DGP Test Files

```
<TestBatch>                                          - a collection of one or more test messages
  <TName>APIRole New Negative</TName>
  <TDescrip>Negative tests of the APIRole New method</TDescrip>
  <TGID>{{TGID}}</TGID>
  <TMsg>                                             - each test message is run sequentially by the test harness
    <TMUserName>{{TMUserName}}</TMUserName>          - the double curly braces indicate a template placeholder value
    <TMPassword>{{TMPassword}}</TMPassword>          - placeholder values are replaced with actual values in each test run
    <TMName>APIRole.New.base</TMName>
    <TMDescrip>empty input parameters</TMDescrip>
    <TMExpAuthCode>OK</TMExpAuthCode>                - the expected authentication code for the user account
    <Meth>                                           - each Meth element is the same as an actual API request message
      <MName>APIRole.New.base</MName>
      <PList>
        <Prm>
          <Name>RoleName</Name>
          <Val>
            <![CDATA[]]>
          </Val>
        </Prm>
        <Prm>
          <Name>RoleDescrip</Name>
          <Val>
            <![CDATA[]]>
          </Val>
        </Prm>
```

```
      </PList>
    </Meth>
    <RList>                              - a list of one or more results returned by the test message
      <Result>
        <RName>APIRole.New.base_DEFAULT</RName>
        <ExpRCode>OK</ExpRCode>          - the expected Result Code
        <ExpDType>TEXT</ExpDType>        - the expected Data Type
        <ExpRVal></ExpRVal>              - the expected Result Value
        <ValMatch></ValMatch>            - flag value indicating if the returned value matches the expected value
        <VarName></VarName>              - a name for the result value returned by the test message, which can be used as if it were
      </Result>                          a template placeholder in test messages run later in the test batch.
    </RList>
  </TMsg>
</TestBatch>
```

The DGP Lattice open-source code also contains test files to test every API method in the system. These test files consist of two main types: the positive test "CRUD" files, along with multiple negative test files that go along with them. Combined, they should provide 100% test coverage of the code in each API method.

Test files will usually contain more than one API method call, each of which are executed sequentially by the API Tester test harness (no batch execution of the multiple method calls like a standard API Request message). This is done so that the results of one method call can be used as an input in later method calls in the same test file. This is accomplished by assigning a value to the VarName element in the Result section of the test file method call.

Doing so adds the return value to the collection of name/value pairs stored in a generic Dictionary. The named value can then be substituted as an input into test file method calls by enclosing it in double curly braces, for example {{VarName}}. Certain variable

names such as {{TMUserName}} and {{TMPassword}} are standard in the API Tester test harness, in this case to run each test using the credentials of the user that connected to the location of a DGP system.
A good place to see working examples of this feature are the _CRUD_proc.xml test files that exist for each set of API methods corresponding to a database table, such as APIMethod_CRUD_proc.xml. These positive path test files are a sequential set of method calls that:

1. Create a new test record, returning its global ID (which is saved as a variable and used in later method calls).
2. Queries for the new record by ID value
3. Queries for the new record by Name value
4. Searches for the new record by Name (includes the Count and Search method calls for server-side pagination)
5. Updates the test record to edit a value
6. Queries for the history of the record (returns all edited and active versions)
7. Deletes the test record (soft delete, which is another update)

All of the CRUD_proc.xml test files follow this basic pattern to call each type of Create, Read, Update and Delete methods, which also "cleans up" after itself by deleting the test record if the entire process runs successfully.

The negative test files call each of the CRUD methods individually, first with empty input parameters to test the required field logic, and then with missing input parameters to test the lower level API message logic. Those two sets of negative tests cover most of the error and exception handling logic of the API methods. Additional tests can pass in the wrong data types for methods that convert the default text inputs to other types such as numbers or dates – but those are fairly uncommon.

The remaining test files cover API methods used in automated processes such as getting a batch of source records, merging a batch of records into a destination table, count checks, duplicate checks, and so on. Most of these methods can only be tested when an environment has more than one location, so a limited form of testing between TestDB1 and TestDB2 can be run on a single computer. These tests only cover the basic "pattern" of the automated process logic, and can only be fully tested using the AutoWork Tester test harness.