HTTP

HTTP Component

The **http:** component provides HTTP based endpoints for consuming 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:

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

URI Format

http:hostname[:port][/resourceUri][?param1=value1][&param2=value2]

Will by default use port **`80`** for HTTP and **`443`** for HTTPS.

camel-http vs camel-jetty
You can only produce to endpoints generated by the HTTP 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, you can use the Jetty Component or the Servlet Component

Examples

Call the URL with the body using **`POST`** and return response as the **`OUT`** message. If body is **`null`** call URL using **`GET`** and return response as **`OUT`** message:

Java DSL	Spring DSL
from("direct:start") .to("http://myhost/mypath");	xml<from uri="direct:start"/> <to uri="http://oldhost"/>

You can override the HTTP endpoint URI by adding a header. Camel will call the http://newhost. This is very handy for e.g. REST URLs:

Java DSL
javafrom("direct:start") .setHeader(Exchange.HTTP_URI, simple("http://myserver/orders/${header.orderId}")) .to("http://dummyhost");

URI parameters can either be set directly on the endpoint URI or as a header:

Java DSL
javafrom("direct:start") .to("http://oldhost?order=123&detail=short"); from("direct:start") .setHeader(Exchange.HTTP_QUERY, constant ("order=123&detail=short")) .to("http://oldhost");

Set the HTTP request method to **`POST`**:

Java DSL	Spring DSL
from("direct:start") .setHeader(Exchange. HTTP_METHOD, constant("POST")) .to("http://www. google.com");	xml<from uri="direct:start"/> <setHeader headerName="CamelHttpMethod"> <constant>POST</constant> </setHeader> <to uri="http://www.google.com"/> <to uri="mock: results"/>

`HttpEndpoint` Options

confluenceTableSmall

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 the option is **`true`**, **`HttpProducer`** will ignore the **`Exchange.HTTP_URI`** header, and use the endpoint's URI for request. You may also set **`throwExceptionOnFailure=false`** to ensure all responses are propagated back to the **`HttpProducer`**. From **Camel 2.3:** when **`true`** the **`HttpProducer`** and **`CamelServlet`** will skip **`gzip`** processing when **`content-encoding=gzip`**.

`disableStreamCache`	`false`	When **`false`** the **`DefaultHttpBinding`** will copy the request input stream into a stream cache and put it into message body which allows it to be read more than once. When **`true`** the **`DefaultHttpBinding`** will set the request input stream direct into the message body. From **Camel 2.17:** this options is now also support by the producer to allow using the response stream directly instead of stream caching as by default.
~~`httpBindingRef`~~	~~`null`~~	~~**Deprecated and removed in Camel 2.17:** Reference to a `org.apache.camel.component.http.HttpBinding` in the Registry. Use the `httpBinding` option instead.~~
`httpBinding`	`null`	From **Camel 2.3:** reference to a **`org.apache.camel.component.http.HttpBinding`** in the Registry.
~~`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`	From **Camel 2.3:** reference to a **`org.apache.camel.component.http.HttpClientConfigurer`** in the Registry.
`httpClient.XXX`	`null`	Use this to option to configure the underlying HttpClientParams. Example: **`httpClient.soTimeout=5000`** will set the **`SO_TIMEOUT`** to 5 seconds.
`clientConnectionManager`	`null`	To use a custom **`org.apache.http.conn.ClientConnectionManager`**.
`transferException`	`false`	From **Camel 2.6:** If enabled 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). On the producer side the exception will be deserialized and thrown as is, instead of the **`HttpOperationFailedException`**. The caused exception will be serialized.
`headerFilterStrategy`	`null`	From **Camel 2.11:** 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`**.
`urlRewrite`	`null`	From **Camel 2.11:** ***Producer only!*** Refers to a custom **`org.apache.camel.component.http.UrlRewrite`** which allows you to rewrite URLs when you bridge/proxy endpoints. See more details at UrlRewrite and How to use Camel as a HTTP proxy between a client and server.
`eagerCheckContentAvailable`	`false`	From **Camel 2.15.3/2.16:** ***Consumer only!*** Whether to eager check whether the HTTP requests has content when **`content-length=0`** or is not present. This option should be set to **`true`** for those HTTP clients that do not send streamed data.
`copyHeaders`	`true`	From **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`	From **Camel 2.16:** the range of HTTP status codes for which a response is considered a success. The values are inclusive. The range must be in the form **`from-to`**, dash included.
`ignoreResponseBody`	`false`	From **Camel 2.16:** when **`true`** the **`HttpProducer`** will not read the response body nor cache the input stream.
`cookieHandler`	`null`	From **Camel: 2.19:** configure a cookie handler to maintain a HTTP session

Authentication and Proxy

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

confluenceTableSmall

Name	Default Value	Description
`authMethod`	`null`	Authentication method, either as **`Basic`**, **`Digest`** or **`NTLM`**.

`authMethodPriority`	`null`	Priority of authentication methods. Is a list separated with comma. For example: **`Basic,Digest`** to exclude **`NTLM`**.
`authUsername`	`null`	Username for authentication.
`authPassword`	`null`	Password for authentication.
`authDomain`	`null`	Domain for **`NTLM`** authentication.
`authHost`	`null`	Optional host for **`NTLM`** authentication.
`proxyHost`	`null`	The proxy host name.
`proxyPort`	`null`	The proxy port number.
`proxyAuthMethod`	`null`	Authentication method for proxy, either as **`Basic`**, **`Digest`** or **`NTLM`**.
`proxyAuthUsername`	`null`	Username for proxy authentication.
`proxyAuthPassword`	`null`	Password for proxy authentication.
`proxyAuthDomain`	`null`	Domain for proxy **`NTLM`** authentication.
`proxyAuthHost`	`null`	Optional host for proxy **`NTLM`** authentication.

When using authentication you **must** provide the choice of method for the **`authMethod`** or **`authProxyMethod`** options. You can configure the proxy and authentication details on either the **`HttpComponent`** or the **`HttpEndoint`**. Values provided on the **`HttpEndpoint`** will take precedence over **`HttpComponent`**. Its most likely best to configure this on the **`HttpComponent`** which allows you to do this once.

The HTTP component uses convention over configuration which means that if you have not explicit set a **`authMethodPriority`** then it will fallback and use the select(ed) **`authMethod`** as priority as well. So if you use **`authMethod.Basic`** then the **`auhtMethodPriority`** will be **`Basic`** only.

Note: **`camel-http`** is based on HttpClient v3.x and as such has only limited support for what is known as NTLMv1, the early version of the NTLM protocol. It does not support NTLMv2 at all. **`camel-http4`** has support for NTLMv2.

`HttpComponent` Options

confluenceTableSmall

Name	Default Value	Description
`httpBinding`	`null`	To use a custom **`org.apache.camel.component.http.HttpBinding`**.
`httpClientConfigurer`	`null`	To use a custom **`org.apache.camel.component.http.HttpClientConfigurer`**.
`httpConnectionManager`	`null`	To use a custom **`org.apache.commons.httpclient.HttpConnectionManager`**.
`httpConfiguration`	`null`	To use a custom **`org.apache.camel.component.http.HttpConfiguration.`**
`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`**. 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.

`HttpConfiguration` contains all the options listed in the table above under the section *HttpConfiguration - Setting Authentication and Proxy*.

Message Headers

confluenceTableSmall

Name	Type	Description
`Exchange.HTTP_URI`	`String`	URI to call. Will override existing URI set directly on the endpoint. This URI is the URI of the HTTP server to call. Its not the same as the Camel endpoint URI, where you can configure endpoint options such as security etc. This header does not support that, its only the URI of the HTTP server.

`Exchange.HTTP_METHOD`	`String`	HTTP method/verb to use. Can be one of: • **`GET`** • **`POST`** • **`PUT`** • **`DELETE`** • **`HEAD`** • **`OPTIONS`** • **`TRACE`**
`Exchange.HTTP_PATH`	`String`	The request URI's path. The header will be used to build the request URI with the **`HTTP_URI`**. From **Camel 2.3.0:** if the path starts with a **`/`**, the **`HttpProducer`** will try to find the relative path based on the **`Exchange.HTTP_BASE_URI`** header or the **`exchange.getFromEndpoint().getEndpointUri();`**.
`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_CHARACTER_ENCODING`	`String`	Character encoding.
`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.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.HTTP_SERVLET_REQUEST`	`HttpServletRequest`	The **`HttpServletRequest`** object.
`Exchange.HTTP_SERVLET_RESPONSE`	`HttpServletResponse`	The **`HttpServletResponse`** object.
`Exchange.HTTP_PROTOCOL_VERSION`	`String`	From **Camel 2.5:** You can set the HTTP protocol version with this header, e.g., **`HTTP/1.0`**. If the header is not present the **`HttpProducer`** will use the default value **`HTTP/1.1`**.

Note: The header names above are constants. For the spring DSL you have to use the value of the constant instead of the name.

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.

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
  The option, **`throwExceptionOnFailure`**, can be set to **`false`** to prevent the **`HttpOperationFailedException`** from being 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 if either **`GET`** or **`POST`** HTTP method should be used:

1. Use method provided in header.
2. **`GET`** if query string is provided in 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:

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

Using Client Timeout - `SO_TIMEOUT`

See the unit test in this link

More Examples

Configuring a Proxy

Java DSL
from("direct:start") .to("http://oldhost?proxyHost=www.myproxy. com&proxyPort=80");

There is also support for proxy authentication via the **`proxyUsername`** and **`proxyPassword`** options.

Using Proxy Settings Outside of the URI

Java DSL	Spring DSL
context.getProperties().put("http.proxyHost", "172.168.18.9"); context.getProperties().put("http.proxyPort" "8080");	<camelContext> <properties> <property key="http.proxyHost" value="172.168.18.9"/> <property key="http.proxyPort" value="8080"/> </properties> </camelContext>

Options on **`Endpoint`** will override options on the context.

Configuring `charset`

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

.setProperty(Exchange.CHARSET_NAME, "iso-8859-1");

Sample with Scheduled Poll

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

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

Getting the Response Code

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

javaExchange exchange = template.send("http://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);

Using `throwExceptionOnFailure=false` To Obtain All Server Responses

In the route below we want to route a message that we enrich with data returned from a remote HTTP call. As we want all responses from the remote server, we set the **`throwExceptionOnFailure=false`** so we get any response in the **`AggregationStrategy`**. As the code is based on a unit test that simulates a HTTP status code 404, there is some assertion code etc.{snippet:id=e1|lang=java|url=camel/tags/camel-2.2.0/components/camel-jetty/src/test/java/org/apache/camel/component/jetty/JettySimplifiedHandle404Test.java}

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 `MaxConnectionsPerHost`

The HTTP Component has a **`org.apache.commons.httpclient.HttpConnectionManager`** where you can configure various global configuration for the given component. By global, we mean that any endpoint the component creates has the same shared **`HttpConnectionManager`**. So, if we want to set a different value for the max connection per host, we need to define it on the HTTP component and *not* on the endpoint URI that we usually use. So here comes:

First, we define the **`http`** component in Spring XML. Yes, we use the same scheme name, **`http`**, because otherwise Camel will auto-discover and create the component with default settings. What we need is to overrule this so we can set our options. In the sample below we set the max connection to 5 instead of the default of 2.{snippet:id=e1|lang=xml|url=camel/tags/camel-2.2.0/tests/camel-itest/src/test/resources/org/apache/camel/itest/http/HttpMaxConnectionPerHostTest-context.xml}And then we can just use it as we normally do in our routes:{snippet:id=e2|lang=xml|url=camel/tags/camel-2.2.0/tests/camel-itest/src/test/resources/org/apache/camel/itest/http/HttpMaxConnectionPerHostTest-context.xml}

Using Pre-Emptive Authentication

If an HTTP server should fail to respond correctly with an expected **`401 Authorization Required`** response for a failed authentication attempt a client can instead use preemptive authentication by specifying the URI option: **`httpClient.authenticationPreemptive=true`**.

Accepting Self-Signed Certificates From Remote Server

See this link from a mailing list discussion with some code to outline how to do this with the Apache Commons HTTP API.

Setting up SSL for HTTP Client

Using the JSSE Configuration Utility

From **Camel 2.8**: the **`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 **`HTTP4`** component.

The version of the Apache HTTP client used in this component resolves SSL/TLS information from a global "protocol" registry. This component provides an implementation, **`org.apache.camel.component.http.SSLContextParametersSecureProtocolSocketFactory`**, of the HTTP client's protocol socket factory in order to support the use of the Camel JSSE Configuration utility. The following example demonstrates how to configure the protocol registry and use the registered protocol information in a route.

javaKeyStoreParameters 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); ProtocolSocketFactory factory = new SSLContextParametersSecureProtocolSocketFactory(scp); Protocol.registerProtocol("https", new Protocol("https", factory, 443)); from("direct:start") .to ("https://mail.google.com/mail/") .to("mock:results");

Configuring Apache HTTP Client Directly

Basically **`camel-http`** component is built on the top of Apache HTTP client, and you can implement a custom **`org.apache.camel.component.http.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:

javaProtocol authhttps = new Protocol("https", new AuthSSLProtocolSocketFactory(new URL("file:my.keystore"), "mypassword", new URL("file:my.truststore"), "mypassword"), 443); Protocol.registerProtocol("https", authhttps);

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 RouteBuilder class you can hook it up like so:

javaHttpComponent httpComponent = getContext().getComponent("http", HttpComponent.class); httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());

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

xml<bean id="myHttpClientConfigurer" class="my.https.HttpClientConfigurer"/> <to uri="https://myhostname.com:443/myURL?httpClientConfigurerRef=myHttpClientConfigurer"/>

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

Endpoint See Also

- Jetty