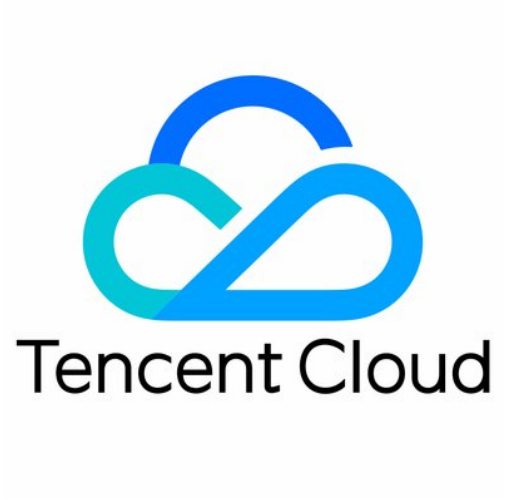
Simple Email Service

SMTP Documentation

Product Documentation



Contents

SMTP Documentation
SMTP Email Sending Guide

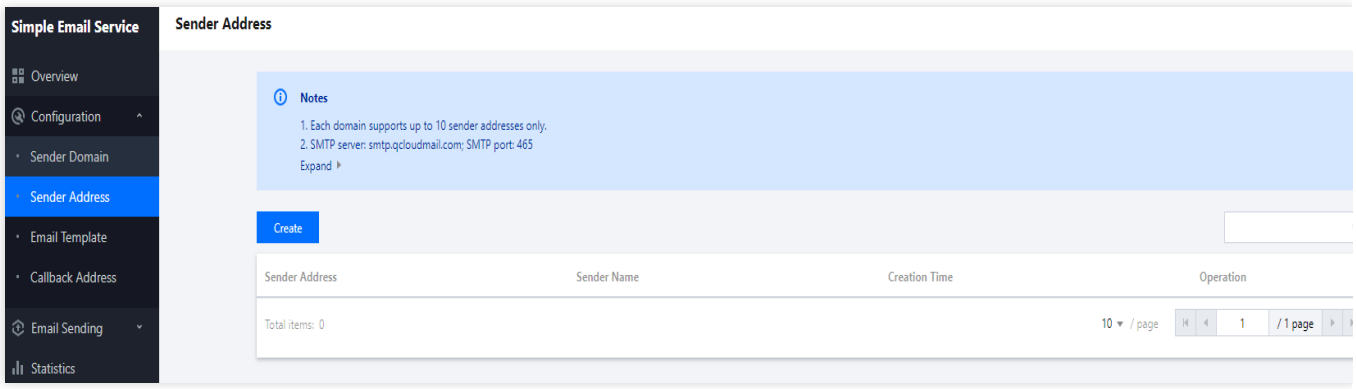
Last updated：2024-09-05 09:47:39

Enabling SMTP Sending Feature

Directions

Step 1. Go to the sender address page

Log in to the SES console and click **Configuration** > **Sender Address** on the left sidebar to enter the sender address page.



Step 2. Configure the SMTP password

1. In the sender address list, find the one for which you want to enable SMTP sending, and click **Set SMTP Password** in the **Operation** column.
2. Enter the SMTP password in the pop-up window and click **OK**.

Sending Email via SMTP API

For the SMTP sample code and specific request parameters, response parameters, and error codes, see SMTP Sample Call.

Note:

Emails with attachments can be sent as instructed in Sending Email with Attachment via SMTP.

SMTP Sending Frequency

The current call rate of the SMTP API is limited to 20 times per second under the same Tencent Cloud account `appId` . In addition, the same sender can send up to 10 emails per hour to the same recipient.
We will deliver emails as soon as possible after receiving them. However, due to the different traffic throttling and reputation protection policies of different email systems, in order to improve your email delivery success rate, we recommend you send emails at a lower frequency.

SMTP Service Address

Last updated：2023-12-22 10:27:15

The SMTP service addresses are as follows:

Address	Description
SMTP service address (Hongkong, Tencent Cloud China)	smtp.qcloudmail.com
SMTP service address (Guangzhou, Tencent Cloud China)	gz-smtp.qcloudmail.com
SMTP service address (Tencent Cloud International)	sg-smtp.qcloudmail.com

The SMTP ports are as follows:

Port	Description
SMTP port	465 (SSL encryption)

Note:

For security reasons, port 25 is currently disabled.

Sample Call for Java

Last updated：2023-12-22 10:27:29

The following sample code is a demo on JDK 1.8:

```
package org.example;

import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.Properties;

public class SampleMail {
    private static final String SMTP_HOST = "sg-smtp.qcloudmail.com";
    private static final String SMTP_PORT = "465";

    public static void main(String[] args) {
        // Configure the environment attributes for email sending
        final Properties props = new Properties();
        // Indicate that SMTP is used to send the email, which requires
authentication
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.host", SMTP_HOST);
        // If SSL is used, remove the configuration of using port 25 and
perform the following configuration:
        props.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
        props.put("mail.smtp.socketFactory.port", SMTP_PORT);
        props.put("mail.smtp.port", SMTP_PORT);
        // Sender account. Enter the sender address configured in the console,
such as xxx@xxx.com
        props.put("mail.user", "xxx@xxx.com");
        // The password that needs to be provided when the SMTP service is
accessed (select the sender address in the console to configure)
        props.put("mail.password", "XXXX");
        props.setProperty("mail.smtp.socketFactory.fallback", "false");
        props.put("mail.smtp.ssl.enable", "true");
        //props.put("mail.smtp.starttls.enable","true");
        // Build the authorization information for SMTP authentication
        Authenticator authenticator = new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                // Username and password
```

```
                String userName = props.getProperty("mail.user");
                String password = props.getProperty("mail.password");
                return new PasswordAuthentication(userName, password);
            }
        };
        // Create the email session with the environment attributes and
authorization information
        Session mailSession = Session.getInstance(props, authenticator);
//        mailSession.setDebug(true);
        //UUID uuid = UUID.randomUUID();
        //final String messageIDValue = "<" + uuid.toString() + ">";
        // Create the email message
        MimeMessage message = new MimeMessage(mailSession) {
            //@Override
            //protected void updateMessageID() throws MessagingException {
            // Set the custom `Message-ID` value
            //setHeader("Message-ID", messageIDValue);
            //}
        };
        try {
            // Set the sender email address and name. Here, enter the sender
address configured in the console (which must be the same as the `mail.user`
above), such as xxx@xxx.com. The name can be customized
            InternetAddress from = new InternetAddress("xxx@xxx.com", "test");
            message.setFrom(from);
            // (Optional) Set the reply-to address
//            Address[] a = new Address[1];
//            a[0] = new InternetAddress("***");
//            message.setReplyTo(a);
            // Set the recipient's email address, such as yyy@yyy.com
            InternetAddress to = new InternetAddress("xxx@xxx.com");
            message.setRecipient(MimeMessage.RecipientType.TO, to);
            // If the email is to be sent to multiple recipients at the same
time, replace the above two lines with the following (due to the restrictions
of some emailing systems, we recommend you try to send the email to one
recipient at a time; plus, the email can be sent to up to 50 recipients at a
time):
            //InternetAddress[] adds = new InternetAddress[2];
            //adds[0] = new InternetAddress("xxx@xxx.com");
            //adds[1] = new InternetAddress("xxx@xxx.com");
            //message.setRecipients(Message.RecipientType.TO, adds);

            // Set the email subject
            message.setSubject("Test email");
            message.setHeader("Content-Transfer-Encoding", "base64");
            // Set the email body type: `text/plain` (plain text) or
`text/html` (HTML document)
```

```
            message.setContent("<!DOCTYPE html>\\n<html>\\n<head>\\n<meta
charset=\\"utf-8\\">\\n<title>hello world</title>\\n</head>\\n<body>\\n " +
                    "<h1>My first heading</h1>\\n    <p>My first paragraph.
</p>\\n</body>\\n</html>", "text/html;charset=UTF-8");
            // Send the email
            Transport.send(message);
        } catch (MessagingException | UnsupportedEncodingException e) {
            String err = e.getMessage();
            err = new String(err.getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8);
            System.out.println(err);
        }
    }

}
```

Sending Attachment

```
package org.example;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.Properties;
import java.util.UUID;

public class SampleMailAttach {
    private static final String SMTP_HOST = "sg-smtp.qcloudmail.com";
    private static final String SMTP_PORT = "465";

    public static void main(String[] args) {
        // Configure the environment attributes for email sending
        final Properties props = new Properties();
        // Indicate that SMTP is used to send the email, which requires
authentication
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.host", SMTP_HOST);
        // If SSL is used, remove the configuration of using port 25 and
perform the following configuration:
        props.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
```

```
        props.put("mail.smtp.socketFactory.port", SMTP_PORT);
        props.put("mail.smtp.port", SMTP_PORT);
        // Sender account. Enter the sender address configured in the console,
such as xxx@xxx.com
        props.put("mail.user", "xxx@xxx.com");
        // The password that needs to be provided when the SMTP service is
accessed (select the sender address in the console to configure)
        props.put("mail.password", "XXXX");
        props.setProperty("mail.smtp.socketFactory.fallback", "false");
        props.put("mail.smtp.ssl.enable", "true");
        //props.put("mail.smtp.starttls.enable","true");
        // Build the authorization information for SMTP authentication
        Authenticator authenticator = new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                // Username and password
                String userName = props.getProperty("mail.user");
                String password = props.getProperty("mail.password");
                return new PasswordAuthentication(userName, password);
            }
        };
        // Create the email session with the environment attributes and
authorization information
        Session mailSession = Session.getInstance(props, authenticator);

        UUID uuid = UUID.randomUUID();
        final String messageIDValue = "<" + uuid.toString() + ">";
        // Create the email message
        MimeMessage message = new MimeMessage(mailSession) {
            @Override
            protected void updateMessageID() throws MessagingException {
                // Set the custom `Message-ID` value
                setHeader("Message-ID", messageIDValue);
            }
        };
        try {
            // Set the sender email address and name. Here, enter the sender
address configured in the console (which must be the same as the `mail.user`
above). The name can be customized, such as test
            InternetAddress from = new InternetAddress("xxx@xxx.com", "test");
            message.setFrom(from);
            // (Optional) Set the reply-to address
            Address[] a = new Address[1];
            a[0] = new InternetAddress("xxx@xxx.com");
            message.setReplyTo(a);
            // Set the recipient's email address, such as yyy@yyy.com
            InternetAddress to = new InternetAddress("xxx@xxx.com");
```

```
            message.setRecipient(MimeMessage.RecipientType.TO, to);
            // If the email is to be sent to multiple recipients at the same
time, replace the above two lines with the following (due to the restrictions
of some emailing systems, we recommend you try to send the email to one
recipient at a time; plus, the email can be sent to up to 50 recipients at a
time):
            /*InternetAddress[] adds = new InternetAddress[2];
            adds[0] = new InternetAddress("xxx@xxx.com");
            adds[1] = new InternetAddress("xxx@xxx.com");
            message.setRecipients(Message.RecipientType.TO, adds);*/

            // Set the email subject
            message.setSubject("Test email");
            // Send the attachment. The total message size does not exceed 10M,
create a message part
            BodyPart messageBodyPart = new MimeBodyPart();
            // Message body: `text/plain` (plain text) or `text/html` (HTML
document)
            messageBodyPart.setText("<!DOCTYPE html>\\n<html>\\n<head>\\n<meta
charset=\\"utf-8\\">\\n<title>hello world</title>\\n</head>\\n<body>\\n " +
                    "<h1>My first heading</h1>\\n    <p>My first paragraph.
</p>\\n</body>\\n</html>");
            messageBodyPart.setHeader("Content-Type", "text/plain;charset=utf-
8");
            // Create the multipart message
            Multipart multipart = new MimeMultipart();
            // Set the text message part
            multipart.addBodyPart(messageBodyPart);
            // Attachment part
            messageBodyPart = new MimeBodyPart();
            // Set the path of the attachment file
            String filename = "/Users/aaa/bbb/a.txt";
            FileDataSource source = new FileDataSource(filename);
            messageBodyPart.setDataHandler(new DataHandler(source));
            // Handle the garbled text problem of the attachment name in
Chinese (attached file path)
            String filenameEncode = MimeUtility.encodeText(filename, "UTF-8",
"base64");
            messageBodyPart.setFileName(filenameEncode);
            messageBodyPart.setHeader("Content-Transfer-Encoding", "base64");
            messageBodyPart.setHeader("Content-Disposition", "attachment");
            messageBodyPart.setHeader("Content-Type", "application/octet-
stream;name=\\"" + filenameEncode + "\\"");
            multipart.addBodyPart(messageBodyPart);

            // Attachment part. Multiple attachments should be divided into
multiple parts
```

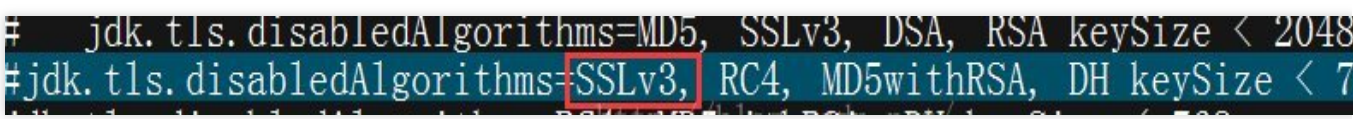
```
            BodyPart messageBodyPart1 = new MimeBodyPart();
            // Set the path of the attachment file
            String filename1 = "/Users/aaa/bbb/b.txt";
            FileDataSource source1 = new FileDataSource(filename1);
            messageBodyPart1.setDataHandler(new DataHandler(source1));
            // Handle the garbled text problem of the attachment name in
Chinese (attached file path)
            String filenameEncode1 = MimeUtility.encodeText(filename1, "UTF-8",
"base64");
            messageBodyPart1.setHeader("Content-Transfer-Encoding", "base64");
            messageBodyPart1.setHeader("Content-Disposition", "attachment");
            messageBodyPart1.setHeader("Content-Type", "application/octet-
stream;name=\\"" + filenameEncode1 + "\\"");
            multipart.addBodyPart(messageBodyPart1);

            // Send the complete message with attachments
            message.setContent(multipart);
            // Code for sending attachments (end)
            // Send the email
            Transport.send(message);
        } catch (MessagingException | UnsupportedEncodingException e) {
            String err = e.getMessage();
            err = new String(err.getBytes(StandardCharsets.ISO_8859_1),
StandardCharsets.UTF_8);
            System.out.println(err);
        }
    }
}
```

FAQs

How do I fix the error "No appropriate protocol (protocol is disabled or cipher suites are inappropriate)"?

Find the `jdk/jre/lib/security/java.security` file and modify it:



Sample Call for Go

Last updated：2024-12-02 16:28:58

The following sample code uses SMTP to send an email in Go language (v1.16):

```
package main

import (
    "crypto/tls"
    "fmt"
    "log"
    "net"
    "net/smtp"
)

// Test465  for port 465
func Test465() error {
    host := "sg-smtp.qcloudmail.com"
    port := 465
    // Sender address created in the console
    email := "abc@cd.com"
    // SMTP password set in the console
    password := "****"
    toEmail := "test@test123.com"
    ccEmail := "cc@test123.com"
    bccEmail := "bcc@test123.com"
    header := make(map[string]string)
    header["From"] = "test " + "<" + email + ">"
    header["To"] = toEmail
    header["Cc"] = ccEmail
    header["Bcc"] = bccEmail
    header["Subject"] = "test subject"
    // HTML email
    header["Content-Type"] = "text/html; charset=UTF-8"
    body := "<!DOCTYPE html>\\n<html>\\n<head>\\n<meta charset=\\"utf-
8\\">\\n<title>hello world</title>\\n</head>\\n<body>\\n " +
        "<h1>My first heading</h1>\\n    <p>My first paragraph.
</p>\\n</body>\\n</html>"
    // Plain text email
    //header["Content-Type"] = "text/plain; charset=UTF-8"
    //body := "test body"
    message := ""
    for k, v := range header {
        message += fmt.Sprintf("%s: %s\\r\\n", k, v)
    }
```

```
    message += "\\r\\n" + body
    auth := smtp.PlainAuth(
        "",
        email,
        password,
        host,
    )
    err := SendMailWithTLS(
        fmt.Sprintf("%s:%d", host, port),
        auth,
        email,
        []string{toEmail},
        []byte(message),
    )
    if err != nil {
        fmt.Println("Send email error:", err)
    } else {
        fmt.Println("Send mail success!")
    }
    return err
}

// Dial return a smtp client
func Dial(addr string) (*smtp.Client, error) {
    conn, err := tls.Dial("tcp", addr, nil)
    if err != nil {
        log.Println("tls.Dial Error:", err)
        return nil, err
    }

    host, _, _ := net.SplitHostPort(addr)
    return smtp.NewClient(conn, host)
}

// SendMailWithTLS send email with tls
func SendMailWithTLS(addr string, auth smtp.Auth, from string,
    to []string, msg []byte) (err error) {
    //create smtp client
    c, err := Dial(addr)
    if err != nil {
        log.Println("Create smtp client error:", err)
        return err
    }
    defer c.Close()
    if auth != nil {
        if ok, _ := c.Extension("AUTH"); ok {
            if err = c.Auth(auth); err != nil {
```

```
                log.Println("Error during AUTH", err)
                return err
            }
        }
    }
    if err = c.Mail(from); err != nil {
        return err
    }
    for _, addr := range to {
        if err = c.Rcpt(addr); err != nil {
            return err
        }
    }
    w, err := c.Data()
    if err != nil {
        return err
    }
    _, err = w.Write(msg)
    if err != nil {
        return err
    }
    err = w.Close()
    if err != nil {
        return err
    }
    return c.Quit()
}

func main() {
    Test465()
}
```

Sample Call for PHP

Last updated：2023-12-22 10:28:04

Notes

1. We recommend you use the PHPMailer package:

If your project is a new one and uses composer, then just add `"phpmailer/phpmailer": "^6.5"` to `composer.json`, or run `composer require phpmailer/phpmailer` and use the following code.

If your project is an old one and does not use composer, you need to manually import PHPMailer.

2. For the service address and port, see SMTP Service Address.

Below is the sample code:

```
<?php

use PHPMailer\\PHPMailer\\PHPMailer;
use PHPMailer\\PHPMailer\\SMTP;
use PHPMailer\\PHPMailer\\Exception;
require './PHPMailer/src/Exception.php';
require './PHPMailer/src/PHPMailer.php';
require './PHPMailer/src/SMTP.php';

$mail = new PHPMailer(true);

try {
    //Server settings
    $mail->SMTPDebug  = SMTP::DEBUG_SERVER;                     //Enable verbose de
    $mail->SMTPAuth   = true;                                   //Enable SMTP authe
    //$mail->AuthType   = 'LOGIN';
    $mail->isSMTP();                                            //Send using SMTP
    $mail->Host       = 'sg-smtp.qcloudmail.com';                  //Set the SMTP s
    $mail->Username   = 'abc@qq.aa.com';          //SMTP username
    $mail->Password   = '123456';                  //SMTP password

    $mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS;            //Enable implicit T
    $mail->CharSet        = PHPMailer::CHARSET_UTF8;
    $mail->CharSet        = 'UTF-8';
    $mail->ContentType = 'text/plain; charset=UTF-8';
    $mail->Encoding   = PHPMailer::ENCODING_BASE64;
    //$mail->Encoding   = '8bit';
    $mail->Port       = 465;                                    //TCP port to conne

    //Recipients
```

```
    $mail->setFrom('abc@qq.aa.com', 'fromName');
    $mail->addAddress('test@test.com', 'toName');    //Add a recipient
    //$mail->addAddress('ellen@example.com');                  //Name is optional
    //$mail->addReplyTo('info@example.com', 'Information');
    //$mail->addCC('cc@example.com');
    //$mail->addBCC('bcc@example.com');

    //Attachments
    $mail->addAttachment('./tmp.txt');                         //Add attachments
    //$mail->addAttachment('/tmp/image.jpg', 'new.jpg');       //Optional name

    //Content
    //$mail->isHTML(true);                                       //Set email forma
    $mail->Subject = 'Here is the subject';
    $mail->Body    = 'This is the HTML message body <b>in bold!</b>';
    //$mail->AltBody = 'This is the body in plain text for non-HTML mail clients';

    $mail->send();
    echo 'Message has been sent';
} catch (Exception $e) {
    echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

Sample Call for Python

Last updated：2024-12-23 14:32:54

python version >= 3.6

```
# -*- coding:utf-8 -*-
import smtplib
import ssl
from email.header import Header
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.utils import formataddr


def send_mail(sender, sender_alias, sender_pwd, recipient_list, subject, body, host
              is_use_ssl=True):
    try:
        message = MIMEMultipart('alternative')
        message['Subject'] = Header(subject, 'UTF-8')
        message['From'] = formataddr([sender_alias, sender])
        message['To'] = ",".join(recipient_list)
        to_addr_list = recipient_list

        mime_text = MIMEText(body, _subtype='html', _charset='UTF-8')
        message.attach(mime_text)

        if is_use_ssl:
            context = ssl.create_default_context()
            context.set_ciphers('DEFAULT')
            client = smtplib.SMTP_SSL(host, port, context=context)
        else:
            client = smtplib.SMTP(host, port)

        client.login(sender, sender_pwd)
        client.sendmail(sender, to_addr_list, message.as_string())
        client.quit()

        print('Send email success!')
    except smtplib.SMTPConnectError as e:
        print('Send email failed,connection error:', e.smtp_code, e.smtp_error)
    except smtplib.SMTPAuthenticationError as e:
        print('Send email failed,smtp authentication error:', e.smtp_code, e.smtp_e
    except smtplib.SMTPSenderRefused as e:
        print('Send email failed,sender refused:', e.smtp_code, e.smtp_error)
    except smtplib.SMTPRecipientsRefused as e:
```

```
        print('Send email failed,recipients refused:', e.recipients)
    except smtplib.SMTPDataError as e:
        print('Send email failed,smtp data error:', e.smtp_code, e.smtp_error)
    except smtplib.SMTPException as e:
        print('Send email failed,smtp exception:', str(e))
    except Exception as e:
        print('Send email failed,other error:', str(e))


if __name__ == '__main__':
    # Sender address created in the console
    from_email = "xxx@xxxx"
    # SMTP password set in the console
    from_email_pwd = "xxx"
    # Recipient email address list
    to_email_list = ["xxx@xxx1", "xxx@xxx2"]

    # Sender email alias
    from_alias = "Test Alias"
    # Email subject
    subject_txt = "[Test Subject]"
    # Email content
    body_content = (
        "<!DOCTYPE html>\\n<html>\\n<head>\\n<meta charset=\\"utf-8\\">\\n<title>he
        "<h1>My First Heading</h1>\\n    <p>My First Paragraph.</p>\\n</body>\\n</h

    #  Use SSL to send emails by default; the port can be 465 or 58
    is_using_ssl = True
    # SMTP service address, Hong Kong=smtp.qcloudmail.com, Singapore=sg-smtp.qcloud
    smtp_host = "sg-smtp.qcloudmail.com"
    # SMTP port number, 465 and 587 use SSL encryption, 25 uses TLS
    smtp_port = 465

    # Use port 25 to send emails
    # is_using_ssl = False
    # smtp_host = "sg-smtp.qcloudmail.com"
    # smtp_port = 25
    send_mail(from_email, from_alias, from_email_pwd, to_email_list, subject_txt, b
              smtp_host, smtp_port, is_using_ssl)
```

Sample Call for C#

Last updated：2024-12-23 14:41:46

Due to the official library's inadequate support for ports 465 and 587, it is recommended to use MailKit, as shown in the following example.

```
using System;
using MimeKit;
using MailKit.Net.Smtp;
using MailKit.Security;

// This code example uses MailKit (MIT License: https://github.com/jstedfast/MailKi

class Program
{
    static void Main()
    {
        try
        {
            // Sender address created in the console
            string fromEmail = "test@example.com";
            // SMTP password set in the console
            string password = "your password";
            string toEmail = "to@example.com";
            string smtpHost = "sg-smtp.qcloudmail.com";
            int smtpPort = 465;
            var message = new MimeMessage();
            message.From.Add(new MailboxAddress("", fromEmail));
            message.To.Add(new MailboxAddress("", toEmail));
            message.Subject = "Test Email with HTML Content";
            message.Body = new TextPart("html")
            {
                Text = @"
                    <html>
                        <body>
                            <h1 style='color:blue;'>Hello!</h1>
                            <p>This is a <b>HTML test email</b> from <i>Tencent Clo
                            <p>Visit <a href='https://intl.cloud.tencent.com/'>this
                        </body>
                    </html>"
            };

            using (var client = new SmtpClient())
            {
                client.Connect(smtpHost, smtpPort, SecureSocketOptions.SslOnConnect
```

```
                client.Authenticate(fromEmail, password);
                client.Send(message);
                client.Disconnect(true);
            }

            Console.WriteLine("Email sent successfully.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }
}
```

Sending Email with Attachment

Last updated：2023-12-22 10:28:23

The method of sending an email with an attachment through SMTP is to construct the content of a MIME-formatted email.

Email MIME Format

For more information, see MIME Protocol.

Note:

A MIME message consists of two parts: email header and email body.

Email Header

The email header contains important information such as sender, recipient, subject, time, MIME version, and body type.

Note:

Each piece of information is called a field, which consists of a domain followed by ":" and the information content and can be in one or multiple lines.

The first line of a field must be written "to the left", i.e., without whitespace characters (spaces and tabs) on the left.

The next lines must start with whitespace characters, one of which must not be inherent in the message itself (that is, it needs to be filtered out during decoding).

Blank lines are not allowed in the email header. If the first line is blank, some emails cannot be recognized by certain email client software, and the original code is displayed.

For example:

Content	Example
Date	Mon, 29 Jun 2009 18:39:03 +0800
From	abc@123.com
To	abc1@123.com
BCC	abc3@123.com
Subject	test
Message-ID	123@123.com

Mime-Version	1.0

Field	Description
Bcc	Blind carbon copy address
Cc	Copy address
Content-Transfer-Encoding	Content transfer encoding method
Content-Type	Content type
Date	Date and time
Delivered-To	Recipient address
From	Sender address
Message-ID	Message ID
MIME-Version	MIME version
Received	Transfer path
Reply-To	Reply-to address
Return-Path	Reply-to address
Subject	Subject
To	Recipient address

Email Body

Field	Description
Content-ID	Content ID
Content-Transfer-Encoding	Content transfer encoding method
Content-Location	Content location (path)
Content-Base	Content base location
Content-Disposition	Content disposition method

Content-Type	Content type

Some fields have parameters in addition to values. Value and parameter as well as parameter and parameter are separated with ";". Parameter name and parameter value are separated with "=".

The email body contains the content of the email, whose type is indicated by the `Content-Type` field in the email header.

Note:

Common simple types include:

text/plain (plain text)

text/html (hypertext)

The multipart type is the essence of MIME emails. The email body is divided into multiple parts, each of which consists of part header and part body separated by a blank line.

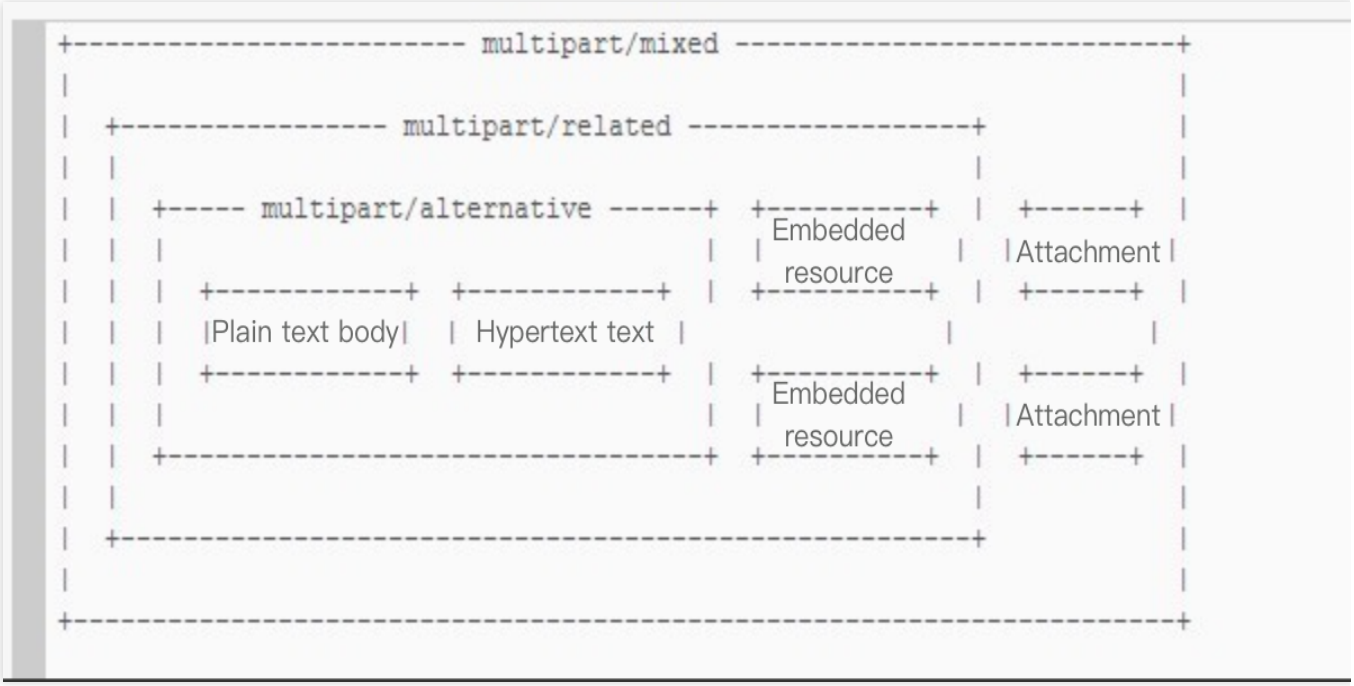
There are three common multipart types:

multipart/mixed

multipart/related

multipart/alternative

The meaning and use of each of these types can be seen from their names. The hierarchical relationship between them can be summarized as shown below:



If you want to add attachments to an email, you must define the `multipart/mixed` part. If there are embedded resources, you must define at least the `multipart/related` part; if plain text and hypertext coexist, you must define at least the `multipart/alternative` part.

Note:

The number of attachments should not exceed 10, the size of a single attachment should not exceed 4 MB, and the total size of all attachments should not exceed 8 MB. For more information, see Data Structure.

Sample Code

```
package main
import (
    "bytes"
    "crypto/tls"
    "encoding/base64"
    "fmt"
    "io/ioutil"
    "log"
    "mime"
    "net"
    "net/smtp"
    "time"
)

// Test465Attachment  for port 465
func Test465Attachment() error {
    boundary := "GoBoundary"
    host := "sg-smtp.qcloudmail.com"
    port := 465
    email := "abc@cd.com"
    password := "***"
    toEmail := "test@test123.com"
    header := make(map[string]string)
    header["From"] = "test " + "<" + email + ">"
    header["To"] = toEmail
    header["Subject"] = "Test465Attachment"
    header["Content-Type"] = "multipart/mixed;boundary=" + boundary
    // This field is not used for the time being. Pass in `1.0` by default
    header["Mime-Version"] = "1.0"
    // This field is not used for the time being
    header["Date"] = time.Now().String()
    bodyHtml := "<!DOCTYPE html>\\n<html>\\n<head>\\n<meta charset=\\"utf-
8\\">\\n<title>hello world</title>\\n</head>\\n<body>\\n " +
        "<h1>My first heading</h1>\\n    <p>My first paragraph.
</p>\\n</body>\\n</html>"
    message := ""
    for k, v := range header {
        message += fmt.Sprintf("%s: %s\\r\\n", k, v)
```

```
    }
    buffer := bytes.NewBuffer(nil)
    buffer.WriteString(message)
    contentType := "Content-Type: text/html" + "; charset=UTF-8"
    body := "\\r\\n--" + boundary + "\\r\\n"
    body += "Content-Type:" + contentType + "\\r\\n"
    body += "\\r\\n" + bodyHtml + "\\r\\n"
    buffer.WriteString(body)

    attachment := "\\r\\n--" + boundary + "\\r\\n"
    attachment += "Content-Transfer-Encoding:base64\\r\\n"
    attachment += "Content-Disposition:attachment\\r\\n"
    attachment += "Content-Type:" + "application/octet-stream" + ";name=\\"" +
mime.BEncoding.Encode("UTF-8",
        "./go.mod") + "\\"\\r\\n"
    buffer.WriteString(attachment)
    writeFile(buffer, "./go.mod")
    // Multiple attachments can be spliced at the end. There can be 10
attachments at most, each of which cannot exceed 5 MB in size. The TOTAL size
of all attachments cannot exceed 8-9 MB; otherwise, EOF will be returned
    attachment1 := "\\r\\n--" + boundary + "\\r\\n"
    attachment1 += "Content-Transfer-Encoding:base64\\r\\n"
    attachment1 += "Content-Disposition:attachment\\r\\n"
    attachment1 += "Content-Type:" + "application/octet-stream" + ";name=\\"" +
mime.BEncoding.Encode("UTF-8",
        "./bbbb.txt") + "\\"\\r\\n"
    buffer.WriteString(attachment1)
    writeFile(buffer, "./bbbb.txt")
    defer func() {
        if err := recover(); err != nil {
            log.Fatalln(err)
        }
    }()

    buffer.WriteString("\\r\\n--" + boundary + "--")
    message += "\\r\\n" + body
    auth := smtp.PlainAuth(
        "",
        email,
        password,
        host,
    )
    err := SendMailWithTLS(
        fmt.Sprintf("%s:%d", host, port),
        auth,
        email,
        []string{toEmail},
```

```
        buffer.Bytes(),
    )
    if err != nil {
        fmt.Println("Send email error:", err)
    } else {
        fmt.Println("Send mail success!")
    }
    return err
}

// Dial return a smtp client
func Dial(addr string) (*smtp.Client, error) {
    conn, err := tls.Dial("tcp", addr, nil)
    if err != nil {
        log.Println("tls.Dial Error:", err)
        return nil, err
    }

    host, _, _ := net.SplitHostPort(addr)
    return smtp.NewClient(conn, host)
}

// SendMailWithTLS send email with tls
func SendMailWithTLS(addr string, auth smtp.Auth, from string,
    to []string, msg []byte) (err error) {
    //create smtp client
    c, err := Dial(addr)
    if err != nil {
        log.Println("Create smtp client error:", err)
        return err
    }
    defer c.Close()
    if auth != nil {
        if ok, _ := c.Extension("AUTH"); ok {
            if err = c.Auth(auth); err != nil {
                log.Println("Error during AUTH", err)
                return err
            }
        }
    }
    if err = c.Mail(from); err != nil {
        return err
    }
    for _, addr := range to {
        if err = c.Rcpt(addr); err != nil {
            return err
        }
```

```
    }
    w, err := c.Data()
    if err != nil {
        return err
    }
    _, err = w.Write(msg)
    if err != nil {
        return err
    }
    err = w.Close()
    if err != nil {
        return err
    }
    return c.Quit()
}

// writeFile read file to buffer
func writeFile(buffer *bytes.Buffer, fileName string) {
    file, err := ioutil.ReadFile(fileName)
    if err != nil {
        panic(err.Error())
    }
    payload := make([]byte, base64.StdEncoding.EncodedLen(len(file)))
    base64.StdEncoding.Encode(payload, file)
    buffer.WriteString("\\r\\n")
    for index, line := 0, len(payload); index < line; index++ {
        buffer.WriteByte(payload[index])
        if (index+1)%76 == 0 {
            buffer.WriteString("\\r\\n")
        }
    }
}

func main() {
    Test465Attachment()
}
```

Error Code

Last updated：2023-12-22 10:28:40

Input Parameters

Field	Description	Remarks
Bcc	Blind carbon copy address	Currently unsupported
Cc	Copy address	Currently unsupported
Content-Transfer-Encoding	Content transfer encoding method	Currently unused. You can leave it empty. Content except attachments doesn't need to be encrypted
Content-Type	Content type	Currently, you can pass in only `text/plain; charset=UTF-8,text/html; charset=UTF-8 multipart/mixed` , `multipart/related` , or `multipart/alternative` ; otherwise, an error will be reported
Date	Date and time	Currently unused
Delivered-To	Recipient address	Currently unused
From	Sender address	Required
Message-ID	Message ID	Currently unused
MIME-Version	MIME version	Currently unused. Leave it empty or pass in 1.0; otherwise, an error will be reported
Received	Transfer path	Currently unused
Reply-To	Reply-to address	Currently unused
Return-Path	Reply-to address	Currently unused
Subject	Subject	Required
To	Recipient address	Required

Attachment parameters (when sending attachment)

Field	Description	Remarks
Content-Type	Content type	We recommend you pass in `application/octet-stream` for files
Content-Transfer-Encoding	Content transfer encoding method	Currently, only Base64 is supported, and an error will be reported if you pass in other values
Content-Disposition	Content disposition method	Currently, you can only pass in `attachment`, and attachments cannot be sent if you pass in other values
Content-ID	Content ID	Currently unsupported
Content-Location	Content location (path)	Currently unsupported
Content-Base	Content base location	Currently unsupported

Note:
The input parameter verification requirements are generally the same as those of SendEmail, including the restrictions on the number of recipients, email body size, attachment format, and attachment size.

Response Parameters

The SMTP API has no response parameters and only supports returning the `err` information. If `nil` is returned, it indicates that the API call is successful, but the actual email sending may not be necessarily successful. To get the sending status, see [GetSendEmailStatus](https://intl.cloud.tencent.com/document/product/1084 /39502).

Error Codes

System errors

1. There are 2,000 or more characters in a single line in the email body.
`554 5.0.0 Error: transaction failed, blame it on the weather: smtp: too longer line in input stream` or other logs that contain `too longer`.
`write tcp *.*.*.*:60575->*.*.*.*:25: write: broken pipe`

2. The attachment is too large.
If the attachment is about 9 MB in size, EOF will be returned. We recommend you keep the total attachment size below 8 MB and keep the total message size below 10 MB. Otherwise, the content will be truncated, and other exception errors such as Base64 decoding failure will be reported.

Business errors

The format of a business error is as follows:

```
554 5.0.0 Error: transaction failed, blame it on the weather: ##SES-response-json: {"Response":{"RequestId":"bee4e9fb-8127-48cc-b606-bbb1e801596b","QcloudError":{"Error":{"Code":"FailedOperation.MissingEmailContent. The operation failed. The content of the email is missing (TemplateData and Simple cannot be both empty).
```

After `##SES-response-json:` is the `json` form of the structure returned by the sending API. The fields are as described below:

Field	Type	Description
RequestId	string	Request ID
QcloudError	stuct	Error structure

QcloudError:

Field	Type	Description
Code	string	Error code
Message	string	Error message

General business error description:

Error Code	Error Description	Remarks
FailedOperation	msg.From is null	The sender is empty
FailedOperation	msg.Subject is null	The subject is empty
FailedOperation	msg.Body is null	The message body is empty
FailedOperation	Content-Transfer-Encoding must in...	Check the `Content-Transfer-Encoding` parameter against the input parameter description
FailedOperation	Content-Type must in...	Check the `Content-Type` parameter in the header against the input parameter

		description
FailedOperation	Mime-Version must in...	Check the `Mime-Version` parameter in the header against the input parameter description
FailedOperation	The email is too large. Remove some content...	The email body other than attachments cannot exceed 1 MB in size
FailedOperation	Incorrect attachment content. Make sure the base64 content is...	The attachment content must be Base64-encoded
FailedOperation	The attachments are too large. Make sure they do not exceed the...	The size of a single attachment exceeds 5 MB, or the total size of all attachments exceeds 10 MB (which may be adjusted)
RequestLimitExceeded.SmtpRateLimit	smtp sending frequency limit...	The SMTP call rate limit is reached

Other business errors

You can refer to the descriptions of error codes in SendEmail.