Learning about DGP Source Code

There are multiple sources for documentation and examples of DGP code, which include the API Documentation, the API Tester test harness and test files, and the free open-source code of DGPDrive itself.

API Documentation

The API section of the documentation contains documentation of every method of every API in a DGP system. The documentation for each method shows all of its input parameters, data types, required inputs, as well as each of its named return values and their data types. Each parameter and return value also has a brief description of its purpose.

The API documentation also contains a list of all the constants used in the code as the equivalent of "string enums" to standardize the input labels and some enum-type values. In DGPDrive, all of these constants are contained in the FieldValues class of the APIUtil project. The same set of constants are used in both client apps and web service API's, since all tiers of DGP are written in C#. If a different language was used for the client applications, a copy of the FieldValues constants would have to be created for that language to be used remotely.

The API documentation does not contain any sample code. Closed-source systems are forced to provide code samples, but since DGPDrive is free and open-source, all of its code across the multiple tiers provide "sample code" showing working examples of how to design and build all the tiers of a DGP system.

Test Files (examples)

The test files used by the API Tester test harness to test all the API methods of a DGP system are one source of working examples of the API Request messages used to call each API method. The <Meth></Meth> content of the test file is the working example of one part of an API Request message, but is wrapped by additional sections used for the processing of test files by the test harness. These examples are useful to see the types of realistic values used to call the various API methods.

DGPDrive Open Source Code (examples)

The best examples of all are the free open-source code of a DGPDrive system. It has working examples of every tier using RPCs to communicate with the other adjacent tiers (RPCs from the client apps to the web services, ADO.NET RPC's from the web services to SQL Server, etc.). These working examples are far more realistic and more extensive than any documentation could ever hope to be.

All of the functionality of a DGPDrive system is approximately 5000 lines of simple code, which reuses a relatively small number of patterns as templates throughout the various tiers. The code to manage the database schemas and core (security) data adds another 4000 lines of code in the _DDL and _DATA classes merged into the DML projects for each database. The idempotent code to manage the core data is the majority of that extra code, since the _DDL classes are small in comparison.

The simplicity of the code extends to the simplicity of the DGP database schemas. A DGPDrive system has 4 databases in total. SysInfo (RBAC security) has 7 tables, the Lattice file storage has 5 tables, SysMetrics logging and metrics has 4 tables, and the SysWork has 2 tables. The largest of all the database tables has 25 columns.

The point being made is that the code is very simple, and there is very little code considering the large set of features, functionality and capabilities that it delivers. The same is true for the database schemas and corresponding data access logic, so using the DGP code as a source of working examples is very easy compared to most other systems.

Integrating DGP Functionality

When building a custom software system using DGP as a foundation, the mechanism for integration must allow both DGP and the custom software to continue to evolve independently of each other without breaking the integration. While this can be done at the code level by integrating DGP projects into the custom code, the best mechanism will generally be for the custom software to include and reference DGP .dll's in order to reuse their functionality.

To avoid the consequences of breaking changes in these .dll's, the DGP code libraries must follow the same immutable append-only convention as all of the other parts of DGP systems. Therefore, each new version of the .dll's will contain all functionality from

previous versions, occasional bug fixes, and the new functionality that justified creating a new version. The date of the .dll files act as the version number to differentiate older versions from newer versions in a human-readable form when they are included in custom software. In the future, the inclusion and update of these .dll's will probably be handled as NuGet packages, but for the beta versions it is a manual process.

Note: the name of the .dll file does not change between versions to avoid breaking the references in the custom code when manually updating .dll file versions. Only the date of the .dll file changes from one version to the next. This convention may change once the update of the .dll files is eventually managed using NuGet packages.

In addition to the .dll files, custom software may also reuse the admin UI's from the Lattice application, along with the DB Setup utility and AutoWork application/service. The other alternative is to build custom versions of all of those applications to replace them.