AutoWork System

DGP's AutoWork automated processing is a federated computing system that acts to schedule calls to web service API methods for all of its functionality. Its processing nodes are able to scale out horizontally within each location to match the growth (and increased automated processing workload) of the main DGP system.

Each instance of the AutoWork app first calls an API method to claim a batch of queue records that configure and control each automated process. It then hands off any claimed records to be run on their own respective thread pool threads. At that point, the work of the scheduler is complete, and it then loops to repeat those two steps using the specified interval between iterations.

Database tables function as queues using special hints such as READPAST, etc. to enforce row-level locking. The records in the queue table serve multiple purposes. First, they contain configuration info needed to run the data-driven automated processes. They also store the global shared mutable state of the processes within a location so that each iteration can be independently executed by any available worker thread. Finally, each queue record serves as a token that is claimed by an instance of the AutoWork application for its own exclusive use.

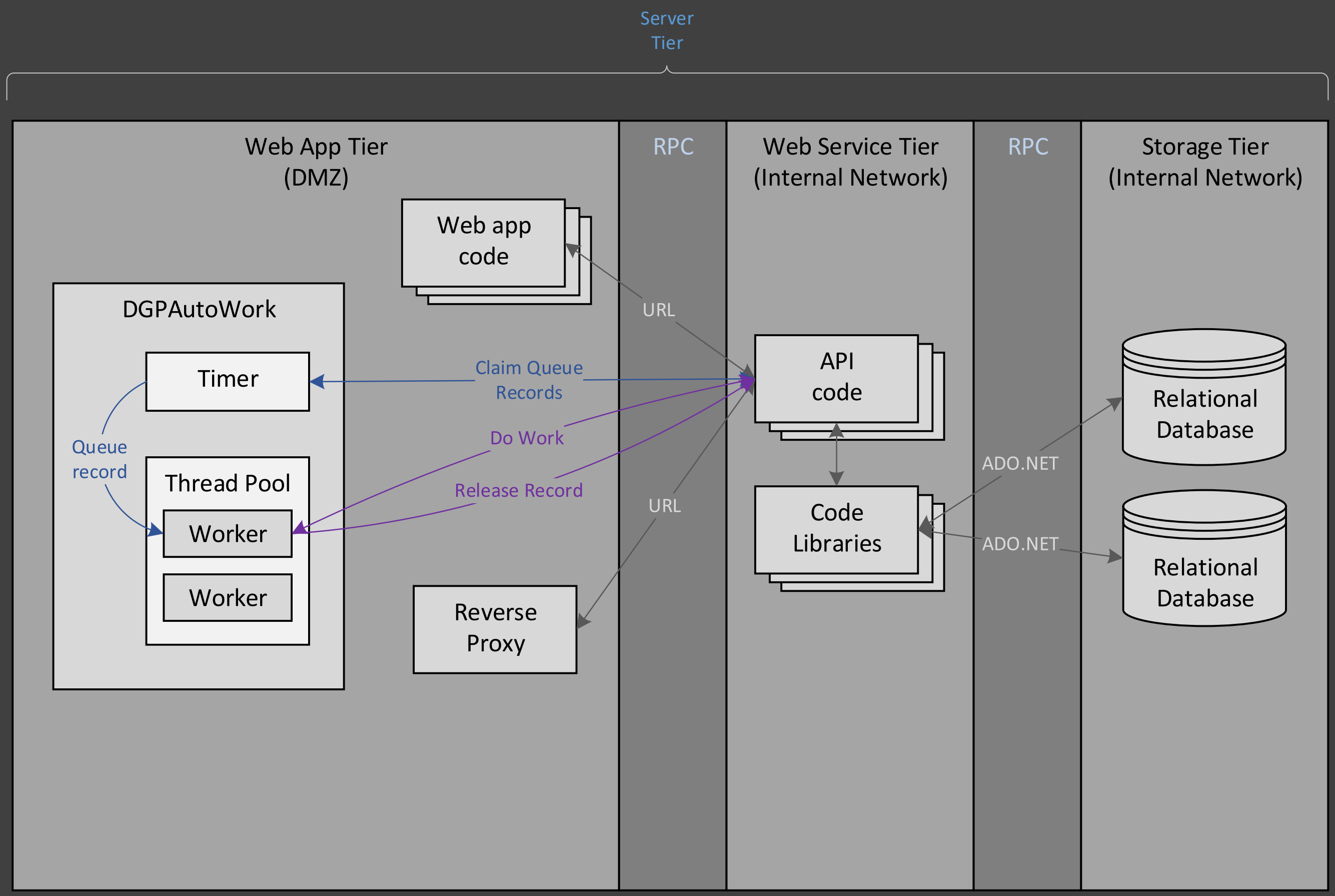
Multiple instances of the AutoWork app race to claim these queue records, which provides fault-tolerance, plus excellent scalability and performance. The contention of this race condition is easily handled by the database engine within each location. Once claimed, the queue record acts as a token granting the exclusive right to execute the automated process it represents to the specific worker thread that claimed it.

All iterations of each process are designed as autonomous one-way asynchronous methods which are guaranteed to be executed "once and only once". In practice, this is achieved by running an "at least once" polling process which loops until each process iteration has been executed, while the idempotent "only once" process logic guarantees that the logic is only executed the first time it is run. Virtually all of the functionality in DGP has been implemented as web service API methods, which includes all of the automated processes. This means that all of the automated work in a DGP system is run by executing RPC's controlled and authorized by the data-driven RBAC security system used by all of the other web service API's. This also allows the logic of all the automated processes to be centrally managed and maintained as part of the DGP web services.

Another big advantage of this approach is that it increases the overall performance and scalability of the AutoWork subsystem through the efficient use of thread pool threads (managed by the Task Parallel Library) combined with IO completion ports for the IO-bound RPC's. The same pattern applies to all IO-bound processing in DGP, including the web services in the system as they make ADO.NET RPC calls to SQL Server (the execution of large processing workloads as a continuous series of small incremental batches of work prevents the various automated processes from becoming CPU-bound).

The DGPAutoWork Windows Service:

- Claims queue records which act as a token controlling the execution of each automated process
- Each claimed record is handed off to its own worker thread to run in parallel
- Each worker thread calls one or more API methods to do the work of the process and then calls another API method to release the queue record



AutoWork Application

The AutoWork application is a .NET Framework Windows Service that can also be run as a console app for debugging when needed. The app is basically a scheduler that calls web service API methods based on timer events, and has very little functionality of its own (for ease of maintenance, extensibility, etc.). The objective of this design is to minimize the need to update the AutoWork Windows service as much as possible. Instead, all updates, fixes, etc. to the automated process logic is done within the API methods.

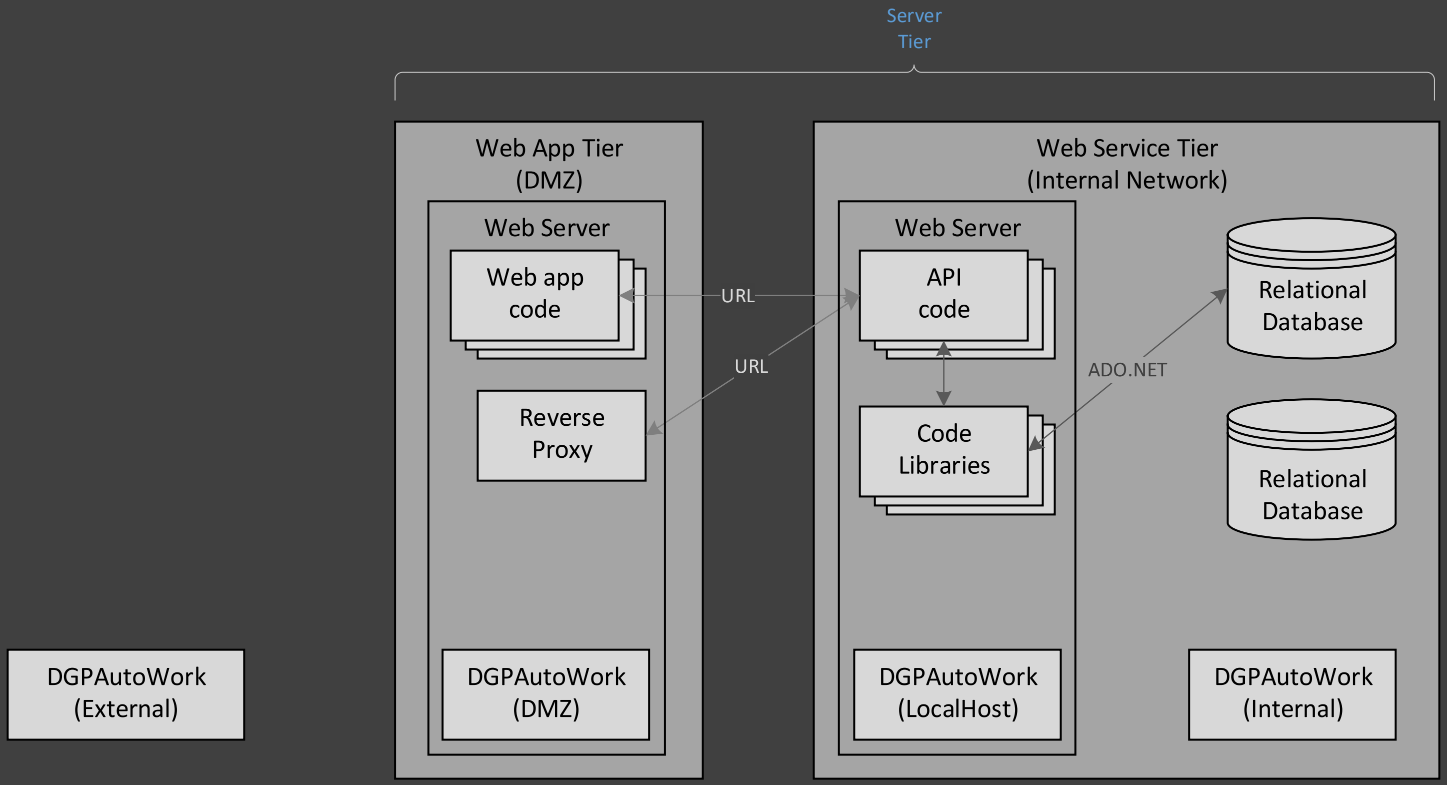
Each of the timers in the app follows the same basic pattern:

1. A Timer event is handled on its own worker thread and calls a web service API method to attempt to claim queue records.
2. A DataTable containing any queue records that were successfully claimed is returned. The timer thread then passes each claimed queue record into its own separate worker thread (Parallel.ForEach) as a one-way “fire and forget” process to run the iteration of the automated work. When each claimed record has been handed off to its own independent worker thread, the timer event handler thread is released.
    a. Each one-way RPC process updates its own claimed work record values to release it for its next iteration.
    b. Exceptions and errors encountered during automated processes are logged, and also affect the data stored in the queue record, which suspends execution of the process until the problem has been resolved.
    c. If logging for the work record is turned on, the RPC process step details are logged to the AutoWorkLog table.

Field Name	Field Values	Description
LocState	ONLINE, OFFLINE	A deprecated mechanism to disable/suspend the AutoWork app
SvcURL		The URL of the DGP web service that the AutoWork app calls to claim work records
AcctName		The DGP user account name used to call the DGP web service
AcctPword		The password of the DGP account used to call the DGP web service
ClaimID		Unique ID value for the application to use when claiming work records.
ReplicaWork	LOCALHOST, INTERNAL, DMZ, EXTERNAL, OFF	The value is one of the network area labels, or OFF to disable the ReplicaWork timer.

ReplicaWorkMS	Int	The number of MS for the ReplicaWork timer interval. Zero pauses the timer, similar to disabling it.
ReplicaMaxBatch	Int	Max number of ReplicaWork records to claim per iteration.
GeneralWork	LOCALHOST, INTERNAL, DMZ, EXTERNAL, OFF	The value is one of the network area labels, or OFF to disable the GeneralWork timer.
GeneralWorkMS	Int	The number of MS for the ReplicaWork timer interval. Zero pauses the timer, similar to disabling it.
GeneralMaxBatch	Int	Max number of GeneralWork records to claim per iteration.
QueueCheck	LOCALHOST, INTERNAL, DMZ, EXTERNAL, OFF	The value is one of the network area labels, or OFF to disable the QueueCheck timer.
QueueCheckMS	Int	The number of MS for the QueueCheck timer interval. Zero pauses the timer, similar to disabling it.
ErrIntervalSec	600	The NextRun schedule interval to use when a process is offline
EventSource	.NET Runtime	Default event source for logging to the local Event Viewer
EventID	1000	Default event ID used for the default event source

The query to claim work records will only search for records that match the network area where it is running (set in the App.config file) in order to insure that the URL's in the claimed work records will be able to be resolved. Also, each iteration to claim queue records has its own unique identifier. Queue records that encounter a problem will not be claimed by subsequent threads from the same instance of the AutoWork service. The QueueCheck automated process scans the queue tables looking for this type of issue, and logs the existence of queue records that have become "stuck" in their processing phase.



There are 4 different network areas within a DGP location.

- LocalHost runs on the same web server as the web service API's
- Internal runs on other computers in the internal network
- DMZ runs on the web servers the DMZ network (or dedicated computers)
- External runs on computers outside of the DGP location

The timers used by the AutoWork app use a separate thread for every timer event handler. The objective is to have the process run by each event handler thread complete its work in less time than the timer interval so that execution of one iteration does not overlap with the next. However, if overlap does occur, it generally does not cause any problems (the event handler logic is reentrant), and is ultimately limited by the total number of work records in a configuration table. Using replication as an example in a system with 3 locations, the ReplicaWork table will have 24 work records (12 total tables in the SysInfo and Lattice databases combined, with one set of 12 records for each replicated location). The GeneralWork table is similar, but its work records are geared toward detecting errors in the data and are scheduled to run much less frequently. The final timer is the QueueScan process which scans the two queue tables themselves for any records that have been claimed for too long without being completed and released.

<u>ReplicaWork</u>

The default interval for the timer to claim ReplicaWork records is 100 MS, or 10 queries per second. It generally takes less than 50 MS to complete an event handler iteration. If the maximum batch size is set to 15 records and there was only a single instance of the AutoWork app, some work records would have to wait 2 iterations to be claimed (200 MS). If two or more instances of the AutoWork app were running, then all records would normally be claimed within a single timer iteration, which is obviously preferable. The average lag to claim queue records should be factored into the interval set for each queue work record.

IMPORTANT NOTE: the interval used to reschedule an iteration is automatically adjusted based on the conditions at runtime. For example, if the number of records returned in an iteration batch equals the maximum number allowed, then a backlog of work is assumed and the NextRun value is set for immediate execution. For most errors, the NextRun value is not set, which along with the error state value suspends execution of the automated process. Some errors are caused by a location being offline, in which case the NexRun interval is slowed down (the ErrIntervalSec value is configurable) in order to minimize the number of error log entries without needing to manually disable/enable all automated processes waiting for a location to come back online.

Finally, the use of claimed work records as a “token” of exclusive ownership causes the various API process methods to be run sequentially (finish to start with the interval value as the lag between iterations). The work record interval and batch size should be balanced so that each iteration is completed in roughly half the time set for the interval, while also making sure that it is able to keep up with the rate at which records are being added to the system. If keeping up with the rate of new records becomes a problem, then optimistic release of the work records can be used. Optimistic release updates and releases the queue records with new state values *before* the process work is started, allowing a degree of parallel execution of the automated work processes.

One potential problem can occur when the same process methods are configured for multiple network areas in a location, or possibly duplicated between a push and a pull configuration in different locations. Since all of the automated process logic is idempotent, the only issue in this situation is that the work record iterations are wasted attempts to run an idempotent processes that do no work.

AutoWork Observability

Running the AutoWork application in console mode will show real-time information about the process to claim work records and the results of the automated work itself in the console screen. In contrast, running the processes in the AutoWork Tester shows the results of the processes and process steps stored in the AutoWorkLog table after they have been completed (assuming logging has been turned ON for the process being observed). The process steps logged in the AutoWorkLog table should be analyzed periodically to observe if the average RunLag and process iteration duration times are within the expected ranges.

In addition, each RPC process has a maximum duration MS set as part of the work record. As a general guideline, it should be set as double the average run duration for a given location. If a process iteration duration exceeds its MaxDurMS value, an error will be logged. The AutoWork app uses the same two-step logging as is used throughout a DGP system, writing the error first to the local Event Viewer and then calling an API method to write the same info to the DGPErrors table.

Stepping through the code of the AutoWork app allows developers to trace the execution of claiming queue records and the orchestration processes calling web service API methods. The logic of at least some of the process API methods themselves can be tested using the API Tester test harness.