Connect Form Overview

One of the first differences people notice when using Lattice is the need to select which location to connect to using the Connect form and the system list file. Since Lattice is a distributed grid system in which all the distributed locations of each environment are active and writable at the same time, a user must decide which location they want to connect to for each session. Once a user connects to a specific location, they will do all of their work in that location during their session.

Any of the work they do while connected to a location that creates data will have that data be replicated in real time to the other locations via background processes run continuously by the auto processing subsystem. If the location a user is connected to experiences any problems, the user would need to use the Connect form again to connect to one of the other locations in order to resume their work (manual failover). In most cases, the work they completed at their original location will have been replicated to the other locations prior to their failover. If that is not the case, the user would need to redo some of their latest work that was not replicated prior to the failure, and then pick up where they left off. Under most circumstances, data is usually replicated between locations within a few seconds, so the window of time to lose work during a manual failover is relatively small.

System List File

The system list file is similar to a favorites list that keeps track of the URL's used by a user to connect to different systems, environments and locations. There are many potential variables for each system regarding how many environments it has (dev, test, QA and prod), how many locations each environment has, as well as the number of URL's per location based on its network topology (internal URL, external URL, reverse proxy pages hosted in a DMZ, and so on). In practice, system list files can be created by technically knowledgeable people and then shared with other users. The content of the system list files should not need to change very often for most systems.

An example of a system list file is shown below:

```xml
<?xml version="1.0"?>
<SysList>
  <Sys>
    <SysName>DGP Lattice</SysName>
    <SysDescrip>Distributed grid platform admin application</SysDescrip>
    <LocList>
      <Loc>
        <LocName>localhost</LocName>
        <LocDescrip>call to web service or proxy using the loopback network address</LocDescrip>
        <EPList>
          <EP>
            <EPName>web service</EPName>
            <EPDescrip>direct access to the web service</EPDescrip>
            <EPURL>http://localhost/DGPWebSvc/DGPCntrl.aspx</EPURL>
          </EP>
          <EP>
            <EPName>web service proxy</EPName>
            <EPDescrip>indirect access to the web service through a proxy</EPDescrip>
            <EPURL>http://localhost/DGPWebApp/DGPProxy.aspx</EPURL>
          </EP>
        </EPList>
      </Loc>
    </LocList>
  </Sys>
</SysList>
```
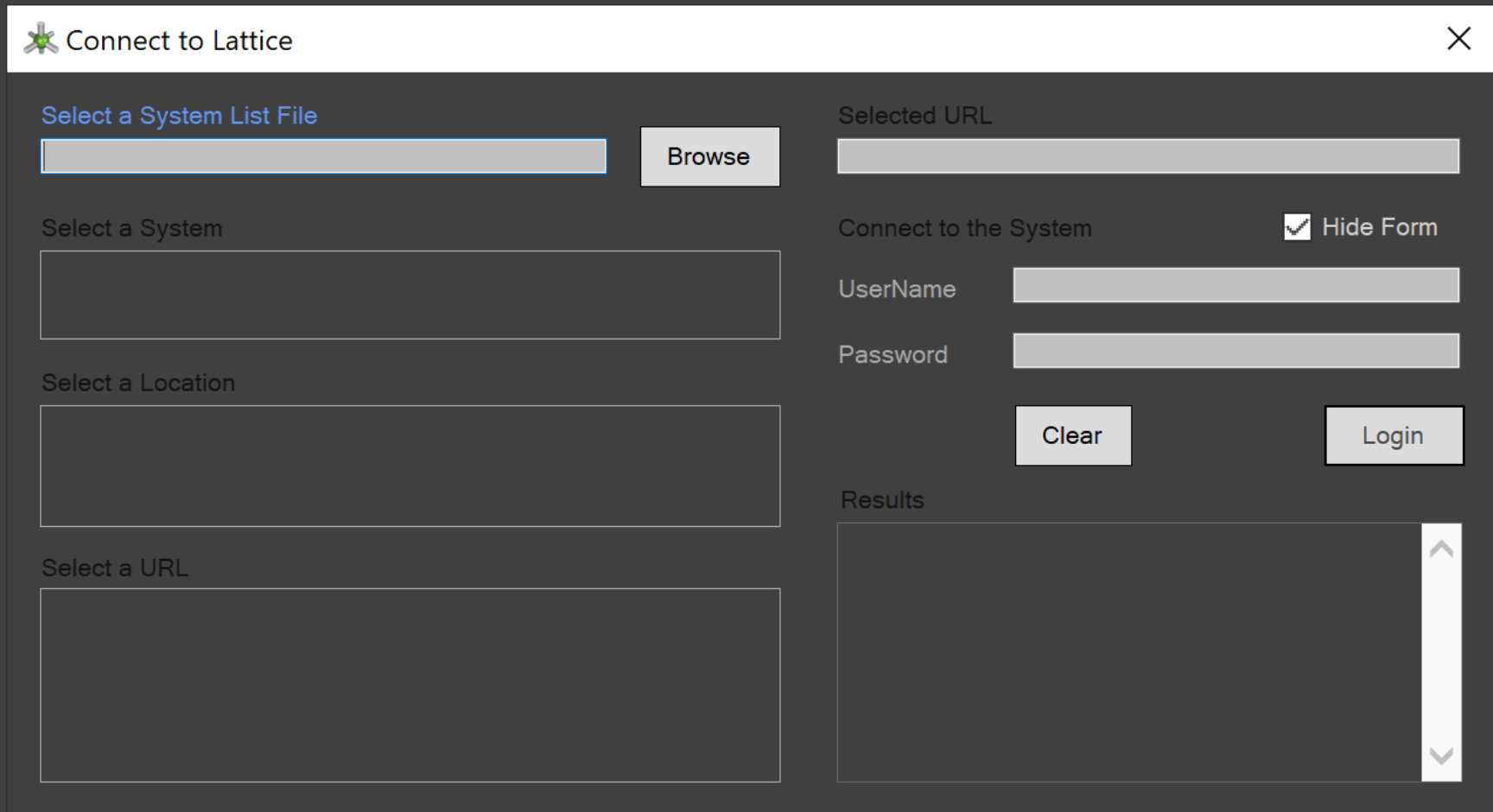
1. When a user first runs Lattice, only two navigation menu items are visible: Connect and Help.
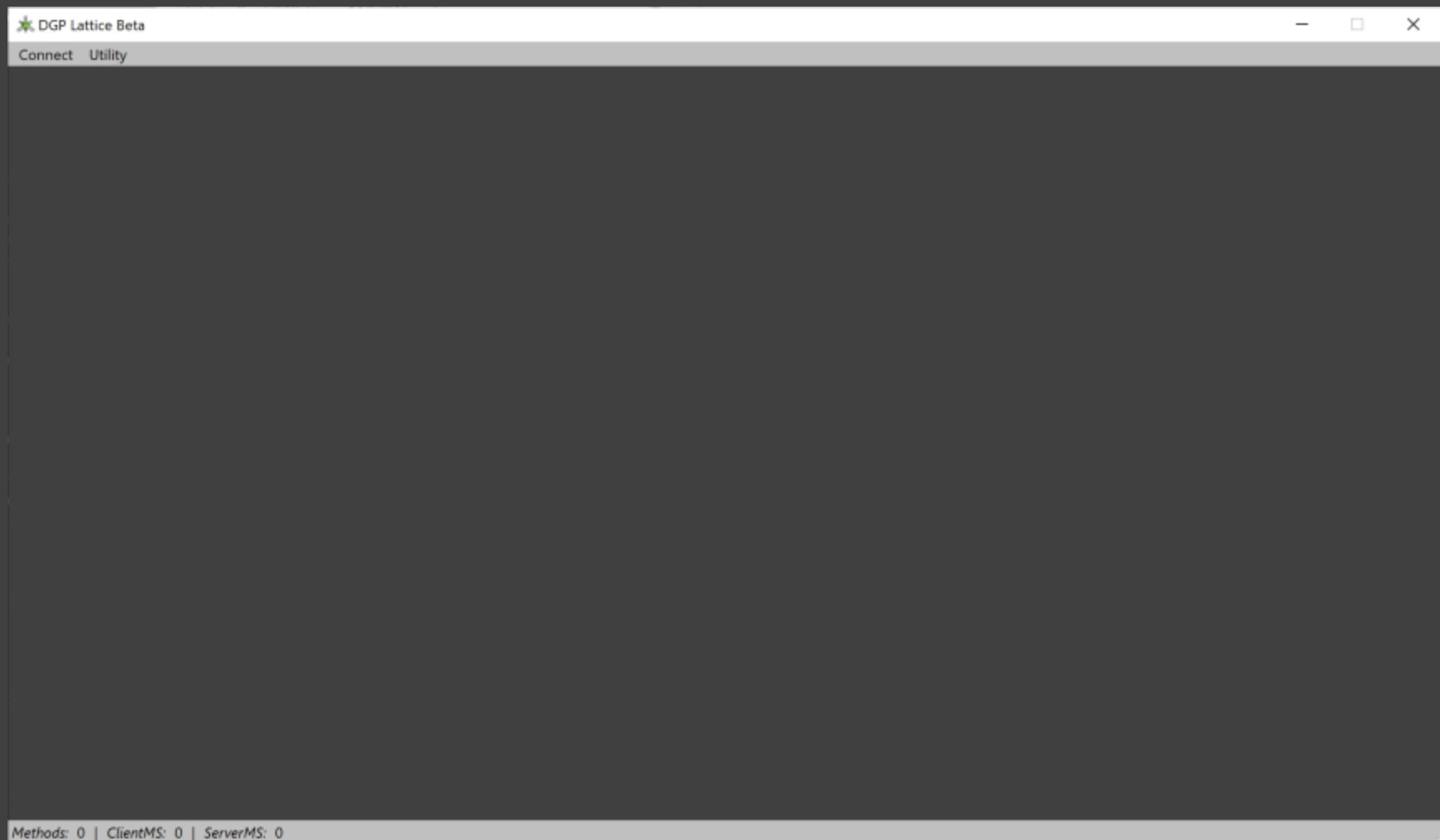
2. When a user opens the Connect form, they see a series of steps that they follow in sequential order:

Connect to Lattice ✕

Select a System List File

[   ] [Browse]

Selected URL

[   ]

Select a System

[   ]

Connect to the System   ☑ Hide Form

UserName [   ]

Password [   ]

Select a Location

[   ]

[Clear]   [Login]

Select a URL

Results

[   ]   [   ]

2.1. Select a System List File – browse for a system list file for the Connect form to read. Selecting a system list file will populate the list of systems used in step 2.2. A user can have a valid account in more than one system, and more than one environment of each system. The purpose of the system list file is to help the users keep track of all of these different URL endpoints for those systems, environments and locations.

2.2. Select System – once the user has opened a system list file, the next step is to select the system they want to use. A system represents the "owner" of all the hardware used to build each separate instance of Lattice and all of the data it contains. Selecting a system populates the list of locations for step 2.3.

2.3. Select Location – after selecting a system, the next step is to select which location to connect to. Selecting a location populates the list of URL endpoints used in step 2.4. Ideally each system would have 3 distributed QA/Prod locations, but dev and test environments will generally only have two. The different environments and their locations are combined into a single list to help simplify the system list itself, since most users will only be connecting to production locations and don't need URL's for other environments.

2.4. Select Endpoint – this is a list of the actual URL's used to connect to a location. Selecting an endpoint displays the selected URL in the Selected URL textbox. Certain URL's will only work when connected to internal networks behind a firewall, while other URL's will only work outside the firewall. They should be named intuitively so that non-technical users can make the correct choice based on where they are when connecting.

2.5. Enter Account Info – the user must enter their account UserName and Password for the system and environment they are connecting to. Clicking the Login button will connect to the selected endpoint URL and call the Login API method. Each environment of each system has its own separate identity store database, so each environment will have its own list of user accounts, roles, and so on. It is recommended that users store info about all of their various accounts in one of the many available password management applications.

2.6. Connection Results – the results of the attempt to connect will be displayed in the Connect form.

    2.6.1.   Each API Request message uses an HMAC hash process to authenticate the user to the system, and then also to authenticate the system to the user in the response message (which helps to defeat "man-in-the-middle" attacks). Currently, only the Login method authenticates the server response to the user. Since the client stays connected to the same location for their entire session, authenticating every response message is unnecessary.

    2.6.2.   Connections can fail due to entering an incorrect UserName and Password, but can also fail due to:

        2.6.2.1.   Expired request message TTLExceeding the account rate limit for the number of method calls per minute

        2.6.2.2.   Exceeding the failed authentication count, which disables the account

    2.6.3.   A successful connection response authenticates the server to the client, caches multiple values from the server in the client application, and then customizes the navigation menu based on the user's role membership.

✳ Connect to Lattice   ✕

Select a System List File

C:\Users\alanrahn\source\repos\DistribwareBeta\C   Browse

Select a System

Name	Descrip
DGP Lattice Beta	Dev, Test and Prod DGP en...

Select a Location

Name	Descrip
localhost	connect to a web service o...
Win10Dev	Windows 10 Hyper-V VM

Select a URL

Name	Descrip	URL
local web service	direct access to th...	http://localhost/D...
local web service ...	indirect access to ...	http://localhost/D...

Selected URL

http://localhost/DGPWebSvc/DGPCntrl.aspx

Connect to the System   ☐ Hide Form

UserName   sysadmin

Password   ********

Clear   Login

Results

Connection:

  All LoginResult values read correctly.

  Server Authentication: True

Login Method

The Login API method is a process that performs a number of different actions. First, it clears any cached UserInfo object that may exist on the server. It then queries for the user's authorization data, and will store it in the user's account record if it is different than the cached data that already exists. This is a "lazy" just-in-time mechanism to update the data cached in each user's account record as it is needed, rather than by running massive batch processes to update all user account records every time RBAC data is updated. Finally, it calls an internal method to create a list of the user account role membership, which is only needed by the login method and is not cached into each user's account record.

<LoginResult>
  <WebSvcVer> the build date of the latest deployed web service, used as a version "number" </WebSvcVer>
  <SvcEncKeyVer> the version label of the current symmetric encryption key used by the web service </SvcEncKeyVer>
  <SvcEncKey> the current 32-byte AES 256 encryption key, used locally for end-to-end file encryption </SvcEncKey>
  <MaxSegSize> the maximum file segment size in bytes accepted by the web service </MaxSegSize>
  <MinSegSize> the minimum file segment size in bytes accepted by the web service</MinSegSize>
  <MinPwordLen> the minimum length of updated passwords accepted by the system </MinPwordLen>
  <RespTime> the response timestamp value created by the web service </RespTime>
  <RespToken> the HMAC hash of the response timestamp value used to authenticate the server to the client </RespToken>
  <UserGID> the user account global ID value </UserGID>
  <ReadGroups> a delimited list of readable DataGroup global ID values </ReadGroups>
  <WriteGroups> a delimited list of writeable DataGroup global ID values </WriteGroups>
  <UserRoles> a delimited list of user account role membership global ID values </UserRoles>
</LoginResult>

*The WebSvcVer (web service version) is used as a version number for the web service, and is actually the date the web service was built and published. This version number is used as a feature toggle by the application to disable any functionality that requires a newer version of the web service API.

Navigation Menu Customization

Lattice navigation menu items are shown to each user based on the user's role membership. Customization of the navigation menu is done to improve the user experience, hiding functionality that an account is not authorized to use.
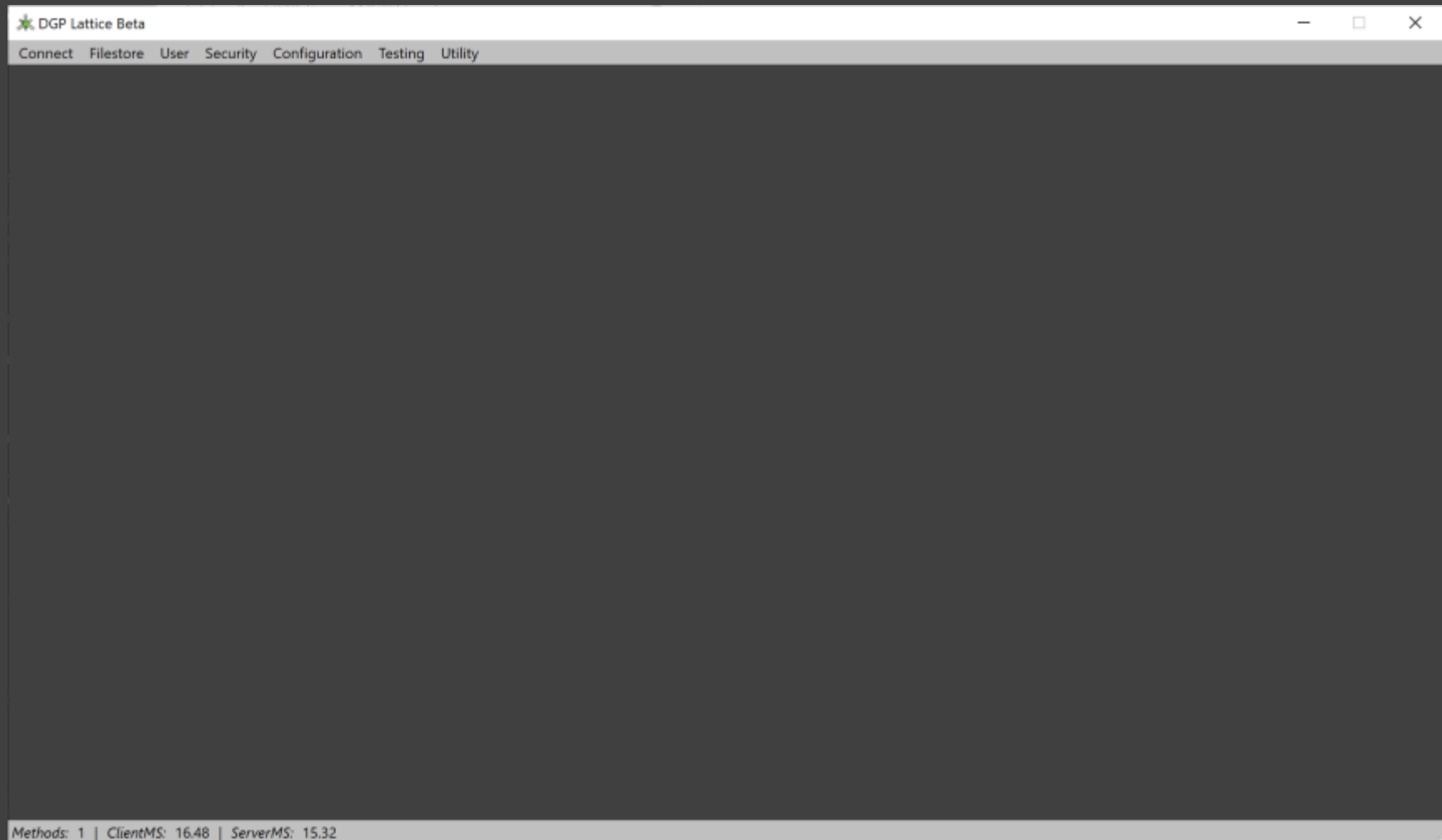
Customization of a user's navigation menu has nothing to do with the security of a system. Client application logic is not trusted by the centralized web services, and all security is enforced within the web services themselves.

Users can still connect to a system with an expired password, which merely restricts API access (and the application navigation menu) so that the user is only allowed to call the ChangePassword API method until their password has been updated. This allows users to update their own expired passwords without the help of an administrator, and without having to worry about making sure to update their password before it expires so they don't get locked out of the system.

The example above shows the navigation menu customized for the sysadmin account (which has full access to all functionality) just after calling the Login API method in the connection form. The FileStore application is an optional distributed file sharing system similar to the original DropBox, but self-hosted. Administrators have to use the security forms to add access to the FileStore role for users assuming the FileStore application has been configured for use in a system environment.