# HTTP4

## HTTP4 Component

**Available as of Camel 2.3**

The **`camel-http4`** component provides HTTP based endpoints for calling external HTTP resources (as a client to call external servers using HTTP).

Maven users will need to add the following dependency to their **`pom.xml`** for this component:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-http4</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

camel-http4 vs camel-http

**`camel-http4`** uses Apache HttpClient 4.x while **`camel-http`** uses Apache HttpClient 3.x.

### URI Format

```
http4:hostname[:port][/resourceUri][?options]
```

### Default Ports

- **`80`** for **`HTTP`**
- **`443`** for **`HTTPS`.**

### Specifying Options

Options should be passed on the URI's query string using the following format: **`?option=value&option=value&...`**

camel-http4 vs camel-jetty

You can only produce to endpoints generated by the **`camel-http4`** component. Therefore it should never be used as input into your Camel Routes. To bind/expose an HTTP endpoint via a HTTP server as input to a Camel route, use the Jetty Component instead.

### `HttpComponent` Options

| Name | Default Value | Description |
|---|---|---|
| `maxTotalConnections` | `200` | The maximum number of connections. |
| `connectionsPerRoute` | `20` | The maximum number of connections per route. |
| `cookieStore` | `null` | **Camel 2.11.2/2.12.0:** To use a custom **`org.apache.http.client.CookieStore`**. By default the **`org.apache.http.impl.client.BasicCookieStore`** is used which is an in-memory only cookie store.<br><br>**Note**: if **`bridgeEndpoint=true`** then the cookie store is forced to be a NOOP cookie store as cookies shouldn't be stored as we are just bridging e.g., acting as a proxy. |
| `httpClientConfigurer` | `null` | Reference to a **`org.apache.camel.component.http.HttpClientConfigurer`** in the Registry. |
| `clientConnectionManager` | `null` | To use a custom **`org.apache.http.conn.ClientConnectionManager`**. |
| `httpBinding` | `null` | To use a custom **`org.apache.camel.component.http.HttpBinding`**. |

| | | |
|---|---|---|
| `httpContext` | `null` | **Camel 2.9.2:** To use a custom `org.apache.http.protocol.HttpContext` when executing requests. |
| `sslContextParameters` | `null` | **Camel 2.8:** To use a custom `org.apache.camel.util.jsse.SSLContextParameters`. See Using the JSSE Configuration Utility.<br><br>**Note:** only one instance of `org.apache.camel.util.jsse.SSLContextParameters` is supported per `HttpComponent`. If two or more instances are needed, a new `HttpComponent` should be created per instance.<br><br>See below for more details. |
| `x509HostnameVerifier` | `BrowserCompatHostnameVerifier` | **Camel 2.7:** You can refer to a different `org.apache.http.conn.ssl.X509HostnameVerifier` instance in the Registry such as `org.apache.http.conn.ssl.StrictHostnameVerifier` or `org.apache.http.conn.ssl.AllowAllHostnameVerifier`. |
| `connectionTimeToLive` | `-1` | **Camel 2.11.0:** The time for connection to live, the time unit is millisecond, the default value is always keep alive. |
| `allowJavaSerializedObject` | `false` | **Camel 2.16.1/2.15.5:** Whether to allow java serialization when a request uses `context-type=application/x-java-serialized-object`. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk. |

## `HttpEndpoint` Options

| Name | Default Value | Description |
|---|---|---|
| `throwExceptionOnFailure` | `true` | Option to disable throwing the `HttpOperationFailedException` in case of failed responses from the remote server. This allows you to get all responses regardless of the HTTP status code. |
| `bridgeEndpoint` | `false` | If `true`, `HttpProducer` will ignore the `Exchange.HTTP_URI` header, and use the endpoint's URI for request.<br><br>If `throwExcpetionOnFailure=false` the `HttpProducer` will return all fault responses to the caller.<br><br>Also, if `true` then `HttpProducer` and `CamelServlet` will skip the gzip processing if the content-encoding is `gzip`. |
| `clearExpiredCookies` | `true` | **Camel 2.11.2/2.12.0:** Whether to clear expired cookies before sending the HTTP request. This ensures the cookies store does not keep growing by adding new cookies which is newer removed when they are expired. |
| `cookieStore` | `null` | **Camel 2.11.2/2.12.0:** To use a custom `org.apache.http.client.CookieStore`. By default the `org.apache.http.impl.client.BasicCookieStore` is used which is an in-memory only cookie store.<br><br>**Note**: if `bridgeEndpoint=true` then the cookie store is forced to be a NOOP cookie store as cookies shouldn't be stored as we are just bridging e.g., acting as a proxy. |
| `disableStreamCache` | `false` | `DefaultHttpBinding` will copy the request input stream into a stream cache and put it into the message body if this option is false to support multiple reads, otherwise `DefaultHttpBinding` will set the request input stream directly in the message body.<br><br>From **Camel 2.17:** this option is also supported by the producer to allow the use of a response stream directly instead of stream caching as by default. |
| `headerFilterStrategy` | `null` | **Camel 2.10.4:** Reference to a instance of `org.apache.camel.spi.HeaderFilterStrategy` in the Registry. It will be used to apply the custom `headerFilterStrategy` on the new create `HttpEndpoint`. |
| `httpBindingRef` | `null` | **Deprecated and will be removed in Camel 3.0:** Reference to a `org.apache.camel.component.http.HttpBinding` in the Registry. Use the `httpBinding` option instead. |
| `httpBinding` | `null` | To use a custom `org.apache.camel.component.http.HttpBinding`. |
| ~~`httpClientConfigurerRef`~~ | ~~`null`~~ | ~~**Deprecated and removed in Camel 2.17:** Reference to a `org.apache.camel.component.http.HttpClientConfigurer` in the Registry. Use the `httpClientConfigurer` option instead.~~ |

| | | |
|---|---|---|
| httpClientConfigurer | null | Reference to a **`org.apache.camel.component.http.HttpClientConfigurer`** in the Registry. |
| ~~httpContextRef~~ | ~~null~~ | ~~**Deprecated and removed in Camel 2.17: Camel 2.9.2:** Reference to a custom `org.apache.http.protocol.HttpContext` in the Registry. Use the `httpContext` option instead.~~ |
| httpContext | null | **Camel 2.9.2:** To use a custom **`org.apache.http.protocol.HttpContext`** when executing requests. |
| httpClient.XXX | null | Setting options on the BasicHttpParams. For instance **`httpClient.soTimeout=5000`** will set the **`SO_TIMEOUT`** to **5** seconds. Look on the setter methods of the following parameter beans for a complete reference: AuthParamBean, ClientParamBean, ConnConnectionParamBean, ConnRouteParamBean, CookieSpecParamBean, HttpConnectionParamBean and HttpProtocolParamBean<br><br>From **Camel 2.13.0: `httpClient`** is changed to configure the HttpClientBuilder and RequestConfig.Builder, please check out API document for a complete reference. e.g., since this version use **`httpClient.socketTimeout=5000`** for setting the socket timeout to 5 seconds. |
| clientConnectionManager | null | To use a custom **`org.apache.http.conn.ClientConnectionManager`**. |
| transferException | false | If **`true`** and an Exchange failed processing on the consumer side, and if the caused **`Exception`** was send back serialized in the response as a **`application/x-java-serialized-object`** content type (for example using Jetty or SERVLET Camel components).<br><br>On the producer side the exception will be deserialized and thrown as is, instead of the **`HttpOperationFailedException`**. The caused exception is required to be serialized. |
| ~~sslContextParametersRef~~ | ~~null~~ | ~~**Deprecated and removed in Camel 2.17: Camel 2.8:** Reference to a `org.apache.camel.util.jsse.SSLContextParameters` in the Registry. **Important:** Only one instance of `org.apache.camel.util.jsse.SSLContextParameters` is supported per HttpComponent. If you need to use 2 or more different instances, you need to define a new HttpComponent per instance you need. See further below for more details. See Using the JSSE Configuration Utility. Use the `sslContextParameters` option instead.~~ |
| sslContextParameters | null | **Camel 2.11.1:** Reference to a **`org.apache.camel.util.jsse.SSLContextParameters`** in the Registry.<br><br>**Note:** only one instance of **`org.apache.camel.util.jsse.SSLContextParameters`** is supported per **`HttpComponent`**. If more instances are required, a new **`HttpComponent`** should be created per instance.<br><br>See below for more details.<br><br>See Using the JSSE Configuration Utility. |
| x509HostnameVerifier | BrowserCompatHostnameVerifier | **Camel 2.7:** You can refer to a different **`org.apache.http.conn.ssl.X509HostnameVerifier`** instance in the Registry such as **`org.apache.http.conn.ssl.StrictHostnameVerifier`** or **`org.apache.http.conn.ssl.AllowAllHostnameVerifier`**. |
| urlRewrite | null | **Camel 2.11: Producer only** Refers to a custom **`org.apache.camel.component.http4.UrlRewrite`** which allows you to rewrite URLs when you bridge/proxy endpoints.<br><br>For more details see UrlRewrite and How to use Camel as a HTTP proxy between a client and server. |
| maxTotalConnections | null | **Camel 2.14**: The maximum number of total connections that the connection manager has. If this option is not set, camel will use the component's setting instead. |
| connectionsPerRoute | null | **Camel 2.14**: The maximum number of connections per route. If this option is not set, camel will use the component's setting instead. |
| authenticationPreemptive | false | **Camel 2.11.3/2.12.2:** If this option is true, **`camel-http4`** sends preemptive basic authentication to the server. |
| eagerCheckContentAvailable | false | **Camel 2.16: Consumer only** Whether to eager check whether the HTTP requests has content if the content-length header is 0 or not present. This can be turned on in case HTTP clients do not send streamed data. |

|  |  |  |
|---|---|---|
| `copyHeaders` | `true` | **Camel 2.16:** If this option is true then **`IN`** exchange headers will be copied to **`OUT`** exchange headers according to copy strategy. Setting this to **`false`**, allows to only include the headers from the HTTP response (not propagating **`IN`** headers). |
| `okStatusCodeRange` | `200-299` | **Camel 2.16:** The status codes which is considered a success response. The values are inclusive. The range must be defined as from-to with the dash included. |
| `ignoreResponseBody` | `false` | **Camel 2.16:** If this option is true, The HTTP producer won't read response body and cache the input stream. |
| `useSystemProperties` | `false` | **Camel 2.18:** If this option is true, The HTTP client will use System Properties to set some parameters of his configuration |
| `mapHttpMessageBody` | `true` | **Camel 2.18:** If this option is true then **`IN`** exchange Body will be mapped to HTTP body. Setting this to false will avoid the HTTP mapping. |
| `mapHttpMessageHeaders` | `true` | **Camel 2.18:** If this option is true then **`IN`** exchange Headers will be mapped to HTTP headers. Setting this to false will avoid the HTTP Headers mapping. |
| `mapHttpMessageFormUrlEncodedBody` | `true` | **Camel 2.18:** If this option is true then **`IN`** exchange Form Encoded body of the exchange will be mapped to HTTP. Setting this to false will avoid the HTTP Form Encoded body mapping. |
| `connectionClose` | `false` | **Camel 2.18:** If this option is true, the producer will add a Connection Close header to HTTP Request |
| `cookieHandler` | `null` | **Camel 2.19:** Configure a cookie handler to maintain a HTTP session |

## Setting Basic Authentication and Proxy

The following authentication options can also be set on the **`HttpEndpoint`**:

Before **Camel 2.8.0**:

| Name | Default Value | Description |
|---|---|---|
| `domain` | `null` | The domain name for authentication. |
| `host` | `null` | The host name authentication. |
| `password` | `null` | Password for authentication. |
| `username` | `null` | Username for authentication. |
| `proxyHost` | `null` | The proxy host name |
| `proxyPort` | `null` | The proxy port number |
| `proxyUsername` | `null` | Username for proxy authentication |
| `proxyPassword` | `null` | Password for proxy authentication |
| `proxyDomain` | `null` | The proxy domain name |
| `proxyNtHost` | `null` | The proxy Nt host name |

From **Camel 2.8.0**:

| Name | Default Value | Description |
|---|---|---|
| `authDomain` | `null` | The domain name for authentication |
| `authHost` | `null` | The host name authentication |
| `authPassword` | `null` | Password for authentication |
| `authUsername` | `null` | Username for authentication |

| | | |
|---|---|---|
| `proxyAuthHost` | `null` | The proxy host name |
| `proxyAuthPort` | `null` | The proxy port number |
| `proxyAuthScheme` | `null` | The proxy scheme, will fallback and use the scheme from the endpoint if not configured. |
| `proxyAuthUsername` | `null` | Username for proxy authentication |
| `proxyAuthPassword` | `null` | Password for proxy authentication |
| `proxyAuthDomain` | `null` | The proxy domain name |
| `proxyAuthNtHost` | `null` | The proxy Nt host name |

## Message Headers

| Name | Type | Description |
|---|---|---|
| `Exchange.CONTENT_ENCODING` | `String` | The HTTP content encoding. Is set on both the **`IN`** and **`OUT`** message to provide a content encoding, such as **`gzip`**. |
| `Exchange.CONTENT_TYPE` | `String` | The HTTP content type. Is set on both the **`IN`** and **`OUT`** message to provide a content type, such as **`text/html`**. |
| `Exchange.HTTP_CHARACTER_ENCODING` | `String` | Character encoding. |
| `Exchange.HTTP_PATH` | `String` | Request URI's path. The header will be used to build the request URI with the **`HTTP_URI`**. |
| `Exchange.HTTP_QUERY` | `String` | URI parameters. Will override existing URI parameters set directly on the endpoint. |
| `Exchange.HTTP_RESPONSE_CODE` | `int` | The HTTP response code from the external server. Is **`200`** for **`OK`**. |
| `Exchange.HTTP_RESPONSE_TEXT` | `String` | The HTTP response text from the external server. |
| `Exchange.HTTP_URI` | `String` | The URI to call. The value of this option will override the existing URI that's set directly on the endpoint. It's not the same as the Camel endpoint URI, where you can configure endpoint options such as security etc. This header does not support that, it's only the URI of the HTTP server. |

Before setting the above, you may wish to read How to avoid sending some or all message headers to prevent inadvertent data "leaks" from your application.

## Message Body

Camel will store the HTTP response from the external server on the **`OUT`** body. All headers from the **`IN`** message will be copied to the **`OUT`** message, so headers are preserved during routing. Additionally Camel will add the HTTP response headers as well to the **`OUT`** message headers.

Using System Properties

When **`useSystemProperties=true`** the **`camel-http4`** client can make use the following system properties:

- `java.home`
- `javax.net.ssl.trustStoreType`
- `javax.net.ssl.trustStore`
- `javax.net.ssl.trustStoreProvider`
- `javax.net.ssl.trustStorePassword`
- `javax.net.ssl.keyStore`
- `javax.net.ssl.keyStoreProvider`
- `javax.net.ssl.keyStorePassword`
- `javax.net.ssl.keyStoreType`
- `http.proxyHost`
- `http.proxyPort`
- `http.nonProxyHosts`
- `http.keepAlive`
- `http.maxConnections`
- `ssl.KeyManagerFactory.algorithm`
- `ssl.TrustManagerFactory.algorithm`

## Response Code

Camel will handle according to the HTTP response code:

- Response code is in the range **`100..299`**, Camel regards it as a success response.
- Response code is in the range **`300..399`**, Camel regards it as a redirection response and will throw a **`HttpOperationFailedException`** with the information.
- Response code is **`400+`**, Camel regards it as an external server failure and will throw a **`HttpOperationFailedException`** with the information.

  throwExceptionOnFailure

  If **`throwExceptionOnFailure=false`** the **`HttpOperationFailedException`** will not be thrown for failed response codes. This allows you to get any response from the remote server.
  There is a sample below demonstrating this.

## `HttpOperationFailedException`

This exception contains the following information:

- The HTTP status code.
- The HTTP status line (text of the status code).
- Redirect location, if server returned a redirect.
- Response body as a **`java.lang.String`**, if server provided a body as response.

## Calling using `GET` or `POST`

The following algorithm is used to determine whether the **`GET`** or **`POST`** HTTP method should be used:

1. Use the method provided in the header.
2. **`GET`** if query string is provided in the header.
3. **`GET`** if endpoint is configured with a query string.
4. **`POST`** if there is data to send (body is not null).
5. **`GET`** otherwise.

## How to Access The `HttpServletRequest` and `HttpServletResponse`

You can get access to these two using the Camel type converter system using

**Note**: you can get the request and response not just from the processor after the **`camel-jetty`** or **`camel-cxf`** endpoint.

```
HttpServletRequest request = exchange.getIn().getBody(HttpServletRequest.class);
HttpServletRequest response = exchange.getIn().getBody(HttpServletResponse.class);
```

## Configuring URI to Call

You can set the HTTP producer's URI directly form the endpoint URI. In the route below, Camel will call out to the external server, **`oldhost`**, using HTTP.

**Java DSL**:

```
from("direct:start")
  .to("http4://oldhost");
```

**Spring DSL**:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <to uri="http4://oldhost"/>
  </route>
</camelContext>
```

You can override the HTTP endpoint URI by adding a header with the key, **`Exchange.HTTP_URI`**, on the message:

```
from("direct:start")
  .setHeader(Exchange.HTTP_URI, constant("http://newhost"))
  .to("http4://oldhost");
```

In the sample above Camel will call the **`http://newhost`** despite the endpoint is configured with **`http4://oldhost`**. If the **`camel-http4`** endpoint is working in bridge mode, it will ignore the header **`Exchange.HTTP_URI`**.

## Configuring URI Parameters

The **`camel-http4`** producer supports URI parameters to be sent to the HTTP server. The URI parameters can either be set directly on the endpoint URI or as a header with the key **`Exchange.HTTP_QUERY`** on the message:

```
from("direct:start")
  .to("http4://oldhost?order=123&detail=short");
```

Or provided via a header:

```
from("direct:start")
  .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
  .to("http4://oldhost");
```

## How To Set The HTTP Method ( `GET` / `PATCH` / `POST` / `PUT` / `DELETE` / `HEAD` / `OPTIONS` / `TRACE` ) on the HTTP Producer

Using the http PATCH method

The HTTP **`PATCH`** method is supported starting with Camel 2.11.3 / 2.12.1.

The **`camel-http4`** specifies the particular HTTP request method via a header:

**Example**:

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD, constant(org.apache.camel.component.http4.HttpMethods.POST))
  .to("http4://www.google.com")
  .to("mock:results");
```

The method can be written a bit shorter using the string constants:

```
.setHeader("CamelHttpMethod", constant("POST"))
```

**Spring DSL**:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <setHeader headerName="CamelHttpMethod">
        <constant>POST</constant>
    </setHeader>
    <to uri="http4://www.google.com"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

## Using Client Timeout - `SO_TIMEOUT`

See the HttpSOTimeoutTest unit test. From **Camel 2.13.0**: See the updated HttpSOTimeoutTest unit test.

## Configuring a Proxy

The **`camel-http4`** component provides a way to configure a proxy.

```
from("direct:start")
  .to("http4://oldhost?proxyAuthHost=www.myproxy.com&proxyAuthPort=80");
```

There is also support for proxy authentication via the **`proxyAuthUsername`** and **`proxyAuthPassword`** options.

### Using Proxy Settings Outside of the URI

To avoid System properties conflicts, you can set proxy configuration only from the **`CamelContext`** or URI.

**Java DSL**:

```
 context.getProperties().put("http.proxyHost", "172.168.18.9");
 context.getProperties().put("http.proxyPort" "8080");
```

**Spring DSL**:

```
   <camelContext>
       <properties>
           <property key="http.proxyHost" value="172.168.18.9"/>
           <property key="http.proxyPort" value="8080"/>
      </properties>
   </camelContext>
```

Camel will first set the settings from Java System or **`CamelContext`** Properties and then the endpoint proxy options if provided. So you can override the system properties with the endpoint options.

**Note**: in **Camel 2.8** there is also a **`http.proxyScheme`** property you can set to explicit configure the scheme to use.

## Configuring `charset`

If you are using **`POST`** to send data you can configure the **`charset`** using the **`Exchange`** property:

```
exchange.setProperty(Exchange.CHARSET_NAME, "ISO-8859-1");
```

### Example: Using a Scheduled Poll

This sample polls the Google homepage every 10 seconds and write the page to the file **`message.html`**:

```
from("timer://foo?fixedRate=true&delay=0&period=10000")
  .to("http4://www.google.com")
  .setHeader(FileComponent.HEADER_FILE_NAME, "message.html")
  .to("file:target/google");
```

### URI Parameters From the Endpoint URI

In this sample we have the complete URI endpoint that is just what you would have typed in a web browser. Multiple URI parameters can of course be set using the **&** character as separator, just as you would in the web browser. Camel does no tricks here.

```
// we query for Camel at the Google page
template.sendBody("http4://www.google.com/search?q=Camel", null);
```

### URI Parameters From the Message

```
Map headers = new HashMap();
headers.put(Exchange.HTTP_QUERY, "q=Camel&lr=lang_en");
// we query for Camel and English language at Google
template.sendBody("http4://www.google.com/search", null, headers);
```

In the header value above notice that it should **not** be prefixed with **?** and you can separate parameters as usual with the **&** char.

### Getting the Response Code

You can get the HTTP response code from the **`camel-http4`** component by getting the value from the **`OUT`** message header with **`Exchange.HTTP_RESPONSE_CODE`**.

```
Exchange exchange = template.send("http4://www.google.com/search", new Processor() {
  public void process(Exchange exchange) throws Exception {
    exchange.getIn().setHeader(Exchange.HTTP_QUERY, constant("hl=en&q=activemq"));
  }
});

Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

## Disabling Cookies

To disable cookies you can set the HTTP Client to ignore cookies by adding this URI option:  **`httpClient.cookiePolicy=ignoreCookies`**

## Advanced Usage

If you need more control over the HTTP producer you should use the **`HttpComponent`** where you can set various classes to give you custom behavior.

### Setting up SSL for HTTP Client

#### Using the JSSE Configuration Utility

From **Camel 2.8**: the **`camel-http4`** component supports SSL/TLS configuration through the Camel JSSE Configuration Utility.  This utility greatly decreases the amount of component specific code you need to write and is configurable at the endpoint and component levels.  The following examples demonstrate how to use the utility with the **`camel-http4`** component.

##### Programmatic Configuration of the Component

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

HttpComponent httpComponent = getContext().getComponent("https4", HttpComponent.class);
httpComponent.setSslContextParameters(scp);
```

##### Spring DSL Based Configuration of Endpoint

```
...
  <camel:sslContextParameters
      id="sslContextParameters">
    <camel:keyManagers
        keyPassword="keyPassword">
      <camel:keyStore
          resource="/users/home/server/keystore.jks"
          password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="https4://127.0.0.1/mail/?sslContextParametersRef=sslContextParameters"/>...
```

#### Configuring Apache HTTP Client Directly

Basically **`camel-http4`** component is built on the top of Apache HttpClient. Please refer to SSL/TLS customization for details or have a look into the **`org.apache.camel.component.http4.HttpsServerTestSupport`** unit test base class.
You can also implement a custom **`org.apache.camel.component.http4.HttpClientConfigurer`** to do some configuration on the **`http`** client if you need full control of it.

However if you *just* want to specify the keystore and truststore you can do this with Apache HTTP **`HttpClientConfigurer`**, for example:

```
KeyStore keystore = ...;
KeyStore truststore = ...;

SchemeRegistry registry = new SchemeRegistry();
registry.register(new Scheme("https", 443, new SSLSocketFactory(keystore, "mypassword", truststore)));
```

And then you need to create a class that implements **`HttpClientConfigurer`**, and registers **`https`** protocol providing a keystore or truststore per example above. Then, from your camel route builder class you can hook it up like so:

```
HttpComponent httpComponent = getContext().getComponent("http4", HttpComponent.class);
httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());
```

If you are doing this using the Spring DSL, you can specify your **`HttpClientConfigurer`** using the URI.

Example:

```
<bean id="myHttpClientConfigurer"
 class="my.https.HttpClientConfigurer">
</bean>

<to uri="https4://myhostname.com:443/myURL?httpClientConfigurer=myHttpClientConfigurer"/>
```

As long as you implement the **`HttpClientConfigurer`** and configure your keystore and truststore as described above, it will work fine.

### Using HTTPS to authenticate gotchas

An end user reported that he had problem with authenticating with HTTPS. The problem was eventually resolved by providing a custom configured **`org.apache.http.protocol.HttpContext`**:

1. Create a (Spring) factory for **`HttpContext`**'s:

```
public class HttpContextFactory {

  private String httpHost = "localhost";
  private String httpPort = 9001;

  private BasicHttpContext httpContext = new BasicHttpContext();
  private BasicAuthCache authCache = new BasicAuthCache();
  private BasicScheme basicAuth = new BasicScheme();

  public HttpContext getObject() {
    authCache.put(new HttpHost(httpHost, httpPort), basicAuth);

    httpContext.setAttribute(ClientContext.AUTH_CACHE, authCache);

    return httpContext;
  }

  // getter and setter
}
```

2. Declare an **`HttpContext`** in the Spring application context file:

```
<bean id="myHttpContext" factory-bean="httpContextFactory" factory-method="getObject"/>
```

3. Reference the context in the **`http4`** URL:

```
<to uri="https4://myhostname.com:443/myURL?httpContext=myHttpContext"/>
```

### Using Different `SSLContextParameters`

The HTTP4 component only support one instance of **`org.apache.camel.util.jsse.SSLContextParameters`** per component. If you need to use two or more different instances, then you need to setup multiple HTTP4 components as shown below. Where we have two components, each using their own instance of **`sslContextParameters`** property.

```
<bean id="http4-foo" class="org.apache.camel.component.http4.HttpComponent">
   <property name="sslContextParameters" ref="sslContextParams1"/>
   <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>

<bean id="http4-bar" class="org.apache.camel.component.http4.HttpComponent">
   <property name="sslContextParameters" ref="sslContextParams2"/>
   <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>
```