAGEIOLATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
337	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.	☑	☑	☑	☑
338	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.	☑	☑	☑	☑
339	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.	☑	☑	☑	☑
340	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.	☑	☑	☑	☑

341	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.	☑	☑	☑	☑
342	PAGELATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
343	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.	☑	☑	☑	☑
344	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.	☑	☑	☑	☑
345	PARALLEL_BACKUP_QUEUE	Occurs when serializing output produced by RESTORE HEADERONLY, RESTORE FILELISTONLY, or RESTORE LABELONLY.	☑	☑	☑	☑
346	PERFORMANCE_COUNTERS_RWLOCK		☑	☑	☑	☑
347	PHYSICAL_SEEDING_DMV		☑	☒	☒	☒
348	PREEMPTIVE_ABR	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
349	PREEMPTIVE_AUDIT_ACCEES_EVENTLOG	Occurs when the SQL Server Operating System (SQLOS) scheduler switches to preemptive mode to write an audit event to the Windows event log.	☑	☒	☑	☑
350	PREEMPTIVE_AUDIT_ACCEES_SECLOG	Occurs when the SQLOS scheduler switches to preemptive mode to write an audit event to the Windows Security log.	☑	☒	☑	☑
351	PREEMPTIVE_CLOSEBACKUPMEDIA	Occurs when the SQLOS scheduler switches to preemptive mode to close backup media.	☑	☑	☑	☑
352	PREEMPTIVE_CLOSEBACKUPTAPE	Occurs when the SQLOS scheduler switches to preemptive mode to close a tape backup device.	☑	☑	☑	☑
353	PREEMPTIVE_CLOSEBACKUPVDIDEVICE	Occurs when the SQLOS scheduler switches to preemptive mode to close a virtual backup device.	☑	☑	☑	☑
354	PREEMPTIVE_CLUSAPI_CLUSTERRESOURCECONTROL	Occurs when the SQLOS scheduler switches to preemptive mode to perform Windows failover cluster operations.	☑	☑	☑	☑
355	PREEMPTIVE_COM_COCREATEINSTANCE	Occurs when the SQLOS scheduler switches to preemptive mode to create a COM object.	☑	☑	☑	☑
356	PREEMPTIVE_COM_COGETCLASSOBJECT		☑	☑	☑	☑
357	PREEMPTIVE_COM_CREATEACCESSOR		☑	☑	☑	☑
358	PREEMPTIVE_COM_DELETEROWS		☑	☑	☑	☑
359	PREEMPTIVE_COM_GETCOMMANDTEXT		☑	☑	☑	☑
360	PREEMPTIVE_COM_GETDATA		☑	☑	☑	☑
361	PREEMPTIVE_COM_GETNEXTROWS		☑	☑	☑	☑
362	PREEMPTIVE_COM_GETRESULT		☑	☑	☑	☑
363	PREEMPTIVE_COM_GETROWSBYBOOKMARK		☑	☑	☑	☑
364	PREEMPTIVE_COM_LBFLUSH		☑	☑	☑	☑
365	PREEMPTIVE_COM_LBLOCKREGION		☑	☑	☑	☑
366	PREEMPTIVE_COM_LBREADAT		☑	☑	☑	☑
367	PREEMPTIVE_COM_LBSETSIZE		☑	☑	☑	☑
368	PREEMPTIVE_COM_LBSTAT		☑	☑	☑	☑
369	PREEMPTIVE_COM_LBUNLOCKREGION		☑	☑	☑	☑
370	PREEMPTIVE_COM_LBWRITEAT		☑	☑	☑	☑
371	PREEMPTIVE_COM_QUERYINTERFACE		☑	☑	☑	☑
372	PREEMPTIVE_COM_RELEASE		☑	☑	☑	☑
373	PREEMPTIVE_COM_RELEASEACCESSOR		☑	☑	☑	☑
374	PREEMPTIVE_COM_RELEASEROWS		☑	☑	☑	☑
375	PREEMPTIVE_COM_RELEASESESSION		☑	☑	☑	☑
376	PREEMPTIVE_COM_RESTARTPOSITION		☑	☑	☑	☑
377	PREEMPTIVE_COM_SEQSTRMREAD		☑	☑	☑	☑
378	PREEMPTIVE_COM_SEQSTRMREADANDWRITE		☑	☑	☑	☑
379	PREEMPTIVE_COM_SETDATAFAILURE		☑	☑	☑	☑
380	PREEMPTIVE_COM_SETPARAMETERINFO		☑	☑	☑	☑

381	PREEMPTIVE_COM_SETPARAMETERPROPERTIES		☑	☑	☑	☑
382	PREEMPTIVE_COM_STRMLOCKREGION		☑	☑	☑	☑
383	PREEMPTIVE_COM_STRMSEEKANDREAD		☑	☑	☑	☑
384	PREEMPTIVE_COM_STRMSEEKANDWRITE		☑	☑	☑	☑
385	PREEMPTIVE_COM_STRMSETSIZE		☑	☑	☑	☑
386	PREEMPTIVE_COM_STRMSTAT		☑	☑	☑	☑
387	PREEMPTIVE_COM_STRMUNLOCKREGION		☑	☑	☑	☑
388	PREEMPTIVE_CONSOLEWRITE		☑	☑	☑	☑
389	PREEMPTIVE_CREATEPARAM		☑	☑	☑	☑
390	PREEMPTIVE_DEBUG		☑	☑	☑	☑
391	PREEMPTIVE_DFSADDLINK		☑	☑	☑	☑
392	PREEMPTIVE_DFSLINKEXISTCHECK		☑	☑	☑	☑
393	PREEMPTIVE_DFSLINKHEALTHCHECK		☑	☑	☑	☑
394	PREEMPTIVE_DFSREMOVELINK		☑	☑	☑	☑
395	PREEMPTIVE_DFSREMOVEROOT		☑	☑	☑	☑
396	PREEMPTIVE_DFSROOTFOLDERCHECK		☑	☑	☑	☑
397	PREEMPTIVE_DFSROOTINIT		☑	☑	☑	☑
398	PREEMPTIVE_DFSROOTSHARECHECK		☑	☑	☑	☑
399	PREEMPTIVE_DTC_ABORT		☑	☑	☑	☑
400	PREEMPTIVE_DTC_ABORTREQUESTDONE		☑	☑	☑	☑
401	PREEMPTIVE_DTC_BEGINTRANSACTION		☑	☑	☑	☑
402	PREEMPTIVE_DTC_COMMITREQUESTDONE		☑	☑	☑	☑
403	PREEMPTIVE_DTC_ENLIST		☑	☑	☑	☑
404	PREEMPTIVE_DTC_PREPAREREQUESTDONE		☑	☑	☑	☑
405	PREEMPTIVE_FILESIZEGET		☑	☑	☑	☑
406	PREEMPTIVE_FSAOLEDB_ABORTTRANSACTION		☑	☑	☑	☑
407	PREEMPTIVE_FSAOLEDB_COMMITTRANSACTION		☑	☑	☒	☑
408	PREEMPTIVE_FSAOLEDB_STARTTRANSACTION		☑	☑	☒	☑
409	PREEMPTIVE_FSRECOVER_UNCONDITIONALUNDO		☑	☑	☑	☑
410	PREEMPTIVE_GETRMINFO		☑	☑	☑	☑
411	PREEMPTIVE_HADR_LEASE_MECHANISM	AlwaysOn Availability Groups lease manager scheduling for CSS diagnostics.	☑	☑	☒	☒
412	PREEMPTIVE_LOCKMONITOR		☑	☑	☑	☑
413	PREEMPTIVE_MSS_RELEASE		☑	☑	☑	☑
414	PREEMPTIVE_ODBCOPS		☑	☑	☑	☑
415	PREEMPTIVE_OLE_UNINIT		☑	☑	☑	☑
416	PREEMPTIVE_OLEDB_ABORTORCOMMITTRAN		☑	☑	☑	☑
417	PREEMPTIVE_OLEDB_ABORTTRAN		☑	☑	☑	☑
418	PREEMPTIVE_OLEDB_GETDATASOURCE		☑	☑	☑	☑
419	PREEMPTIVE_OLEDB_GETLITERALINFO		☑	☑	☑	☑
420	PREEMPTIVE_OLEDB_GETPROPERTIES		☑	☑	☑	☑
421	PREEMPTIVE_OLEDB_GETPROPERTYINFO		☑	☑	☑	☑
422	PREEMPTIVE_OLEDB_GETSCHEMALOCK		☑	☑	☑	☑
423	PREEMPTIVE_OLEDB_JOINTRANSACTION		☑	☑	☑	☑
424	PREEMPTIVE_OLEDB_RELEASE		☑	☑	☑	☑

425	PREEMPTIVE_OLEDB_SETPROPERTIES		☑	☑	☑	☑
426	PREEMPTIVE_OLEDBOPS		☑	☑	☑	☑
427	PREEMPTIVE_OS_ACCEPTSECURITYCONTEXT		☑	☑	☑	☑
428	PREEMPTIVE_OS_ACQUIRECREDENTIALSHANDLE		☑	☑	☑	☑
429	PREEMPTIVE_OS_AUTHENTICATIONOPS		☑	☑	☑	☑
430	PREEMPTIVE_OS_AUTHORIZATIONOPS		☑	☑	☑	☑
431	PREEMPTIVE_OS_AUTHZGETINFORMATIONFROMCONTEXT		☑	☑	☑	☑
432	PREEMPTIVE_OS_AUTHZINITIALIZECONTEXTFROMSID		☑	☑	☑	☑
433	PREEMPTIVE_OS_AUTHZINITIALIZERESOURCEMANAGER		☑	☑	☑	☑
434	PREEMPTIVE_OS_BACKUPREAD		☑	☑	☑	☑
435	PREEMPTIVE_OS_CLOSEHANDLE		☑	☑	☑	☑
436	PREEMPTIVE_OS_CLUSTEROPS		☑	☑	☑	☑
437	PREEMPTIVE_OS_COMOPS		☑	☑	☑	☑
438	PREEMPTIVE_OS_COMPLETEAUTHTOKEN		☑	☑	☑	☑
439	PREEMPTIVE_OS_COPYFILE		☑	☑	☑	☑
440	PREEMPTIVE_OS_CREATEDIRECTORY		☑	☑	☑	☑
441	PREEMPTIVE_OS_CREATEFILE		☑	☑	☑	☑
442	PREEMPTIVE_OS_CRYPTACQUIRECONTEXT		☑	☑	☑	☑
443	PREEMPTIVE_OS_CRYPTIMPORTKEY		☑	☑	☑	☑
444	PREEMPTIVE_OS_CRYPTOPS		☑	☑	☑	☑
445	PREEMPTIVE_OS_DECRYPTMESSAGE		☑	☑	☑	☑
446	PREEMPTIVE_OS_DELETEFILE		☑	☑	☑	☑
447	PREEMPTIVE_OS_DELETESECURITYCONTEXT		☑	☑	☑	☑
448	PREEMPTIVE_OS_DEVICEIOCONTROL		☑	☑	☑	☑
449	PREEMPTIVE_OS_DEVICEOPS		☑	☑	☑	☑
450	PREEMPTIVE_OS_DIRSVC_NETWORKOPS		☑	☑	☑	☑
451	PREEMPTIVE_OS_DISCONNECTNAMEDPIPE		☑	☑	☑	☑
452	PREEMPTIVE_OS_DOMAINSERVICESOPS		☑	☑	☑	☑
453	PREEMPTIVE_OS_DSGETDCNAME		☑	☑	☑	☑
454	PREEMPTIVE_OS_DTCOPS		☑	☑	☑	☑
455	PREEMPTIVE_OS_ENCRYPTMESSAGE		☑	☑	☑	☑
456	PREEMPTIVE_OS_FILEOPS		☑	☑	☑	☑
457	PREEMPTIVE_OS_FINDFILE		☑	☑	☑	☑
458	PREEMPTIVE_OS_FLUSHFILEBUFFERS		☑	☑	☑	☑
459	PREEMPTIVE_OS_FORMATMESSAGE		☑	☑	☑	☑
460	PREEMPTIVE_OS_FREECREDENTIALSHANDLE		☑	☑	☑	☑
461	PREEMPTIVE_OS_FREELIBRARY		☑	☑	☑	☑
462	PREEMPTIVE_OS_GENERICOPS		☑	☑	☑	☑
463	PREEMPTIVE_OS_GETADDRINFO		☑	☑	☑	☑
464	PREEMPTIVE_OS_GETCOMPRESSEDFILESIZE		☑	☑	☑	☑
465	PREEMPTIVE_OS_GETDISKFREESPACE		☑	☑	☑	☑
466	PREEMPTIVE_OS_GETFILEATTRIBUTES		☑	☑	☑	☑
467	PREEMPTIVE_OS_GETFILESIZE		☑	☑	☑	☑
468	PREEMPTIVE_OS_GETLONGPATHNAME		☑	☑	☑	☑

469	PREEMPTIVE_OS_GETPROCADDRESS		☑	☑	☑	☑
470	PREEMPTIVE_OS_GETVOLUMENAMEFORVOLUMEMOUNTPOINT		☑	☑	☑	☑
471	PREEMPTIVE_OS_GETVOLUMEPATHNAME		☑	☑	☑	☑
472	PREEMPTIVE_OS_INITIALIZESECURITYCONTEXT		☑	☑	☑	☑
473	PREEMPTIVE_OS_LIBRARYOPS		☑	☑	☑	☑
474	PREEMPTIVE_OS_LOADLIBRARY		☑	☑	☑	☑
475	PREEMPTIVE_OS_LOGONUSER		☑	☑	☑	☑
476	PREEMPTIVE_OS_LOOKUPACCOUNTSID		☑	☑	☑	☑
477	PREEMPTIVE_OS_MESSAGEQUEUEOPS		☑	☑	☑	☑
478	PREEMPTIVE_OS_MOVEFILE		☑	☑	☑	☑
479	PREEMPTIVE_OS_NETGROUPGETUSERS		☑	☑	☑	☑
480	PREEMPTIVE_OS_NETLOCALGROUPGETMEMBERS		☑	☑	☑	☑
481	PREEMPTIVE_OS_NETUSERGETGROUPS		☑	☑	☑	☑
482	PREEMPTIVE_OS_NETUSERGETLOCALGROUPS		☑	☑	☑	☑
483	PREEMPTIVE_OS_NETUSERMODALSGET		☑	☑	☑	☑
484	PREEMPTIVE_OS_NETVALIDATEPASSWORDPOLICY		☑	☑	☑	☑
485	PREEMPTIVE_OS_NETVALIDATEPASSWORDPOLICYFREE		☑	☑	☑	☑
486	PREEMPTIVE_OS_OPENDIRECTORY		☑	☑	☑	☑
487	PREEMPTIVE_OS_PDH_WMI_INIT		☑	☑	☒	☒
488	PREEMPTIVE_OS_PIPEOPS		☑	☑	☑	☑
489	PREEMPTIVE_OS_PROCESSOPS		☑	☑	☑	☑
490	PREEMPTIVE_OS_QUERYCONTEXTATTRIBUTES		☑	☑	☒	☒
491	PREEMPTIVE_OS_QUERYREGISTRY		☑	☑	☑	☑
492	PREEMPTIVE_OS_QUERYSECURITYCONTEXTTOKEN		☑	☑	☑	☑
493	PREEMPTIVE_OS_REMOVEDIRECTORY		☑	☑	☑	☑
494	PREEMPTIVE_OS_REPORTEVENT		☑	☑	☑	☑
495	PREEMPTIVE_OS_REVERTTOSELF		☑	☑	☑	☑
496	PREEMPTIVE_OS_RSFXDEVICEOPS		☑	☑	☑	☑
497	PREEMPTIVE_OS_SECURITYOPS		☑	☑	☑	☑
498	PREEMPTIVE_OS_SERVICEOPS		☑	☑	☑	☑
499	PREEMPTIVE_OS_SETENDOFFILE		☑	☑	☑	☑
500	PREEMPTIVE_OS_SETFILEPOINTER		☑	☑	☑	☑
501	PREEMPTIVE_OS_SETFILEVALIDDATA		☑	☑	☑	☑
502	PREEMPTIVE_OS_SETNAMEDSECURITYINFO		☑	☑	☑	☑
503	PREEMPTIVE_OS_SQLCLROPS		☑	☑	☑	☑
504	PREEMPTIVE_OS_SQMLAUNCH		☑	☑	☑	☑
505	PREEMPTIVE_OS_VERIFYSIGNATURE		☑	☑	☑	☑
506	PREEMPTIVE_OS_VSSOPS		☑	☑	☑	☑
507	PREEMPTIVE_OS_WAITFORSINGLEOBJECT		☑	☑	☑	☑
508	PREEMPTIVE_OS_WINSOCKOPS		☑	☑	☑	☑
509	PREEMPTIVE_OS_WRITEFILE		☑	☑	☑	☑
510	PREEMPTIVE_OS_WRITEFILEGATHER		☑	☑	☑	☑
511	PREEMPTIVE_OS_WSASETLASTERROR		☑	☑	☑	☑
512	PREEMPTIVE_REENLIST		☑	☑	☑	☑

513	PREEMPTIVE_RESIZELOG		☑	☑	☑	☑
514	PREEMPTIVE_ROLLFORWARDREDO		☑	☑	☑	☑
515	PREEMPTIVE_ROLLFORWARDUNDO		☑	☑	☑	☑
516	PREEMPTIVE_SB_STOPENDPOINT		☑	☑	☑	☑
517	PREEMPTIVE_SERVER_STARTUP		☑	☑	☑	☑
518	PREEMPTIVE_SETRMINFO		☑	☑	☑	☑
519	PREEMPTIVE_SHAREDMEM_GETDATA		☑	☑	☑	☑
520	PREEMPTIVE_SNIOPEN		☑	☑	☑	☑
521	PREEMPTIVE_SOSHOST		☑	☑	☑	☑
522	PREEMPTIVE_SOSTESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
523	PREEMPTIVE_SP_SERVER_DIAGNOSTICS		☑	☑	☒	☒
524	PREEMPTIVE_STARTRM		☑	☑	☑	☒
525	PREEMPTIVE_STREAMFCB_CHECKPOINT		☑	☑	☑	☒
526	PREEMPTIVE_STREAMFCB_RECOVER		☑	☑	☑	☒
527	PREEMPTIVE_STRESSDRIVER	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☒
528	PREEMPTIVE_TESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☒
529	PREEMPTIVE_TRANSIMPORT		☑	☑	☑	☒
530	PREEMPTIVE_UNMARSHALPROPAGATIONTOKEN		☑	☑	☑	☒
531	PREEMPTIVE_VSS_CREATESNAPSHOT		☑	☑	☑	☒
532	PREEMPTIVE_VSS_CREATEVOLUMESNAPSHOT		☑	☑	☑	☒
533	PREEMPTIVE_XE_CALLBACKEXECUTE		☑	☑	☑	☒
534	PREEMPTIVE_XE_CX_FILE_OPEN		☑	☒	☒	☒
535	PREEMPTIVE_XE_CX_HTTP_CALL		☑	☒	☒	☒
536	PREEMPTIVE_XE_DISPATCHER		☑	☑	☑	☑
537	PREEMPTIVE_XE_ENGINEINIT		☑	☑	☑	☑
538	PREEMPTIVE_XE_GETTARGETSTATE		☑	☑	☑	☑
539	PREEMPTIVE_XE_SESSIONCOMMIT		☑	☑	☑	☑
540	PREEMPTIVE_XE_TARGETFINALIZE		☑	☑	☑	☑
541	PREEMPTIVE_XE_TARGETINIT		☑	☑	☑	☑
542	PREEMPTIVE_XE_TIMERRUN		☑	☑	☑	☑
543	PREEMPTIVE_XETESTING	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
544	PRINT_ROLLBACK_PROGRESS	Used to wait while user processes are ended in a database that h	☑	☑	☑	☑
545	PRU_ROLLBACK_DEFERRED		☑	☑	☒	☒
546	PWAIT_ALL_COMPONENTS_INITIALIZED		☑	☑	☒	☒
547	PWAIT_COOP_SCAN		☑	☑	☒	☒
548	PWAIT_EVENT_SESSION_INIT_MUTEX		☑	☑	☒	☒
549	PWAIT_HADR_ACTION_COMPLETED		☑	☑	☒	☒
550	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.	☑	☑	☒	☒
551	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	☑	☑	☒	☒
552	PWAIT_HADR_FAILOVER_COMPLETED		☑	☒	☒	☒
553	PWAIT_HADR_JOIN		☑	☒	☒	☒

554	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.	☑	☑	☒	☒
555	PWAIT_HADR_ONLINE_COMPLETED	An AlwaysOn create or failover availability group operation is waiting for the target availability group to come online.	☑	☑	☒	☒
556	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.	☑	☑	☒	☒
557	PWAIT_HADR_SERVER_READY_CONNECTIONS		☑	☑	☒	☒
558	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.	☑	☑	☒	☒
559	PWAIT_LOG_CONSOLIDATION_IO		☑	☒	☒	☒
560	PWAIT_LOG_CONSOLIDATION_POLL		☑	☒	☒	☒
561	PWAIT_MD_LOGIN_STATS	Occurs during internal synchronization in metadata on login stats.	☑	☑	☒	☒
562	PWAIT_MD_RELATION_CACHE	Occurs during internal synchronization in metadata on table or index.	☑	☑	☒	☒
563	PWAIT_MD_SERVER_CACHE	Occurs during internal synchronization in metadata on linked servers.	☑	☑	☒	☒
564	PWAIT_MD_UPGRADE_CONFIG	Occurs during internal synchronization in upgrading server wide configurations.	☑	☑	☒	☒
565	PWAIT_PREEMPTIVE_AUDIT_ACCESS_WINDOWSLOG		☑	☑	☒	☒
566	PWAIT_QRY_BPMEMORY		☑	☑	☒	☒
567	PWAIT_REPLICA_ONLINE_INIT_MUTEX		☑	☑	☒	☒
568	PWAIT_RESOURCE_SEMAPHORE_FT_PARALLEL_QUERY_SYNC		☑	☑	☒	☒
569	PWAIT_SECURITY_CACHE_INVALIDATION		☑	☑	☒	☒
570	PWAIT_XTP_FSSTORAGE_MAINTENANCE		☑	☒	☒	☒
571	PWAIT_XTP_HOST_STORAGE_WAIT		☑	☒	☒	☒
572	QDS_ASYNC_CHECK_CONSISTENCY_TASK		☑	☒	☒	☒
573	QDS_ASYNC_PERSIST_TASK		☑	☒	☒	☒
574	QDS_ASYNC_PERSIST_TASK_START		☑	☒	☒	☒
575	QDS_BCKG_TASK		☑	☒	☒	☒
576	QDS_CLEANUP_STALE_QUERIES_TASK_MAIN_LOOP_SLEEP		☑	☒	☒	☒
577	QDS_CTXS		☑	☒	☒	☒
578	QDS_DB_DISK		☑	☒	☒	☒
579	QDS_DYN_VECTOR		☑	☒	☒	☒
580	QDS_LOADDB		☑	☒	☒	☒
581	QDS_PERSIST_TASK_MAIN_LOOP_SLEEP		☑	☒	☒	☒
582	QDS_SHUTDOWN_QUEUE		☑	☒	☒	☒
583	QDS_STMT		☑	☒	☒	☒
584	QDS_STMT_DISK		☑	☒	☒	☒
585	QDS_TASK_SHUTDOWN		☑	☒	☒	☒
586	QDS_TASK_START		☑	☒	☒	☒
587	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.	☑	☑	☑	☑
588	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.	☑	☑	☑	☑

589	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.	☑	☑	☑	☑
590	QRY_PARALLEL_THREAD_MUTEX		☑	☑	☒	☒
591	QUERY_ERRHDL_SERVICE_DONE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☒	☑	☑
592	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.	☑	☑	☑	☑
593	QUERY_NOTIFICATION_MGR_MUTEX	Occurs during synchronization of the garbage collection queue in the Query Notification Manager.	☑	☑	☑	☑
594	QUERY_NOTIFICATION_SUBSCRIPTION_MUTEX	Occurs during state synchronization for transactions in Query Notifications.	☑	☑	☑	☑
595	QUERY_NOTIFICATION_TABLE_MGR_MUTEX	Occurs during internal synchronization within the Query Notification Manager.	☑	☑	☑	☑
596	QUERY_NOTIFICATION_UNITTEST_MUTEX	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
597	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.	☑	☑	☑	☑
598	QUERY_TASK_ENQUEUE_MUTEX		☑	☑	☒	☒
599	QUERY_TRACEOUT	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
600	QUERY_WAIT_ERRHDL_SERVICE	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☒	☑	☑
601	RECOVER_CHANGEDB	Occurs during synchronization of database status in warm standby database.	☑	☑	☑	☑
602	REDO_THREAD_PENDING_WORK		☑	☑	☒	☒
603	REDO_THREAD_SYNC		☑	☑	☒	☒
604	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.	☑	☑	☑	☑
605	REPL_HISTORYCACHE_ACCESS		☑	☑	☑	☑
606	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.	☑	☑	☑	☑
607	REPL_TRANFSINFO_ACCESS		☑	☒	☒	☒
608	REPL_TRANHASHTABLE_ACCESS		☑	☑	☑	☑
609	REPL_TRANTEXTINFO_ACCESS		☑	☒	☒	☑
610	REPLICA_WRITES	Occurs while a task waits for completion of page writes to database snapshots or DBCC replicas.	☑	☑	☑	☑
611	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.	☑	☑	☑	☑
612	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.	☑	☑	☑	☑
613	RESMGR_THROTTLED	Occurs when a new request comes in and is throttled based on the GROUP_MAX_REQUESTS setting.	☑	☑	☑	☑
614	RESOURCE_GOVERNOR_IDLE		☑	☑	☒	☑
615	RESOURCE_QUEUE	Occurs during synchronization of various internal resource queues.	☑	☑	☑	☑
616	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.	☑	☑	☑	☑
617	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.	☑	☑	☑	☑
618	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.	☑	☑	☑	☑

619	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.	☑	☒	☑	☑
620	RG_RECONFIG		☑	☑	☑	☑
621	RTDATA_LIST		☑	☒	☒	☒
622	SCAN_CHAR_HASH_ARRAY_INITIALIZATION		☑	☑	☒	☒
623	SEC_DROP_TEMP_KEY	Occurs after a failed attempt to drop a temporary security key before a retry attempt.	☑	☑	☑	☑
624	SECURITY_CRYPTO_CONTEXT_MUTEX		☑	☑	☒	☒
625	SECURITY_KEYRING_RWLOCK		☑	☑	☒	☒
626	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.	☑	☑	☑	☑
627	SECURITY_RULETABLE_MUTEX		☑	☑	☒	☒
628	SEMPLAT_DSI_BUILD		☑	☑	☒	☒
629	SEQUENCE_GENERATION		☑	☒	☒	☒
630	SEQUENTIAL_GUID	Occurs while a new sequential GUID is being obtained.	☑	☑	☑	☑
631	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.	☑	☑	☑	☑
632	SERVER_RECONFIGURE		☑	☑	☑	☒
633	SHUTDOWN	Occurs while a shutdown statement waits for active connections to exit.	☑	☑	☑	☑
634	SLEEP_BPOOL_FLUSH	Occurs when a checkpoint is throttling the issuance of new I/Os in order to avoid flooding the disk subsystem.	☑	☑	☑	☑
635	SLEEP_DBSTARTUP	Occurs during database startup while waiting for all databases to recover.	☑	☑	☑	☑
636	SLEEP_DCOMSTARTUP	Occurs once at most during SQL Server instance startup while waiting for DCOM initialization to complete.	☑	☑	☑	☑
637	SLEEP_MASTERDBREADY		☑	☑	☒	☒
638	SLEEP_MASTERMDREADY		☑	☑	☒	☒
639	SLEEP_MASTERUPGRADED		☑	☑	☒	☒
640	SLEEP_MSDBSTARTUP	Occurs when SQL Trace waits for the msdb database to complete startup.	☑	☑	☑	☑
641	SLEEP_SYSTEMTASK	Occurs during the start of a background task while waiting for tempdb to complete startup.	☑	☑	☑	☑
642	SLEEP_TASK	Occurs when a task sleeps while waiting for a generic event to occur.	☑	☑	☑	☑
643	SLEEP_TEMPDBSTARTUP	Occurs while a task waits for tempdb to complete startup.	☑	☑	☑	☑
644	SLO_UPDATE		☑	☒	☒	☒
645	SNI_CONN_DUP		☑	☒	☒	☒
646	SNI_CRITICAL_SECTION	Occurs during internal synchronization within SQL Server networking components.	☑	☑	☑	☑
647	SNI_HTTP_WAITFOR_0_DISCON	Occurs during SQL Server shutdown, while waiting for outstanding HTTP connections to exit.	☑	☑	☑	☑
648	SNI_LISTENER_ACCESS	Occurs while waiting for non-uniform memory access (NUMA) nodes to update state change. Access to state change is serialized.	☑	☑	☑	☑
649	SNI_TASK_COMPLETION	Occurs when there is a wait for all tasks to finish during a NUMA node state change.	☑	☑	☑	☑
650	SOAP_READ	Occurs while waiting for an HTTP network read to complete.	☑	☑	☑	☑
651	SOAP_WRITE	Occurs while waiting for an HTTP network write to complete.	☑	☑	☑	☑
652	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.	☑	☑	☑	☑
653	SOS_DISPATCHER_MUTEX	Occurs during internal synchronization of the dispatcher pool. This includes when the pool is being adjusted.	☑	☑	☑	☑

654	SOS_LOCALLOCATORLIST	Occurs during internal synchronization in the SQL Server memory manager.	☑	☒	☑	☑
655	SOS_MEMORY_TOPLEVELBLOCKALLOCATOR		☑	☑	☒	☒
656	SOS_MEMORY_USAGE_ADJUSTMENT	Occurs when memory usage is being adjusted among pools.	☑	☑	☑	☑
657	SOS_OBJECT_STORE_DESTROY_MUTEX	Occurs during internal synchronization in memory pools when destroying objects from the pool.	☑	☑	☑	☑
658	SOS_PHYS_PAGE_CACHE	Accounts for the time a thread waits to acquire the mutex it must acquire before it allocates physical pages or before it returns those pages to the operating system. Waits on this type only appear if the instance of SQL Server uses AWE memory.	☑	☑	☒	☒
659	SOS_PROCESS_AFFINITY_MUTEX	Occurs during synchronizing of access to process affinity settings.	☑	☑	☑	☑
660	SOS_RESERVEDMEMBLOCKLIST	Occurs during internal synchronization in the SQL Server memory manager.	☑	☒	☑	☑
661	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.	☑	☑	☑	☑
662	SOS_SMALL_PAGE_ALLOC	Occurs during the allocation and freeing of memory that is managed by some memory objects.	☑	☑	☑	☑
663	SOS_STACKSTORE_INIT_MUTEX	Occurs during synchronization of internal store initialization.	☑	☑	☑	☑
664	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.	☑	☑	☑	☑
665	SOS_VIRTUALMEMORY_LOW	Occurs when a memory allocation waits for a resource manager to free up virtual memory.	☑	☑	☑	☑
666	SOSHOST_EVENT	Occurs when a hosted component, such as CLR, waits on a SQL Server event synchronization object.	☑	☑	☑	☑
667	SOSHOST_INTERNAL	Occurs during synchronization of memory manager callbacks used by hosted components, such as CLR.	☑	☑	☑	☑
668	SOSHOST_MUTEX	Occurs when a hosted component, such as CLR, waits on a SQL Server mutex synchronization object.	☑	☑	☑	☑
669	SOSHOST_RWLOCK	Occurs when a hosted component, such as CLR, waits on a SQL Server reader-writer synchronization object.	☑	☑	☑	☑
670	SOSHOST_SEMAPHORE	Occurs when a hosted component, such as CLR, waits on a SQL Server semaphore synchronization object.	☑	☑	☑	☑
671	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.	☑	☑	☑	☑
672	SOSHOST_TRACELOCK	Occurs during synchronization of access to trace streams.	☑	☑	☑	☑
673	SOSHOST_WAITFORDONE	Occurs when a hosted component, such as CLR, waits for a task to complete.	☑	☑	☑	☑
674	SP_PREEMPTIVE_SERVER_DIAGNOSTICS_SLEEP		☑	☑	☒	☒
675	SP_SERVER_DIAGNOSTICS_BUFFER_ACCESS		☑	☑	☒	☒
676	SP_SERVER_DIAGNOSTICS_INIT_MUTEX		☑	☑	☒	☒
677	SP_SERVER_DIAGNOSTICS_SLEEP		☑	☑	☒	☒
678	SQLCLR_APPDOMAIN	Occurs while CLR waits for an application domain to complete startup.	☑	☑	☑	☑
679	SQLCLR_ASSEMBLY	Occurs while waiting for access to the loaded assembly list in the appdomain.	☑	☑	☑	☑
680	SQLCLR_DEADLOCK_DETECTION	Occurs while CLR waits for deadlock detection to complete.	☑	☑	☑	☑
681	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.	☑	☑	☑	☑
682	SQLSORT_NORMMUTEX	Occurs during internal synchronization, while initializing internal sorting structures.	☑	☑	☑	☑
683	SQLSORT_SORTMUTEX	Occurs during internal synchronization, while initializing internal sorting structures.	☑	☑	☑	☑
684	SQLTRACE_BUFFER_FLUSH	Occurs when a task is waiting for a background task to flush trace buffers to disk every four seconds.	☑	☒	☒	☑
685	SQLTRACE_FILE_BUFFER	Occurs during synchronization on trace buffers during a file trace.	☑	☑	☑	☒
686	SQLTRACE_FILE_READ_IO_COMPLETION		☑	☑	☑	☒
687	SQLTRACE_FILE_WRITE_IO_COMPLETION		☑	☑	☑	☒
688	SQLTRACE_INCREMENTAL_FLUSH_SLEEP		☑	☑	☑	☒
689	SQLTRACE_LOCK		☑	☒	☒	☑
690	SQLTRACE_PENDING_BUFFER_WRITERS		☑	☑	☑	☒
691	SQLTRACE_SHUTDOWN	Occurs while trace shutdown waits for outstanding trace events to complete.	☑	☑	☑	☑

692	SQLTRACE_WAIT_ENTRIES	Occurs while a SQL Trace event queue waits for packets to arrive on the queue.	☑	☑	☑	☑
693	SRVPROC_SHUTDOWN	Occurs while the shutdown process waits for internal resources to be released to shutdown cleanly.	☑	☑	☑	☑
694	STARTUP_DEPENDENCY_MANAGER		☑	☑	☒	☒
695	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.	☑	☑	☑	☑
696	TERMINATE_LISTENER		☑	☒	☒	☒
697	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.	☑	☑	☑	☑
698	TIMEPRIV_TIMEPERIOD	Occurs during internal synchronization of the Extended Events timer.	☑	☑	☑	☑
699	TRACE_EVTNOTIF		☑	☑	☑	☑
700	TRACEWRITE	Occurs when the SQL Trace rowset trace provider waits for either a free buffer or a buffer with events to process.	☑	☑	☑	☑
701	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.	☑	☑	☑	☑
702	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.	☑	☑	☑	☑
703	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.	☑	☑	☑	☑
704	TRAN_MARKLATCH_NL	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
705	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.	☑	☑	☑	☑
706	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.	☑	☑	☑	☑
707	TRANSACTION_MUTEX	Occurs during synchronization of access to a transaction by multiple batches.	☑	☑	☑	☑
708	UCS_ENDPOINT_CHANGE		☑	☑	☒	☒
709	UCS_MANAGER		☑	☑	☒	☒
710	UCS_MEMORY_NOTIFICATION		☑	☑	☒	☒
711	UCS_SESSION_REGISTRATION		☑	☑	☒	☒
712	UCS_TRANSPORT		☑	☑	☒	☒
713	UCS_TRANSPORT_STREAM_CHANGE		☑	☑	☒	☒
714	UTIL_PAGE_ALLOC	Occurs when transaction log scans wait for memory to be available during memory pressure.	☑	☑	☑	☑
715	VDI_CLIENT_COMPLETECOMMAND		☑	☒	☒	☒
716	VDI_CLIENT_GETCOMMAND		☑	☒	☒	☒
717	VDI_CLIENT_OPERATION		☑	☒	☒	☒
718	VDI_CLIENT_OTHER		☑	☒	☒	☒
719	VERSIONING_COMMITTING		☑	☑	☒	☒
720	VIA_ACCEPT	Occurs when a Virtual Interface Adapter (VIA) provider connection is completed during startup.	☑	☑	☑	☑
721	VIEW_DEFINITION_MUTEX	Occurs during synchronization on access to cached view definitions.	☑	☑	☑	☑
722	WAIT_FOR_RESULTS	Occurs when waiting for a query notification to be triggered.	☑	☑	☑	☑
723	WAIT_SCRIPTDEPLOYMENT_REQUEST		☑	☒	☒	☒
724	WAIT_SCRIPTDEPLOYMENT_WORKER		☑	☒	☒	☒
725	WAIT_XTP_ASYNC_TX_COMPLETION		☑	☒	☒	☒
726	WAIT_XTP_CKPT_AGENT_WAKEUP		☑	☒	☒	☒
727	WAIT_XTP_CKPT_CLOSE	Occurs when waiting for a checkpoint to complete.	☑	☒	☒	☒

728	WAIT_XTP_CKPT_ENABLED	Occurs when checkpointing is disabled, and waiting for checkpointing to be enabled.	☑	☒	☒	☒
729	WAIT_XTP_CKPT_STATE_LOCK	Occurs when synchronizing checking of checkpoint state.	☑	☒	☒	☒
730	WAIT_XTP_GUEST	Occurs when the database memory allocator needs to stop receiving low-memory notifications.	☑	☑	☒	☒
731	WAIT_XTP_HOST_WAIT	Occurs when waits are triggered by the database engine and implemented by the host.	☑	☒	☒	☒
732	WAIT_XTP_OFFLINE_CKPT_BEFORE_REDO		☑	☒	☒	☒
733	WAIT_XTP_OFFLINE_CKPT_LOG_IO	Occurs when offline checkpoint is waiting for a log read IO to complete.	☑	☒	☒	☒
734	WAIT_XTP_OFFLINE_CKPT_NEW_LOG	Occurs when offline checkpoint is waiting for new log records to scan.	☑	☒	☒	☒
735	WAIT_XTP_PROCEDURE_ENTRY	Occurs when a drop procedure is waiting for all current executions of that procedure to complete.	☑	☒	☒	☒
736	WAIT_XTP_RECOVERY		☑	☒	☒	☒
737	WAIT_XTP_TASK_SHUTDOWN	Occurs when waiting for an In-Memory OLTP thread to complete.	☑	☑	☒	☒
738	WAIT_XTP_TRAN_COMMIT	Occurs when execution of a natively compiled stored procedure is waiting for an XTP transaction to commit (waiting for transactions dependent on for instance).	☑	☑	☒	☒
739	WAIT_XTP_TRAN_DEPENDENCY	Occurs when waiting for transaction dependencies.	☑	☒	☒	☒
740	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.	☑	☑	☑	☑
741	WAITFOR_PER_QUEUE		☑	☑	☒	☒
742	WAITFOR_TASKSHUTDOWN	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
743	WAITSTAT_MUTEX	Occurs during synchronization of access to the collection of statistics used to populate sys.dm_os_wait_stats.	☑	☑	☑	☑
744	WCC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
745	WINFAB_API_CALL		☑	☒	☒	☒
746	WINFAB_REPLICA_BUILD_OPERATION		☑	☒	☒	☒
747	WORKTBL_DROP	Occurs while pausing before retrying, after a failed worktable drop.	☑	☑	☑	☑
748	WRITE_COMPLETION	Occurs when a write operation is in progress.	☑	☑	☑	☑
749	WRITELOG	Occurs while waiting for a log flush to complete. Common operations that cause log flushes are checkpoints and transaction commits.	☑	☑	☑	☑
750	XACT_OWN_TRANSACTION	Occurs while waiting to acquire ownership of a transaction.	☑	☑	☑	☑
751	XACT_RECLAIM_SESSION	Occurs while waiting for the current owner of a session to release ownership of the session.	☑	☑	☑	☑
752	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.	☑	☑	☑	☑
753	XACTWORKSPACE_MUTEX	Occurs during synchronization of defections from a transaction, as well as the number of database locks between enlist members of a transaction.	☑	☑	☒	☑
754	XDES_HISTORY		☑	☑	☒	☒
755	XDES_OUT_OF_ORDER_LIST		☑	☑	☒	☒
756	XDES_SNAPSHOT		☑	☑	☒	☒
757	XDESTSVERMGR		☑	☑	☒	☒
758	XE_BUFFERMGR_ALLPROCESSED_EVENT	Occurs when Extended Events session buffers are flushed to targets. This wait occurs on a background thread.	☑	☑	☑	☑
759	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.	☑	☑	☑	☑
760	XE_CALLBACK_LIST		☑	☑	☒	☒
761	XE_CX_FILE_READ		☑	☑	☒	☒

762	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.	☑	☑	☑	☑
763	XE_DISPATCHER_JOIN	Occurs when a background thread that is used for Extended Events sessions is terminating.	☑	☑	☑	☑
764	XE_DISPATCHER_WAIT	Occurs when a background thread that is used for Extended Events sessions is waiting for event buffers to process.	☑	☑	☑	☑
765	XE_LIVE_TARGET_TVF		☑	☑	☒	☑
766	XE_MODULEMGR_SYNC	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
767	XE_OLS_LOCK	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☑	☑	☑
768	XE_PACKAGE_LOCK_BACKOFF	Identified for informational purposes only. Not supported. Future compatibility is not guaranteed.	☑	☒	☑	☑
769	XE_SERVICES_EVENTMANUAL		☑	☑	☑	☑
770	XE_SERVICES_MUTEX		☑	☑	☑	☑
771	XE_SERVICES_RWLOCK		☑	☑	☑	☑
772	XE_SESSION_CREATE_SYNC		☑	☑	☑	☑
773	XE_SESSION_FLUSH		☑	☑	☑	☑
774	XE_SESSION_SYNC		☑	☑	☑	☑
775	XE_STM_CREATE		☑	☑	☑	☑
776	XE_TIMER_EVENT		☑	☑	☑	☑
777	XE_TIMER_MUTEX		☑	☑	☑	☑
778	XE_TIMER_TASK_DONE		☑	☒	☑	☑
779	XTP_HOST_DB_COLLECTION		☑	☒	☒	☒
780	XTP_HOST_LOG_ACTIVITY		☑	☒	☒	☒
781	XTPPROC_CACHE_ACCESS	Occurs when for accessing all natively compiled stored procedure cache objects.	☑	☒	☒	☒
782	XTPPROC_PARTITIONED_STACK_CREATE	Occurs when allocating per-NUMA node natively compiled stored procedure cache structures (must be done single threaded) for a given procedure.	☑	☑	☒	☒