## DB Setup Prerequisites

DGP initially maintained its database schemas and core data using SQL scripts. However, maintaining these scripts proved to be very difficult, especially when variables were introduced to eliminate default admin account credentials, static naming conventions, etc. Also, maintaining the idempotent core data as SQL scripts quickly became unworkable.

The DB Setup app was designed and built as the solution to these problems. The idempotent SQL schema scripts were embedded into "_DDL" classes and merged with the other SQL data access classes. The idempotent core data scripts were rebuilt as "_DATA" classes using C# to eliminate much of the redundancy as well as manage the embedded variable values, and then also merged with the other data access classes. The idempotent core data was handled as if it were being replicated from a virtual master database that only exists in code. This allows the core data to be repeatedly updated in existing systems without interfering with real replicated data.

<u>DBSetup App.config keys</u>

| Field Name | Field Values | Description |
|---|---|---|
| AppName | DGPSetup | The app name used in logging info and errors |
| EventSource | .NET Runtime | The default event source to use for logging info to the Event Viewer if a custom event source has not been created |
| EventID | 1000 | The event ID to use when logging info to the default Event Viewer |

Using SQL Server Management Studio (SSMS), create the following 5 empty databases if they do not already exist:

- DGP_hostname_SysInfo
- DGP_hostname_SysMetrics
- DGP_hostname_SysWork
- DGP_hostname_FileStore
- DGP_hostname_FileShard1
- DGP_hostname_FileShard2

Naming conventions will vary greatly from one organization to another. DGP differs from most systems in that:

1. There are no default names, accounts, paths, etc. in a DGP system. Instead, pretty much everything is configurable.
2. This is done mainly to improve security. Default admin accounts, admin passwords, etc. are supposed to be reset by the administrators of a system, but in many/most cases – they are not. DGP solves this problem by creating a super admin account name, password, server names, app names, and so on as part of the setup, using the DBSetup utility.
3. This also allows for a DGP system to use organization naming conventions, rather than impose default DGP names.

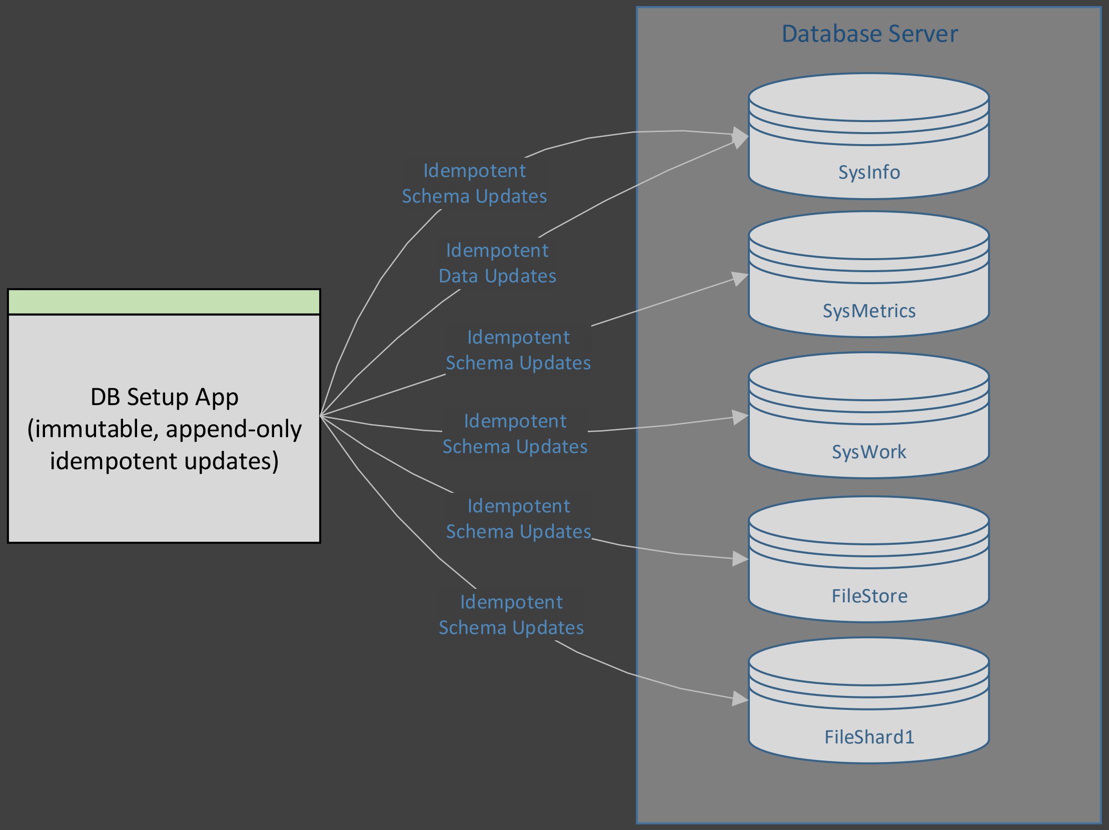
The downside of this approach is that admins can lock themselves out of their own DGP system if they lose the customized credentials and other secret info created during setup. Also, there are many options when creating databases, such as where the .mdf and .ldf files will be stored, which Filegroups will be used, what collation will be used, and so on. The choices made for these options depend a great deal on which environment the databases belong to. A developer will just use all the default values to quickly set up the databases on their development workstation, and use the local SQL Server "sa" account in their connection strings (all of which is fine for development), but none of those shortcuts are acceptable when setting up a QA or production environment.

Therefore, the only rule is that the name of each database must be unique in all of the locations for the environment it belongs to. It is a very good idea to create a naming convention for unique names that also differentiate between the different systems, environments and locations. By default, the names of host servers must already be unique in most cases, so a simple schemaname_hostname convention works pretty well. Short abbreviations for systems, environments and locations can be added as necessary. Try to keep the names as short as possible while still being human-readable, and most of all – unique.

## DB Setup Utility

After creating the empty databases configured in a way that is appropriate for their environment, the next step is to run the DB Setup utility to build the tables in the empty databases and populate the SysInfo tables with core (security) data.

The DB Setup application is the UI for all of the Master Data Management functionality in a DGP system. It is used not only to set up database schemas for the first time, but is run repeatedly to apply ongoing immutable append-only changes to the schemas and core data in a system over time.


Database Server
DB Setup App (immutable, append-only idempotent updates)
Idempotent Schema Updates
Idempotent Data Updates
Idempotent Schema Updates
Idempotent Schema Updates
Idempotent Schema Updates
Idempotent Schema Updates
SysInfo
SysMetrics
SysWork
FileStore
FileShard1

Keeping all of the database schemas and core data synchronized on every dev workstation, every test server, every QA server, and every production server, in multiple distributed locations is a challenge unique to distributed grid systems. The purpose of the DB Setup utility is to make all of that work fairly easy to accomplish.

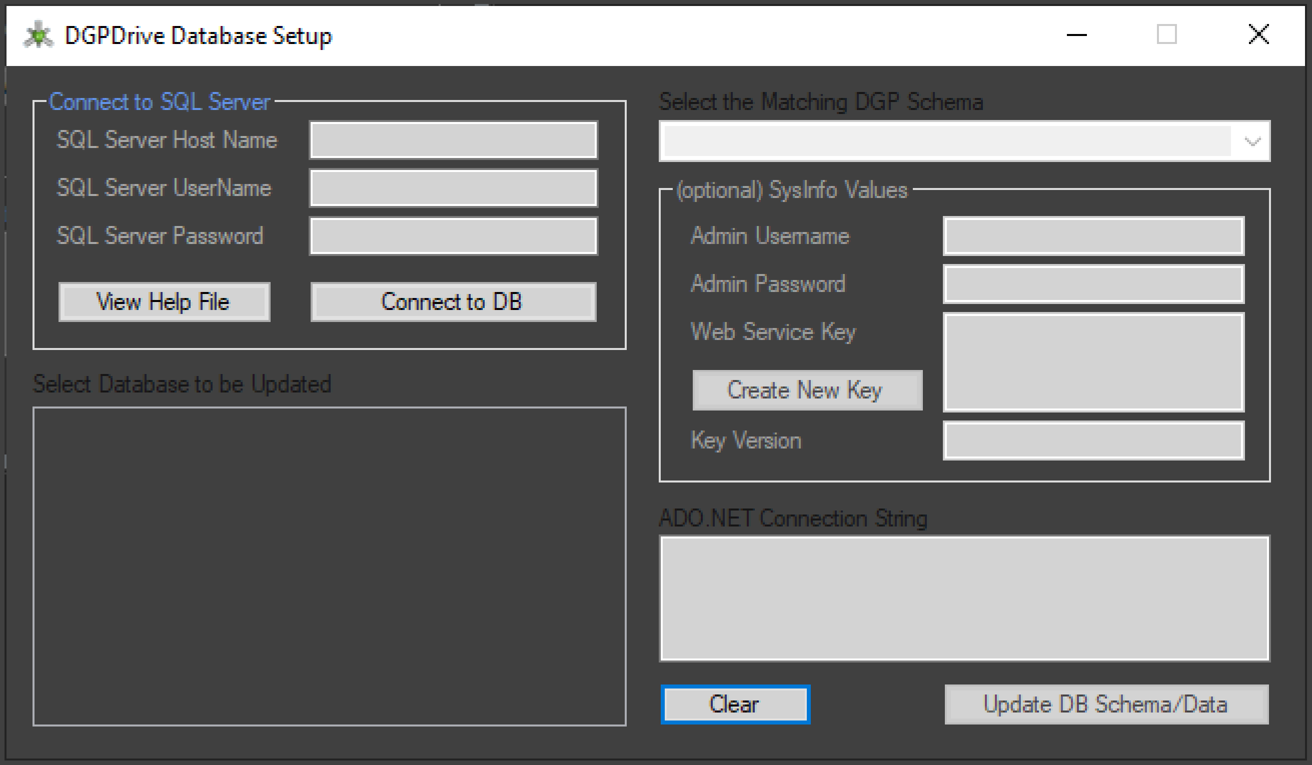
First, the database schemas for every table are maintained in code, specifically the tablename_ddl classes, which are SQL scripts embedded in the C# code and executed over an ADO.NET connection. They are designed to build the tables and apply alter table changes only if those steps have not already been run (idempotent processes for all parts of the database schemas). DGP replica schemas follow an immutable append-only convention, so all changes to a schema in the tablename_ddl classes can only ADD new elements to the table schema, and never change or delete existing elements. For example, adding new columns, new indexes, etc. to an existing table, or adding a new table to an existing database, and so on.

Second, idempotent processes do the same for all the core data in a system, which currently consist of the security data stored in the tables of the SysInfo security database. DGP systems use a data-driven Role-Based Access Control (RBAC) subsystem for account authentication and API method authorization. In order to be able to use a new system, or a new location added to a system, the data that defines all of the API methods, base roles, admin user accounts and data groups must be populated in the SysInfo database, or the DGP system will not work.

The DB Setup utility handles this by populating all of the core records in each table *as if they had been replicated* from a master database that in actuality only exists in the code of the utility. This allows the security records to be identical in every database of each location of each environment in a system, while using the customized configuration and secret data of each location/environment for their respective ADO.NET connection strings, etc. In addition, the "replicated" core data is safely ignored by the real replication processes synching all of the data created by the users of that system.

As a system evolves over time, changes to the schemas and additional core data are added to the DB Setup utility _ddl and _data classes, which then adds them to all of the databases of the system. Changing the database schemas can easily cause deadlocks if they are done while the system is in use, therefore the DB Setup utility should only be run when the database server has been taken offline for maintenance and patching.

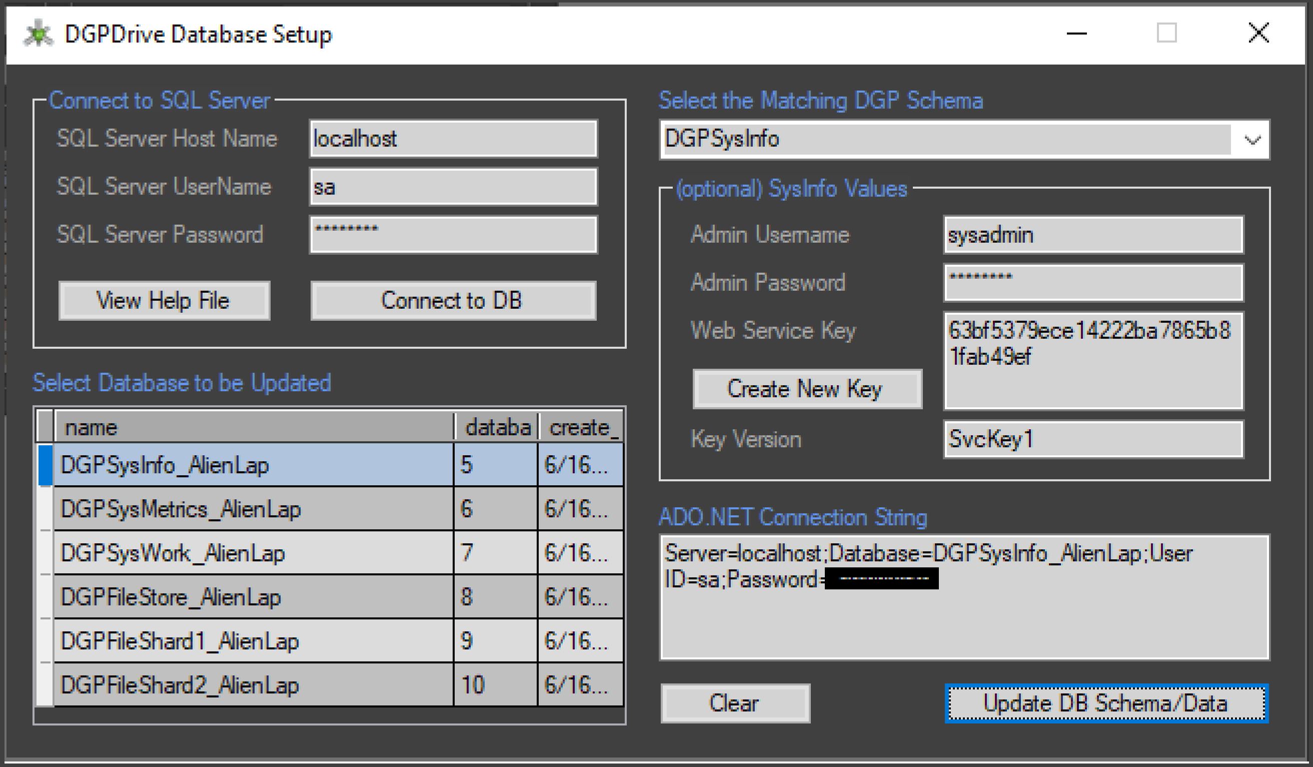
In summary, the core security data "replicated" by the utility into each database schema is only intended for security data and admin accounts needed to access a new system. Once that has been accomplished, another option is to use DGP replication to synchronize data between locations, and in some cases between different environments.



Some of the values that will be generated and displayed in this UI are system "secrets" and must be saved securely. A good password manager can be used to save not only the DGP super admin account name and password, but also the other values as notes linked to that entry in that software. Exactly how the data is saved does not matter, but losing it means being locked out of your own system.

*Step 1* when using the utility is to connect to SQL Server. The DB Setup utility must be run on the internal network which has direct connectivity to the SQL Server database host. Use one of the SQL Server admin accounts and click the Connect to DB button.

Entering the credentials from the Mixed Mode configuration of SQL Server allows the utility to connect to the SQL Server database engine, and returns a list of all the non-system databases hosted by that instance. This list should include the 6 empty databases created in a previous step.



| | |
|---|---|
| SQL Server Host Name | localhost |
| SQL Server UserName | sa |
| SQL Server Password | ******** |
| Select the Matching DGP Schema | DGPSysInfo |
| Admin Username | sysadmin |
| Admin Password | ******** |
| Web Service Key | 63bf5379ece14222ba7865b81fab49ef |
| Key Version | SvcKey1 |

| name | databa | create_ |
|---|---|---|
| DGPSysInfo_AlienLap | 5 | 6/16... |
| DGPSysMetrics_AlienLap | 6 | 6/16... |
| DGPSysWork_AlienLap | 7 | 6/16... |
| DGPFileStore_AlienLap | 8 | 6/16... |
| DGPFileShard1_AlienLap | 9 | 6/16... |
| DGPFileShard2_AlienLap | 10 | 6/16... |

```
Server=localhost;Database=DGPSysInfo_AlienLap;User
ID=sa;Password=[redacted]
```

*Step 2* is to select a specific database to be maintained from the list of databases.

*Step 3* is to then choose the DGP schema that matches the selected database from the combo box. Having no default names or naming conventions in DGP means the admin must select which schema matches which database.

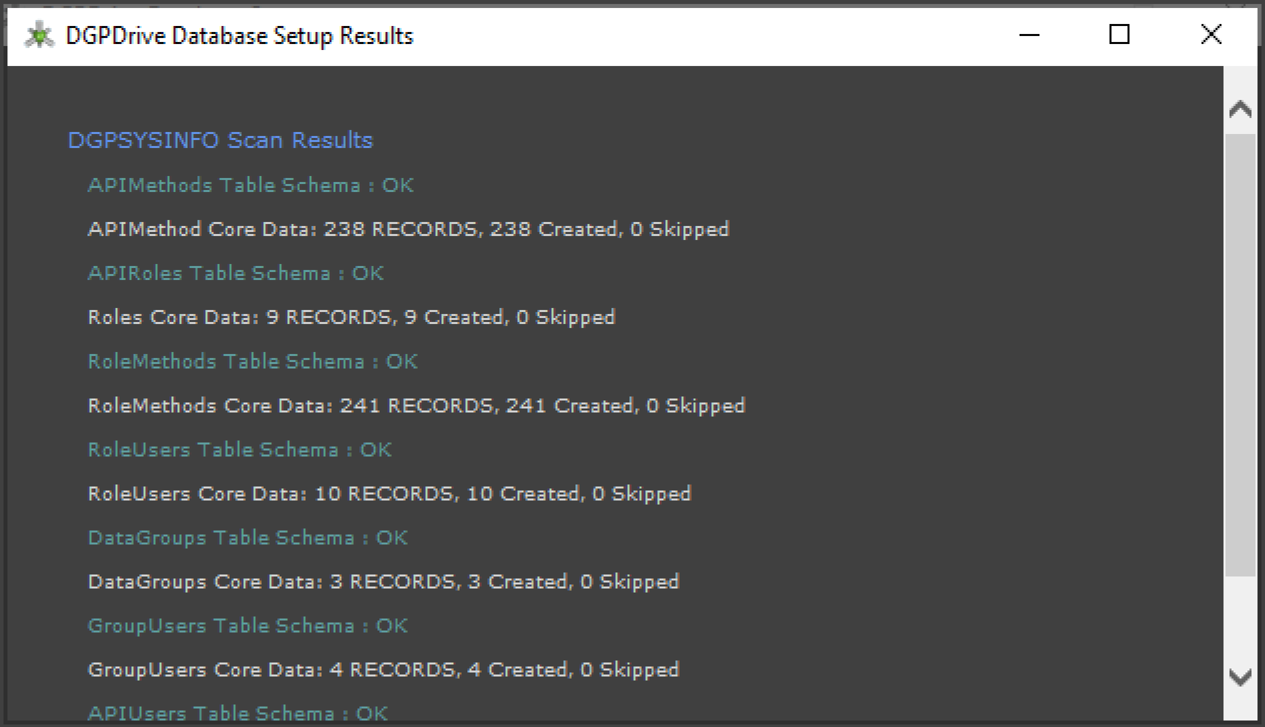
If the SysInfo schema is chosen in step 3, an additional step is required to enter (or create) the name of the DGP super admin account, and its password. Also, the current service encryption key must be pasted into the Web Service Key field (or created new the first time a system is configured) along with a key version label.  The encryption key is needed to encrypt the password of the DGP system admin account before it is stored, which is part of the zero-knowledge encryption used in DGP systems.

The account info, encryption key and key version must be stored securely so that those values are not lost.  Losing the credentials can lock you out of your DGP system.  Losing the encryption key means that the data stored in the system cannot be decrypted or used.

Every time the DB Setup utility is run for an existing SysInfo database, the correct super admin username, password, encryption key and key version values must be entered/copied into the UI.  If any of those values are missing, the UI will not run any core data processes from the "_DATA" classes, and will only run the schema update processes from the "_DDL" classes.

***Step 4*** is to click the update schema button, which will display the results of the idempotent update processes for the schema (and core data when applicable).  If data records are found to already exist, they are skipped.  If they are not found, they are created.

The ADO.NET connection string displayed in the UI should be saved so it can be copied into the Web.config file of the DGP web service.  It will use the SQL Server account name and password used to connect the DB Setup app to SQL Server to run the SQL scripts that maintain the schema and core data.  The username and password in the connection string should be edited to the SQL Server system account that is intended to be used for each database (that is why the credentials are displayed in clear text – but the DB Setup utility can only be run by admins when they are inside the location network, so the risk to security should be minimal).



```
DGPSYSINFO Scan Results
  APIMethods Table Schema : OK
  APIMethod Core Data: 238 RECORDS, 238 Created, 0 Skipped
  APIRoles Table Schema : OK
  Roles Core Data: 9 RECORDS, 9 Created, 0 Skipped
  RoleMethods Table Schema : OK
  RoleMethods Core Data: 241 RECORDS, 241 Created, 0 Skipped
  RoleUsers Table Schema : OK
  RoleUsers Core Data: 10 RECORDS, 10 Created, 0 Skipped
  DataGroups Table Schema : OK
  DataGroups Core Data: 3 RECORDS, 3 Created, 0 Skipped
  GroupUsers Table Schema : OK
  GroupUsers Core Data: 4 RECORDS, 4 Created, 0 Skipped
  APIUsers Table Schema : OK
```

Important Note: Many types of modifications to DGP schemas will use a SCH-M (schema modification) lock. In addition, some modifications will cause a table to be rebuilt, including all of its data – which can take a long time to complete if the table contains many records. For this reason, it is best to run the DB Setup modifications as part of a monthly patching and maintenance process, and NOT as part of a more frequent deployment and testing process. In other words, the database and/or host server being maintained should be offline for the duration of the DB Setup modifications.

<u>DGPWebSvc Web.config Keys</u>

| Web.Config Key | Sample Value | Description |
|---|---|---|
| SvcKeyVersion | SvcKeyV1 | The label of the current encryption key |
| SvcKeyV1 | (32-byte encryption key) | The value of the specified encryption key (maintains a list of current and all previous encryption keys) |
| | | |
| SysInfo | | The ADO.NET connection string for the DGPSysInfo database |
| SysWork | | The ADO.NET connection string for the DGPSysWork database |
| SysMetrics | | The ADO.NET connection string for the DGPSysMetrics database |
| FileStore | | The ADO.NET connection string for the DGPFileStore database |
| FileShard1 | | The ADO.NET connection string for the DGPFileShard1 database |
| FileShard2 | | The ADO.NET connection string for the DGPFileShard2 database |

The connection strings displayed in the DB Setup utility, the Web Service Key and Key Version values all need to be saved into the Web.config files of each web service in a location. In addition, it is recommended that these values be saved securely elsewhere as a backup. The SvcKey and SvcKeyVersion must be the same in all locations of an environment (those values are used by the data replicated between locations). Also, the DGP system admin username and password must be saved in some form of password manager software for safekeeping, and is the same in all locations of an environment as well. The connection strings, on the other hand, are only valid within each individual location.

Every new SvcKey encryption key must be *ADDED* (appended) to the list of key versions in the Web.config files in yet another example of the immutable append-only convention. Historical data encrypted with older key versions will use those encryption keys to be decrypted (referenced by their name/label), as long as they exist in the Web.config file. Data will be encrypted with the latest version of the encryption key whenever it is saved in the system, so in theory the older keys will eventually no longer be used and can be removed from the list at that time.