Mail

Mail Component

The mail component provides access to Email via Spring's Mail support and the underlying JavaMail system.

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-mail</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency> Geronimo mail .jar
We have discovered that the geronimo mail `.jar` (v1.6) has a bug when polling mails with attachments. It cannot correctly identify the `Content-Type`. So, if you attach a `.jpeg` file to a mail and you poll it, the `Content-Type` is resolved as `text/plain` and not as `image/jpeg`. For that reason, we have added an `org.apache.camel.component.ContentTypeResolver` SPI interface which enables you to provide your own implementation and fix this bug by returning the correct Mime type based on the file name. So if the file name ends with `jpeg/jpg`, you can return `image/jpeg`.

You can set your custom resolver on the `MailComponent` instance or on the `MailEndpoint` instance.
POP3 or IMAP
POP3 has some limitations and end users are encouraged to use IMAP if possible.
Using mock-mail for testing
You can use a mock framework for unit testing, which allows you to test without the need for a real mail server. However you should remember to not include the mock-mail when you go into production or other environments where you need to send mails to a real mail server. Just the presence of the mock-javamail.jar on the classpath means that it will kick in and avoid sending the mails.

URI format

Mail endpoints can have one of the following URI formats (for the protocols, SMTP, POP3, or IMAP, respectively):

smtp://[username@]host[:port][?options] pop3://[username@]host[:port][?options] imap://[username@]host[:port][?options]

The mail component also supports secure variants of these protocols (layered over SSL). You can enable the secure protocols by adding `s` to the scheme:

smtps://[username@]host[:port][?options] pop3s://[username@]host[:port][?options] imaps://[username@]host[:port][?options]

You can append query options to the URI in the following format, `?option=value&option=value&...`

Sample endpoints

Typically, you specify a URI with login credentials as follows (taking SMTP as an example):

smtp://[username@]host[:port][?password=somepwd]

Alternatively, it is possible to specify both the user name and the password as query options:

smtp://host[:port]?password=somepwd&username=someuser

For example:

smtp://mycompany.mailserver:30?password=tiger&username=scott

DefaultPortsDefault ports

Default port numbers are supported. If the port number is omitted, Camel determines the port number to use based on the protocol.

confluenceTableSmall

Protocol	Default Port Number
`SMTP`	`25`
`SMTPS`	`465`
`POP3`	`110`
`POP3S`	`995`
`IMAP`	`143`
`IMAPS`	`993`

Options

confluenceTableSmall

Property	Default	Description
`host`		The host name or IP address to connect to.

`port`	See #DefaultPorts	The TCP port number to connect on.
`username`		The user name on the email server.
`password`	`null`	The password on the email server.
`ignoreUriScheme`	`false`	If `false`, Camel uses the scheme to determine the transport protocol (POP, IMAP, SMTP etc.)
`contentType`	`text/plain`	The mail message content type. Use `text/html` for HTML mails.
`folderName`	`INBOX`	The folder to poll.
`destination`	`username@host`	**@deprecated** Use the `to` option instead. The `TO` recipients (receivers of the email).
`to`	`username@host`	The TO recipients (the receivers of the mail). Separate multiple email addresses with a comma. Email addresses containing special characters such as "&" will need to be handled differently - see How do I configure password options on Camel endpoints without the value being encoded.
`replyTo`	`alias@host`	As of **Camel 2.8.4, 2.9.1+**, the Reply-To recipients (the receivers of the response mail). Separate multiple email addresses with a comma.
`cc`	`null`	The CC recipients (the receivers of the mail). Separate multiple email addresses with a comma.
`bcc`	`null`	The BCC recipients (the receivers of the mail). Separate multiple email addresses with a comma.
`from`	`camel@localhost`	The FROM email address.
`subject`		As of **Camel 2.3**, the Subject of the message being sent. Note: Setting the subject in the header takes precedence over this option.
`peek`	`true`	**Camel 2.11.3/2.12.2:** Consumer only. Will mark the `javax.mail.Message` as peeked before processing the mail message. This applies to `IMAPMessage` messages types only. By using peek the mail will not be eager marked as `SEEN` on the mail server, which allows us to rollback the mail message if there is an error processing in Camel.
`delete`	`false`	Deletes the messages after they have been processed. This is done by setting the `DELETED` flag on the mail message. If `false`, the `SEEN` flag is set instead. As of **Camel 2.10** you can override this configuration option by setting a header with the key `delete` to determine if the mail should be deleted or not.
`unseen`	`true`	It is possible to configure a consumer endpoint so that it processes only unseen messages (that is, new messages) or all messages. Note that Camel always skips deleted messages. The default option of `true` will filter to only unseen messages. POP3 does not support the `SEEN` flag, so this option is not supported in POP3; use IMAP instead. **Important:** This option is **not** in use if you also use `searchTerm` options. Instead if you want to disable unseen when using `searchTerm`'s then add `searchTerm.unseen=false` as a term.
`copyTo`	`null`	**Camel 2.10:** Consumer only. After processing a mail message, it can be copied to a mail folder with the given name. You can override this configuration value, with a header with the key `copyTo`, allowing you to copy messages to folder names configured at runtime.
`fetchSize`	`-1`	Sets the maximum number of messages to consume during a poll. This can be used to avoid overloading a mail server, if a mailbox folder contains a lot of messages. Default value of `-1` means no fetch size and all messages will be consumed. Setting the value to 0 is a special corner case, where Camel will not consume any messages at all.
`alternativeBodyHeader`	`CamelMailAlternativeBody`	Specifies the key to an IN message header that contains an alternative email body. For example, if you send emails in `text/html` format and want to provide an alternative mail body for non-HTML email clients, set the alternative mail body with this key as a header.
`debugMode`	`false`	Enable debug mode on the underlying mail framework. The SUN Mail framework logs the debug messages to `System.out` by default.
`connectionTimeout`	`30000`	The connection timeout in milliseconds. Default is 30 seconds.
`consumer.initialDelay`	`1000`	Milliseconds before the polling starts.
`consumer.delay`	`60000`	Camel will poll the mailbox only once a minute by default to avoid overloading the mail server.

`consumer.useFixedDelay`	`false`	Set to `true` to use a fixed delay between polls, otherwise fixed rate is used. See ScheduledExecutorService in JDK for details.
`disconnect`	`false`	**Camel 2.8.3/2.9:** Whether the consumer should disconnect after polling. If enabled this forces Camel to connect on each poll.
`closeFolder`	`true`	**Camel 2.10.4:** Whether the consumer should close the folder after polling. Setting this option to `false` and having `disconnect=false` as well, then the consumer keep the folder open between polls.
`mail.XXX`	`null`	Set any additional java mail properties. For instance if you want to set a special property when using POP3 you can now provide the option directly in the URI such as: `mail.pop3.forgettopheaders=true`. You can set multiple such options, for example: `mail.pop3.forgettopheaders=true&mail.mime.encodefilename=true`.
`mapMailMessage`	`true`	**Camel 2.8:** Specifies whether Camel should map the received mail message to Camel body/headers. If set to true, the body of the mail message is mapped to the body of the Camel IN message and the mail headers are mapped to IN headers. If this option is set to false then the IN message contains a raw `javax.mail.Message`. You can retrieve this raw message by calling `exchange.getIn().getBody(javax.mail.Message.class)`.
`maxMessagesPerPoll`	`0`	Specifies the maximum number of messages to gather per poll. By default, no maximum is set. Can be used to set a limit of e.g. 1000 to avoid downloading thousands of files when the server starts up. Set a value of 0 or negative to disable this option.
`javaMailSender`	`null`	Specifies a pluggable org.apache.camel.component.mail.JavaMailSender instance in order to use a custom email implementation.
`ignoreUnsupportedCharset`	`false`	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then `charset=XXX` (where XXX represents the unsupported charset) is removed from the `content-type` and it relies on the platform default instead.
`sslContextParameters`	`null`	**Camel 2.10:** Reference to a `org.apache.camel.util.jsse.SSLContextParameters` in the Registry. This reference overrides any configured SSLContextParameters at the component level. See Using the JSSE Configuration Utility.
`searchTerm`	`null`	**Camel 2.11:** Refers to a `javax.mail.search.SearchTerm` which allows to filter mails based on search criteria such as subject, body, from, sent after a certain date etc. See further below for examples.
`searchTerm.xxx`	`null`	**Camel 2.11:** To configure search terms directly from the endpoint uri, which supports a limited number of terms defined by the `org.apache.camel.component.mail.SimpleSearchTerm` class. See further below for examples.
`sortTerm`	`null`	**Camel 2.15:** To configure the sortTerms that IMAP supports to sort the searched mails**.** You may need to define an array of `com.sun.mail.imap.sortTerm` in the registry first and #name to reference it in this URI option. **Camel 2.16:** You can also specify a comma separated list of sort terms on the URI that Camel will convert internally. For example, to sort descending by date you would use `sortTerm=reverse,date`. You can use any of the sort terms defined in com.sun.mail.imap.SortTerm.
`postProcessAction`	`null`	**Camel 2.15:** Refers to a`org.apache.camel.component.mail.MailBoxPostProcessAction` for doing post processing tasks on the mailbox once the normal processing ended.
`skipFailedMessage`	`false`	**Camel 2.15.1:** If the mail consumer cannot retrieve a given mail message, then this option allows to skip the message and move on to retrieve the next mail message. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.
`handleFailedMessage`	`false`	**Camel 2.15.1:** If the mail consumer cannot retrieve a given mail message, then this option allows to handle the caused exception by the consumer's error handler. By enable the bridge error handler on the consumer, then the Camel routing error handler can handle the exception instead. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.
`dummyTrustManager`	`false`	**Camel 2.17:**To use a dummy security setting for trusting all certificates. Should only be used for development mode, and not production.
`idempotentRepository`	`null`	**Camel 2.17:** A pluggable repository org.apache.camel.spi.IdempotentRepository which allows to cluster consuming from the same mailbox, and let the repository coordinate whether a mail message is valid for the consumer to process.
`idempotentRepositoryRemoveOnCommit`	`true`	**Camel 2.17:** When using idempotent repository, then when the mail message has been successfully processed and is committed, should the message id be removed from the idempotent repository (default) or be kept in the repository. By default its assumed the message id is unique and has no value to be kept in the repository, because the mail message will be marked as seen/moved or deleted to prevent it from being consumed again. And therefore having the message id stored in the idempotent repository has little value. However this option allows to store the message id, for whatever reason you may have.

`mailUidGenerator`		**Camel 2.17:** A pluggable MailUidGenerator that allows to use custom logic to generate UUID of the mail message.

SSL support

The underlying mail framework is responsible for providing SSL support. You may either configure SSL/TLS support by completely specifying the necessary Java Mail API configuration options, or you may provide a configured SSLContextParameters through the component or endpoint configuration.

Using the JSSE Configuration Utility

As of **Camel 2.10**, the mail 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 mail component.

Programmatic configuration of the endpoint

KeyStoreParameters ksp = new KeyStoreParameters(); ksp.setResource("/users/home/server/truststore.jks"); ksp.setPassword("keystorePassword"); TrustManagersParameters tmp = new TrustManagersParameters(); tmp.setKeyStore(ksp); SSLContextParameters scp = new SSLContextParameters(); scp.setTrustManagers(tmp); Registry registry = ... registry.bind("sslContextParameters", scp); ... from(...)   .to("smtps://smtp.google.com?username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters");

Spring DSL based configuration of endpoint

xml... <camel:sslContextParameters id="sslContextParameters"> <camel:trustManagers> <camel:keyStore resource="/users/home/server/truststore.jks" password="keystorePassword"/> </camel:trustManagers> </camel:sslContextParameters>... ... <to uri="smtps://smtp.google.com?username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters"/>...

Configuring JavaMail Directly

Camel uses SUN JavaMail, which only trusts certificates issued by well known Certificate Authorities (the default JVM trust configuration). If you issue your own certificates, you have to import the CA certificates into the JVM's Java trust/key store files, override the default JVM trust/key store files (see `SSLNOTES.txt` in JavaMail for details).

Mail Message Content

Camel uses the message exchange's IN body as the MimeMessage text content. The body is converted to `String.class`.

Camel copies all of the exchange's IN headers to the MimeMessage headers. You may wish to read How to avoid sending some or all message headers to prevent inadvertent data "leaks" from your application.

The subject of the MimeMessage can be configured using a header property on the IN message. The code below demonstrates this:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mail/src/test/java/org/apache/camel/component/mail/MailSubjectTest.java}The same applies for other MimeMessage headers such as recipients, so you can use a header property as `To`:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mail/src/test/java/org/apache/camel/component/mail/MailUsingHeadersTest.java}**Since Camel 2.11** When using the MailProducer the send the mail to server, you should be able to get the message id of the MimeMessage with the key `CamelMailMessageId` from the Camel message header.

Headers take precedence over pre-configured recipients

The recipients specified in the message headers always take precedence over recipients pre-configured in the endpoint URI. The idea is that if you provide any recipients in the message headers, that is what you get. The recipients pre-configured in the endpoint URI are treated as a fallback.

In the sample code below, the email message is sent to `davsclaus@apache.org`, because it takes precedence over the pre-configured recipient, `info@mycompany.com`. Any `cc` and `bcc` settings in the endpoint URI are also ignored and those recipients will not receive any mail. The choice between headers and pre-configured settings is all or nothing: the mail component *either* takes the recipients exclusively from the headers or exclusively from the pre-configured settings. It is not possible to mix and match headers and pre-configured settings.

java Map<String, Object> headers = new HashMap<String, Object>(); headers.put("to", "davsclaus@apache.org"); template.sendBodyAndHeaders ("smtp://admin@localhost?to=info@mycompany.com", "Hello World", headers);

Multiple recipients for easier configuration

It is possible to set multiple recipients using a comma-separated or a semicolon-separated list. This applies both to header settings and to settings in an endpoint URI. For example:

java Map<String, Object> headers = new HashMap<String, Object>(); headers.put("to", "davsclaus@apache.org ; jstrachan@apache.org ; ningjiang@apache.org");

The preceding example uses a semicolon, `;`, as the separator character.

Setting sender name and email

You can specify recipients in the format, `name <email>`, to include both the name and the email address of the recipient.

For example, you define the following headers on the a Message:

Map headers = new HashMap(); map.put("To", "Claus Ibsen <davsclaus@apache.org>"); map.put("From", "James Strachan <jstrachan@apache.org>"); map.put("Subject", "Camel is cool");

JavaMail API (ex SUN JavaMail)

JavaMail API is used under the hood for consuming and producing mails.
We encourage end-users to consult these references when using either POP3 or IMAP protocol. Note particularly that POP3 has a much more limited set of features than IMAP.

- JavaMail POP3 API
- JavaMail IMAP API
- And generally about the MAIL Flags

Samples

We start with a simple route that sends the messages received from a JMS queue as emails. The email account is the `admin` account on `mymailserver.com`.

from("jms://queue:subscription").to("smtp://admin@mymailserver.com?password=secret");

In the next sample, we poll a mailbox for new emails once every minute. Notice that we use the special `consumer` option for setting the poll interval, `consumer.delay`, as 60000 milliseconds = 60 seconds.

from("imap://admin@mymailserver.com password=secret&unseen=true&consumer.delay=60000") .to("seda://mails");

In this sample we want to send a mail to multiple recipients:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mail/src/test/java/org/apache/camel/component/mail/MailRecipientsTest.java}

Sending mail with attachment sample

Attachments are not support by all Camel components
The *Attachments API* is based on the Java Activation Framework and is generally only used by the Mail API. Since many of the other Camel components do not support attachments, the attachments could potentially be lost as they propagate along the route. The rule of thumb, therefore, is to add attachments just before sending a message to the mail endpoint.

The mail component supports attachments. In the sample below, we send a mail message containing a plain text message with a logo file attachment.{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mail/src/test/java/org/apache/camel/component/mail/MailAttachmentTest.java}

SSL sample

In this sample, we want to poll our Google mail inbox for mails. To download mail onto a local mail client, Google mail requires you to enable and configure SSL. This is done by logging into your Google mail account and changing your settings to allow IMAP access. Google have extensive documentation on how to do this.

from("imaps://imap.gmail.com?username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD" + "&delete=false&unseen=true&consumer.delay=60000").to("log:newmail");

The preceding route polls the Google mail inbox for new mails once every minute and logs the received messages to the `newmail` logger category. Running the sample with `DEBUG` logging enabled, we can monitor the progress in the logs:

2008-05-08 06:32:09,640 DEBUG MailConsumer - Connecting to MailStore imaps//imap.gmail.com:993 (SSL enabled), folder=INBOX 2008-05-08 06:32:11,203 DEBUG MailConsumer - Polling mailfolder: imaps//imap.gmail.com:993 (SSL enabled), folder=INBOX 2008-05-08 06:32:11,640 DEBUG MailConsumer - Fetching 1 messages. Total 1 messages. 2008-05-08 06:32:12,171 DEBUG MailConsumer - Processing message: messageNumber=[332], from=[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[... 2008-05-08 06:32:12,187 INFO newmail - Exchange[MailMessage: messageNumber=[332], from=[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...

Consuming mails with attachment sample

In this sample we poll a mailbox and store all attachments from the mails as files. First, we define a route to poll the mailbox. As this sample is based on google mail, it uses the same route as shown in the SSL sample:

from("imaps://imap.gmail.com?username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD" + "&delete=false&unseen=true&consumer.delay=60000").process(new MyMailProcessor());

Instead of logging the mail we use a processor where we can process the mail from java code:

public void process(Exchange exchange) throws Exception { // the API is a bit clunky so we need to loop Map<String, DataHandler> attachments = exchange.getIn().getAttachments(); if (attachments.size() > 0) { for (String name : attachments.keySet()) { DataHandler dh = attachments.get(name); // get the file name String filename = dh.getName(); // get the content and convert it to byte[] byte[] data = exchange.getContext().getTypeConverter() .convertTo(byte[].class, dh.getInputStream()); // write the data to a file FileOutputStream out = new FileOutputStream(filename); out.write(data); out.flush(); out.close(); } } }

As you can see the API to handle attachments is a bit clunky but it's there so you can get the `javax.activation.DataHandler` so you can handle the attachments using standard API.

How to split a mail message with attachments

In this example we consume mail messages which may have a number of attachments. What we want to do is to use the Splitter EIP per individual attachment, to process the attachments separately. For example if the mail message has 5 attachments, we want the Splitter to process five messages, each having a single attachment. To do this we need to provide a custom Expression to the Splitter where we provide a List<Message> that contains the five messages with the single attachment.

The code is provided out of the box in Camel 2.10 onwards in the `camel-mail` component. The code is in the class: `org.apache.camel.component.mail.SplitAttachmentsExpression`, which you can find the source code here

In the Camel route you then need to use this Expression in the route as shown below:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mail/src/test/java/org/apache/camel/component/mail/MailSplitAttachmentsTest.java}If you use XML DSL then you need to declare a method call expression in the Splitter as shown below

xml<split> <method beanType="org.apache.camel.component.mail.SplitAttachmentsExpression"/> <to uri="mock:split"/> </split>

From Camel 2.16 onwards you can also split the attachments as byte[] to be stored as the message body. This is done by creating the expression with boolean true

SplitAttachmentsExpression split = SplitAttachmentsExpression(true);

And then use the expression with the splitter eip.

Using custom SearchTerm

Available as of Camel 2.11

You can configure a `searchTerm` on the `MailEndpoint` which allows you to filter out unwanted mails.

For example to filter mails to contain Camel in either Subject or Text you can do as follows:

xml<route> <from uri="imaps://mymailseerver?username=foo&password=secret&searchTerm.subjectOrBody=Camel"/> <to uri="bean:myBean"/> </route>

Notice we use the `"searchTerm.subjectOrBody"` as parameter key to indicate that we want to search on mail subject or body, to contain the word "Camel".
The class `org.apache.camel.component.mail.SimpleSearchTerm` has a number of options you can configure:

Or to get the new unseen emails going 24 hours back in time you can do. Notice the "now-24h" syntax. See the table below for more details.

xml<route> <from uri="imaps://mymailseerver?username=foo&password=secret&searchTerm.fromSentDate=now-24h"/> <to uri="bean:myBean"/> </route>

You can have multiple searchTerm in the endpoint uri configuration. They would then be combined together using AND operator, eg so both conditions must match. For example to get the last unseen emails going back 24 hours which has Camel in the mail subject you can do:

xml<route> <from uri="imaps://mymailseerver?username=foo&password=secret&searchTerm.subject=Camel&searchTerm.fromSentDate=now-24h"/> <to uri="bean:myBean"/> </route> confluenceTableSmall

Option	Default	Description
unseen	`true`	Whether to limit by unseen mails only.
subjectOrBody	`null`	To limit by subject or body to contain the word.
subject	`null`	The subject must contain the word.
body	`null`	The body must contain the word.
from	`null`	The mail must be from a given email pattern.
to	`null`	The mail must be to a given email pattern.
fromSentDate	`null`	The mail must be sent after or equals (GE) a given date. The date pattern is `yyyy-MM-dd HH:mm:SS`, eg use `"2012-01-01 00:00:00"` to be from the year 2012 onwards. You can use `"now"` for current timestamp. The "now" syntax supports an optional offset, that can be specified as either + or - with a numeric value. For example for last 24 hours, you can use `"now - 24h"` or without spaces `"now-24h"`. Notice that Camel supports shorthands for hours, minutes, and seconds.
toSentDate	`null`	The mail must be sent before or equals (BE) a given date. The date pattern is `yyyy-MM-dd HH:mm:SS`, eg use `"2012-01-01 00:00:00"` to be before the year 2012. You can use `"now"` for current timestamp. The "now" syntax supports an optional offset, that can be specified as either + or - with a numeric value. For example for last 24 hours, you can use `"now - 24h"` or without spaces `"now-24h"`. Notice that Camel supports shorthands for hours, minutes, and seconds.

The `SimpleSearchTerm` is designed to be easily configurable from a POJO, so you can also configure it using a <bean> style in XML

<bean id="mySearchTerm" class="org.apache.camel.component.mail.SimpleSearchTerm"> <property name="subject" value="Order"/> <property name="to" value="acme-order@acme.com"/> <property name="fromSentDate" value="now"/> </bean>

You can then refer to this bean, using #beanId in your Camel route as shown:

xml<route> <from uri="imaps://mymailseerver?username=foo&password=secret&searchTerm=#mySearchTerm"/> <to uri="bean:myBean"/> </route>

In Java there is a builder class to build compound `SearchTerm}}s using the {{org.apache.camel.component.mail.SearchTermBuilder` class.
This allows you to build complex terms such as:

// we just want the unseen mails which is not spam SearchTermBuilder builder = new SearchTermBuilder(); builder.unseen().body(Op.not, "Spam").subject (Op.not, "Spam") // which was sent from either foo or bar .from("foo@somewhere.com").from(Op.or, "bar@somewhere.com"); // .. and we could continue building the terms SearchTerm term = builder.build();

Endpoint See Also