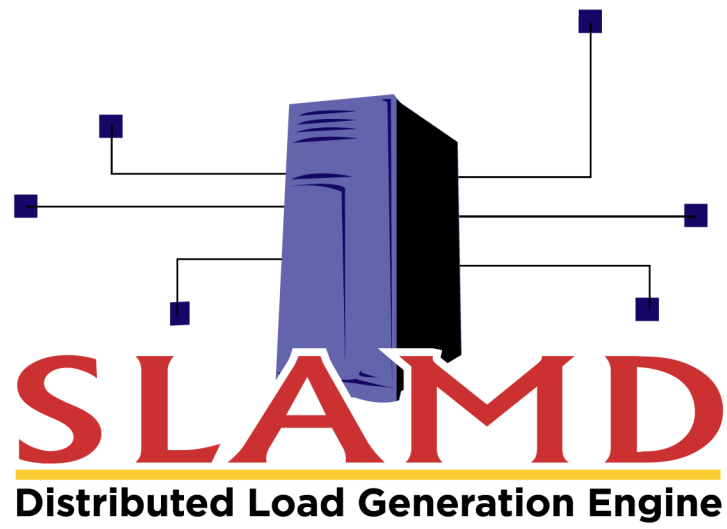


# Tools Guide



February 2009

# Table of Contents

<b><a href="#">Tools Overview</a></b> .....	<b>3</b>
<b><a href="#">Administrative Tools</a></b> .....	<b>4</b>
<b><a href="#">LDAP Performance Tools</a></b> .....	<b>6</b>
<a href="#">searchrate</a> .....	7
<a href="#">modrate</a> .....	8
<a href="#">authrate</a> .....	8
<b><a href="#">LDAPDecoder</a></b> .....	<b>10</b>
<a href="#">Running in Proxy Mode</a> .....	11
<a href="#">Running in Offline Mode</a> .....	18
<a href="#">Generating SLAMD Job Scripts</a> .....	21
<b><a href="#">LDIFStructure</a></b> .....	<b>25</b>
<b><a href="#">MakeLDIF</a></b> .....	<b>26</b>
<a href="#">Running MakeLDIF</a> .....	26
<a href="#">The Template File Format</a> .....	31
<a href="#">Template File Tags</a> .....	35
<a href="#">Defining Custom Tags</a> .....	43
<b><a href="#">TCPCapture and TCPReplay</a></b> .....	<b>47</b>
<a href="#">Running TCPCapture</a> .....	47
<a href="#">Running TCPReplay</a> .....	48
<b><a href="#">ThroughputTest</a></b> .....	<b>49</b>
<a href="#">The ThroughputTest Server</a> .....	49
<a href="#">The ThroughputTest Client</a> .....	49
<b><a href="#">UDPPing</a></b> .....	<b>51</b>
<a href="#">Running the UDPPing Client</a> .....	51
<a href="#">Running the UDPPing Server</a> .....	52
<b><a href="#">SLAMD License</a></b> .....	<b>53</b>

# Tools Overview

In addition to the web-based administrative interface that may be used to schedule jobs and view results, SLAMD comes with a number of useful tools that are related to ways in which you might want to use SLAMD. Some of these tools may be used to generate data to use when testing, some of them may be used to perform tests in a standalone manner, and others may be used to interact with the administrative interface from the command line. This document provides an overview of some of the tools included with SLAMD and information about how to use them.

For all tools provided with SLAMD, you may get some help by invoking the corresponding command with the "-h" argument. This will display a list of all supported arguments for that command, along with descriptions for those arguments.

# Administrative Tools

SLAMD is provided with a number of administrative tools that can be used to perform certain tasks from the command line rather as an alternative to the web-based interface. Some of those tools include:

- `add-job-class` -- This may be used to add a new job class to the SLAMD server so that jobs of that type may be scheduled.
- `cancel-job` -- This may be used to request that a pending or running job be cancelled.
- `create-folder` -- This may be used to create a job folder in the SLAMD server.
- `delete-folder` -- This may be used to delete a job folder in the SLAMD server.
- `disable-job` -- This may be used to disable a pending job so that it is not eligible to start.
- `enable-job` -- This may be used to enable a job that has been disabled.
- `export-data` -- This may be used to export job data from the SLAMD server in a form that may be imported into another SLAMD instance.
- `export-job` -- This may be used to export statistics collected by a job in tab-delimited text form in a manner that may be used by other applications (e.g., imported into a spreadsheet).
- `get-job` -- This may be used to display information for a specified job.
- `import-data` -- This may be used to import data into the SLAMD server that was exported using the `export-data` tool.
- `install-job-pack` -- This may be used to install a job pack into the SLAMD server, which is a collection of job classes that make it possible to schedule jobs of that type.
- `list-folders` -- This may be used to obtain a list of folders defined in the SLAMD server.

- `restart-slamd` -- This may be used to restart the SLAMD server without restarting the underlying web application container.
- `run-script` -- This may be used to execute a script written in the SLAMD scripting language.
- `slamd-status` -- This may be used to obtain status information from the SLAMD server.
- `start-slamd` -- This may be used to start the SLAMD server if it is not running but the underlying web container is running.
- `stop-slamd` -- This may be used to stop the SLAMD server without stopping the underlying web application container.
- `upload-file` -- This may be used to upload a file into a folder in the SLAMD server.
- `view-log` -- This may be used to view information from the SLAMD server log.

For each of these tools, you may obtain usage information by invoking the tool with the "-h" option. Most tools include the following options:

- `-h {address}` -- Specifies the address of the SLAMD server. By default, it will attempt to access the server on the local system.
- `-p {port}` -- Specifies the port of the SLAMD server. By default, it will use port 8080.
- `-A {authID}` -- Specifies the username to use to authenticate to the SLAMD server if that capability is enabled.
- `-P {password}` -- Specifies the password to use to authenticate to the SLAMD server if that capability is enabled.
- `-u {uri}` -- Specifies the base URI for the SLAMD administrative interface. By default, a URI of `"/slamd"` will be used.
- `-s` -- Specifies that communication with the SLAMD server should be encrypted with SSL.

# LDAP Performance Tools

While the primary purpose of SLAMD is to provide a graphical interface for running jobs across one or more client systems, it does provide a few tools which may be used to perform simple performance tests against LDAP directory servers from a single client. This is often useful for spot-checking, and these tools provide instantaneous real-time feedback as the job runs so you can examine the results as they are obtained.

Each of these tools provide the ability to target multiple entries through the use of value patterns. Value patterns are strings which may contain numeric ranges so that value patterns may be used to generate multiple values. Each numeric range should be enclosed in square brackets, with upper and lower bounds separated by either a colon (to indicate that values in the range should be chosen in sequential order) or a dash (to indicate that values in the range should be chosen randomly). If sequential ordering is to be used, you can follow the upper bound with an "x" character followed by an integer to indicate that values should use the specified interval rather than the default interval of one. In either sequential or random ordering, you can also follow the upper bound with a "%" character followed by a format string (as used by the `java.text.DecimalFormat` class) that indicates how the numeric value may be formatted. Examples of value patterns include:

- `(uid=user.[1:1000])` -- This indicates that values between 1 and 1000 should be selected in sequential order, so that the first value generated will be `"(uid=user.1)"`, the second will be `"(uid=user.2)"`, etc., up to `"(uid=user.1000)"`, at which point it will wrap back around to `"(uid=user.[1:1000])"`.
- `(uid=user.[1-1000])` -- This indicates that values between 1 and 1000 should be chosen at random and used in place of the bracketed range.
- `(uid=user.[1:1000x2])` -- This indicates that values between 1 and 1000 should be selected in sequential order, but using increments of 2. The first value generated will be `"(uid=user.1)"`, the second will be `"(uid=user.3)"`, and so on.
- `(uid=user.[1-1000%0000])` -- This indicates that values between 1 and 1000 should be chosen at random and should be formatted so that all values will be four

digits long, padded with leading zeros as necessary (so if a value of "12" is chosen, then the resulting value will be "(uid=user.0012)").

## searchrate

The `searchrate` tool provides the ability to perform repeated LDAP searches against a directory server and will report the recent and average rate of the searches, the recent and average times (in milliseconds, with microsecond accuracy) required to perform the searches, the number of entries returned per search, and the number of errors encountered during processing.

Some of the arguments supported by this tool include:

- `-h {address}` -- Specifies the address of the directory server to use. By default, it will use the local system.
- `-p {port}` -- Specifies the port of the directory server to use. By default, it will use port 389.
- `-D {bindDN}` -- Specifies the DN to use to bind to the directory server (if using simple authentication). By default, no authentication will be performed.
- `-w {password}` -- Specifies the password to use to bind to the directory server (if using simple authentication). By default, no authentication will be performed.
- `-z` -- Specifies that communication should be encrypted with SSL.
- `-b {baseDN}` -- Specifies the base DN to use for the searches. This may be a single DN, or it may be a value pattern.
- `-s {scope}` -- Specifies the scope to use for the searches. This may be one of "base", "one", "sub", or "subord".
- `-f {filter}` -- Specifies the filter to use for the searches. This may be a single filter value, or it may be a value pattern.
- `-t {num}` -- Specifies the number of concurrent threads to use to generate the load.

## modrate

The `modrate` tool provides the ability to perform repeated LDAP modifications against a directory server and will report the recent and average rate of the modifications, the recent and average times (in milliseconds, with microsecond accuracy) required to perform the modifications, and the number of errors encountered during processing.

Some of the arguments supported by this tool include:

- `-h {address}` -- Specifies the address of the directory server to use. By default, it will use the local system.
- `-p {port}` -- Specifies the port of the directory server to use. By default, it will use port 389.
- `-D {bindDN}` -- Specifies the DN to use to bind to the directory server (if using simple authentication). By default, no authentication will be performed.
- `-w {password}` -- Specifies the password to use to bind to the directory server (if using simple authentication). By default, no authentication will be performed.
- `-z` -- Specifies that communication should be encrypted with SSL.
- `-b {baseDN}` -- Specifies the DN of the entry to modify. This may be either a single DN or a value pattern.
- `-A {name}` -- Specifies the name of the attribute to modify. This argument may be provided multiple times with different attribute names if multiple attributes should be updated in each modification.
- `-l {length}` -- Specifies the number of characters to include in the values used in the modifications.
- `-t {num}` -- Specifies the number of concurrent threads to use to generate the load.

## authrate

The `authrate` tool provides the ability to perform repeated LDAP authentication sequences against a directory server, where each authentication consists of a search to find a user entry followed by a bind as that user to verify the credentials. It will report the recent and average rate of the authentications, the recent and average times (in



milliseconds, with microsecond accuracy) required to perform the authentications, and the number of errors encountered during processing.

Some of the arguments supported by this tool include:

- `-h {address}` -- Specifies the address of the directory server to use. By default, it will use the local system.
- `-p {port}` -- Specifies the port of the directory server to use. By default, it will use port 389.
- `-D {bindDN}` -- Specifies the DN to use to bind to the directory server for the connections used to perform the searches. By default, no authentication will be performed.
- `-w {password}` -- Specifies the password to use to bind to the directory server for the connections used to perform the searches. By default, no authentication will be performed.
- `-z` -- Specifies that communication should be encrypted with SSL.
- `-b {baseDN}` -- Specifies the base DN to use for the searches. This may be a single DN, or it may be a value pattern.
- `-s {scope}` -- Specifies the scope to use for the searches. This may be one of "base", "one", "sub", or "subord".
- `-f {filter}` -- Specifies the filter to use for the searches. This may be a single filter value, or it may be a value pattern.
- `-c {password}` -- Specifies the password to use when binding as the users identified by the searches.
- `-a {authType}` -- Specifies the type of authentication to perform. Supported authentication types are "SIMPLE", "CRAM-MD5", "DIGEST-MD5", and "PLAIN".
- `-t {num}` -- Specifies the number of concurrent threads to use to generate the load.

# LDAPDecoder

Many network protocols like HTTP and SMTP are text-based, which means that it is relatively simple to decode that information if it is intercepted over the wire. LDAP, however, is binary protocol that uses the ASN.1 basic encoding rules specification to encode all communication. While some components of LDAP communication (e.g., distinguished names) may be decipherable, it is significantly more difficult to interpret other data elements.

To address this problem, the LDAPDecoder utility provides a means of interpreting LDAP communication and displaying it in a human-readable form. This can be very useful for debugging problems with the interaction between LDAP clients and a directory server, or to simply gain a better understanding of the structure of LDAP traffic.

The LDAPDecoder offers two modes of operation:

- It can be configured as a very simple proxy, in which LDAP clients communicate with the LDAPDecoder rather than the directory server. The LDAPDecoder will decode the request before forwarding it on to the directory server, and it will decode the response from the server before forwarding it back to the client.
- It can be used to decode network packet captures taken from utilities like snoop on Solaris or tcpdump on Linux.

Although the first mode of operation does require that clients be reconfigured to communicate with the LDAPDecoder rather than directly with the directory server, this mode of operation does offer some advantages that may not be otherwise available:

- The LDAPDecoder can be configured to use SSL for its communication, in which case it could offer the ability to decode encrypted LDAP traffic.
- The LDAPDecoder can make it possible to decode communication for the case in which the LDAP client and directory server are on the same system. Some operating systems like Solaris do not offer the ability to capture traffic over the loopback interface.

- If configured to listen on a port greater than 1024, the LDAPDecoder does not require root access to operate. In general, network packet capture utilities do require root access.
- It can be used to more easily target the communication between a specific LDAP client and the directory server by simply changing that one client to communicate with the LDAPDecoder rather than the directory server. Depending on how it was obtained, a network packet capture may include traffic from a large number of clients, or even non-LDAP communication.
- Large LDAP requests and responses may be too big to fit into a single packet and therefore would need to be split into multiple packets. The LDAPDecoder cannot interpret LDAP communication in which the entire request is not present in a single packet.
- In some cases, particularly when processing a search that matches multiple entries, a multiple LDAP messages may be bundled into the same packet. The LDAPDecoder cannot interpret LDAP communication in which multiple LDAP messages are present in a single packet.

This document discusses the use of the LDAPDecoder utility in both modes of operation.

## Running in Proxy Mode

When the LDAPDecoder is configured to operate in proxy mode, LDAP communication between clients and the directory server must pass through the decoder. As it intercepts the communication, it will interpret and display the information it contains in a human-readable form.

To start the LDAPDecoder in proxy mode, use the command:

```
$ tools/ldap-decoder.sh -L {listenPort}
```

where *{listenPort}* is the port on which the LDAPDecoder should listen for client connections. By default, this will listen on all interfaces on the specified port, and will forward all traffic to a directory server listening on port 389 on the same system. However, this can be changed by providing additional command-line arguments. The other arguments that may be used are:

- **-h** *{serverAddress}* -- This specifies the address of the directory server to which all communication should be forwarded. It may be either an IP address or resolvable name. By default, an address of "127.0.0.1" will be used.
- **-p** *{serverPort}* -- This specifies the port of the directory server to which all communication should be forwarded. By default, a port of 389 will be used.
- **-l** *{listenAddress}* -- This specifies the address on which the LDAPDecoder will listen for connections from clients. By default, the LDAPDecoder will listen on all interfaces using an address of "0.0.0.0".
- **-f** *{outputFile}* -- This specifies the path to the output file to which the decoded LDAP communication should be written. By default, the information will be written to standard output.
- **-F** *{outputFile}* -- This specifies the path to the output file to which decoded communication will be written as a SLAMD job script. By default, no script file will be written.
- **-m** -- This specifies that each client connection should be written to a separate output file. By default, all communication captured will be written to the same place (either standard output or a specified output file). If this option is used, then an output file must be specified with the **-f** option, and the address and port of the client connection will be appended to this filename for each connection.
- **-s** -- This specifies that the communication between the LDAPDecoder and the directory server should be encrypted using SSL. More information on using SSL will be provided later in this section. By default, SSL will not be used for this communication.
- **-S** -- This specifies that the communication between the clients and the LDAPDecoder should be encrypted using SSL. More information on using SSL will be provided later in this section. By default, SSL will not be used for this communication.
- **-b** -- This specifies that the raw bytes of the LDAP communication should be included in the output. By default, only the human-readable interpretation of the traffic will be output. Including the raw bytes received can be useful for debugging purposes. Note that only the actual TCP data will be included. Other data in the packet received (i.e., transport-layer, IP, and TCP headers) will not be included in the output.

- `-v` -- This specifies that the LDAPDecoder should operate in verbose mode. This will cause more debugging information to be generated that can help diagnose any problems that may occur.

When the LDAPDecoder is started in proxy mode, it will display a message indicating that it is listening for client connections, like:

```
$ tools/ldap-decoder.sh -L 5389
Listening on 0.0.0.0:5389 for client connections
```

At this point, whenever a client establishes a connection to the LDAPDecoder, the LDAPDecoder will establish a corresponding connection to the directory server. Then, any requests from the client will be decoded before being passed onto the directory server, and responses from the server will be decoded before being sent to the client. For example, if the client issues the command:

```
$ ldapsearch -p 5389 -D "cn=Directory Manager" -w password \
  -b "dc=example,dc=com" -s base "(objectClass=*)"
```

the output produced by the LDAPDecoder may be:

```
[26/Feb/2004:14:40:55.221 -0600] -- New client connection from 127.0.0.1:38694
[26/Feb/2004:14:40:55.269 -0600] -- Read data from the client
Decoded Data from Client:
  LDAP Bind Request
    Message ID: 1
    LDAP Bind Request Protocol Op
      LDAP Version: 3
      Bind DN: cn=Directory Manager
      Authentication Data:
        Authentication Type: Simple
        Bind Password: password

[26/Feb/2004:14:40:55.325 -0600] -- Read data from the server
Decoded Data from Server:
  LDAP Bind Response
    Message ID: 1
    LDAP Bind Response Protocol Op
      Result Code: 0 (Success)

[26/Feb/2004:14:40:55.330 -0600] -- Read data from the client
Decoded Data from Client:
  LDAP Search Request
    Message ID: 2
```

```

LDAP Search Request Protocol Op
  Base DN:  dc=example,dc=com
  Scope:    0 (baseObject)
  Deref Aliases:  0 (neverDerefAliases)
  Size Limit:    0
  Time Limit:    0
  Types Only:    false
  Filter:  (objectClass=*)
  Attributes:

[26/Feb/2004:14:40:55.348 -0600] -- Read data from the server
Decoded Data from Server:
  LDAP Search Result Entry
    Message ID:  2
    LDAP Search Result Entry Protocol Op
      dn: dc=example,dc=com
      dc: example
      objectClass: top
      objectClass: domain

[26/Feb/2004:14:40:55.357 -0600] -- Read data from the server
Decoded Data from Server:
  LDAP Search Result Done
    Message ID:  2
    LDAP Search Result Done Protocol Op
      Result Code:  0 (Success)

[26/Feb/2004:14:40:55.391 -0600] -- Read data from the client
Decoded Data from Client:
  LDAP Unbind Request
    Message ID:  3
    LDAP Unbind Request Protocol Op

[26/Feb/2004:14:40:55.396 -0600] -- Error reading data from the client
[26/Feb/2004:14:40:55.397 -0600] -- Connection from 127.0.0.1 closed

```

## Using SSL Between the LDAPDecoder and Directory Server

Normally, whenever a client connects to a directory server using SSL, all communication between them is encrypted so that it is indecipherable to anyone else that might be able to see this traffic. However, if the LDAPDecoder is run in proxy mode, it becomes an endpoint for the communication with the client, which means that it can interpret the encrypted communication from the client. SSL is supported for communication between the client and the LDAPDecoder

as well as between the LDAPDecoder and the directory server. The LDAPDecoder uses JSSE to provide SSL support, but before it may be used it is necessary to configure the JSSE keystore.

Providing support for SSL between the LDAPDecoder and the directory server is simpler than between the client and the LDAPDecoder, so that will be discussed first. In this case, it is only necessary to ensure that the LDAPDecoder trusts the SSL certificate used by the directory server. If the directory server's certificate has been generated by a commercial certificate authority (e.g., VeriSign or Thawte), then JSSE may already trust the certificate by default. However, if that is not the case, then it will be necessary to create a certificate trust store with an appropriate certificate from the directory server's certificate database.

To do this for a Sun ONE Directory Server instance, the `certutil` tool provided in the `shared/bin` directory under the install root may be used. First, list all the certificates in the certificate database using the command:

```
$ certutil -L -d ../../alias -P "slapd-{instanceName}-"
```

where `{instanceName}` is the name of the directory server instance with which the LDAPDecoder will communicate using SSL. There will likely be either one or two certificates listed in this database. If there is only one, then that certificate is probably self-signed, and that certificate will be the one that should be imported into the JSSE trust store. If there are two, then they will likely be the server certificate (probably named "`server-cert`") and the certificate of the certificate authority that signed the server certificate. In that case, the CA certificate is the certificate that should be used.

Once the name of the appropriate certificate has been identified, it may be retrieved in ASCII form using another form of the `certutil` command:

```
$ certutil -L -d ../../alias -P "slapd-{instanceName}-" -a -n {nickname}
```

where `{nickname}` is the nickname of the certificate to be exported. If all goes well, then the public key information for the certificate should be displayed, like:

```
-----BEGIN CERTIFICATE-----
MIICiDCCAFGgAwIBAgIFAPtZ6NcWdQYJKoZIhvcNAQEEBQAwezELMAkGA1UEBhMC
VVMxJDAJBGNVBAgTBVRleGFzMQ8wDQYDVQQHEwZBdXN0aW4xGTAXBGNVBAoTEFN1
biBNawNybn3N5c3RlbXMxEDAOBgNVBASTB1N1biBPTkUxHjAcBgNVBAMTFXB1Y29z
LmN1bnRyYWwuc3VuLmNvbTAeFw0wMzEyMDkxODAyNTdaFw0wNTAzMDkxODAyNTda
MHsxZCZAJBgNVBAYTAlVTMQ4wDAYDVQQIEwVUZXhhczEPMA0GA1UEBxMGQXVzdGlu
MRkwFwYDVQQKEwBTdW4gTW1jcm9zeXN0ZW1zMRAwDgYDVQQLEwdTdW4gT05FMR4w
HAYDVQQDExVwZW50cmF5LnN1bi5jb20wgZ8wDQYJKoZIhvcNAQEEBQAD
gY0AMIGJAoGBALp4iX1vphWemXPCAXC/zcAlmZ/8rzbr+1j+a0hjrG/UoFFPcyMn
H7xhXD31FzMlhDehSroHJe3VFPgBCNKR4IcCv139lhDi1ip475+fO+7ZI36dbats
I0gWB9MHYcm+LtUx6I3SRoKsB/DK1HEKtUqTEQNvEr1v8D4KeMRW44nxAgMBAAGj
GDAWMBQGCGSAGG+EIBAQB/wQEAwIGwDANBgkqhkiG9w0BAQQFAAOBgQBMnAzN
1sYJcGJVE0TdRiRpW03E2LIEpn1wy/Cnr3OpCkvwK9kWsp3hxAYvXlJ/29CNad5/
22QT/3VVFZ6kKtxatVnOTb/7c5/tE98VcDChFAFkJncSksxtrA4047+wKFFqa7cP
vh0fgXD4emYTPASxHmK5zC2eiRpBjY91MVNsrg==
-----END CERTIFICATE-----
```

This output (including the lines containing **BEGIN CERTIFICATE** and **END CERTIFICATE**) should be copied into a text file that the LDAPDecoder can access. It may then be imported into a JSSE trust store using the **keytool** utility provided with the Java installation, like:

```
$ keytool -import -alias ds-cert -file {certFile} -keystore trust.store
```

where *{certFile}* is the path to the file containing the ASCII representation of the certificate obtained above.

Once the trust store has been created, an argument should be provided on the command line that will tell Java to use that trust store when communicating with the server over SSL. This can be done like:

```
$ env JAVA_ARGS="-Djavax.net.ssl.trustStore=trust.store" \
  tools/ldap-decoder.sh -L 5389 -h ssl-ds.example.com -p 636 -s
```

This will configure the LDAPDecoder so that it will communicate over SSL with the directory server *sslds.example.com* on port 636, while clients will still communicate with the LDAPDecoder itself without using SSL. Note that there cannot be any spaces in the **"-Djavax.net.ssl.trustStore=trust.store"** portion of the command line.

## Using SSL Between Clients and the LDAPDecoder

While providing the ability for the LDAPDecoder to communicate securely with the directory server is important, it is of limited usefulness without the ability for clients to communicate security with the LDAPDecoder. This can be done, but it is more complex and requires that the LDAPDecoder itself be issued a server certificate. This certificate may be either self-signed (which is free, but may require that the client be provided with the certificate so it may be trusted) or signed by an external CA (which may or may not be free, but could be easier to use with clients).

Regardless of whether a self-signed or externally-signed certificate will be used, the first step is to create a certificate request. This may be done using the **keytool** utility, like:

```
$ keytool -genkey -alias decoder-cert -dname {subject} -validity 365 \
  -keyalg rsa -keystore key.store
```

where *x* is the subject of the certificate. The subject should be something like **"CN={fqdn},O={companyName},C={countryCode}"**, where *{fqdn}* is the fully-qualified domain name of the system on which the LDAPDecoder will be run, *{companyName}* is the name of the organization with which the system is associated, and *{countryCode}* is the two-character country code for the country in which it will be used (e.g., "US" for the United States). After prompting for a password (which may be any value if this is a new keystore), this will generate a private key for the certificate in the keystore file **key.store**.



If the certificate used by the LDAPDecoder is to be self-signed, then that certificate can be generated using the command:

```
$ keytool -selfcert -alias decoder-cert -validity 365 \  
-keystore key.store
```

After being prompted for a password (which should be the same password used when generating the private key), the certificate will be signed. At this point, it will likely need to be exported in ASCII form so that it may be imported into a certificate database for use by LDAP clients. This may be done using the command:

```
$ keytool -list -rfc -alias decoder-cert -keystore key.store
```

If the certificate is going to be signed by an external CA, then it will be necessary to extract the certificate request from the keystore. This can be done using the command:

```
$ keytool -certreq -alias decoder-cert -file decoder.csr \  
-keystore key.store
```

The certificate signing request will be written to the file decoder.csr, which can then be provided to a certificate authority to be signed. Once the request has been signed and is available (preferably as a PKCS#7 certificate chain), the certificate may be imported into the certificate database using the command:

```
$ keytool -import -trustcacerts -alias decoder-cert \  
-file decoder.cert -keystore key.store
```

Once the keystore contains a valid certificate (either self-signed or signed by an external CA), then the LDAPDecoder may be configured to accept SSL-based connections from clients. To do this, it must be provided with both the location of the JSSE keystore and the password required to access the private key that it contains. This requires providing two additional properties on the command line, like:

```
$ env JAVA_ARGS="-Djavax.net.ssl.keyStore=key.store  
-Djavax.net.ssl.keyStorePassword=password  
-Djavax.net.ssl.trustStore=trust.store" \  
tools/ldap-decoder.sh -L 5636 -S -h ssllds.example.com -p 636 -s
```

At this point, any communication between the clients and the LDAPDecoder will be encrypted using SSL, and any communication between the LDAPDecoder and the directory server will be encrypted using SSL. Note that it is not necessary to have both enabled at the same time, as either segment can be SSL-enabled independently of the other.

## Running in Offline Mode

While running the LDAPDecoder in proxy mode is useful for debugging problems under controlled conditions, it is not feasible in all cases (e.g., when it is not possible to reconfigure the clients to communicate with the LDAPDecoder rather than directly with the directory server). For that reason, the LDAPDecoder offers the ability to operate in offline mode, in which case it can parse and interpret data captured in either a snoop version 2 (like that used by the Solaris `snoop` utility) or libpcap 2.4 (like that used by `tcpdump` on Linux and other platforms) capture file format. Note that because the traffic is not passing through the LDAPDecoder in this case, it is not possible to decode SSL-encrypted communication. However, non-SSL communication may be examined and interpreted.

In order to start the LDAPDecoder in offline mode, the following command should be used:

```
$ tools/ldap-decoder.sh -i {captureFile}
```

where *{captureFile}* is the path to the snoop or libpcap capture file containing the data to be interpreted. By default, the LDAPDecoder will attempt to interpret any TCP packets with a source or destination port of 389 as LDAP traffic. However, this behavior may be changed using command line arguments. The arguments supported for offline mode include:

- **-h {serverAddress}** -- This specifies the address of the system on which the directory server is running. By default, any TCP traffic with the appropriate source or destination port will be examined, which can be useful if the capture contains data from interaction with multiple directory servers. However, if a server address is provided, then only communication with that system will be examined.
- **-p {serverPort}** -- This specifies the port on which the directory server is running. By default, a value of 389 will be used.
- **-f {outputFile}** -- This specifies the path to the output file into which the decoded LDAP traffic should be written. By default, this information will be sent to standard output.
- **-F {outputFile}** -- This specifies the path to the output file to which decoded communication will be written as a SLAMD job script. By default, no script file will be written.
- **-b** -- This specifies that the raw bytes of the communication should be included in the output in addition to the human-readable parsed LDAP messages. This can be useful for debugging purposes.

- **-v --** This specifies that the LDAPDecoder should operate in verbose mode, which can provide useful debugging information if problems arise.

The output generated by the LDAPDecoder when run in offline mode will be the same as the output generated when it is run in proxy mode.

Note that the LDAPDecoder currently does impose some limitations on the kind of packet captures that may be examined. In particular, only IPv4 is currently supported, so any communication using IPv6 cannot be interpreted. Further, only captures obtained over Ethernet will be allowed. Other data link devices (e.g., token ring, FDDI, HDLC, ARCNet, SLIP, PPP, etc.) are not supported. Note that the Linux loopback device is treated as Ethernet, so captures obtained over the loopback interface on Linux systems can be examined.

## Using snoop to Capture LDAP Communication

Solaris includes a utility called **snoop** that can capture network traffic and store it in a form that may be examined using the LDAPDecoder. While the manual page for **snoop** does include all the information necessary to use the utility, a brief overview will be provided here as well.

If snoop is run on the command line with no arguments, it will print a limited amount of information about each packet that it sees to standard output. The output in this form is not of any use to the LDAPDecoder, as it needs access to the raw data contained in those packets. Therefore, the **-o** option must be used to specify the file into which the capture data will be written, like:

```
# snoop -o snoop.capture
```

This will write information about each packet captured into the **snoop.capture** file in a binary format described in RFC 1761.

By default, **snoop** will listen on the first non-loopback interface that it finds in the system. On a multi-homed system, this may not be the interface over which the LDAP traffic will be transferred. In that case, the specific interface to use can be specified using the **-d** argument. For example:

```
# snoop -o snoop.capture -d hme0
```

will cause **snoop** to listen on the "hme0" interface rather than any other interface that might be present in the system. If a system does have multiple network interfaces, then it is recommended that the "-d" argument always be provided so that the correct interface will be used.

Also by default, **snoop** will listen using promiscuous mode, which means that it will capture information about any network traffic that it sees on the target interface, regardless of whether that traffic is actually coming from or going to the system running **snoop**. That is, if **snoop** is running in promiscuous mode on an unswitched network, then it may capture traffic between other systems on the same network segment. In some cases this may be desirable, but in others it may cause the capture to be unnecessarily large or even introduce confusion into the process of interpreting the data if another system on the same network segment is also running an LDAP directory server. Therefore, it is possible to configure **snoop** to not listen in promiscuous mode by providing the "-P" argument on the command line.

Finally, **snoop** will capture all network traffic that it sees, which may include packets that have nothing to do with the directory server. While the LDAPDecoder will simply ignore such packets, including this information can dramatically increase the size of the capture file, particularly for captures running for an extended period of time. Therefore, it may be desirable to include an expression on the command line to indicate that only traffic meeting a certain criteria should be captured. For example, an expression of "**tcp port 389**" will restrict the capture to only include TCP traffic in which the source or destination port number is TCP port 389. Consult the **snoop** man page for additional information on what may be present in this expression.

Using this information, an appropriate command line to use in order to capture LDAP communication using **snoop** might be:

```
# snoop -P -d hme0 -o snoop.capture tcp port 389
```

When the desired information has been captured, simply use Ctrl+C to stop the capture. The **snoop.capture** file may then be provided to the LDAPDecoder so that it may be interpreted.

## Using tcpdump to Capture LDAP Communication

The **tcpdump** utility is very similar to **snoop** in the way that it operates and the features that it provides. The primary differences in usage are in the command-line arguments that should be provided to accomplish certain tasks. For example:

- To send the data captured by **tcpdump** to a binary file rather than printing summary information to standard output, use the "-w {filename}" argument.

- To specify which interface to use when capturing data, use the "**-i** {interface}" argument.
- To disable promiscuous mode, use the "**-p**" argument.
- Capture expressions are still allowed, and in this case "**tcp port 389**" is still a valid capture expression. However, there may be syntax differences between the capture expressions used by **snoop** and **tcpdump** in some cases.

One other important note is that in its default mode of operation **tcpdump** may truncate packets so that the entire packet is not captured. To prevent that, make sure to add "**-s 0**" to the command line, which indicates that the entire packet should be captured.

As an example, a **tcpdump** command line to capture LDAP communication could be as follows:

```
# tcpdump -p -i eth0 -w tcpdump.capture -s 0 tcp port 389
```

Consult the **tcpdump** man page for further information on using this utility.

## Generating SLAMD Job Scripts

By default, whenever the LDAPDecoder is used, it will generate human-readable output that can be used to better understand the kinds of operations that the client performs when communicating with the directory server. This information can then be used by a developer to create a custom SLAMD job class or script based on that access pattern. However, as of SLAMD 1.8.0 the LDAPDecoder also provides the ability to write captured data to a job script that can be executed in the SLAMD scripting engine with or without modification. If the generated script is not modified, then executing the script will simply replay the exact set of requests that the client sent to the server. If the script is edited, then there are a lot more options for making it more generic (e.g., choosing users at random rather than always using the same one as the client that performed the capture) as well as to allow for the possibility for running those operations in a loop to turn a relatively small set of requests into a much larger one.

The LDAPDecoder will automatically record any captured requests to a SLAMD script if you use the "**-F** {outputFile}" argument on the command line (it will also send a human-readable version of the output to either standard output or the file specified using the "**-f** {outputFile}" argument, although this can be set to something like **/dev/null** if that output is not desired). The generated script should be completely executable on its own to reproduce the behavior captured by the client. For example, issuing the following **ldapsearch** command:

```
$ ldapsearch -p 2389 -D "cn=Directory Manager" -w password \
    -b "dc=example,dc=com" -s base "(objectClass=*)"
```

can result in a job script that looks like:

```
# This script was dynamically generated by the the SLAMD LDAPDecoder tool.
# Generation Date:  Fri Oct 08 11:05:12 CDT 2004
```

```
# Make the LDAP data types available for use.
use com.sun.slamd.scripting.ldap.LDAPAttributeVariable;
use com.sun.slamd.scripting.ldap.LDAPAttributeVariable;
use com.sun.slamd.scripting.ldap.LDAPConnectionVariable;
use com.sun.slamd.scripting.ldap.LDAPEntryVariable;
use com.sun.slamd.scripting.ldap.LDAPModificationVariable;
use com.sun.slamd.scripting.ldap.LDAPModificationSetVariable;
```

```
# Define the variables that we will use.
variable boolean          useSSL;
variable int              resultCode;
variable int              port;
variable LDAPConnection   conn;
variable LDAPEntry        entry;
variable LDAPModification mod;
variable LDAPModificationSet modSet;
variable string           bindDN;
variable string           bindPW;
variable string           host;
variable string           message;
variable StringArray      searchAttrs;
```

```
# Read the values of all the configuration arguments.
host  = script.getScriptArgument("host", "127.0.0.1");
port  = script.getScriptIntArgument("port", 1389);
useSSL = script.getScriptBooleanArgument("useSSL", false);
bindDN = script.getScriptArgument("bindDN", "");
bindPW = script.getScriptArgument("bindPW", "");
```

```
# Indicate that the connection should collect and report statistics.
conn.enableAttemptedOperationCounters();
conn.enableSuccessfulOperationCounters();
conn.enableFailedOperationCounters();
conn.enableOperationTimers();
```

```
# Establish the connection that will be used for all the work.  If the
# connection attempt fails, then exit with an error.
resultCode = conn.connect(host, port, bindDN, bindPW, 3, useSSL);
if resultCode.notEqual(conn.success())
begin
    message = "Unable to connect.  Result code was:  ";
```

```

    message = message.append(resultCode.toString());
    script.logMessage(message);
    script.exitWithError();
end;

#### Bind request captured at Fri Oct 08 11:05:36 CDT 2004
# Bind DN:  cn=Directory Manager
# Authentication Type:  Simple
# Authentication Password:  password
resultCode = conn.bind("cn=Directory Manager", "password");

#### Bind response captured at Fri Oct 08 11:05:36 CDT 2004
# Result code:  0

#### Search request captured at Fri Oct 08 11:05:36 CDT 2004
# Search Base:  dc=example,dc=com
# Scope:  baseObject
# Deref Policy:  neverDerefAliases
# Size Limit:  0
# Time Limit:  0
# Types Only:  false
# Filter:  (objectClass=*)
searchAttrs.removeAll();
resultCode = conn.search("dc=example,dc=com", conn.scopeBase(),
"(objectClass=*)", searchAttrs, 0, 0);

#### Search result entry captured at Fri Oct 08 11:05:36 CDT 2004
# dn: dc=example,dc=com
# dc: example
# objectClass: top
# objectClass: domain

#### Search result done captured at Fri Oct 08 11:05:36 CDT 2004
# Result code:  0

#### Unbind request captured at Fri Oct 08 11:05:36 CDT 2004
# Not acutally going to unbind to prevent problems with future operations.
# If you actually want the unbind processed, then uncomment the next line:
# conn.disconnect();

#### LDAPDecoder shutdown detected at Fri Oct 08 11:05:45 CDT 2004
# Close the connection to the directory server.
conn.disconnect();

```

As can be seen from this script, only the requests from the client are actually included as executable code. The responses from the server are included as comments, but as they will not be replayed by the client as part of the load generation process, then there is no need to include any processing associated with them in the script. The client will automatically read all response messages associated with a request (including all search result entries), but by default will not do any processing on them. Also note that unbind requests sent by the client will be included in the script but commented out so that connection closures will not prevent future operations from being processed.

This captured information can be used as the basis of complete SLAMD job scripts that can accurately simulate real-world clients. See the SLAMD Scripting Language Guide for more information on the scripting language features that are available.



# LDIFStructure

The LDIFStructure tool is designed to help analyze an LDIF file and summarize its contents. This is very helpful for generating a MakeLDIF template from real-world data, but it can have a number of other uses as well when trying to understand the characteristics of the data.

The LDIFStructure tool provides both command-line and graphical modes. The graphical mode is the simplest way to view information about the structure of the analyzed LDIF file, and it may be invoked using the command:

```
$ tools/ldif-structure.sh -l {ldifFile}
```

This will analyze the contents of the specified LDIF file and will display a graphical representation of its contents. The left pane will show the tree structure for the data. The upper right pane will show the unique sets of object classes for entries immediately below the selected node in the tree structure. The lower right pane will provide summary information about the entries with that combination of object classes that are immediately below the selected node in the tree structure.

To start the LDIFStructure tool in text mode, issue the command:

```
$ tools/ldif-structure.sh -l {ldifFile} -o {outputFile} [-a]
```

This will analyze the contents of the provided LDIF file and will write information about the structure to the specified output file. If the "-a" argument is provided, then the information written to the output file will be an aggregate of all entries immediately below each parent entry in the LDIF file. If the "-a" argument is not provided, then a separate summary will be written for each unique combination of object classes.

# MakeLDIF

MakeLDIF is a Java-based utility for generating LDIF files. It may be used to generate sample data to import into an LDAP directory server. Although other utilities exist for this purpose (e.g., `dbgen.pl`), MakeLDIF offers a number of powerful features not available in other tools:

- MakeLDIF is highly customizable. LDIF files are generated based on templates defined by the user. This means that it is easy to include custom attributes, model complex DIT structures, and generate realistic data.
- User-defined templates can make use of a wide range of tags that make it possible to dynamically generate different kinds of data. In addition, an API is available that makes it possible to define custom tags to handle different kinds of processing not offered by the standard tags.
- MakeLDIF is written in Java. This allows it to be used on a wide range of platforms without recompilation like native code, but does not suffer from the lack of large file support in most Perl interpreters.
- MakeLDIF makes it possible to provide additional information while it is in the process of generating the LDIF file. For example, you can write an additional file with the list of the DNs of all the entries created, or you can generate a list of potential search filters that can be used to access the data once it has been imported.

MakeLDIF can easily be used as a standalone utility for generating LDIF data for a number of purposes. However, because it has been designed for use with the SLAMD distributed load generation engine, it offers a number of features that make it especially useful for generating data to use in a directory server for running many of the SLAMD jobs that work with LDAP directory servers.

## Running MakeLDIF

In order to run MakeLDIF, the minimum required command line is:

```
$ tools/make-ldif.sh -t {template} -o {output}
```

where *{template}* is the path to the template file that describes the way in which the LDIF file should be generated and *{output}* is the path to the output file that should be created. For example:

```
$ tools/make-ldif.sh -t example.template -o example.ldif
```

will cause MakeLDIF to generate an LDIF file named **example.ldif** based on the information contained in the **example.template** template file.

Although only the **-t** and **-o** options were illustrated above, a number of command-line arguments may be used with MakeLDIF to customize its behavior. The full set of supported arguments is as follows:

- **-t {filename}** -- Specifies the path to the template file that describes the way in which the LDIF file should be generated. This is a required parameter. Information about the format to use for MakeLDIF template files will be provided in later sections.
- **-o {filename}** -- Specifies the path to the output file to which the LDIF data will be written. This is a required parameter. If the **-m** parameter is also used, then this filename will be the name of the first file that is created, and all subsequent files will have a numeric extension appended to it.
- **-c {filename}** -- Specifies the path to a CSV or other delimited text file that contains data that will be used in the process of generating the LDIF. This must be specified if the template makes use of the "**<csvfield:{number}>**" tag.
- **-c {delimiter}** -- Specifies the delimiter that should be used when parsing the delimited text file. If this is not specified, then it will be assumed that the file will use comma-separated values. A tab may be specified as the delimiter by using the character sequence **"\t"**.
- **-f {filename}** -- Specifies the path to the file containing the list of first names to be used when generating the data. By default, MakeLDIF will use a file named **first.names** in the current working directory, although this option can be used to specify a different file. If a different file is to be used, then it should contain one name per line and it should not contain any duplicates.
- **-l {filename}** -- Specifies the path to the file containing the list of last names to be used when generating the data. By default, MakeLDIF will use a file named **last.names** in the current working directory, although this option can be used to specify a different

file. If a different file is to be used, then it should contain one name per line and it should not contain any duplicates.

- **-d {filename}** -- Specifies the path and name of a file into which the DNs of the entries generated will be written. This DN file can then be used in conjunction with other utilities like the SLAMD ModRate job that have the ability to operate on a list of entry DNs. If this option is not provided, then no DN file will be generated.
- **-b {filename}** -- Specifies the path and name of a file into which bind information for the generated entries will be written. Each line written to this file will be in the form *{dn}{tab}{password}*, where *{dn}* is the DN of the user entry, *{tab}* is the tab character, and *{password}* is the value of the **userPassword** attribute for the user entry. If this option is not provided, then no bind information file will be generated.
- **-L {filename}** -- Specifies the path and name of a file into which login information is to be written. Each line written to this file will be in the form *{loginID}{tab}{password}*, where *{loginID}* is the value of the login ID attribute, *{tab}* is the tab character, and *{password}* is the value of the **userPassword** attribute for the user entry. If this option is not provided, then no login information file will be generated.
- **-i {attribute}** -- Specifies the name of the attribute that should be used to hold the login ID for the user if the **-L** option is used. By default, the **uid** attribute is assumed to hold the login ID, although this option can be used to specify a different attribute.
- **-F {filename}** -- Specifies the path and name of a file into which filter information is to be written. If this option is used, then the **-T** option must also be used to specify the types of filters to create, and the information will be written with one filter per line. If this is used in conjunction with the **-m** option, then each filter file created will append the name of the attribute and index type to this base filename.
- **-T {type}** -- Specifies the attributes and index types that should be included in the filter file. The format of the filter type should be *{attribute}::type*, where *{attribute}* is the name of the attribute for which the filters are to be generated and *{type}* is the index type for which the filters are to be generated. The index types that may be specified are **eq** for equality indexes (each unique value of the attribute will be included in the filter list), **subInitial** (for substring filters containing the first three characters of a value followed by an asterisk), **subAny** (for substring filters containing each unique three-character combination present in any of the values surrounded by asterisks), **subFinal** (for substring filters containing an asterisk followed by the last three characters in the filter list), and **sub** (which is a shorthand notation for including **subInitial**, **subAny**, and **subFinal** filter types). It is possible to specify that multiple filter types should be used in the filter list by separating them with commas. For example, "**-T cn:eq,subInitial**" will generate both equality and subInitial filters based on values of

the cn attribute. Multiple `-t` options can be provided on the same command line to indicate that filters should be generated for multiple attributes.

- `-n {value}` -- Specifies the number of characters that should be included in substring filters generated by MakeLDIF. This applies to all substring filter types (subInitial, subAny, and subFinal). The default value is 3, but any positive integer value is allowed.
- `-N {value}` -- Specifies the minimum number of entries in the LDIF file that will be required to match a filter before it will be included in the filter file. This applies to all filter types (equality as well as all substring types). The default value is 1, but any positive integer value is allowed.
- `-x {value}` -- Specifies the maximum number of entries in the LDIF file that will be allowed to match a filter before it is excluded from the filter file. This applies to all filter types (equality as well as all substring types). By default, there is no limit, but specifying any positive value greater than or equal to the minimum number of entries to match (as specified with the `-N` option) will enforce a maximum limit.

This can be very useful for directory servers that have an ALLIDs threshold or similar feature in which the server will not maintain an index key that matches more than a specified number of entries.

- `-s {value}` -- Specifies a numeric seed to use for the random number generator. By default, the random number generator will be seeded based on the current time, which means that every time the LDIF file is generated, the random values will be different. However, if a random seed is provided, then the LDIF file will consistently contain the same sequence of "random" values. That is, if the same template file is used and the same random seed is provided, then the LDIF files generated by MakeLDIF will be identical. Note that this may not be true if certain kinds of tags are used in the template that may use their own random number generator (e.g., the `exec` tag).
- `-m {value}` -- Specifies the maximum number of entries that should be written to a single LDIF file. By default, all entries generated will be written to a single LDIF file (the one specified using the `-o` argument), but if the `-m` argument is provided, then at most `{value}` entries will be written to any single file. This can be useful if there is a possibility that the LDIF data will need to be accessed by a utility that does not provide large file support.
- `-X {value}` -- Specifies the maximum number of entries that should be created for each template under each branch. This may be used to produce a small version of the LDIF file that can be used to validate that the template has been defined properly to produce the desired result.

- **-w --** Specifies that long lines in the LDIF output should be wrapped at a column length of 75 characters. By default, no wrapping is performed.
- **-m --** Specifies that a different filter file should be created for each index type of each attribute specified using the **-t** argument. By default, all filters will be written to the same filter file (the file specified by the **-f** option), but this argument will cause each the filter list for each attribute and index type to be written to a separate file. The name of each filter file will be the base filter file name followed by a period, the attribute name, another period, and the index type.
- **-s --** Specifies that MakeLDIF should skip branch entries when generating the LDIF output. That is, only template entries will be written to the LDIF file but not the parent entry for those template entries. By default, the LDIF file generated will contain both branch and template entries.
- **-d --** Specifies that MakeLDIF should operate in debug mode, which will cause it to provide additional debugging information for some errors that may occur while using MakeLDIF.
- **-h --** Specifies that MakeLDIF should print usage information and exit without performing any other action. If this argument is provided, then no other options are required.
- **-v --** Specifies that MakeLDIF should print version information and exit without performing any other action. If this argument is provided, then no other options are required.

## The Template File Format

As mentioned earlier, MakeLDIF uses template files to define the way in which the LDIF files should be generated. This is a much more powerful approach than those taken by other LDIF generators because it allows for a great deal of flexibility without the need to alter any code to produce the desired result.

Template files contain three sections:

- Global replacement variables
- Branch definitions
- Template definitions

The remainder of this section will discuss each separately.

## Global Replacement Variables

The first section that should be present in the template file is the section that defines the global replacement variables. Global replacement variables are used to define strings of text that can be referenced later in the template file and will be automatically replaced as each line is read into memory. For example, the replacement variable definition:

```
define suffix=dc=example,dc=com
```

will create a global replacement variable named "**suffix**" with a value of "**dc=example,dc=com**". Once a global replacement variable has been defined, any case in which that replacement variable name appears in square brackets (e.g., "[**suffix**]"), then that token will be replaced with the value that has been defined for the replacement variable.

Once all the replacement variable definitions have been read (as signified by the first blank line following one or more replacement variable definitions), all remaining lines that are read from the template file will be processed on a line-by-line basis and any occurrences of a replacement variable name in square brackets will be replaced with the value of that variable. Because that replacement is done as the template file is read into memory, replacement variables may occur in any point, including branch definitions that are not parsed for other tags.

If there are global replacement variables defined in the template file, then they must appear at the top of the file. However, it is not required that there be any replacement variables, and if there are not then the template file should start with the branch definitions.

## Branch Definitions

Branch definitions are used in the template file to define the basic structure to use for the LDIF file. They specify the entry or entries that should appear at the top of the hierarchy and the number and kinds of entries that should appear below those parent entries.

The most basic form of a branch definition is simply:

```
branch: dc=example,dc=com
```

This specifies that an entry "dc=example,dc=com" should be created. The entry will look like:

```
dn: dc=example,dc=com
objectclass: top
objectclass: domain
dc: example
```

The basic structure of the entry is defined by the RDN attribute of **dc** specified in the DN of the branch definition. MakeLDIF automatically associates the **dc** RDN attribute with the **domain** objectclass. It has similar default definitions for other common RDN attributes in branch entries:

- **o** -- Creates an entry with the **organization** objectclass
- **ou** -- Creates an entry with the **organizationalUnit** objectclass
- **c** -- Creates an entry with the **country** objectclass
- **l** -- Creates an entry with the **locality** objectclass

In addition, possible to use any other kind of RDN attribute for a branch entry. For branch entries with an RDN attribute other than one specified above, the entry will be created with the **extensibleObject** objectclass.

The branch definition provided above will not cause any additional entries to be created below that branch entry. In order to do that, it is necessary to specify one or more **subordinateTemplate** lines. For example:

```
branch: ou=People,dc=example,dc=com
subordinateTemplate: person:1000
```

This will cause the "ou=People,dc=example,dc=com" entry to be created and then 1000 other entries created below that branch modeled after the **person** template. The **person** template should be defined later in the template file, and the format used to do so will be provided in the next section.

Branch entries are not limited to just one **subordinateTemplate** definition. It is possible to specify multiple **subordinateTemplate** definitions by simply including them on separate lines of the branch definition. The following example will create 1000 entries based on the **person** template and an additional 100 entries based on the **certificatePerson** template:

```
branch: ou=People,dc=example,dc=com
subordinateTemplate: person:1000
subordinateTemplate: certificatePerson: 100
```

In all cases above, the branch entries themselves will contain only the DN, the two objectclass values, and the RDN attribute. It is possible to include any other attributes you wish in the branch entry by simply including them in the branch definition in the template file. For example:



```
branch: dc=example,dc=com
description: This is the description.
```

will create the entry:

```
dn: dc=example,dc=com
objectclass: top
objectclass: domain
dc: example
description: This is the description.
```

This additional text is static, and in fact unlike template definitions that may occur later in the template file, branch definitions are not dynamically parsed. However, it is possible to use global replacement variables in branch definitions to make it somewhat more manageable. For example, if the top of the template file looks like:

```
define      suffix=dc=example,dc=com

branch: [suffix]

branch: ou=People,[suffix]
subordinateTemplate: 1000
```

then the following entries will be created in the LDIF file with 1000 additional entries below them based on the `person` template:

```
dn: dc=example,dc=com
objectclass: top
objectclass: domain
dc: example

dn: ou=People,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
ou: People
```

## Template Definitions

The heart of the MakeLDIF template file are the actual template definitions. Template definitions define the structure of the entries that will be generated. They specify the set of attributes to include in the entries and the types of values that those attributes should contain. The specification of the values is handled through the use of tags that will be parsed by MakeLDIF and replaced with the appropriate values for those tags.

A sample template definition might look like:

```

template: person
rdnAttr: uid
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
givenName: <first>
sn: <last>
cn: {givenName} {sn}
initials: {givenName:1}{sn:1}
uid: {givenName}.{sn}
mail: {uid}@[maildomain]
userPassword: <random:alphanumeric:8>
telephoneNumber: <random:telephone>
homePhone: <random:telephone>
pager: <random:telephone>
mobile: <random:telephone>
employeeNumber: <sequential:100000>
street: <random:numeric:5> <file:streets> Street
l: <file:cities>
st: <file:states>
postalCode: <random:numeric:5>
postalAddress: {cn}${street}${l}, {st} {postalCode}
description: This is the description for {cn}.

```

The actual tags that may be included in a template definition will be discussed in a later section. However, this example does illustrate some of the flexibility that MakeLDIF offers when generating LDIF data.

At the top of the template definition are two lines that provide information about the template itself and are not actually included in entries created from this template. The first line specifies the name of the template. This is the name that is referenced in the `subordinateTemplate` lines of the branch definition. The second line specifies the name of the attribute that should be used as the RDN attribute for the entry. This attribute must be assigned a value lower in the template definition, and the way in which the value is assigned must ensure that the value will be unique.

Note that it is possible to use multivalued RDNs by separating the attribute names with a plus sign, like:

```
rdnAttr: uid+employeeNumber
```

If multivalued RDNs are used, then the combination of RDN attribute values must be unique, but it is possible for one or more of the attributes in the RDN to be non-unique as long as the combination is never duplicated.

In addition to the `template` and `rdnAttr` lines, it is also possible to include one or more `subordinateTemplate` lines. This makes it possible to include dynamically-generated

entries below other entries that have been dynamically generated (e.g., if each user entry has one or more entries below it), and can allow for some rather complex hierarchies. While there is no limit placed on this level of nesting, it is important to ensure that no recursive loops are created by having a `subordinateTemplate` that either directly or indirectly will create additional entries using the same template.

Template definitions also support the concept of inheritance through the use of the `extends` keyword. For example, entries generated from the template definition:

```
template: certificatePerson
rdnAttr: uid
extends: person
userCertificate;binary:: <random:base64:1000>
```

will include all of the attributes defined in the person template as well as `userCertificate;binary` with the specified format. Multiple inheritance is allowed (by including multiple lines with the `extends` keyword), but as with the `subordinateTemplate` keyword it is important not to create a recursive loop in which a template could either directly or indirectly extend itself.

## Template File Tags

In order to ensure that MakeLDIF provides the ability to generate LDIF files that may be used to simulate a wide variety of deployments, a large number of tags have been defined for use in template definitions. This section describes the standard set of tags that may be used in a MakeLDIF template file. It is possible to use custom tags in template files as well, but the process for developing custom tags is defined in a later section.

### Standard Replacement Tags

The tags that may be used in a template file include:

- `<presence:{percent}>` -- Indicates how likely the associated attribute value is to be included in any given entry generated from this template. The value specified for `{percent}` should be an integer between 0 and 100, inclusive. This should only be used with attributes that are not required by the objectclasses used in the entry, and there should be something else included in the value of the attribute that will be present in entries that are chosen to include this attribute value. The `presence` tag itself is replaced with an empty string.

- `<ifpresent:{attribute}>` -- Indicates that this attribute value is to be included in an entry only if the entry contains one or more values for attribute `{attribute}`. Note that if this feature is used, then `{attribute}` must be assigned a value in the template before the line that checks for its presence. The `ifpresent` tag itself is replaced with an empty string.
- `<ifpresent:{attribute}:{value}>` -- Indicates that this attribute value is to be included in an entry only if the entry contains attribute `{attribute}` with a value of `{value}`. Note that if this feature is used, then `{attribute}` must be assigned a value in the template before the line that checks for its presence. The `ifpresent` tag itself is replaced with an empty string.
- `<ifabsent:{attribute}>` -- Indicates that this attribute value is to be included in an entry only if the entry does not contain any values for attribute `{attribute}`. Note that if this feature is used, then `{attribute}` must be assigned a value in the template before the line that checks for its presence. The `ifabsent` tag itself is replaced with an empty string.
- `<ifabsent:{attribute}:{value}>` -- Indicates that this attribute value is to be included in an entry only if the entry does not contain attribute `{attribute}` with a value of `{value}`. Note that if this feature is used, then `{attribute}` must be assigned a value in the template before the line that checks for its presence. The `ifabsent` tag itself is replaced with an empty string.
- `<first>` -- Replaces the tag with a value from the first name file. If both a first name and a last name are included in an entry, then the combination of the first and last name is guaranteed to be unique. This is, no two entries in the same LDIF file will have the same combination of first and last name values. Note that in order to guarantee this, it is necessary to ensure that the first name file does not contain any duplicate values, the last name file does not contain any duplicate values, and the first and last name values are used in their entirety (i.e., you cannot use the substring feature of the attribute value replacements of the form `{givenName:5}` discussed below).
- `<last>` -- Replaces the tag with a value from the last name file. If both a first name and a last name are included in an entry, then the combination of the first and last name is guaranteed to be unique. This is, no two entries in the same LDIF file will have the same combination of first and last name values. Note that in order to guarantee this, it is necessary to ensure that the first name file does not contain any duplicate values, the last name file does not contain any duplicate values, and the first and last name values are used in their entirety (i.e., you cannot use the substring feature of the attribute value replacements of the form `{sn:5}` discussed below).

- **<dn>** -- Replaces the tag with the distinguished name (DN) of the current entry. Note that in order for this to work properly, the RDN attribute for the entry must be assigned a value on an earlier line of the template.
- **<parentdn>** -- Replaces the tag with the DN of the parent entry.
- **<ancestordn:{depth}>** -- Replaces the tag with the DN of the entry's ancestor at the specified depth. A depth of 1 will return the DN of the entry's immediate parent, a depth of 2 will return the DN of the entry's grandparent, and so on. If the entry does not have an ancestor at the specified depth, then the "**<ancestordn:{depth}>**" tag will be replaced with an empty string.
- **<parent:{attr}>** -- Replaces the tag with the value of the specified attribute from the parent entry, provided that the parent entry was generated using a template rather than a branch. If the parent does not have any values for the specified attribute, or if information about the contents of the parent entry are not available, then this tag will be replaced with an empty string. If the parent has multiple values for the specified attribute, then the first value will be used.
- **<csvfield:{fieldName}>** -- Replaces the tag with the specified field from the CSV or delimited text file. The first field in the CSV file will be field number zero. Note that if this tag is used, the "**-c {filename}**" argument must be provided on the MakeLDIF command line to specify the path to the data file.
- **<exec:{command}>** -- Replaces the tag with the information sent to standard output when the command *{command}* is executed on the system. Note that because this requires a separate process to be invoked for each entry created using this template, using this tag can make the LDIF generation process proceed much more slowly than if the **exec** tag is not used.
- **<exec:{command},{arg1},{arg2},...,{argN}>** -- Replaces the tag with the information sent to standard output when the command *{command}* is executed on the system with the provided set of arguments. Note that because this requires a separate process to be invoked for each entry created using this template, using this tag can make the LDIF generation process proceed much more slowly than if the **exec** tag is not used.
- **<random:chars:{characters}:{length}>** -- Replaces the tag with *{length}* characters from the character set *{characters}*. The character set *{characters}* can contain any character other than the colon.

- `<random:chars:{characters}:{minLength}:{maxLength}>` -- Replaces the tag with between `{minLength}` and `{maxLength}` (inclusive) characters from the character set `{characters}`. The character set `{characters}` can contain any character other than the colon.
- `<random:alpha:{length}>` -- Replaces the tag with a string of `{length}` randomly-chosen alphabetic characters.
- `<random:alpha:{minLength}:{maxLength}>` -- Replaces the tag with a string of between `{minLength}` and `{maxLength}` (inclusive) randomly-chosen alphabetic characters.
- `<random:numeric:{length}>` -- Replaces the tag with a string of `{length}` randomly-chosen numeric digits.
- `<random:numeric:{minValue}:{maxValue}>` -- Replaces the tag with an integer value between `{minValue}` and `{maxValue}` (inclusive). Note that the integer value will not be padded with leading zeroes, so if that is desired then the "`<random:numeric:{minValue}:{maxValue}:{minLength}>`" tag should be used.
- `<random:numeric:{minValue}:{maxValue}:{minLength}>` -- Replaces the tag with an integer value between `{minValue}` and `{maxValue}` (inclusive). If the integer value chosen contains less than `{minLength}` digits, then it will be padded with leading zeroes to the required minimum length.
- `<random:alphanumeric:{length}>` -- Replaces the tag with `{length}` randomly-chosen alphanumeric characters.
- `<random:alphanumeric:{minLength}:{maxLength}>` -- Replaces the tag with between `{minLength}` and `{maxLength}` (inclusive) randomly-chosen alphanumeric characters.
- `<random:hex:{length}>` -- Replaces the tag with `{length}` randomly-chosen hexadecimal digits.
- `<random:hex:{minLength}:{maxLength}>` -- Replaces the tag with between `{minLength}` and `{maxLength}` (inclusive) randomly-chosen hexadecimal digits.
- `<random:base64:{length}>` -- Replaces the tag with `{length}` randomly-chosen characters from the base64 character set. Note that if `{length}` is not a multiple of 4, then the generated value will be padded with equal signs so that the total length is a multiple of 4 as per the base64 specification.
- `<random:base64:{minLength}:{maxLength}>` -- Replaces the tag with between `{minLength}` and `{maxLength}` (inclusive) randomly-chosen characters from the base64 character set.

Note that if the selected length is not a multiple of 4, then the generated value will be padded with equal signs so that the total length is a multiple of 4 as per the base64 specification.

- `<random:telephone>` -- Replaces the tag with a string of randomly-chosen numeric digits in the form "123-456-7890". This uses a US-style telephone number, but it is possible to generate telephone numbers in other formats by combining other kinds of tags (e.g., to generate a telephone number in the UK format, you could use "+44<random:numeric:4> <random:numeric:6>").
- `<random:month>` -- Replaces the tag with the name of a randomly-chosen month.
- `<random:month:{length}>` -- Replaces the tag with the first *{length}* characters from the name of a randomly-chosen month.
- `<guid>` -- Replaces the tag with a GUID (globally-unique identifier) value containing hexadecimal digits in the form "12345678-90ab-cdef-1234-567890abcdef". GUID values should be unique within the same LDIF file.
- `<sequential>` -- Replaces the tag with a sequentially-increasing numeric value. The first entry generated using this tag will have a value of 0, the second a value of 1, and so on. Note that sequential counters are maintained on a per-attribute and per-template basis, so it is possible to use multiple sequential counters in different attributes of the same entry without impacting each other, and it is also possible to use sequential counters for the same attribute in different templates without impacting each other. However, it is not possible to use multiple sequential counters for the same attribute in the same template without them impacting each other.
- `<sequential:{initial}>` -- Replaces the tag with a sequentially-increasing numeric value, starting at specified initial value *{initial}*. The first entry generated using this tag will have a value of *{initial}*, the second a value of *{initial}*+1, and so on. Note that sequential counters are maintained on a per-attribute and per-template basis, so it is possible to use multiple sequential counters in different attributes of the same entry without impacting each other, and it is also possible to use sequential counters for the same attribute in different templates without impacting each other. However, it is not possible to use multiple sequential counters for the same attribute in the same template without them impacting each other.
- `<list:{value1},{value2},...,{valueN}>` -- Replaces the tag with a randomly-chosen value from the provided comma-delimited list. Each value in the list provided will have an equal chance of being selected.

- **<list:{value1}:{weight1},{value2}:{weight2},...,{valueN}:{weightN}>** -- Replaces the tag with a randomly-chosen value from the provided comma-delimited weighted list. The weight associated with each list item determines how likely that value is to be chosen. A list item with a weight of 2 is twice as likely to be chosen as an item with a weight of 1. The weights specified must be positive integers.
- **<file:{filename}>** -- Replaces the tag with a randomly-chosen value from the specified file. There should be one value per line of the file. It is not possible to assign weights to the values in the file, but a value can be weighted artificially by including it in the file multiple times. For example, a value that appears in the file three times will be three times as likely to be chosen as a value that appears only once.
- **<base64:{value}>** -- Replaces the tag with the base64-encoded representation of *{value}*. The value *{value}* will be converted into a byte array using the UTF-8 character set, and then that byte array will be base64 encoded.
- **<base64:{charset}:{value}>** -- Replaces the tag with the base64-encoded representation of *{value}*. The value *{value}* will be converted into a byte array using the rules of the Java character set *{charset}*, and then that byte array will be base64-encoded. See the Java API documentation to determine the names of the character sets that may be used.
- **<loop:{lowerBound}:{upperBound}>** -- Creates  $(\{upperBound\} - \{lowerBound\} + 1)$  copies of this line with this tag replaced in each copy with a sequentially-incrementing number starting at *{lowerBound}* in the first copy,  $(\{lowerBound\} + 1)$  in the second copy, and so on, so that in the last copy this tag will be replaced with *{upperBound}*. Note that it is possible to include multiple **loop** tags on the same line (even with different *{lowerBound}* values), but only the first tag will be used to determine the number of copies to create.
- **<custom:{class}>** -- Replaces the tag with the value generated by invoking the custom tag whose implementation is provided in the class *{class}*. The provided class name *{class}* must be fully-qualified (i.e., including the package name if applicable), and that class must exist in the classpath used to run MakeLDIF.
- **<custom:{class}:{arg1},{arg2},...,{argN}>** -- Replaces the tag with the value generated by invoking the custom tag whose implementation is provided in the class *{class}* with the specified argument list. The provided class name *{class}* must be fully-qualified (i.e., including the package name if applicable), and that class must exist in the classpath used to run MakeLDIF.



## Attribute Value Reference Tags

In addition to the standard replacement tags listed above, it is possible to use tags that reference the values of other attributes in the same entry. These tags are called attribute value reference tags and they may be used by simply enclosing the name of the desired attribute in curly braces. When these tags are encountered in the template, they will be replaced with the value of the specified attribute. If the specified attribute has not been assigned a value for the current entry, then this tag will be replaced with an empty string.

For example, consider the following excerpt from a template:

```
givenName: <first>
sn: <last>
uid: {givenName}.{sn}
cn: {givenName} {sn}
mail: {uid}@example.com
```

If the value chosen for the first name is "John" and the last name is "Doe", then the following LDIF output would result:

```
givenName: John
sn: Doe
uid: John.Doe
cn: John Doe
mail: John.Doe@example.com
```

Note that in order for this to work properly, it is necessary to assign a value to an attribute before it may be referenced in this manner. If, for example, the definition of the `mail` attribute had appeared before the definition of the `uid` attribute, then the resulting e-mail address would have been "`@example.com`" because the "`{uid}`" tag would not have a value and therefore would have been replaced with an empty string.

It is also possible to place a colon after the name of the attribute followed by a positive integer value *{length}*. This will cause at most the first *{length}* characters of the value from the specified attribute to be used instead of the entire value. For example, the template excerpt:

```
givenName: <first>
sn: <last>
initials: {givenName:1}{sn:1}
```

would produce the following LDIF for a first name of "John" and a last name of "Doe":

```
givenName: John
sn: Doe
initials: JD
```

If the specified *{length}* is longer than the value of the named attribute, then the entire value of the attribute will be used and no padding will be added.

## Tag Evaluation Order

The order in which tags are evaluated while parsing a template definition is important because changing the evaluation order can change the way that the output is produced. Although it is not possible for the end user to change this evaluation order, it is important to understand how template definitions are processed so that LDIF files are generated in the appropriate manner.

In most cases, it is not possible to nest tags. That is, the output of one tag cannot be used as input to another. For example, consider the following:

```
description: <random:alpha:<random:numeric:5:10>>
```

One might hope that this would first evaluate the "<random:numeric:5:10>" tag to create a numeric value between 5 and 10 (for example, 7) and then evaluate the outer tag as "<random:alpha:7>". However, this is not the case. Because of the way that MakeLDIF parses template definitions, this will fail because it will try to interpret "<random" as an integer value. Fortunately, this is not a problem in most cases because enough variations of the standard replacement tags have been provided to deal with this. For example, to achieve the desired result attempted by the above template line, you can instead use:

```
description: <random:alpha:5:10>
```

For the purposes of this discussion, MakeLDIF parses template files in the following order:

1. All standard replacement tags other than **base64**, **custom**, and **loop**.
2. All attribute value reference tags.
3. The **base64** standard replacement tags ("**<base64:{value}>**" and "**<base64:{charset}:{value}>**").
4. The **custom** standard replacement tags ("**<custom:{class}>**" and "**<custom:{class}:{arg1},{arg2},...,{argN}>**").
5. The **loop** standard replacement tag ("**<loop:{lowerBound}:{upperBound}>**").

Based on this order of operations, any tag at one level may be used as input to any tag at a higher level. For example, the following is legal because `base64` standard replacement tags are evaluated after attribute value reference tags:

```
description:: <base64:{givenName}>
```

## Defining Custom Tags

One of the benefits of MakeLDIF is the large number of tags that it provides that can be used to customize the process of creating LDIF files. However, even with the large set of tags provided, it may be desirable to extend the functionality even further. This is possible through the use of custom tags.

Custom tags can be easily defined by creating a Java class that extends the abstract `CustomTag` class. This class defines three methods:

- `public void initialize()` -- Provides the ability to perform any one-time initialization that may be necessary when this tag is first created. This is an optional step, and by default no initialization is performed.
- `public void reinitialize()` -- Provides the ability to perform additional initialization every time the template is used to start processing a new branch. This is also optional, and by default no reinitialization is performed.
- `public String generateOutput(String[] tagArguments)` -- Generates the output that should appear whenever this custom tag is invoked in the template. The provided set of arguments can be used to customize the output as necessary. This method must be implemented in all custom tags.

## A Sample Custom Tag Implementation

To demonstrate the ease with which a custom tag may be implemented for use with MakeLDIF, this section provides an implementation for a simple custom tag that can be used to multiple integer values together. The integer values to be added will be provided as arguments to the custom tag, and that set of integer values is the only information necessary to perform this task. Therefore, only the `generateOutput` method needs a real implementation.

This custom tag may be implemented as follows:

```
/**
```

```

* This class provides an implementation of a MakeLDIF custom tag that will be
* used to calculate the sum of all the integer values provided as arguments
* to the tag.
*/
public class SumCustomTag
    extends CustomTag
{
    /**
     * Performs any necessary one-time initialization that should be performed
     * when this custom tag is first created. In this case, no initialization
     * is performed.
     */
    public void initialize()
    {
        // No implementation required.
    }

    /**
     * Performs any initialization that should be performed each time the LDIF
     * generation starts working on a new branch (e.g., to reset any internal
     * variables that might have been in use). In this case, no
     * reinitialization is performed.
     */
    public void reinitialize()
    {
        // No implementation required.
    }

    /**
     * Parses the list of arguments, converts the values to integers, and totals
     * those values.
     *
     * @param tagArguments The arguments containing the numeric values to be
     *                     totaled.
     *
     * @return The string representation of the total of all the argument
     *         values.
     */
    public String generateOutput(String[] tagArguments)
    {
        int sum = 0;

        for (int i=0; i < tagArguments.length; i++)
        {
            sum += Integer.parseInt(tagArguments[i]);
        }

        return String.valueOf(sum);
    }
}

```

## Using the Example Custom Tag

Once the custom tag class has been implemented and compiles properly, it may be used by specifying that class in a `custom` tag in the template file. For example, to use the custom tag we just implemented, then something like the following could be placed in the template file:

```
cn: <custom:SumCustomTag:1,2>
```

When MakeLDIF is run using a template that contains this definition, the `generateOutput` method of the `SumCustomTag` class will be invoked and it will produce output containing:

```
cn: 3
```

Because static values were provided as arguments to the tag, the output will be exactly the same every time. This is not all that useful in this case, but it doesn't have to be that way. As indicated earlier, custom tags are parsed after most other kinds of tags. This means that it is possible to use something like:

```
cn: <custom:SumCustomTag:<random:numeric:5>,<random:numeric:5>>
```

to use the output from other tags as arguments to the custom tag. In this case, each "`<random:numeric:5>`" tag will generate a 5-digit integer and then the custom tag will add those values together. Unlike the previous example, the output in this case may be different for each entry because the arguments are randomly-chosen values. Of course, even then this particular custom tag is not all that useful, but it is a simple example that can be used as the foundation for creating more useful custom tags for real-world purposes.

# TCPCapture and TCPReplay

The TCPCapture and TCPReplay tools provide a simple means of recording TCP network communication so that it may be replayed later in an attempt to generate the same load. It is useful for certain types of load which may be generated by reproducing exactly the same network traffic.

## Running TCPCapture

The TCPCapture tool serves as a simple proxy that can record TCP traffic as it passes through to the target server. It records only the client side of the communication so that it may be replayed later using the TCPReplay tool, or using the TCP Replay job.

To start the TCPCapture tool, use the following command:

```
$ tools/tcp-capture.sh {options}
```

where *{options}* should include the following:

- **-l** *{port}* -- Specifies the port on which the TCPCapture utility should accept connections from clients.
- **-h** *{address}* -- Specifies the address of the server to which captured data should be forwarded.
- **-p** *{port}* -- Specifies the port of the server to which captured data should be forwarded.
- **-o** *{file}* -- Specifies the output file to which the captured data is to be written.

This utility will capture data and write it to the capture file until it is interrupted (e.g., using Ctrl +C). Also, note that while the TCPCapture utility will forward communication in both directions between the client and the backend server, it will only actually record the communication from the client since that is the only data that will need to be replayed.

# Running TCPReplay

The TCPReplay tool provides a means of replaying requests from a client to generate load against a server. The data to replay should be provided in a capture file that was obtained using the TCPCapture utility.

To start the TCPReplay tool, use the following command:

```
$ tools/tcp-replay.sh {options}
```

where *{options}* should include the following:

- **-h** *{address}* -- Specifies the address of the server to which the data should be replayed.
- **-p** *{port}* -- Specifies the port of the server to which the data is to be replayed.
- **-i** *{path}* -- Specifies the path to the capture file containing the data to be replayed.
- **-P** -- Indicates that the TCPReplay tool should attempt to preserve the original timing between the requests captured. By default, a fixed delay will be inserted between each packet.
- **-m** *{multiplier}* -- Specifies a floating-point value that should be used as the multiplier to apply to the original timing. This is only applicable if the "-P" argument is also provided. Values less than one will cause the tool to attempt to replay operations faster than the original timing (e.g., a value of 0.5 will wait half as long between request packets, so it should go about twice as fast), while values greater than one will cause the replay to be slower than the original communication. By default, a value of 1.0 will be used.
- **-d** *{delay}* -- Specifies a fixed delay (in milliseconds) to use between each packet that is replayed. By default, no fixed delay will be inserted.
- **-I** *{count}* -- Specifies the number of times that the entire data set should be replayed. By default, it will be replayed once.
- **-D** *{delay}* -- Specifies the delay (in milliseconds) that should be inserted between consecutive replays of the data set. By default, no delay will be inserted.
- **-T** *{duration}* -- Specifies the maximum length of time (in milliseconds) that the replay should be allowed to take. By default, no maximum time limit will be enforced.
- **-t** *{count}* -- Specifies the number of threads to use to replay the data. Each thread will replay the same communication independently on separate connections to the same server. By default, a single thread will be used.

# ThroughputTest

The ThroughputTest utility may be used in an attempt to measure the network bandwidth between two systems. It consists of client and server processes, and when the client establishes a connection to the server, the server will send data to the client as quickly as possible. The client will measure the amount of data that was transferred and the length of time required to do so and will report the overall rate of data transfer. Note that this tool will measure the rate at which actual data may be sent, and does not include TCP, IP, and other network headers.

## The ThroughputTest Server

The following command may be used to start the ThroughputTest server:

```
$ tools/throughput-test-server.sh {options}
```

where *{options}* may include:

- **-p {port}** -- Specifies the port on which the ThroughputTest server will listen for client connections. By default, it will listen on port 3333.
- **-b {size}** -- Specifies the size in bytes of the buffer that should be used when sending data to the client. The default buffer size is 8192 bytes.
- **-N** -- Indicates that the TCP\_NODELAY socket option should be used when sending data to the client so that data from the server is flushed immediately to the client rather than using Nagle's algorithm to wait for a brief period to see if more data might be sent and could be included in the same packet.

## The ThroughputTest Client

The following command may be used to start the ThroughputTest client:

```
$ tools/throughput-test-client.sh {options}
```

where *{options}* may include:



- **-h** *{address}* -- Specifies the address of the ThroughputTest server.
- **-p** *{port}* -- Specifies the port number for the ThroughputTest server.
- **-b** *{size}* -- Specifies the total amount of data in bytes that should be transferred from the server to the client. If this is not provided, then there will be no limit on the amount of data transferred.
- **-d** *{time}* -- Specifies the length of time in seconds that the test should be allowed to run. If this is not provided, then no time limit will be enforced.
- **-B** *{size}* -- Specifies the size in bytes of the buffer to use when sending data from the server. The default buffer size is 8192 bytes.

Either the "**-b** *{size}*" or the "**-d** *{time}*" option must be specified in order to indicate when the client should stop running. If both are provided, then client will stop as soon as either limit has been reached.

When the test has completed, the client will report the network throughput measured using multiple units (bits, bytes, kilobits, kilobytes, megabits, and megabytes per second). For example:

```
$ java -jar ThroughputTestClient.jar -h server.example.com -d 60
Read 4038716240 bytes in 60008 milliseconds
Bits per second:      538495498.667
Bytes per Second:     67311937.333
Kilobits per Second:  525874.510
Kilobytes per Second: 65734.314
Megabits per Second:  513.549
Megabytes per Second: 64.194
```

# UDPPing

The UDPPing utility provides a mechanism for measuring network latency between two hosts. The client accomplishes this by sending UDP messages to the echo service (or an instance of the UDPPing server) on the server system and tracking the length of time required to receive the response. It is most accurate when there is significant latency, as when the client and server are separated by a WAN, or are communicating over the internet.

## Running the UDPPing Client

The UDPPing client may be run using the following command:

```
$ tools/udp-ping.sh {options}
```

where *{options}* may include:

- **-h** *{address}* -- Specifies the address of the server to which to send the requests.
- **-p** *{port}* -- Specifies the port of the server on which to send the requests. By default, it will use UDP port 7, which is the port used by the standard echo service.
- **-c** *{count}* -- Specifies the total number of ping requests to send. By default, there will be no limit.
- **-i** *{interval}* -- Specifies the interval in milliseconds between requests. By default, an interval of 1000 milliseconds will be used.
- **-s** *{size}* -- Specifies the size in bytes of the UDP datagrams used in the ping requests. This may not be less than 12 bytes. By default, a value of 64 bytes will be used.
- **-t** *{timeout}* -- Specifies the maximum length of time in milliseconds to wait for a response to a ping requests before assuming that either the request or response packet was lost. By default, a timeout of 1000 milliseconds will be used.
- **-R** -- Indicates that the UDPPing client should use a high-resolution timer to increase the accuracy of the measurements.

## Running the UDPPing Server

If the UDP-based echo service is running on the target server, then the UDPPing client can use it to measure network latency. However, if this service is not enabled, then you may run the UDPPing server on the target system to achieve the same result. It may be run using the following command:

```
$ tools/udp-ping-server {options}
```

where *{options}* may include:

- **-p {port}** -- Specifies the port on which to listen for requests. By default, it will listen on port 7777.
- **-v** -- Indicates that the server should operate in verbose mode.

# SLAMD License

The SLAMD Distributed Load Generation Engine (SLAMD) is distributed under the Sun Public License Version 1.0. This is an OSI-approved Open Source license, and more information about such licenses may be found at <http://www.opensource.org/>. The full text of the Sun Public License Version 1.0 is provided below for reference.

SUN PUBLIC LICENSE Version 1.0

## 1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof and corresponding documentation released with the source code.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated documentation, interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

## 2. Source Code License.

### 2.1 The Initial Developer Grant.

The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused

by:

i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

## 2.2. Contributor Grant.

Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

## 3. Distribution Obligations.

### 3.1. Application of License.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

### 3.2. Availability of Source Code.

Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

### 3.3. Description of Modifications.

You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

### 3.4. Intellectual Property Matters.

#### (a) Third Party Claims.

If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.

If Contributor's Modifications include an application programming interface ("API") and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.

Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

### 3.5. Required Notices.

You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

### 3.6. Distribution of Executable Versions.

You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ



from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

### 3.7. Larger Works.

You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

### 4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

### 5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

### 6. Versions of the License.

#### 6.1. New Versions.

Sun Microsystems, Inc. ("Sun") may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

#### 6.2. Effect of New Versions.

Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Sun. No one other than Sun has the right to modify the terms applicable to Covered Code created under this License.

#### 6.3. Derivative Works.

If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must: (a) rename Your license so that the phrases "Sun," "Sun Public License," or "SPL" or any confusingly

similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Sun Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

## 7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

## 8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

(a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b)

and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

#### 9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

#### 10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

#### 11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of,

or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

## 12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

## 13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as "Multiple-Licensed". "Multiple-Licensed" means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.

### Exhibit A -Sun Public License Notice.

The contents of this file are subject to the Sun Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. A copy of the License is available at <http://www.sun.com/>

The Original Code is \_\_\_\_\_. The Initial Developer of the Original Code is \_\_\_\_\_. Portions created by \_\_\_\_\_ are Copyright (C) \_\_\_\_\_. All Rights Reserved.

Contributor(s): \_\_\_\_\_.

Alternatively, the contents of this file may be used under the terms of the \_\_\_\_\_ license (the "[\_\_\_\_\_] License"), in which case the provisions of [\_\_\_\_\_] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [\_\_\_\_\_] License and not to allow others to use your version of this file under the SPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [\_\_\_\_\_] License. If you do not delete the provisions above, a recipient may use your version of this file under either the SPL or the [\_\_\_\_\_] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of

the notices in the Source Code files of the Original Code. You should use the text of this Exhibit A rather than the text found in the Original Code Source Code for Your Modifications.]