HTTP Configuration

This topic explains how to configure message proxies that send and receive messages over HTTP (including HTTPS).

> **Version Compatibility**
>
> HTTP traffic files recorded inVirtualize or SOAtest9.10.x cannot be used with 9.9.x verions and earlier.

In this section:

You will need the host, port, and path of the service you are testing or creating a virtual service forto set up basic HTTP connection options. Use the settings you would normally use to directly message the service.

Connection Settings

To specify basic HTTP connection options, set the following service and listening details in the Proxy Settings Connection tab.

Proxy Settings (incoming)

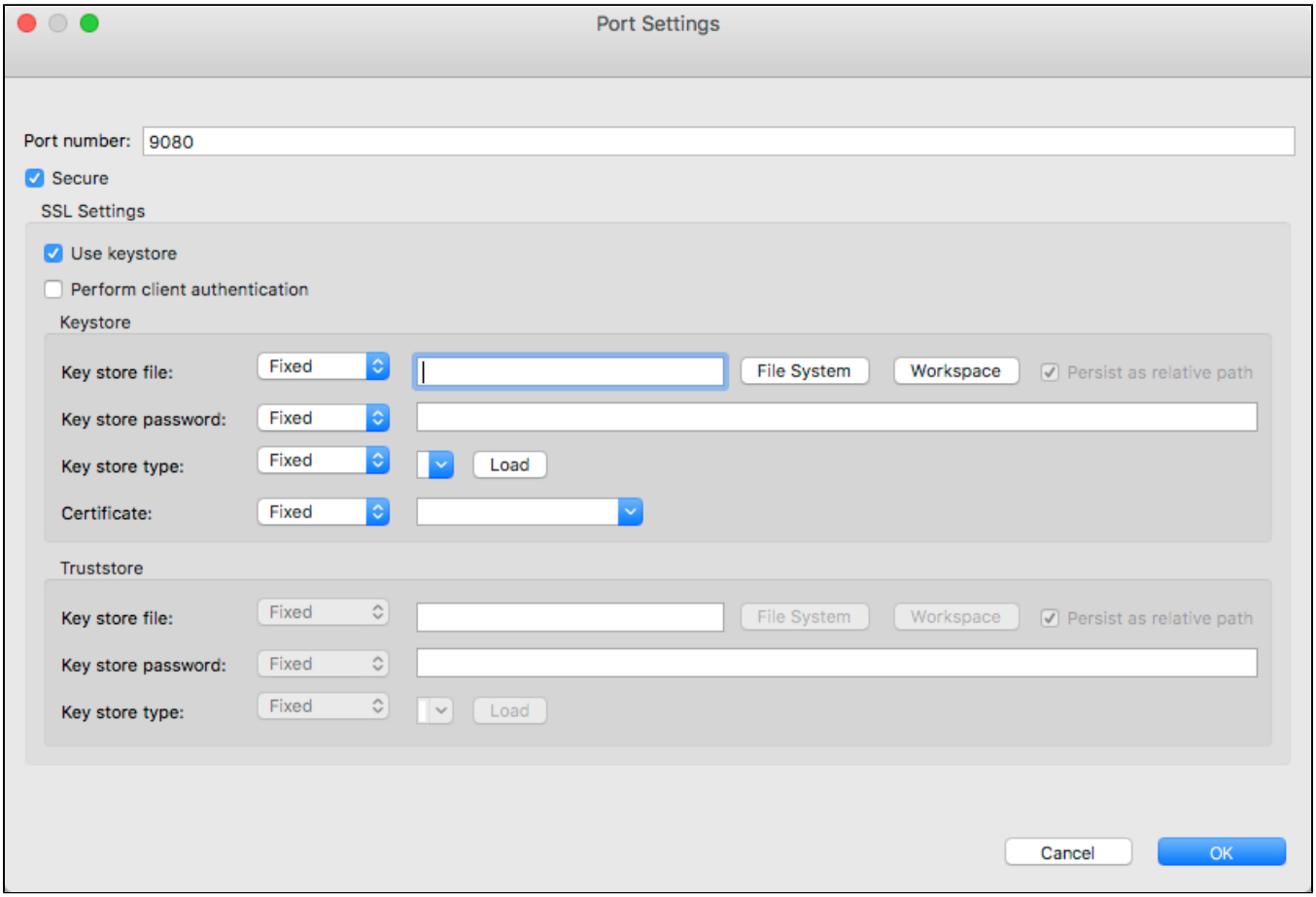
These settings specify where messages from the client should connect in order to communicate with the application under test.

Listener	You can choose an HTTP listener that you defined when creating the proxy (see Creating Proxies) from the drop-down menu or use the default listener. You can add HTTP listeners in these section: 1. Click **New** and specify a name for the listener. 2. Click **Add Port** and enter a port number. You can enable the message proxy to automatically assign a port by specifying `0` as the port number. When the proxy is enabled, the assigned port number will appear in the console. The port is randomly assigned every time the message proxy is changed/enabled. You can also send a GET request to the `messageProxies` API endpoint to return the automatically-assigned port number. See Testing through the REST API for additional information. 3. If the client sends traffic over SSL, enable the **Secure** option and enable your verification options. See SSL Settings for Listener Ports for details. 4. Click **OK** to exit the port editor. 5. Click **Add** to add additional ports to the listener or **OK** again to finish adding the listener.

Proxy listen path	Enter the path where the proxy should listen for incoming connections. No two message proxies can have HTTP connections with the same proxy path or with a path that matches an existing virtual asset's HTTP path. See Service Forward Path and Proxy Listen Path for more details.
Proxy URL	Displays the URL that should be given to the AUT.See Directing AUTs to Proxies for additional information.

SSL Settings for Listener Ports

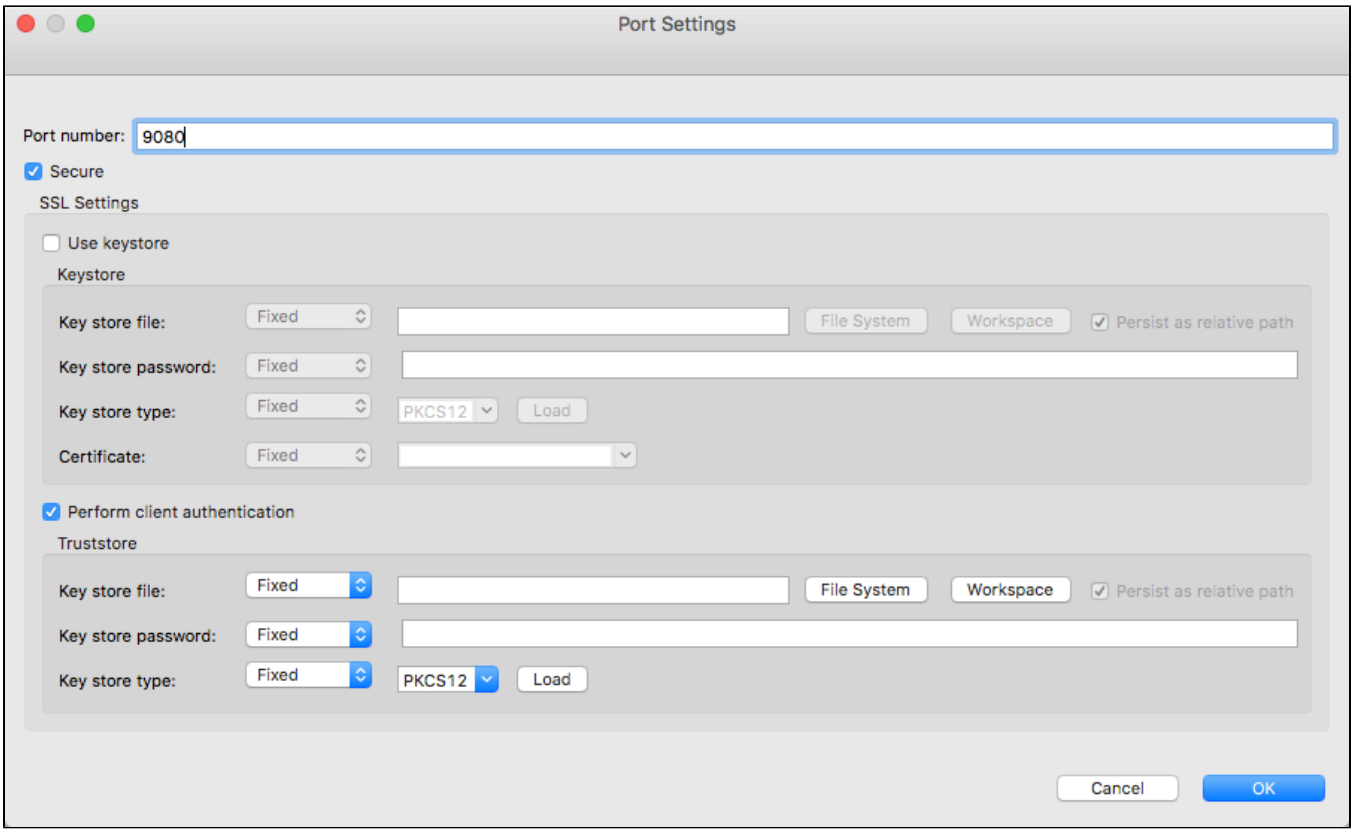
Enable the **Use keystore** option to configure server side SSL settings.



Configure the following settings:

Key store file	Specify the path to the key store file. You can enter the path manually or browse the file system or workspace.
Key store password	Specify the password to access the keystore.
Key store type	Choose a keystore type from the drop-down menu and click **Load**.
Certificate	From the **Certificate** drop-down menu, choose the certificate alias to present to the server. The menu will be empty if there are configuration errors, such as incorrect password or keystore type.

Enable the **Perform client authentication** option to configure client side SSL settings.



Configure the following settings:

Key store file	Specify the path to the key store file. You can enter the path manually or browse the file system or workspace.
Key store password	Specify the password to access the keystore.
Key store type	Choose a keystore type from the drop-down menu and click **Load**.

Primary Connection (outgoing)

Service URL	Contains the full URL for the target application (comprising the service host, service port, and forward path). You can enter a complete URL here and/or edit specific components in the following fields. Updates made in one area will be propagated to the other (e.g., if you modify the port in the URL, the value in the Service port field will be updated automatically).
Service host	Enter the host name of the machine where the service resides. This is the machine to which the proxy will send messages. If you want the proxy to foward to a virtual asset on the local server without consuming an HTTP connection, enter `localhost` or `127.0.0.1` rather than the actual host name.
Service port	Enter the port where the service is listening. This is the port to which the proxy will send messages.
Service forward path	(Optional) Enter the path to which the proxy should forward the messages that it receives. If blank, this defaults to the value in the **Proxy listen path** field. If the HTTP proxy is sending to messages localhost, you must enter a **Service forward** path because the proxy doesn't allow forwarding to itself. If the **Service forward path** sends a redirect, the proxy will follow the redirect and then respond back. It will not pass the redirect back to the client. See Service Forward Path and Proxy Listen Path for more details.

Secondary Connection (outgoing)

Enable the **Use secondary connection if primary fails** option if you want traffic to be redirected to a secondary proxy endpoint when the primary connection fails or the responder is not available. The connection will be considered "failed" if the response status code is 400 level or higher.

For instance, you may specify a secondary endpoint if you want a virtual asset to be used whenever the live endpoint is unavailable. If a virtual asset does not handle a given use case, you may also use a secondary endpoint to redirect traffic to the live endpoint and record the live traffic for creating a new virtual asset to cover that use case.

If this option is not enabled, the primary connection will be used for recording.

If this option is enabled, you can select from the available recording options (described below).

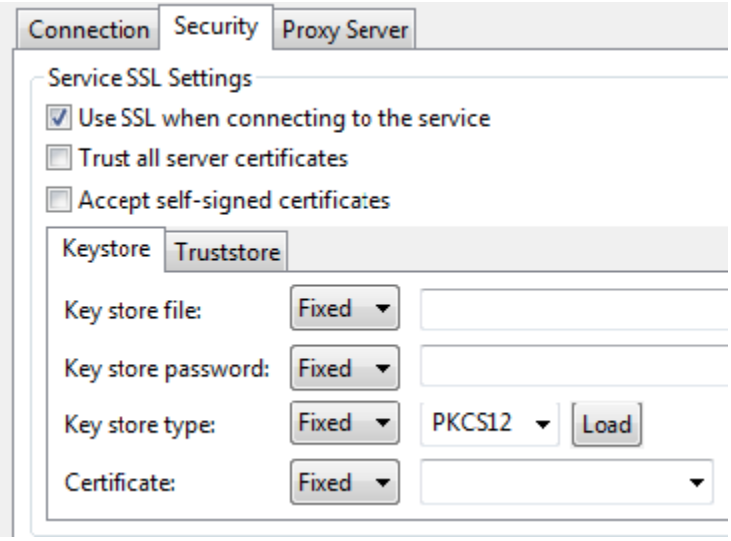
Recording options

These settings determine how to record traffic when a secondary endpoint is specified.

- **Record on both connections:** Records traffic for both the primary and secondary connections. Error messages from the primary will not be recorded; instead the messages will be sent to the secondary and the responses will be recorded. Errors reported by the secondary will be recorded.
- **Record on primary connection only:** Records traffic for the primary connection. Does not record errors.
- **Record on secondary connection only:** Records traffic for the secondary connection, including errors.

Security Tab Settings

The Service SSL section needs to be completed *only if the service you are working with uses SSL*.



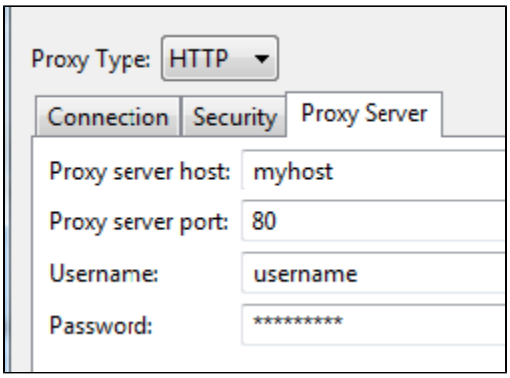
If theservice being virtualized and/or the application under testuses SSL and/or other authentication (basic/digest, Kerberos, NTLM), additional configuration may be required.

For details on completing the Service SSL fields immediately below these fields, see Security Configuration.

Proxy Server Tab Settings

You can specify a server for the message proxy in this tab, which enables you to configure a different proxy server for traffic to and from different message proxies. This configuration provides control over which proxy server handles traffic between the application under test and a specific message proxy.

To specify proxy settings at this level, provide the appropriate details in the proxy's Proxy Server tab.



SSL is not supported.

Service Forward Path and Proxy Listen Path

In the simplest case, you can set **Proxy listen path** to the path of your service and leave **Service forward path** empty. With this configuration, the proxy will automatically forward all messages it receives on that path to the same path at the **Service host** and **Service port**.

If you need the proxy to listen on a different path than the path of your service, set **Service forward path** to the actual path where you want received messages to be sent. The proxy will forward the path and any query parts to the target service.

If the **Proxy listen path** and the **Service forward path** are different, then any segments in the request after the **Proxy listen path** will be appended to the forwarded request. The **Proxy listen path** is essentially being replaced with the **Service forward path** so that the entire path (as received by the proxy) gets sent to the service.

Using Wildcard Characters in Paths

You can use wildcard characters to specify dynamic path segments. For example, configuring the path as `/path/*/service` would enable the following paths to go to the same proxy:

```
/path/1/service

/path/2/service
```

Wildcards can be used to replace an entire path segment. For example:

- `/path/*/service` — valid
- `/path/1*2/service` — not valid

The wildcard can only be used for one segment of the path. A path configured as `/path/*/service` will NOT match `/path/1/2/service`. If you want your path to match both `/path/1/2/service` and `/path/3/4/service`, use the pattern `/path/*/*/service`.

You can use the dynamic segment from the listen path as part of the forward path. There are two ways to do this:

- If you want traffic to be forwarded to the path where the request was received, leave the forward path empty.
- If the forward path is different and the dynamic value needs to be used, configure it using environment variable syntax. For example:
  ```
  listen path: /path/*/service
  forward path: /asset/path/${1}/service
  ```

The dynamic listen path segments represented by the wildcards can be accessed using environment variable syntax. The occurrence of the wildcard is used as the variable name; in other words, the first occurrence of the wildcard is ${1}, the second is ${2}, the third is ${3}, etc. In the following example, ${2} refers to the second occurrence of the wildcard (which was "bank" in the request path):

```
listen path: /path/*/service/*/account
forward path: /asset/path/${2}/service/${1}/account

request path: /path/1/service/bank/account
forward path: /asset/path/bank/service/1/account
```

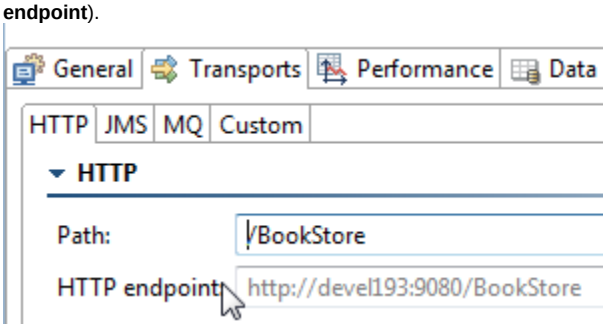
Header Alterations

In most cases, the proxy will pass all headers directly to and from the target service. Some content-length related headers may be changed to suit how the proxy server works. For example, the proxy server doesn't support responding with the "chunked" transfer encoding and will replace "host" headers sent to match its own host name (however, it does support receiving chunked requests and chunked responses from the target service).

Setting the Target Service

A proxy can use a virtual or test asset on a server as its target service.

1. Set the service host and port to that of the Virtualize Server where the virtual asset is deployed. If you want the proxy to foward to a virtual asset on the local Virtualize server without consuming an HTTP connection, enter `localhost` or `127.0.0.1` rather than the actual host name.
2. Set the proxy connection's **Service forward path** to the virtual asset's path (found in the virtual asset's **Transports> HTTP** tab, under **HTTP endpoint**).



Chunking/Unchunking Behavior

The proxy supports receiving chunked requests and chunked responses from the target service. If a service response uses HTTP chunking, proxies will unchunk responses before they are returned to the original caller/AUT.

Additional HTTP Message Proxy Configuration

You can create a properties file to configure additional HTTP listener settings. The properties file allows you to fine-tune HTTP listener performance and specify additional SSL settings.

1. Create a plain text file and specify the properties you want to set (see HTTP Listener Message Proxy Performance Properties and HTTP Listener Message Proxy SSL Properties).
2. Save the file to theVirtualAssets or TestAssets directory in your workspace and name it `embeddedServer.properties`.

The file will be read every time a proxy with HTTP listeners is enabled.

HTTP Listener Message Proxy Properties

embedded.connector. maxHttpHeaderSize	Specify the maximum size of the request and response HTTP header (bytes). If not specified, this attribute is set to `8192` (8 KB).

HTTP Listener Message Proxy Performance Properties

All properties are optional and non-integer values will be ignored.

embedded. connector. maxThreads	Specify the maximum number of request-processing threads the connector should create. This property determines the maximum number of simultaneous requests that can be handled. Default is `200`.
embedded. connector. minThreads	Specify the minimum number of threads that should always be running. Default is `10`.
embedded. connector.acceptors	Specify the number of threads that should be used to accept connections. Increase this value on machines with multiple CPUs or when you are using several non-keep-alive connections. `1` or `2` is adequate in most cases. Default is `1`.
embedded. connector. idleTimeout	Specify how many milliseconds the connector should wait for another HTTP request before closing the connection. Specify `-1` to allow the connector to wait indefinitely. Default is `use connectTimeout`.
embedded. connector. soLingerTime	Specify how many milliseconds the sockets used by the connector should linger when the sockets are closed. Lingering is disabled by default.
embedded. connector. acceptorPriorityDelta	Specify the priority of the acceptor threads used to accept new connections. See the java.lang.Thread JavaDoc for additional details. Default is `5`.

embedded.connector.acceptQueueSize	Specify the maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is at capacity will be refused. Default is `100`.
embedded.connector.connectTimeout	Specify how many milliseconds the connector should wait for the request URI line to be presented after accepting a connection. Enter `-1` to allow the connector to wait indefinitely. Default is `60000`.

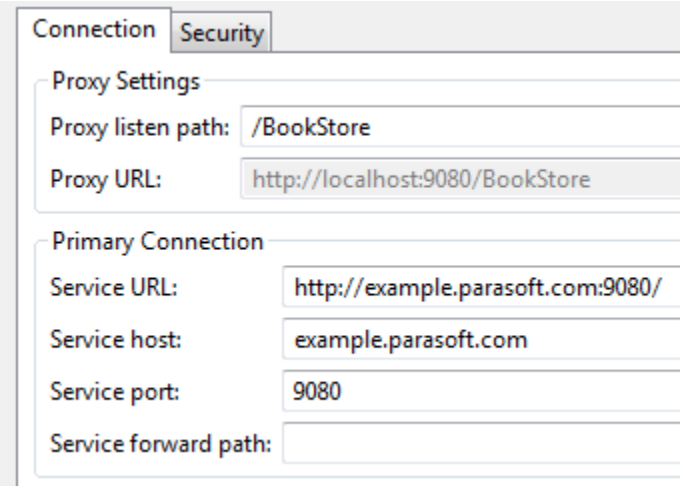
HTTP Listener Message Proxy SSL Properties

embedded.ssl.includeProtocols	Comma-separated list of SSL protocols to support for HTTPS connections. If specified, only the protocols in the list will be supported by the SSL implementation in the JVM. If this property is not set, the protocols supported by the JVM are used (excluding SSLv2 and SSLv3 if the JVM enables either or both of them by default).
embedded.ssl.includeCipherSuites	Comma-separated list of encryption ciphers to support for HTTPS connections. Use the JSSE cipher naming convention to specify the ciphers. If the property is set, only the ciphers in the list will be supported by the SSL implementation. If this property is not set, the JVM's default cipher suites, except for suites that are not considered secure, will be supported. As a result, a very limited set of ciphers will be available by default for older JVMs.
embedded.ssl.useCipherSuitesOrder	Set to `true` to enforce the server's cipher order (from the ciphers setting). Set to `false` to choose the first acceptable cipher suite presented by the client. This feature requires Java 8 or later. Default is `undefined`, which will result in the order being defined by the JSSE implementation.
embedded.ssl.maxCertPathLength	The maximum number of intermediate certificates that will be allowed when validating client certificates. Default is `5`.
embedded.ssl.crlPath	Sets the path to the file containing the certificate revocation list used for verifying client certificates. If not defined, client certificates will not be checked against a certificate revocation list.
embedded.ssl.keyManagerAlgorithm	Sets the certificate encoding algorithm to be used. By default, `KeyManagerFactory.getDefaultAlgorithm()` is used, which returns `SunX509` for Sun JVMs. IBM JVMs return `IbmX509`.
embedded.ssl.trustStoreAlgorithm	Sets the algorithm used for truststore. If not specified, the default value returned by `javax.net.ssl.TrustManagerFactory.getDefaultAlgorithm()` is used.
embedded.ssl.useKeyManagerAlgorithmForTruststore	If set to `true`, the key manager algorithm is used as the truststore algorithm. This takes precedent over the `embedded.ssl.truststoreAlgorithm` property. Default is `false`.

Example: Sending Traffic to a Service

Assume that we want to create a message for a service that is normally accessed at http://example.parasoft.com:9080/BookStore. We could create an HTTP proxy with the settings:

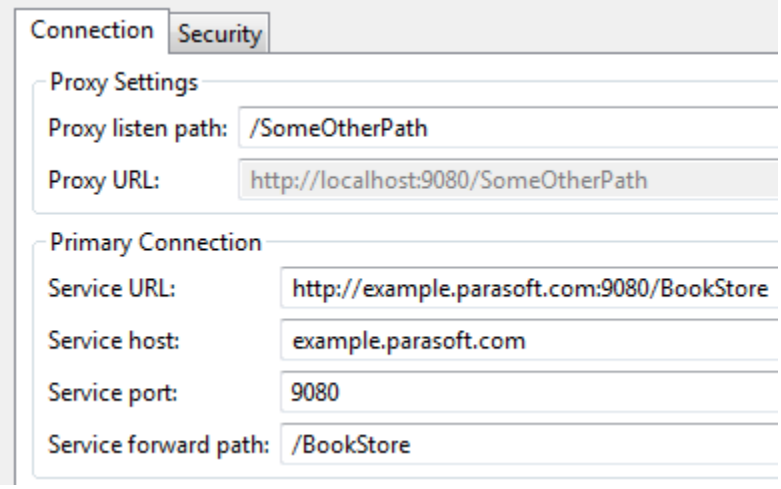
- **Proxy listen path:** /BookStore
- **Service host:**example.parasoft.com
- **Service port:** 9080
- **Service forward path:** [empty]



This configuration will listen on /BookStore and forward all traffic to the actual book store service.

If we wanted to listen on a path other than the service path, we would configure the proxy as follows:

- **Service forward path:** /BookStore
- **Proxy listen path:** /SomeOtherPath



This would route traffic from /SomeOtherPath on the proxy to the actual book store service.

In both cases, all traffic that goes to those paths (including sub paths) is sent to the service. In the second example, traffic sent to /SomeOtherPath /SubPath would be sent to /BookStore/SubPath. Because queries are preserved, /SomeOtherPath?param=value would be sent to /BookStore? param=value.

Forwarding Requests to Services that Require Case-sensitive Header Names

If you are using the desktop's server, your message proxy headers will be lowercased and may fail to function properly when requests are forwarded to a service that requires case-sensitive HTTP header names.

You can create a _global.headers file in the /VirtualAssets/ directory and add the headers using the specific capitalization you want to pass. Right-click the server in the UI and re-deploy all virtual assets to begin using this file in your environment.

You can also address HTTP header capitalization issues for specific message proxies. Create a <proxy-name>.headers file in the /VirtualAssets/ directory and add the headers using the specific capitalization you want to pass for the specific proxy. This file will apply to specific proxy and override the global file. Right-click the server in the UI and re-deploy all virtual assets to begin using this file in your environment.

Example Headers File

```
X-AUTHORIZATION
ANOTHER-HEADER
HEADER3
```

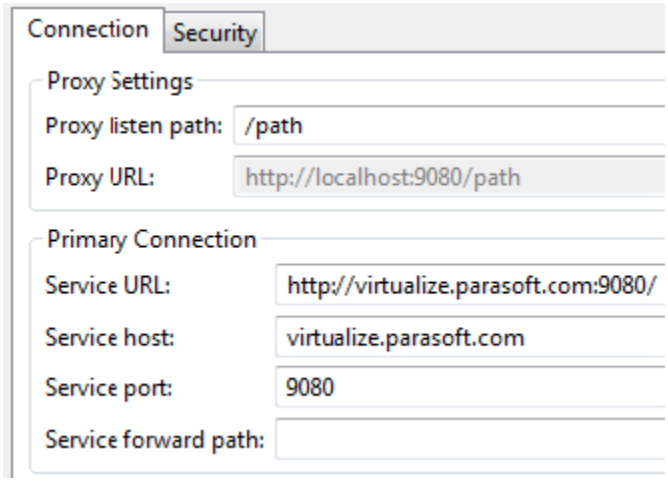
Example: Sending Traffic to a Virtual Asset

A proxy can also send traffic to a virtual asset. In that case, enter the host and port information for the Virtualize Server as if were just another service.

For example, to send traffic to a virtual asset on a remote server, you might use:

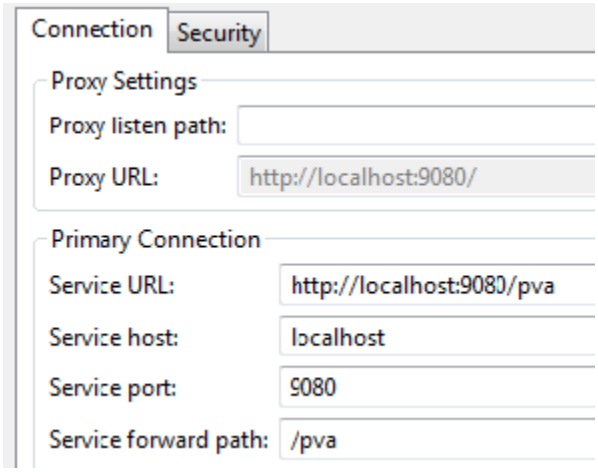
- **Service host:** virtualize.parasoft.com
- **Service port:** 9080

- **Proxy listen path:** /path



To send traffic to a virtual asset on a local server, you might use:

- **Service host:** localhost
- **Service port:** 9080
- **Service forward path:** /pva



The HTTP proxy makes no distinction between a virtual asset and an actual service. Both are configured the same way. However if the HTTP proxy is sending to localhost, you must specify **Service forward path** because the proxy doesn't allow forwarding to itself.

> **Forwarding Requests to Services that Require Case-sensitive Header Names**
>
> If you are using the Virtualize desktop's server, your message proxy headers will be lowercased and may fail to function properly when requests are forwarded to a service that requires case-sensitive HTTP header names. See Forwarding Requests to Services that Require Case-sensitive Header Names for details.

Security Configuration

There are two aspects of configuration for security:

- **If the service to which you are forwarding the traffic uses SSL and/or access authentication**, you need to perform proxy-level configuration (from the proxy's UI controls).
- **If the AUT uses SSL and/or access authentication**, you need to perform server-level Configuration (for the Tomcat-based server).

Depending on your configuration, you configure either security setting or both.

Proxy-level Configuration

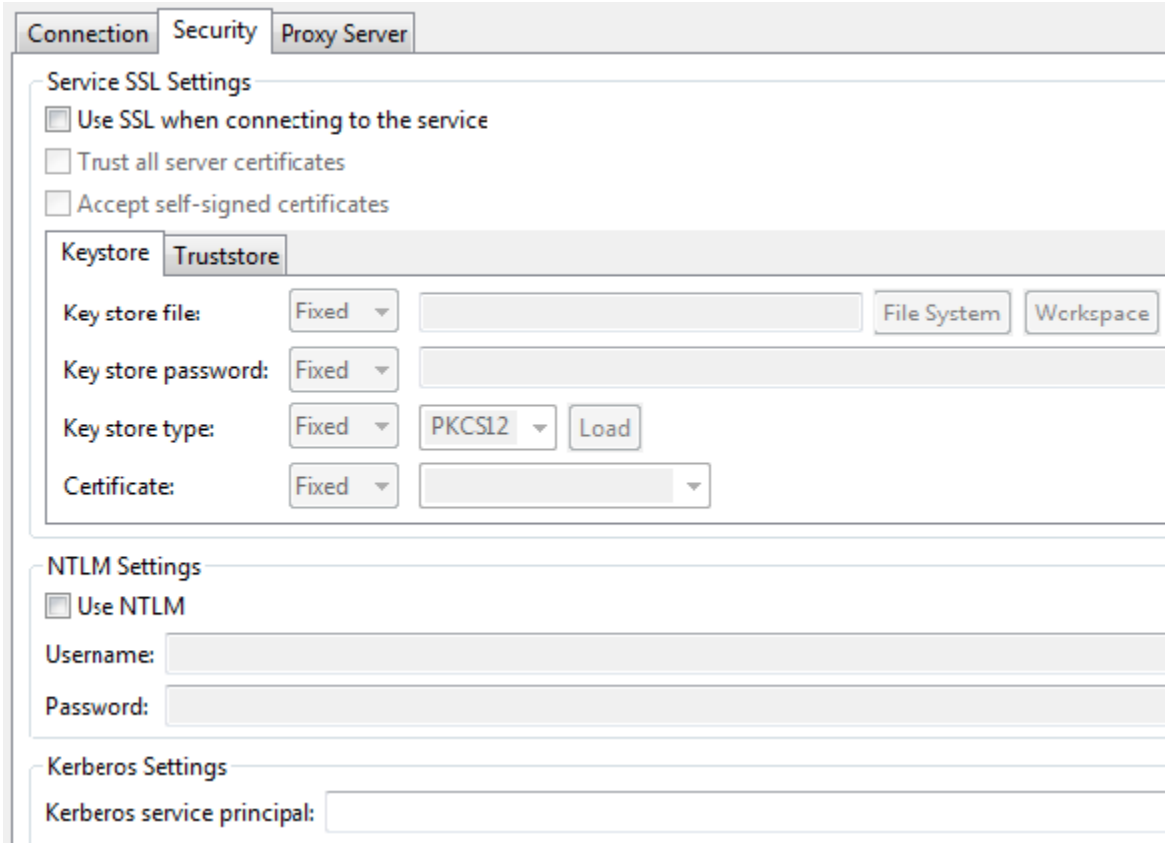
Proxy-level security configuration—which is relevant if the service to which you are forwarding the traffic uses SSL and/or access authentication—covers:

SSL

For SSL, additional settings are required. In the lower part of the configuration panel, you need to specify whether to:

- Enable trust for self signed certificates.
- Enable trust for all certificate.
- Set a trust store to validate server certificates.

Additionally, you need to provide the information needed to use a specific key store and certificate so Parasoft can determine which certificate to present to a server (e.g., for 2-way SSL).



SSL Configuration Panel Fields

Option	Description
Use SSL when connecting to the service	Enables the SSL.
Trust all server certificates	If enabled, any certificate will be accepted. No validation will be performed. This option disables trust validation when the message proxy establishes the connection with the service; it makes it accept the connection with any certificate that is presented by the service. In general, this option should be enabled if certificate trust is not a focus for the environment where is deployed.
Accept self-signed certificates	If enabled, certificates will be accepted as long as the validation method java.security.cert.X509Certificate.checkValidity() returns true on them, which effectively checks if the current date and time are within the validity period given in the certificate. The certificate trust path will not be evaluated and the provided Trust-store configuration will not be applied. This option determines if certificates presented by the service and are not signed by a trusted certificate authority are trusted. In general, this option should be enabled if certificate trust is not a focus for the environment whereSOAtest or Virtualizeis deployed.

Note that the Truststore configuration (described below) is applicable only if **Trust all server certificates** and **Accept self-signed certificates** are both unselected.

Keystore Configuration Panel Fields

Option	Description
Key store file	Specifies the path to the key store file. The keystore determines the certificates and keys that are presented by the message proxy to the service during the SSL handshake.
Key store password	Specifies the password to access the key store.
Key store type	Specifies the type of the key store.
Certificate	Specifies the alias of the certificate to use when authenticating to a server.

After completing the keystore detail, click the **Load** and choose the certificate alias to present to the server from the **Certificate** drop-down. If the **Certificate** dropdown is not populated when you click **Load**, you may have entered the incorrect password or keystore type.

Truststore Configuration Panel Fields

These fields are applicable only if **Trust all** and **Accept self-signed certificates** (described above) are both unselected.

Option	Description
Key store file	Specifies the path to the truststore file. The truststore determines the certificates that the message proxy should trust when hand-shaking with the service. If the service presents a certificate that is not included in this store, the connection will be refused. If no truststore is provided, the default JRE truststore will be used. This option is only applicable if **trust all server certificates** is NOT selected.
Key store password	Specifies the password to access the truststore.
Key store type	Specifies the type of the truststore.

NTLM Settings Panel Fields

Option	Description
Use NTLM	Specifies whether the service requires NTLM authentication.
Username	Specifies the username for NTLM authentication.
Password	Specifies the password for NTLM authentication.

Kerberos Panel Fields

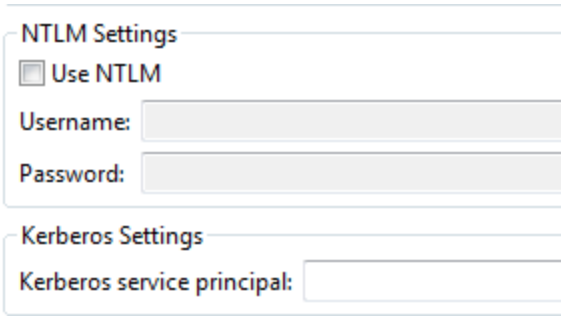
Option	Description
Kerberos service principal	Specifies the service principal to authenticate the request.

Basic/Digest Authentication

If the application under test provides Basic and Digest authentication credentials as part of the request and transmits them as part of the HTTP header, the proxy will pass them along to the service unmodified (the same way it handles other HTTP headers).

NTLM Authentication

If your service requires NTLM authentication, provide the username/password in the NTLM section.



Kerberos Authentication

If your service requires Kerberos authentication, set the **Kerberos Service Principal** in the Kerberos Authentication section.

Server-level Configuration

Server-level security configuration—which is relevant if the AUT uses SSL and/or access authentication—can involve:

- Configuring the Virtualize Server
- Using NTLM with WAFFLE
- Using Unsupported Configurations (Kerberos with WAFFLE, JAAS)

Configuring the SOAtest and Virtualize Server

You can change the default port number (9080), enable SSL, and configure other server settings. See Server Configuration for details.

Using NTLM with WAFFLE

To use the third-party library WAFFLE for NTLM:

1. Copy the following jar files into tomcat's `lib` directory: `jna.jar, platform.jar, wafflejna.jar`.
2. Add the following to `tomcat/conf/server.xml`.

```
<Context>
  <Valve className="waffle.apache.NegotiateAuthenticator" principalFormat="fqn" roleFormat="both"/>
  <Realm className="waffle.apache.WindowsRealm" /> </Context>
```

> **Where is server.xml?**
>
> **If you installed SOAtest but not Virtualize:** Launch SOAtest, ensure that at least one responder has been created, then modify the server.xml file at [SOAtest install dir]/eclipse/plugins/com.parasoft.xtest.libs.web_[version]/root/tomcat/conf/server.xml
>
> **If you installed Virtualize but not SOAtest:** Launch Virtualize, ensure that at least one responder has been created, then modify the server.xml file at [Virtualize install dir]/eclipse/plugins/com.parasoft.xtest.libs.web_[version]/root/tomcat/conf/server.xml
>
> **If you installed Virtualize and SOAtest together:** Launch Virtualize, ensure that at least one responder has been created, then modify the server.xml file at [SOAtest install dir]/eclipse/plugins/com.parasoft.xtest.libs.web_[SOAtest_ver]/root/tomcat/conf/server.xml

3. Add the following to `tomcat/conf/web.xml`.

```
<security-role>
  <role-name>Everyone</role-name>
</security-role>
<security-constraint>
  <display-name>Waffle Security Constraint</display-name>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Everyone</role-name>
  </auth-constraint>
</security-constraint>
```

4. Restart the server.

For more details, see the Single Sign-On: Tomcat Negotiate Authenticator (Kerberos + NTLM) w/ Waffle Tutorial and the WAFFLE home page.

Using Unsupported Configurations (Kerberos with WAFFLE, JAAS)

WAFFLE includes support for Kerberos authentication; see the WAFFLE home page for directions and support.

JAAS is another option for configuring the Tomcat-based Virtualize Server to perform Kerberos authentication. Consult Tomcat for directions on how to use JAAS.