Extended OVSDB Plugin Configuration

If desired, you can fine tune the configuration of the OVSDB plugin using the information in this appendix.

This appendix includes the following sections:

Performing Basic OVSDB Plugin Operations

The OVSDB plugin supports two modes of operation:

- Single switch mode—The OVSDB plugin connects to only one switch and one controller or controller cluster.
- Dual switch mode—The OVSDB plugin connects to a pair of switches and one controller or controller cluster.

To perform basic OVSDB plugin operations, use these commands:

Command	Purpose
guestshell run sudo ovsdb-plugin service start	Starts the OVSDB plugin. **Note** Check for startup errors using **cat nohup.out** or tail the logs using **tail -f log/ovsdb-plugin.log**.
guestshell run sudo ovsdb-plugin service stop	Stops the OVSDB plugin.
guestshell run sudo ovsdb-plugin service restart	Restarts the OVSDB plugin.
guestshell run sudo ovsdb-plugin service version	Displays the version of the OVSDB plugin.
guestshell run sudo ovsdb-plugin config set -h	Displays information on how to configure the OVSDB plugin.

Command	Purpose
service enable	Auto-starts the OVSDB plugin on switch bootup.
service disable	Disables auto-start of the OVSDB plugin on switch bootup.

This example shows the OVSDB configuration help:

```
switch# guestshell run sudo ovsdb-plugin config set -h
usage: ovsdb-plugin config set [-h]
                                         [--switch-description SWITCH_DESCRIPTION]
                                         [--keep-test-config]
                                         [--log-level {error,warn,info,debug,trace}]
                                         [--log-server ADDRESS]
                                         [--log-type {file,udp}]
                                         [--max-json-peers INT]
                                         [--run-in-switch]
                                         [--schema-version {1.3.99,1.3.0}]
                                         [--vrf NAME] [--xms INT] [--xmx INT]
                                         [-v]
                                         ct_addr1[,ct_addr2]
                                         sw_addr1[,sw_addr2] user1[,user2]
                                         pswd1[,pswd2] name

positional arguments:
  ct_addr1[,ct_addr2]   Controller cluster address in IP:PORT format. Port
                        defaults to 6632 if not included. To specify two or
                        more controllers, separate them with a comma (no
                        spaces). E.g. 10.21.1.10:6632,10.21.1.11:6632
  sw_addr1[,sw_addr2]   Switch address in IP:PORT format. Port defaults to 443
                        if not includes. In VPC mode, specify two switches by
                        separating them with a comma (no spaces). E.g.
                        10.21.1.10:443,10.21.1.11:443
  user1[,user2]         Switch username(s). To specify a different username
                        for a second switch separate with a comma (no spaces).
                        E.g. user1,user2. If no username is given for the
                        second switch, the first one will be used
  pswd1[,pswd2]         Switch password(s). To specify a different password
                        for a second switch, separate with a comma (no
                        spaces). E.g. pswd1,pswd2. If no password is given for
                        the second switch, the first one will be used
  name                  Switch name.

optional arguments:
  -h, --help            show this help message and exit
  --switch-description SWITCH_DESCRIPTION
                        Switch description.
  --keep-test-config    Keep the test config
  --log-level {error,warn,info,debug,trace}
                        Log level to use. Defaults to info
  --log-server ADDRESS  When --log-type is set to UDP, use this to specify the
                        remote where the logs will be sent. The address must
                        be in IP:PORT format. PORT defaults to 514 if not
                        included
  --log-type {file,udp}
                        The type of logging to use. When set to file, the path
                        is always PLUGIN_ROOT/log/ovsdb-plugin.log. Defaults
                        to file
  --max-json-peers INT  Maximum number of JSON peers. Defaults to 6 if not
                        included. Set to -1 to have the plugin auto-compute
                        the value based on the number of controllers
```

```
--run-in-switch       Configures the plugin for running in the switch
--schema-version {1.3.99,1.3.0}
                      Set the schema version to use. Defaults to 1.3.0
--vrf NAME            Used only when --run-in-switch is set. This configures
                      the plugin to use the given VRF name when
                      communicating with the controller. Defaults to
                      management
--xms INT             Set initial Java heap size in MB. Defaults to 2048
--xmx INT             Set maximum Java heap size in MB. Defaults to 2048
-v, --verbose         Show extended information
```

Generating the SSL Keys

Step 1 Choose one of these methods to generate the SSL keys:

- In non-vPC mode without Guest Shell or remote vPC mode, enter the **sudo ovsdb-plugin cert bootstrap** command and skip the remaining steps.
- In non-vPC mode with Guest Shell, enter the **guestshell run sudo ovsdb-plugin cert bootstrap** command and skip the remaining steps.
- In local vPC mode, enter the **guestshell run sudo ovsdb-plugin cert bootstrap --receive** command on the first instance, and go to the next step.

Step 2 Copy the temporary certificate displayed on the terminal.

Step 3 Enter the **guestshell run sudo ovsdb-plugin cert bootstrap --send IP_OF_FIRST** command on the second instance.

Step 4 Paste the temporary certificate when prompted.

Note You can also pipe in the temporary certificate to the second instance (for example, **cat tempcert.txt | guestshell run sudo ovsdb-plugin cert bootstrap --send**).

What to do next

If you ever need to reset the SSL keys, follow the previous steps but enter the **guestshell run sudo ovsdb-plugin cert reset** command in place of the **guestshell run sudo ovsdb-plugin cert bootstrap** command.

Configuring the Startup config.json File

Normally, the startup config.json file is created and edited by the CLI code (using information provided by the CLI user). The following sections document many configuration file settings that are not directly supported by the CLI.

Caution Pay serious attention to the syntax when directly editing the configuration file. For example, an extra space can cause the application to be unable to run.

On startup, the application validates the configuration file. If the configuration file is invalid, the application sends an alert to the console, and the output is captured in the nohup.out file. If the application fails to start after directly editing the file, review the nohup.out file. It should provide suggestions as to what part of the configuration file has changed.

Always make a backup of the config.json file before directly editing the file. For nearly every value, a restart of the application is required for the application to fully act upon the changed values. The exceptions are the default log level and the switch username and password. To have the running application see a change to the default log level or the switch usernames and passwords, you must run the **ovsdb-plugin service reloadconfig** command.

The startup config.json file supports the following sections:

- loggingInfo—Specifies whether and how to log.
- tlsInfo—Specifies SSL keys and certificates.
- systemInfo—Specifies db schema and running-in-switch flags.
- supportedSwitchType—Specifies the supported switch types.
- switchInfo—Contains information on the single switch to be used.
- managerInfo—Contains information on the inbound and outbound connections that the OVSDB plugin will support.
- otherInfo—Specifies optional flags and settings.

loggingInfo

Several different styles and configurations can be configured. This application uses the Log4j 1.2logging library. Many of the settings describe values that the application will use to auto-build a Log4j configuration file, which is used to create the applications logging. Logging can be turned off, written to disk, sent remotely (via SysLog), or configured in a separate Log4j-styled configuration file.

To turn off all logging:

```
disableAllLogging: true
        - Disables all logging.
        - No other settings are needed, validated, or used.
```

To write log files to disk:

```
loggingStyle: 1
        baseDir: "/path/to/base/dir/no/ending/slash"    <<-- note: on Windows, use / not \
        maxFileSizeInMb: 5
        maxNumFiles: 10
```

Details on each setting:

```
disableAllLogging: values are true or false.  Default = false.
        - Setting this disables all logging.
        - This setting will override ALL the other logging settings.

    loggingStyle: this is an integer value between 1 and 31. Default = 1
        - the value can represent multiple settings
        - the value represents the AND-ing of one ore more bit values
        - however, some settings are EXCLUSIVE (i.e., cannot be combined)
        - The standard settings are:
            1 - simple logging to local file
        * location is specified in baseDir [see below]
```

```
            4 - Syslog Logging
        * Host location is specified in sysLogHost [see below]
        * Facility is specified in sysLogFacility [see below]
           16 - Log4J Logging
        * User-specified log4J file is specified in loggingConfigFile [see below]
        - Note that Log4j Logging is "exclusive" in that it cannot be combined with another

          logging style.  However the other settings can be "mixed and matched".
          - For example, a setting of 5 would mean Syslog logging AND Simple Local Logging.

            - Note also that 17 would NOT mean simple local logging AND Log4j logging
              because Log4j logging is exclusive.

    The ovsdb plugin uses an asynchronous logging facility:
    - Logging will asynchronously be written to the log location
    - Note that async also means that if the logging device is full then the application
          will continue even though it cannot write to the log.
        - Note that the asyncBufferSize setting controls how many messages can be buffered
        - Scenario:
        - Application is running.
        - Logging device becomes full
        - A count of AsyncBufferSize more logging statements are made
        - The buffer is now full; this FIFO buffer will now beging dropping logging
statements.
        - The running application continues on; it is not affected by the device full issue.

        - The current contents of the async buffer will be flushed to the logging device
        - Logging will continue.
        - The running application continues on; however, now it is successfully logging.

    asyncBufferSize: number of messages that can be buffered IF async = True
    - Once this number of messages are buffered, the system will begin to drop logging
statements
    - Successful logging will resume once the logging device is no longer full.

    baseDir: "/path/to/base/dir/no/ending/slash"
        - REQUIRED value if loggingStyle AND 1 = 1 (i.e., simple local logging is set)
        - On windows, use / and not \
        - Should NOT end with a trailing slash
        - Note that although this is called "local" logging, it just have to be a path that
 is accessible
          to the application; so the actual device can be "remote" yet locally accesible.

    defaultLogLevel: logLevel
        - REQUIRED value
        - logLevel must be one of FATAL, ERROR, WARN, INFO, DEBUG, TRACE
        - Capitalization is not significant
        - This sets the DEFAULT log level for all application logging.
       - Logging on a case-by-case basis can be overridden by usage of the overrideLogLevels
 section [see below].

    loggingConfigFile: path to the "override" log4j config file
        - Most of the rest of the settings in the LoggingInfo section describe
          settings that will be used to "auto generate" a Log4j-style config file.
        - However, the user can decide to hand-carve their own config file.
        - To do this:
        - Set logging style = 16 (aka Log4j Logging)
        - provide a path to your own config file via the loggingConfigFile setting
        - The application will only validate that a file exists at this location.
          It will NOT do ANY validation of the contents of the file.  It will simply
          tell the Log4j logging system where your config file is.

    overrideLogLevels: this section allows the DEFAULT log level to be overridden
        - This section can consist of 1 or more overrides.
        - You can override the default by going higher or by going lower.
        - You can override the log level by referencing a java class or family of classes.
        - For example, there is a class called:
com.nexus.andromeda.ovsdb.jsonrpc.util.JJsonPeer
        - You could set an override directly on that class, or you could set an override
          in the com.nexus.andromeda.ovsdb.jsonrpc.util package or at the
          com.nexus.andromeda.ovsdb.jsonrpc.util level.
        - Therefore, the label "CLASSNAME" is kind of mis-named.
        - As with everything in this file, exact spacing is CRITICAL. (Never use tabs).
        - For example, here is a default override:
           {
                "className": "org.apache.commons.beanutils",
                "level": "FATAL"
            }
        - This shows that the class or package "tree" (you can't really tell here) named
          org.apache.commons.beanutils should be set to FATAL level.
        - In this case, the reason is that package is incredibly "noisy" and we only ever
          care about this package if it logs something at the FATAL level.

    maxFileSizeInMb: integer value for max file size before roll over
        - REQUIRED value if loggingStyle is localDebug or localBuffered

    maxNumFiles: integer max number of files to roll over
        - REQUIRED value if loggingStyle is localDebug or localBuffered
        - NOTE: Max size of all log files will be
          (maxFileSizeInMb x maxNumFiles) in MB.

    messageFiltering: this section provides the ability to filter out specific messages
        - There are certain logging statements that occur fairly frequently.
        - These messages are specially programmed to read this messageFiltering section.
        - The theClass field describes the class generating the logging statement.
        - The regex describes the line to be "filtered out".
        - The count indicates how often to let the statement show up in the logs.
        - For example, a count of 10 means only write this statement to the log every 10th
          time it occurs.

    sysLogHost - the Host IP of the computer that is support syslog logging
        - Only applicable if loggingStyle AND 4 = 4 (i.e., syslog logging is set)

    sysLogFacility - the facility that is supporting syslog logging
        - Only applicable if loggingStyle AND 4 = 4 (i.e., syslog logging is set)
```

tlsInfo

```
tlsInfo:
    privateKeyFile:      "C:/ovsdb-plugin/config/jay-server.p12"
    certificateFile:     "C:/ovsdb-plugin/config/ContrllerCaCert.der"

    privateKeyPassword:  "ovsdb-plugin"
    certificatePassword: "mypassword"

    # optional flag
    turnOnSSLDebugging:   true
```

systemInfo

```
systemInfo:
    schemaVersion: value
    runningInSwitch: boolean-flag
```

```
# Current legal schema versions are "1.3.0" and 1.3.99"

# Running in switch command indicates whether or not we are
configured to run in the switch.  This affects how certain
decisions are made in the system.
```

supportedSwitchType

Defines an array of all the legal types of switches to be used.

```
supportedSwitchType:
    - switchType: "N9K-C9396PX"
    - switchType: "N9K-C93128TX"
    - switchType: "N9K-C9372TX"
    - switchType: "N9K-C9332PQ"
    - switchType: "N9K-C9372PX"
   etc.
```

Note You can use a wildcard in the switchType definition, which would replace all of the previous definitions.

```
supportedSwitchType:
    - switchType: "N9K-C93*"
```

This definition matches all of the switches that start with "N9K-C93". Alternatively, the SwitchType could be set to just *, which would match all switches. However, the wildcard character can occur only as the last character in the string, so "N9K-C93*X" would not be supported.

switchInfo

Defines an array of the single switch to run against.

```
- host: 10.21.111.17
          user: admin
          password: mySwitchPassword
          certificateFile: path-to-CA-certificate-file
```

managerInfo

Defines an array of the connections that the OVSDB plugin will support.

```
managerInfo:
    - name: "controller"
      target: ptcp
      port: 6640
      max_backoff: 1000
      inactivity_probe: 0
```

Details on each setting:

```
name: theName
        - Required value defining the name of this connection.
        - Both configured and learned controllers must have the word "Controller" in their
          name.

    target: theTarget
        - Required value describing the connection.
        - One of PTCP, TCP, PSSL or SSL.
           - TCP and PTCP are TCP connections.
           - SSL and PSSL are SSL connections.
           - The leading p indicates that the OVSDB plugin is to act as the server.
           - The lack of p indicates that the OVSDB plugin is to act as the client.

    port: integer portNumber
        - Required port number value.

    max_backoff: integer backValue in Milliseconds
        - Required value.
        - Indicates how long the OVSDB plugin should wait after a broken connection before
          it establishes a new connection.

    inactivity_probe: integer inactivity_probe in Milliseconds
        - Required value.
        - Indicates how long the OVSDB plugin should wait when not receiving any data on a
          connection before it sends out an echo request.
        - Once an echo request is sent, if this same amount of time expires without a
          response, the OVSDB plugin will close the socket connection.
       - For example, if inactivity_probe is set to 60000, the OVSDB plugin, after waiting

          for 120 seconds with no data received, will close the socket.
      - A value of 0 indicates no inactivity probes should be sent on the given interface.
```

otherInfo

This class allows last-second configuration without having to generate or update class files to support the new fields. It is both YAML and Json compatible, and it allows you to treat non-YAML-standard embedded strings as key-value pairs.

Note The flags must include double quotes (for example, flag: "*VariableName*: 123" and flag: "*AnotherVariableName*: x45").

Currently defined values (flags in the current config.json file):

```
- flag: "StartUp_PopulateDb: True"
       * This is used by the ovsdb-plugin startup code.  If not present, the value defaults
 to False.
          This value tells the plugin to attempt to build its own OVSDB based on existing
switch configuration.
          Sometimes it might be useful to toggle to False to allow the plugin to take a
fresh
          configuration from the controller.
        * The default value is True.
```

Other available configuration flags that are supported but are not currently being set (that is, the default values are being used):

```
- flag: "DB_minElapsedTimeBeforeLogging: NNN"
        * At one point in time, we were concerned about database performance.
          This flag allowed us to track every database call that took MORE than
          a certain time.  For example, setting this value to 400 would cause any
          db call that took MORE than 400 ms to be logged (so it could be tracked).
        * The default value is 150 aka 150 msecs.

   - flag: "JP_HowOftenToLogSocketConnectFailuresInMs: NNNNN"
        * For each connection (or ManagerInfo) in the config.json file, certain code
          will attempt to make a socket connection to each of those records.
         For some connections (like the utility connection used to support the ovsdb cli),

          it is VERY VERY likely NOT to make a connection.  However, each connection
          failure causes logging.  This setting HOW OFTEN (in ms) we should SHOW an error.
        Use of this feature will reduce the amount of uninformative logging data generated.

        * Note that the current defalut value is grossly = Show Errors every 20th time.

   - flag: "JP_OffsetFromInactivyProbeTimeInMs: NNNNN"
        * This is a millisecond value used by the JSON Peers.
        * The ManagerInfo record in the config file (as well as the data conveyed to us by
 a controller)
          contains an InactivityProbe value.
       * This value describes how long (in milliseconds) it will take of inactivity before
 the
          specified connection will send us a probe (aka echo) message "to keep the socket
 warm"
          (i.e., to maintain the connection".
        * The JSON Peer uses this same value.  If it sees this InactivityProbe amount of
time without
          any activity, it will decide to send an echo.
        * So, back to this flag.  This flag allows us to control who will send the echo.
          (because, basically, both sides are "agreeing" to send it at, say, 60 seconds;
so it is
        INCREDIBLY arbitrary which "end" will end up sending the actual echo/probe message).

        * This configured value ends up being SUBTRACTED from the configured (or provided)
 inactivity
          probe value and it is this calculated value that ends up being used in the JSON
Peers
          calculations.
        * (Assume here that the configured Inactivity Probe Time = 60 seconds).
        * So, if you set this value to 2000, you are telling the JSON Peer to use, say,
60-2 or 58
          seconds as its calculated value.  That means, most likely, that the JSON Peer
will end up
          sending the echo to the controller.
        * Alternatively, you could set this value to -2000.  This means that the JSON Peer
 will use
         ( 60 - (-2) ) or 62 seconds as its probe time.  This would mean that, most likely,
 the
          controller will end up sending echoes to the JSON Peer.
        * This value defaults -2000 aka -2 seconds aka the controller will probably send
us echoes.

   - flag: "JP_ManageJsonPeersLoopTimeInMs: NNNNN"
        * This millisecond value determines what the delay (in milliseconds) between
subsequent loops
          for the ManagerJJsonPeers code should be.
        * This value defaults to 1500 (aka 1.5 seconds).

   - flag: "JP_ReportManageJJsonPeerStatusTimeInSeconds: 20000"
        * This SECONDS value determines approximately how often the "Manager Json Peer
logging report"
          should be generated.
        * This should be a value more than 45 seconds, since the "report" is not really
that useful.
```

```
        * Also, this thread is constantly running; setting this to N seconds means that you
 will get a
          1 line log of ManagerJJsonPeer status information FOR THE ENTIRE ANDROMEDA RUN.
        * This value defaults to 120 which means you would get a report every 2 minutes.
        * Setting this value to 0 means generate no reports.

   - flag: "JP_ReportThreadCountTimeInMs: NNNNN"
        * The system has an ability to occasionally generate a log statement which
          indicates how many Threads are currently in use and what the peak usage has been.

          This flag defaults = -1 = do not display.
        * Note that this is not a "fine tuned" time value; I would suggest not to expect
          any more accuracy (or granularity) than about a second (aka 1000).
        * I have typically set this to 10000 or 15000 (aka 10 or 15 seconds).
        * The log statment would look like:
              (timestamp) [ManageJJsonPeers] INFO  jsonrpc.util.ManageJJsonPeers | Java
Thread Count:
          Current = 30, Peak = 33

    - flag: "JP_ReportThreadInfoTimeInMs: NNNNN"
        * The system has an ability to occasionally generate a log statement which
          provides *detailed* information on *every* thread that is currently running.
       * This option takes a bit of time to generate and can easily generate over 100 lines
 of logging output.
          Therefore, it is felt that this should not be set to any value less than about
20000.
        * This flag defaults = -1 = do not display.
        * Note that this is not a "fine tuned" time value; I would suggest not to expect
          any more accuracy (or granularity) than about a second (aka 1000).
        * I have typically set this to 20000 or 30000 (aka 20 or 30 seconds).

    - flag: "JP_SocketTimeoutValue: 60000"
        * This is used by the JJsonPeer to set the SocketTimeout value used by the JSON
Peers.
        * This value defaults to 0.
        * 0 means NO TIMEOUT (i.e., block forever)

    - flag: "JPSR_SingleSocketReadSize: 10000"
        * This value determines the size for each read attempt on the JSON Peer socket".
        * A too large value will slow the average read times down; a too small value will
cause too many
          reads to be performed.
        * There is no "absolulutely correct" value.
        * However, values < about 2000 or > about 20000 probably are not very efficient.
        * If not set, this value defaults to 5000.

    - flag: "JpSupport_ThreadPoolSize: 10"
        * This value determines how many threads are used to support JSON Peer support
threads.
        * If not set, this value defaults the "maximumNumJJsonPeerThreads" value + 2.

    - flag: "JpSupportJP_WaitPoolTermination: 5"
        * This weirdly-named value determines how long the Jp Support shutdown code waits
for the thread
          pool to terminate.
        * If not set, this value defaults to 3.

    - flag: "JP_TimeBetweenLongRunningCmdsLoggingInMs: NNNNN"
        * When the JJSonPeer receives a command, it passes it to the "SA" to determine a
response.
          Another thread (specifically, ManageJJsonPeers) watches over the JJsonPeers.
          If this value is set to NNN, any time we are waiting for LONGER than NNN
milliseconds
          for the SAL to generate a response, a log statement would be generated.
        * This defaults to 10000 aka 10 seconds.
        * This log statement will look something like:
         (timestamp) [(who)] WARN  jsonrpc.util.JJsonPeerStatus | JP=(connection descriptor):
 has been
          building a Response to a rcvd command for (amountOfTime)  ms. !!!!!!

    - flag: "LoadDbFromSwitchMaxWaitTimeInMs: NNNN"
        * This value is only to be used in a Worst Case Scenario.
        * The value indicates the Maximum Time that the "Load DB" code should take under
ANY circumstances.
        * If the code EVER takes this long,
            - the code waiting for this process to complete will be returned to
            - a failure will be returned.
        * This integer number of milliseconds value defaults to 5*60*1000 (aka 5 minutes).

    - flag: "LoadDbFromSwitchSingleSleepTimeInMs: NNNN"
        * This value indicates how long each sleep time should be while waiting for the
"Load DB" code to complete.
        * If the value is configured too small the code waiting will spend too much CPU
waiting and looping.
        * If the value is configured too large then it is likely that (on average) half of
 the configured time
          will be "wasted" because the code is done but the waiting code is unneccessarily
 sleeping too long.
        * This integer number of milliseconds value defaults to 500.

    - flag: "MAIN_DelayBeforeRunMatrixInSecs: NN"
        * This value is used by the main routine (AndromedaMain) to determine how many
seconds to delay
          (after everything else is done) before calling The Matrix.
        * This integer number of seconds value defaults to 0.

    - flag: "maximumNumJJsonPeerThreads: NNN”
        * This is used by the JJsonPeerThreadBoss to determine the maximum
          number of JJsonPeer threads that should exist at any one time.
        * Note, without this flag present, (or if it has a value of -1) the value is auto
calculated.

    - flag: "NOTIF_ThreadPoolSize: 10"
        * This value determines how many threads are used to support the internal event
Notifiation system.
        * If not set, this value defaults to 6.

    - flag: "NOTIF_WaitPoolTermination: 5"
       * This value determines how long the notification shutdown code waits for the thread
 pool to terminate.
        * If not set, this value defaults to 3.

    - flag: "NX_AaaRefreshCycleInSecs: NNNNN"
       * This value determines how often the "AAA Refresh" (to the switch) should be done.

        * This value defaults to 540 seconds.

    - flag: "NX_AaaRefreshStartDelayInMsecs: NNNNN"
        * This value determines a start delay the "AAA Refresh" (to the switch) should be
done.
        * This value defaults to 5500 seconds.

    - flag: "NX_CheckConnectivityTimeout: NNNNN"
       * This millisecond value determines how long to "sleep" between subsequent attempts
 to connect
          to the switch.
        * This value defaults to 10000 (aka 10 seconds).
    - flag: "NX_ConnectionRetryCount: N"
        * This value determines how many times several low-level NXAPI calls are made BEFORE
 an error is logged.
        * The affected routines (in the NXAPITLServiceImpl class) are:
           + attemptToReconnectAsNeeded
           + ensureConnectedToWebsocket
           + checkSocketConnectivity
        * This value defaults to 2.

    - flag: "NX_ConnectTimeout: NNNNN"
        * This millisecond value determines what the connect timeout value for the socket
to the switch should be.
        * This value defaults to 20000 (aka 20 seconds).

    - flag: "NX_ReadTimeout: NNNNN"
        * This millisecond value determines what the read timeout value for the socket to
the switch should be.
        * This value defaults to 20000 (aka 20 seconds).

    - flag: "Port_Vlan_Batching_Config: NNN"
        * This value determines how many port-vlan pairs are gathered together in a single
 batched NXAPI call.
       * The code sends the SMALLER of (the num avail and this configured size) in a single
 NXAPI call.
        * This value defaults to 100.

    - flag: "Vnid_Vlan_Batching_Config: NNN"
        * This value determines how many vlan-vnid pairs are gathered together in a single
 batched NXAPI call.
       * The code sends the SMALLER of (the num avail and this configured size) in a single
 NXAPI call.
        * This value defaults to 100.

    - flag: "SHUT_OkToStopLoggingAtShutdown: False"
       * This TRUE/FALSE value determines whether or to suppress all logging once a shutdown
 occurs.
        * If not set, this value defaults to TRUE.

    - flag: "StartUp_PopulateDb: True"
        * This flag is used by the "Load DB" code to determine whether or not the code
should run at all.
        * If not set, this value defaults to TRUE.
        * If set to False, "Load DB" simply returns without doing anything.

    - flag: "StartUp_SystemReadyDelayInMs: 60000"
        * This is used by the ovsdb-plugin code.  It used to be used at startup, but now
it is used
          whenever we need to wait for a given switch to return a good ConfigReady value.
        * If not present, the value defaults to 10000.
        * Note that there is no range-checking on this value.

    - flag: "SUB_GoodSubscriptionRefreshRateInMsecs: NNNNN"
       * This millisecond value determines approximately how often the Subscription Refresh

          threads run "when things are going well".
        * This value defaults to 40000 (aka 40 seconds).

    - flag: "SUB_BadSubscriptionRefreshRateInMsecs: NNNNN"
       * This millisecond value determines approximately how often the Subscription Refresh

          threads run "when things are going poorly".
        * It seems like this value should be smaller than the "good" value
          of SUB_GoodSubscriptionRefreshRateInMsecs.
        * This value defaults to 15000 (aka 15 seconds).
    - flag: "SUB_NumErrsBeforeDeclareBad: N"
        * This counter value determines how many consecutive errors a Subscription Refresh
          Thread should see before it declares the NXAPI connection as down.
       * Note that crossing this threshhold puts the Subscription Refresh thread in "probe
 mode",
          where it is in a loop, probing the nxapi connection and waiting for it to come
up.
        * See NumSuccessesToEscapeProbeMode for related info.
        * This value defaults to 4.

    - flag: "SUB_NumSuccessesToEscapeProbeMode: 5"
       * This counter value determines how many consecutive successes a Subscription Refresh

          Thread should see (once it has declared the connection down and is in "probe
mode").
        * See SUB_NumFailuresToDeclareBad for related info.
        * This value defaults to 5.

    - flag: "SUB_SocketConnectTimeoutInMsecs: 60000"
        * This is used by the SubscriptionRefreshThread as its "connection timeout" value.
       * This value is only used by the Single Subscription code in the Subscription Refresh
 Thread.
        * This value defaults to 20000 aka 20 seconds.

    - flag: "SUB_SocketReadTimeoutInMsecs: NNNNN"
        * This millisecond value determines what the read timeout value for the socket to
the switch should be.
       * This value is only used by the Single Subscription code in the Subscription Refresh
 Thread.
        * This value defaults to 20000 (aka 20 seconds).

    - flag: "SUB_ProbeModeDelayInMsecs: NNNNN"
       * This millisecond value determines how often the "probe the nxapi connection until
 it comes back up"
          code will loop.
        * This value defaults to 10000 (aka 10 seconds).

    - flag: "TL_ConnectionDownSleepInMs: NNNNN"
       * This millisecond value is used by the AsyncTaskWork (which sends batch information
 to the switch).
        * The AsyncTaskWorker, once it finds the NXAPI connection to be down, it sits in a
 loop and keeps waiting
          for an UP.
        * This value is the sleep time (in milliseconds) between subsequent connection
checks.
        * This value defaults to 20000 aka 20 seconds.
```

The following flags are not supported:

```
- flag: "DoNewAndromedaStatusFormat: True"
    - flag: "JP_NumBadMsgCounterThreshHold: 10"
    - flag: "JP_NumEndOfStreamReadsThreshHold: 10000"
    - flag: "JP_NumGenSocketExceptThreshHold: 10"
    - flag: "JP_NumSocketTimeoutExceptThreshHold: 10"
    - flag: "NX_MaxDebugLogOutput: NNNNN"
    - flag: "Startup_SystemReadyIgnoreResults: False"
    - flag: "StartUp_SystemReadyNumTries: 3"
    - flag: "VPC_BeanRequestDelayInSecs: NNN"
    - flag: "VPC_BeanRequestTimeoutInSecs: NNNN"
    - flag: "VPC_DoWeAllowNonVpcSwitches: True"
    - flag: "VPC_HowManyBeanErrsBeforeDeclarePrimary: NNN"
```