AutoWork System (optional)

DGP's AutoWork automated processing is a distributed, highly scalable system that schedules calls to web service API methods to execute all of its automated functionality. Its processing nodes (web servers) are able to scale out horizontally within each location to match the growth (and increased automated processing workload) of the main DGP system. The practice of breaking up large workloads into multiple sequential iterations of small batches of work is the key that enables this type of distributed automation system to be feasible. The use of SOAP one-way (fire-and-forget) API methods to execute each iteration of the various processes allows the stateless web server farms to process all of the automated workload, relying on the web servers to handle the multi-threading and parallel execution of the iterations of many processes.

The AutoWork Windows service first calls an API method to claim a batch of queue records which are used to configure and control each automated process iteration. It then calls the SOAP one-way controller specified by the claimed record, which executes its iteration on its own respective thread pool thread (thanks to the web server hosting the one-way web service). At that point, due to the request/acknowledge pattern of the one-way methods, the work of the scheduler service 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 process iteration. They also store the global shared mutable state of the processes so that each iteration can be independently executed by any available worker thread. Finally, each queue record acts as a token granting the exclusive right to execute an iteration of the automated process it represents to the specific worker thread that claimed it.

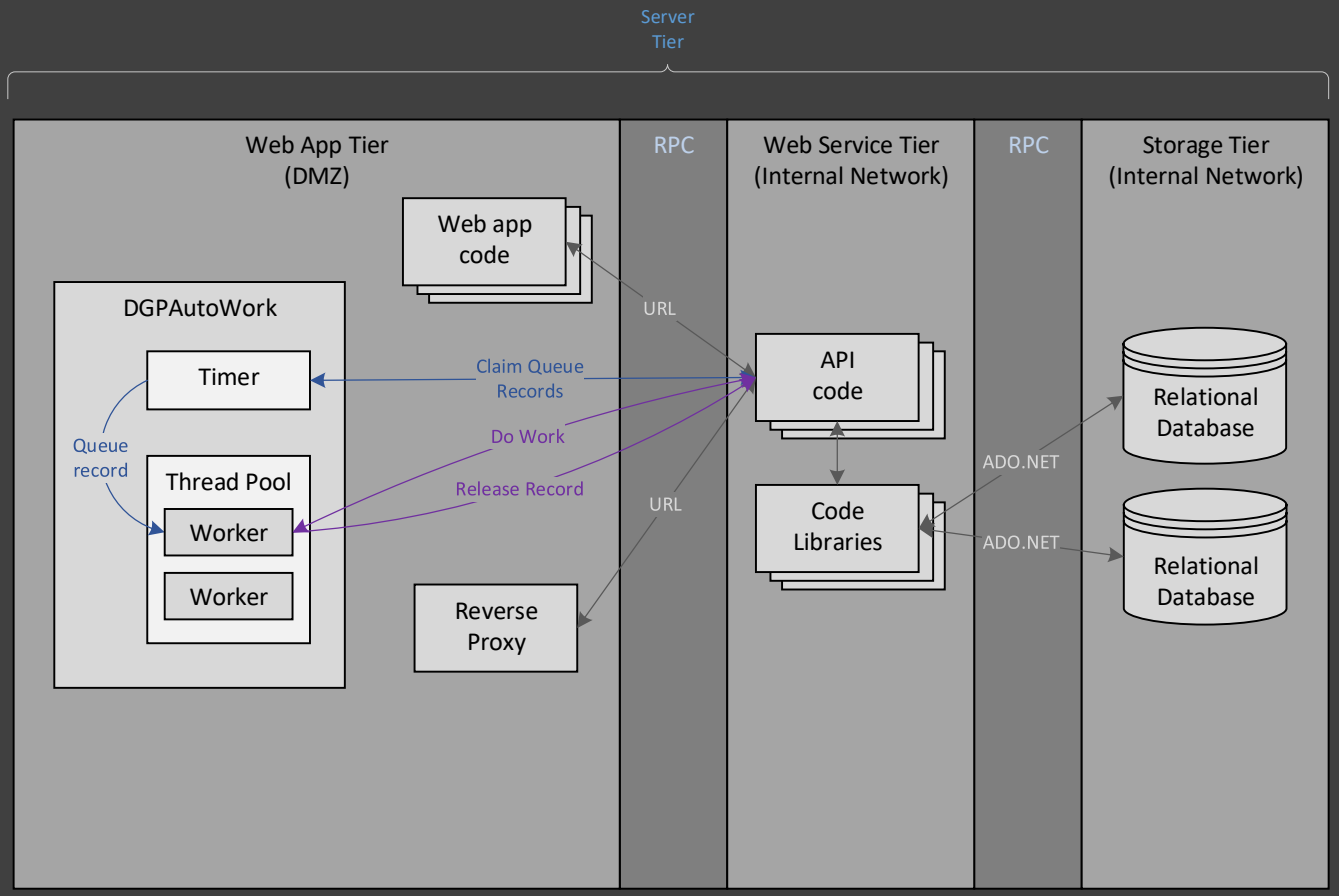
Multiple instances of the AutoWork service race to claim these queue records, which provides fault-tolerance, plus excellent overall scalability and performance. The contention of this race condition is easily handled by the database engine within each location.

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

same 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.

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



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 thanks to the IO completion ports that are used by the Windows OS 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.

AutoWork Scheduler

The AutoWork scheduler 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 service, since its basic scheduler functionality does not change.

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

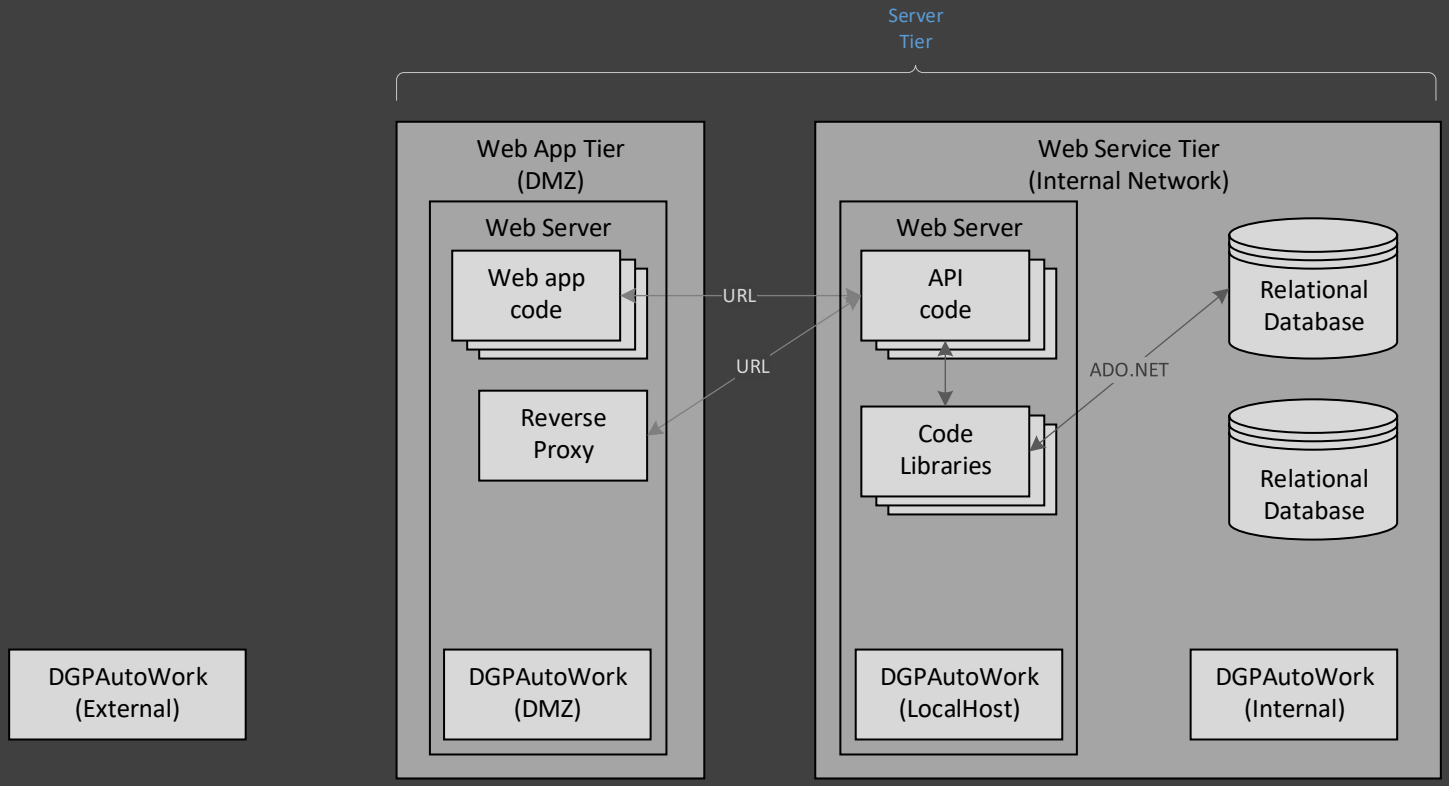
1. A Timer event is handled on its own worker thread and calls a web service API method to attempt to claim queue records.
2. A DataTable containing any queue records that were successfully claimed is returned. The timer thread then passes each claimed queue record as an input into a one-way "fire and forget" API method to run the iteration of the automated work. When each claimed record has been handed off to a one-way API method, the timer event handler thread is released.
    a. Each one-way API method 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. If the fire-and-forget process encounters a problem, the queue record will not be released. 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, and then releases those records.

This mechanism could generate a large number of error messages in the error log, so a system that notifies administrators of errors must be added to production systems (Splunk is one example). These types of systems can be used to scan various sources of logged error messages in order to notify admins when problems occur.



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

When scheduling automated work, the objective is to have the iteration run by each one-way API 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, the queue record will not be available, and will have to wait for the next scheduler iteration (slightly increasing the iteration lag time).

Using replication as an example, each location has a total of 14 tables in the SysInfo and other various DGPDrive databases combined (some databases such as SysMetrics and SysWork are not replicated between locations). If a location is configured for both push and pull replication, it can have as many as 28 records for replication and the same number for verification. The GeneralWork table is similar, but its work records have little or no state to be maintained, and are geared toward detecting errors in the data, etc. 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.

ReplicaWork

The ReplicaWork timer interval and batch size should be adjusted so that all queue records will have an opportunity to be claimed within one second. For example, the default interval for the timer to claim ReplicaWork records is 200 MS, or 5 queries of the queue table to claim queue records per second. The current DGP system has 14 replica database tables. If push replication was used at each location, they would have 28 ReplicaWork records for replication, and 28 records for verification. If the queue batch size was set to 10 records, all 28 queue records would normally be claimed within one second during a backlog of work.

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 batch size, then a backlog of work is assumed and the NextRun value is set for immediate execution. Otherwise, it's NextRun is scheduled using the interval value.

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 a variable amount of lag between iterations). The work record interval and batch size should be adjusted so that each iteration is completed in roughly the time set for the interval, while also making sure that it is able to keep up with the average rate at which records are being added to the specific table. Also, the maximum replication batch size is determined by the number of records that can fit within the 80K maximum size of API request and response messages. Ideally, the iteration batch size should be adjusted so that the average duration of the process is slightly less than the scheduled interval.