using System;

using System.Collections.Generic;

using System.Linq;

using System.Globalization;

using System.Text;

using System.Text.RegularExpressions;

namespace QRCoder

{

public static class PayloadGenerator

{

public abstract class Payload

{

public virtual int Version { get { return -1; } }

public virtual QRCodeGenerator.ECCLevel EccLevel { get { return QRCodeGenerator.ECCLevel.M; } }

public virtual QRCodeGenerator.EciMode EciMode { get { return QRCodeGenerator.EciMode.Default; } }

public abstract override string ToString();

}

public class WiFi : Payload

{

private readonly string ssid, password, authenticationMode;

private readonly bool isHiddenSsid;

/// <summary>

/// Generates a WiFi payload. Scanned by a QR Code scanner app, the device will connect to the WiFi.

/// </summary>

/// <param name="ssid">SSID of the WiFi network</param>

/// <param name="password">Password of the WiFi network</param>

/// <param name="authenticationMode">Authentification mode (WEP, WPA, WPA2)</param>

/// <param name="isHiddenSSID">Set flag, if the WiFi network hides its SSID</param>

public WiFi(string ssid, string password, Authentication authenticationMode, bool isHiddenSSID = false)

{

this.ssid = EscapeInput(ssid);

this.ssid = isHexStyle(this.ssid) ? "\"" + this.ssid + "\"" : this.ssid;

this.password = EscapeInput(password);

this.password = isHexStyle(this.password) ? "\"" + this.password + "\"" : this.password;

this.authenticationMode = authenticationMode.ToString();

this.isHiddenSsid = isHiddenSSID;

}

public override string ToString()

{

return

$"WIFI:T:{this.authenticationMode};S:{this.ssid};P:{this.password};{(this.isHiddenSsid ? "H:true" : string.Empty)};";

}

public enum Authentication

{

WEP,

WPA,

nopass

}

}

public class Mail : Payload

{

private readonly string mailReceiver, subject, message;

private readonly MailEncoding encoding;

/// <summary>

/// Creates an empty email payload

/// </summary>

/// <param name="mailReceiver">Receiver's email address</param>

/// <param name="encoding">Payload encoding type. Choose dependent on your QR Code scanner app.</param>

public Mail(string mailReceiver, MailEncoding encoding = MailEncoding.MAILTO)

{

this.mailReceiver = mailReceiver;

this.subject = this.message = string.Empty;

this.encoding = encoding;

}

/// <summary>

/// Creates an email payload with subject

/// </summary>

/// <param name="mailReceiver">Receiver's email address</param>

/// <param name="subject">Subject line of the email</param>

/// <param name="encoding">Payload encoding type. Choose dependent on your QR Code scanner app.</param>

public Mail(string mailReceiver, string subject, MailEncoding encoding = MailEncoding.MAILTO)

{

this.mailReceiver = mailReceiver;

this.subject = subject;

this.message = string.Empty;

this.encoding = encoding;

}

/// <summary>

/// Creates an email payload with subject and message/text

/// </summary>

/// <param name="mailReceiver">Receiver's email address</param>

/// <param name="subject">Subject line of the email</param>

/// <param name="message">Message content of the email</param>

/// <param name="encoding">Payload encoding type. Choose dependent on your QR Code scanner app.</param>

public Mail(string mailReceiver, string subject, string message, MailEncoding encoding = MailEncoding.MAILTO)

{

this.mailReceiver = mailReceiver;

this.subject = subject;

this.message = message;

this.encoding = encoding;

}

public override string ToString()

{

switch (this.encoding)

{

case MailEncoding.MAILTO:

return

$"mailto:{this.mailReceiver}?subject={System.Uri.EscapeDataString(this.subject)}&body={System.Uri.EscapeDataString(this.message)}";

case MailEncoding.MATMSG:

return

$"MATMSG:TO:{this.mailReceiver};SUB:{EscapeInput(this.subject)};BODY:{EscapeInput(this.message)};;";

case MailEncoding.SMTP:

return

$"SMTP:{this.mailReceiver}:{EscapeInput(this.subject, true)}:{EscapeInput(this.message, true)}";

default:

return this.mailReceiver;

}

}

public enum MailEncoding

{

MAILTO,

MATMSG,

SMTP

}

}

public class SMS : Payload

{

private readonly string number, subject;

private readonly SMSEncoding encoding;

/// <summary>

/// Creates a SMS payload without text

/// </summary>

/// <param name="number">Receiver phone number</param>

/// <param name="encoding">Encoding type</param>

public SMS(string number, SMSEncoding encoding = SMSEncoding.SMS)

{

this.number = number;

this.subject = string.Empty;

this.encoding = encoding;

}

/// <summary>

/// Creates a SMS payload with text (subject)

/// </summary>

/// <param name="number">Receiver phone number</param>

/// <param name="subject">Text of the SMS</param>

/// <param name="encoding">Encoding type</param>

public SMS(string number, string subject, SMSEncoding encoding = SMSEncoding.SMS)

{

this.number = number;

this.subject = subject;

this.encoding = encoding;

}

public override string ToString()

{

switch (this.encoding)

{

case SMSEncoding.SMS:

return $"sms:{this.number}?body={System.Uri.EscapeDataString(this.subject)}";

case SMSEncoding.SMS\_iOS:

return $"sms:{this.number};body={System.Uri.EscapeDataString(this.subject)}";

case SMSEncoding.SMSTO:

return $"SMSTO:{this.number}:{this.subject}";

default:

return "sms:";

}

}

public enum SMSEncoding

{

SMS,

SMSTO,

SMS\_iOS

}

}

public class MMS : Payload

{

private readonly string number, subject;

private readonly MMSEncoding encoding;

/// <summary>

/// Creates a MMS payload without text

/// </summary>

/// <param name="number">Receiver phone number</param>

/// <param name="encoding">Encoding type</param>

public MMS(string number, MMSEncoding encoding = MMSEncoding.MMS)

{

this.number = number;

this.subject = string.Empty;

this.encoding = encoding;

}

/// <summary>

/// Creates a MMS payload with text (subject)

/// </summary>

/// <param name="number">Receiver phone number</param>

/// <param name="subject">Text of the MMS</param>

/// <param name="encoding">Encoding type</param>

public MMS(string number, string subject, MMSEncoding encoding = MMSEncoding.MMS)

{

this.number = number;

this.subject = subject;

this.encoding = encoding;

}

public override string ToString()

{

switch (this.encoding)

{

case MMSEncoding.MMSTO:

return $"mmsto:{this.number}?subject={System.Uri.EscapeDataString(this.subject)}";

case MMSEncoding.MMS:

return $"mms:{this.number}?body={System.Uri.EscapeDataString(this.subject)}";

default:

return "mms:";

}

}

public enum MMSEncoding

{

MMS,

MMSTO

}

}

public class Geolocation : Payload

{

private readonly string latitude, longitude;

private readonly GeolocationEncoding encoding;

/// <summary>

/// Generates a geo location payload. Supports raw location (GEO encoding) or Google Maps link (GoogleMaps encoding)

/// </summary>

/// <param name="latitude">Latitude with . as splitter</param>

/// <param name="longitude">Longitude with . as splitter</param>

/// <param name="encoding">Encoding type - GEO or GoogleMaps</param>

public Geolocation(string latitude, string longitude, GeolocationEncoding encoding = GeolocationEncoding.GEO)

{

this.latitude = latitude.Replace(",",".");

this.longitude = longitude.Replace(",", ".");

this.encoding = encoding;

}

public override string ToString()

{

switch (this.encoding)

{

case GeolocationEncoding.GEO:

return $"geo:{this.latitude},{this.longitude}";

case GeolocationEncoding.GoogleMaps:

return $"http://maps.google.com/maps?q={this.latitude},{this.longitude}";

default:

return "geo:";

}

}

public enum GeolocationEncoding

{

GEO,

GoogleMaps

}

}

public class PhoneNumber : Payload

{

private readonly string number;

/// <summary>

/// Generates a phone call payload

/// </summary>

/// <param name="number">Phonenumber of the receiver</param>

public PhoneNumber(string number)

{

this.number = number;

}

public override string ToString()

{

return $"tel:{this.number}";

}

}

public class SkypeCall : Payload

{

private readonly string skypeUsername;

/// <summary>

/// Generates a Skype call payload

/// </summary>

/// <param name="skypeUsername">Skype username which will be called</param>

public SkypeCall(string skypeUsername)

{

this.skypeUsername = skypeUsername;

}

public override string ToString()

{

return $"skype:{this.skypeUsername}?call";

}

}

public class Url : Payload

{

private readonly string url;

/// <summary>

/// Generates a link. If not given, http/https protocol will be added.

/// </summary>

/// <param name="url">Link url target</param>

public Url(string url)

{

this.url = url;

}

public override string ToString()

{

return (!this.url.StartsWith("http") ? "http://" + this.url : this.url);

}

}

public class WhatsAppMessage : Payload

{

private readonly string number, message;

/// <summary>

/// Let's you compose a WhatApp message and send it the receiver number.

/// </summary>

/// <param name="number">Receiver phone number</param>

/// <param name="message">The message</param>

public WhatsAppMessage(string number, string message)

{

this.number = number;

this.message = message;

}

/// <summary>

/// Let's you compose a WhatApp message. When scanned the user is asked to choose a contact who will receive the message.

/// </summary>

/// <param name="message">The message</param>

public WhatsAppMessage(string message)

{

this.number = string.Empty;

this.message = message;

}

public override string ToString()

{

return ($"whatsapp://send?phone={this.number}&text={Uri.EscapeDataString(message)}");

}

}

public class Bookmark : Payload

{

private readonly string url, title;

/// <summary>

/// Generates a bookmark payload. Scanned by an QR Code reader, this one creates a browser bookmark.

/// </summary>

/// <param name="url">Url of the bookmark</param>

/// <param name="title">Title of the bookmark</param>

public Bookmark(string url, string title)

{

this.url = EscapeInput(url);

this.title = EscapeInput(title);

}

public override string ToString()

{

return $"MEBKM:TITLE:{this.title};URL:{this.url};;";

}

}

public class ContactData : Payload

{

private readonly string firstname;

private readonly string lastname;

private readonly string nickname;

private readonly string org;

private readonly string phone;

private readonly string mobilePhone;

private readonly string workPhone;

private readonly string email;

private readonly DateTime? birthday;

private readonly string website;

private readonly string street;

private readonly string houseNumber;

private readonly string city;

private readonly string zipCode;

private readonly string stateRegion;

private readonly string country;

private readonly string note;

private readonly ContactOutputType outputType;

private readonly AddressOrder addressOrder;

/// <summary>

/// Generates a vCard or meCard contact dataset

/// </summary>

/// <param name="outputType">Payload output type</param>

/// <param name="firstname">The firstname</param>

/// <param name="lastname">The lastname</param>

/// <param name="nickname">The displayname</param>

/// <param name="phone">Normal phone number</param>

/// <param name="mobilePhone">Mobile phone</param>

/// <param name="workPhone">Office phone number</param>

/// <param name="email">E-Mail address</param>

/// <param name="birthday">Birthday</param>

/// <param name="website">Website / Homepage</param>

/// <param name="street">Street</param>

/// <param name="houseNumber">Housenumber</param>

/// <param name="city">City</param>

/// <param name="stateRegion">State or Region</param>

/// <param name="zipCode">Zip code</param>

/// <param name="country">Country</param>

/// <param name="addressOrder">The address order format to use</param>

/// <param name="note">Memo text / notes</param>

public ContactData(ContactOutputType outputType, string firstname, string lastname, string nickname = null, string phone = null, string mobilePhone = null, string workPhone = null, string email = null, DateTime? birthday = null, string website = null, string street = null, string houseNumber = null, string city = null, string zipCode = null, string country = null, string note = null, string stateRegion = null, AddressOrder addressOrder = AddressOrder.Default, string org = null)

{

this.firstname = firstname;

this.lastname = lastname;

this.nickname = nickname;

this.org = org;

this.phone = phone;

this.mobilePhone = mobilePhone;

this.workPhone = workPhone;

this.email = email;

this.birthday = birthday;

this.website = website;

this.street = street;

this.houseNumber = houseNumber;

this.city = city;

this.stateRegion = stateRegion;

this.zipCode = zipCode;

this.country = country;

this.addressOrder = addressOrder;

this.note = note;

this.outputType = outputType;

}

public override string ToString()

{

string payload = string.Empty;

if (outputType == ContactOutputType.MeCard)

{

payload += "MECARD+\r\n";

if (!string.IsNullOrEmpty(firstname) && !string.IsNullOrEmpty(lastname))

payload += $"N:{lastname}, {firstname}\r\n";

else if (!string.IsNullOrEmpty(firstname) || !string.IsNullOrEmpty(lastname))

payload += $"N:{firstname}{lastname}\r\n";

if (!string.IsNullOrEmpty(org))

payload += $"ORG:{org}\r\n";

if (!string.IsNullOrEmpty(phone))

payload += $"TEL:{phone}\r\n";

if (!string.IsNullOrEmpty(mobilePhone))

payload += $"TEL:{mobilePhone}\r\n";

if (!string.IsNullOrEmpty(workPhone))

payload += $"TEL:{workPhone}\r\n";

if (!string.IsNullOrEmpty(email))

payload += $"EMAIL:{email}\r\n";

if (!string.IsNullOrEmpty(note))

payload += $"NOTE:{note}\r\n";

if (birthday != null)

payload += $"BDAY:{((DateTime)birthday).ToString("yyyyMMdd")}\r\n";

string addressString = string.Empty;

if(addressOrder == AddressOrder.Default)

{

addressString = $"ADR:,,{(!string.IsNullOrEmpty(street) ? street + " " : "")}{(!string.IsNullOrEmpty(houseNumber) ? houseNumber : "")},{(!string.IsNullOrEmpty(zipCode) ? zipCode : "")},{(!string.IsNullOrEmpty(city) ? city : "")},{(!string.IsNullOrEmpty(stateRegion) ? stateRegion : "")},{(!string.IsNullOrEmpty(country) ? country : "")}\r\n";

}

else

{

addressString = $"ADR:,,{(!string.IsNullOrEmpty(houseNumber) ? houseNumber + " " : "")}{(!string.IsNullOrEmpty(street) ? street : "")},{(!string.IsNullOrEmpty(city) ? city : "")},{(!string.IsNullOrEmpty(stateRegion) ? stateRegion : "")},{(!string.IsNullOrEmpty(zipCode) ? zipCode : "")},{(!string.IsNullOrEmpty(country) ? country : "")}\r\n";

}

payload += addressString;

if (!string.IsNullOrEmpty(website))

payload += $"URL:{website}\r\n";

if (!string.IsNullOrEmpty(nickname))

payload += $"NICKNAME:{nickname}\r\n";

payload = payload.Trim(new char[] { '\r', '\n' });

}

else

{

var version = outputType.ToString().Substring(5);

if (version.Length > 1)

version = version.Insert(1, ".");

else

version += ".0";

payload += "BEGIN:VCARD\r\n";

payload += $"VERSION:{version}\r\n";

payload += $"N:{(!string.IsNullOrEmpty(lastname) ? lastname : "")};{(!string.IsNullOrEmpty(firstname) ? firstname : "")};;;\r\n";

payload += $"FN:{(!string.IsNullOrEmpty(firstname) ? firstname + " " : "")}{(!string.IsNullOrEmpty(lastname) ? lastname : "")}\r\n";

if (!string.IsNullOrEmpty(org))

{

payload += $"ORG:" + org + "\r\n";

}

if (!string.IsNullOrEmpty(phone))

{

payload += $"TEL;";

if (outputType == ContactOutputType.VCard21)

payload += $"HOME;VOICE:{phone}";

else if (outputType == ContactOutputType.VCard3)

payload += $"TYPE=HOME,VOICE:{phone}";

else

payload += $"TYPE=home,voice;VALUE=uri:tel:{phone}";

payload += "\r\n";

}

if (!string.IsNullOrEmpty(mobilePhone))

{

payload += $"TEL;";

if (outputType == ContactOutputType.VCard21)

payload += $"HOME;CELL:{mobilePhone}";

else if (outputType == ContactOutputType.VCard3)

payload += $"TYPE=HOME,CELL:{mobilePhone}";

else

payload += $"TYPE=home,cell;VALUE=uri:tel:{mobilePhone}";

payload += "\r\n";

}

if (!string.IsNullOrEmpty(workPhone))

{

payload += $"TEL;";

if (outputType == ContactOutputType.VCard21)

payload += $"WORK;VOICE:{workPhone}";

else if (outputType == ContactOutputType.VCard3)

payload += $"TYPE=WORK,VOICE:{workPhone}";

else

payload += $"TYPE=work,voice;VALUE=uri:tel:{workPhone}";

payload += "\r\n";

}

payload += "ADR;";

if (outputType == ContactOutputType.VCard21)

payload += "HOME;PREF:";

else if (outputType == ContactOutputType.VCard3)

payload += "TYPE=HOME,PREF:";

else

payload += "TYPE=home,pref:";

string addressString = string.Empty;

if(addressOrder == AddressOrder.Default)

{

addressString = $";;{(!string.IsNullOrEmpty(street) ? street + " " : "")}{(!string.IsNullOrEmpty(houseNumber) ? houseNumber : "")};{(!string.IsNullOrEmpty(zipCode) ? zipCode : "")};{(!string.IsNullOrEmpty(city) ? city : "")};{(!string.IsNullOrEmpty(stateRegion) ? stateRegion : "")};{(!string.IsNullOrEmpty(country) ? country : "")}\r\n";

}

else

{

addressString = $";;{(!string.IsNullOrEmpty(houseNumber) ? houseNumber + " " : "")}{(!string.IsNullOrEmpty(street) ? street : "")};{(!string.IsNullOrEmpty(city) ? city : "")};{(!string.IsNullOrEmpty(stateRegion) ? stateRegion : "")};{(!string.IsNullOrEmpty(zipCode) ? zipCode : "")};{(!string.IsNullOrEmpty(country) ? country : "")}\r\n";

}

payload += addressString;

if (birthday != null)

payload += $"BDAY:{((DateTime)birthday).ToString("yyyyMMdd")}\r\n";

if (!string.IsNullOrEmpty(website))

payload += $"URL:{website}\r\n";

if (!string.IsNullOrEmpty(email))

payload += $"EMAIL:{email}\r\n";

if (!string.IsNullOrEmpty(note))

payload += $"NOTE:{note}\r\n";

if (outputType != ContactOutputType.VCard21 && !string.IsNullOrEmpty(nickname))

payload += $"NICKNAME:{nickname}\r\n";

payload += "END:VCARD";

}

return payload;

}

/// <summary>

/// Possible output types. Either vCard 2.1, vCard 3.0, vCard 4.0 or MeCard.

/// </summary>

public enum ContactOutputType

{

MeCard,

VCard21,

VCard3,

VCard4

}

/// <summary>

/// define the address format

/// Default: European format, ([Street] [House Number] and [Postal Code] [City]

/// Reversed: North American and others format ([House Number] [Street] and [City] [Postal Code])

/// </summary>

public enum AddressOrder

{

Default,

Reversed

}

}

public class BitcoinLikeCryptoCurrencyAddress : Payload

{

private readonly BitcoinLikeCryptoCurrencyType currencyType;

private readonly string address, label, message;

private readonly double? amount;

/// <summary>

/// Generates a Bitcoin like cryptocurrency payment payload. QR Codes with this payload can open a payment app.

/// </summary>

/// <param name="currencyName">Bitcoin like cryptocurrency address of the payment receiver</param>

/// <param name="address">Bitcoin like cryptocurrency address of the payment receiver</param>

/// <param name="amount">Amount of coins to transfer</param>

/// <param name="label">Reference label</param>

/// <param name="message">Referece text aka message</param>

public BitcoinLikeCryptoCurrencyAddress(BitcoinLikeCryptoCurrencyType currencyType, string address, double? amount, string label = null, string message = null)

{

this.currencyType = currencyType;

this.address = address;

if (!string.IsNullOrEmpty(label))

{

this.label = Uri.EscapeUriString(label);

}

if (!string.IsNullOrEmpty(message))

{

this.message = Uri.EscapeUriString(message);

}

this.amount = amount;

}

public override string ToString()

{

string query = null;

var queryValues = new KeyValuePair<string,string>[]{

new KeyValuePair<string, string>(nameof(label), label),

new KeyValuePair<string, string>(nameof(message), message),

new KeyValuePair<string, string>(nameof(amount), amount.HasValue ? amount.Value.ToString("#.########", CultureInfo.InvariantCulture) : null)

};

if (queryValues.Any(keyPair => !string.IsNullOrEmpty(keyPair.Value)))

{

query = "?" + string.Join("&", queryValues

.Where(keyPair => !string.IsNullOrEmpty(keyPair.Value))

.Select(keyPair => $"{keyPair.Key}={keyPair.Value}")

.ToArray());

}

return $"{Enum.GetName(typeof(BitcoinLikeCryptoCurrencyType), currencyType).ToLower()}:{address}{query}";

}

public enum BitcoinLikeCryptoCurrencyType

{

Bitcoin,

BitcoinCash,

Litecoin

}

}

public class BitcoinAddress : BitcoinLikeCryptoCurrencyAddress

{

public BitcoinAddress(string address, double? amount, string label = null, string message = null)

: base(BitcoinLikeCryptoCurrencyType.Bitcoin, address, amount, label, message) { }

}

public class BitcoinCashAddress : BitcoinLikeCryptoCurrencyAddress

{

public BitcoinCashAddress(string address, double? amount, string label = null, string message = null)

: base(BitcoinLikeCryptoCurrencyType.BitcoinCash, address, amount, label, message) { }

}

public class LitecoinAddress : BitcoinLikeCryptoCurrencyAddress

{

public LitecoinAddress(string address, double? amount, string label = null, string message = null)

: base(BitcoinLikeCryptoCurrencyType.Litecoin, address, amount, label, message) { }

}

public class SwissQrCode : Payload

{

//Keep in mind, that the ECC level has to be set to "M" when generating a SwissQrCode!

//SwissQrCode specification:

// - (de) https://www.paymentstandards.ch/dam/downloads/ig-qr-bill-de.pdf

// - (en) https://www.paymentstandards.ch/dam/downloads/ig-qr-bill-en.pdf

//Changes between version 1.0 and 2.0: https://www.paymentstandards.ch/dam/downloads/change-documentation-qrr-de.pdf

private readonly string br = "\r\n";

private readonly string alternativeProcedure1, alternativeProcedure2;

private readonly Iban iban;

private readonly decimal? amount;

private readonly Contact creditor, ultimateCreditor, debitor;

private readonly Currency currency;

private readonly DateTime? requestedDateOfPayment;

private readonly Reference reference;

private readonly AdditionalInformation additionalInformation;

/// <summary>

/// Generates the payload for a SwissQrCode v2.0. (Don't forget to use ECC-Level=M, EncodingMode=UTF-8 and to set the Swiss flag icon to the final QR code.)

/// </summary>

/// <param name="iban">IBAN object</param>

/// <param name="currency">Currency (either EUR or CHF)</param>

/// <param name="creditor">Creditor (payee) information</param>

/// <param name="reference">Reference information</param>

/// <param name="debitor">Debitor (payer) information</param>

/// <param name="amount">Amount</param>

/// <param name="requestedDateOfPayment">Requested date of debitor's payment</param>

/// <param name="ultimateCreditor">Ultimate creditor information (use only in consultation with your bank - for future use only!)</param>

/// <param name="alternativeProcedure1">Optional command for alternative processing mode - line 1</param>

/// <param name="alternativeProcedure2">Optional command for alternative processing mode - line 2</param>

public SwissQrCode(Iban iban, Currency currency, Contact creditor, Reference reference, AdditionalInformation additionalInformation = null, Contact debitor = null, decimal? amount = null, DateTime? requestedDateOfPayment = null, Contact ultimateCreditor = null, string alternativeProcedure1 = null, string alternativeProcedure2 = null)

{

this.iban = iban;

this.creditor = creditor;

this.ultimateCreditor = ultimateCreditor;

this.additionalInformation = additionalInformation != null ? additionalInformation : new AdditionalInformation();

if (amount != null && amount.ToString().Length > 12)

throw new SwissQrCodeException("Amount (including decimals) must be shorter than 13 places.");

this.amount = amount;

this.currency = currency;

this.requestedDateOfPayment = requestedDateOfPayment;

this.debitor = debitor;

if (iban.IsQrIban && reference.RefType != Reference.ReferenceType.QRR)

throw new SwissQrCodeException("If QR-IBAN is used, you have to choose \"QRR\" as reference type!");

if (!iban.IsQrIban && reference.RefType == Reference.ReferenceType.QRR)

throw new SwissQrCodeException("If non QR-IBAN is used, you have to choose either \"SCOR\" or \"NON\" as reference type!");

this.reference = reference;

if (alternativeProcedure1 != null && alternativeProcedure1.Length > 100)

throw new SwissQrCodeException("Alternative procedure information block 1 must be shorter than 101 chars.");

this.alternativeProcedure1 = alternativeProcedure1;

if (alternativeProcedure2 != null && alternativeProcedure2.Length > 100)

throw new SwissQrCodeException("Alternative procedure information block 2 must be shorter than 101 chars.");

this.alternativeProcedure2 = alternativeProcedure2;

}

public class AdditionalInformation

{

private readonly string unstructuredMessage, billInformation, trailer;

/// <summary>

/// Creates an additional information object. Both parameters are optional and must be shorter than 141 chars in combination.

/// </summary>

/// <param name="unstructuredMessage">Unstructured text message</param>

/// <param name="billInformation">Bill information</param>

public AdditionalInformation(string unstructuredMessage = null, string billInformation = null)

{

if (((unstructuredMessage != null ? unstructuredMessage.Length : 0) + (billInformation != null ? billInformation.Length : 0)) > 140)

throw new SwissQrCodeAdditionalInformationException("Unstructured message and bill information must be shorter than 141 chars in total/combined.");

this.unstructuredMessage = unstructuredMessage;

this.billInformation = billInformation;

this.trailer = "EPD";

}

public string UnstructureMessage

{

get { return !string.IsNullOrEmpty(unstructuredMessage) ? unstructuredMessage.Replace("\n", "") : null; }

}

public string BillInformation

{

get { return !string.IsNullOrEmpty(billInformation) ? billInformation.Replace("\n", "") : null; }

}

public string Trailer

{

get { return trailer; }

}

public class SwissQrCodeAdditionalInformationException : Exception

{

public SwissQrCodeAdditionalInformationException()

{

}

public SwissQrCodeAdditionalInformationException(string message)

: base(message)

{

}

public SwissQrCodeAdditionalInformationException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class Reference

{

private readonly ReferenceType referenceType;

private readonly string reference;

private readonly ReferenceTextType? referenceTextType;

/// <summary>

/// Creates a reference object which must be passed to the SwissQrCode instance

/// </summary>

/// <param name="referenceType">Type of the reference (QRR, SCOR or NON)</param>

/// <param name="reference">Reference text</param>

/// <param name="referenceTextType">Type of the reference text (QR-reference or Creditor Reference)</param>

public Reference(ReferenceType referenceType, string reference = null, ReferenceTextType? referenceTextType = null)

{

this.referenceType = referenceType;

this.referenceTextType = referenceTextType;

if (referenceType == ReferenceType.NON && reference != null)

throw new SwissQrCodeReferenceException("Reference is only allowed when referenceType not equals \"NON\"");

if (referenceType != ReferenceType.NON && reference != null && referenceTextType == null)

throw new SwissQrCodeReferenceException("You have to set an ReferenceTextType when using the reference text.");

if (referenceTextType == ReferenceTextType.QrReference && reference != null && (reference.Length > 27))

throw new SwissQrCodeReferenceException("QR-references have to be shorter than 28 chars.");

if (referenceTextType == ReferenceTextType.QrReference && reference != null && !Regex.IsMatch(reference, @"^[0-9]+$"))

throw new SwissQrCodeReferenceException("QR-reference must exist out of digits only.");

if (referenceTextType == ReferenceTextType.QrReference && reference != null && !ChecksumMod10(reference))

throw new SwissQrCodeReferenceException("QR-references is invalid. Checksum error.");

if (referenceTextType == ReferenceTextType.CreditorReferenceIso11649 && reference != null && (reference.Length > 25))

throw new SwissQrCodeReferenceException("Creditor references (ISO 11649) have to be shorter than 26 chars.");

this.reference = reference;

}

public ReferenceType RefType {

get { return referenceType; }

}

public string ReferenceText

{

get { return !string.IsNullOrEmpty(reference) ? reference.Replace("\n", "") : null; }

}

/// <summary>

/// Reference type. When using a QR-IBAN you have to use either "QRR" or "SCOR"

/// </summary>

public enum ReferenceType

{

QRR,

SCOR,

NON

}

public enum ReferenceTextType

{

QrReference,

CreditorReferenceIso11649

}

public class SwissQrCodeReferenceException : Exception

{

public SwissQrCodeReferenceException()

{

}

public SwissQrCodeReferenceException(string message)

: base(message)

{

}

public SwissQrCodeReferenceException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class Iban

{

private string iban;

private IbanType ibanType;

/// <summary>

/// IBAN object with type information

/// </summary>

/// <param name="iban">IBAN</param>

/// <param name="ibanType">Type of IBAN (normal or QR-IBAN)</param>

public Iban(string iban, IbanType ibanType)

{

if (ibanType == IbanType.Iban && !IsValidIban(iban))

throw new SwissQrCodeIbanException("The IBAN entered isn't valid.");

if (ibanType == IbanType.QrIban && !IsValidQRIban(iban))

throw new SwissQrCodeIbanException("The QR-IBAN entered isn't valid.");

if (!iban.StartsWith("CH") && !iban.StartsWith("LI"))

throw new SwissQrCodeIbanException("The IBAN must start with \"CH\" or \"LI\".");

this.iban = iban;

this.ibanType = ibanType;

}

public bool IsQrIban

{

get { return ibanType == IbanType.QrIban; }

}

public override string ToString()

{

return iban.Replace("-", "").Replace("\n", "").Replace(" ","");

}

public enum IbanType

{

Iban,

QrIban

}

public class SwissQrCodeIbanException : Exception

{

public SwissQrCodeIbanException()

{

}

public SwissQrCodeIbanException(string message)

: base(message)

{

}

public SwissQrCodeIbanException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class Contact

{

private static readonly HashSet<string> twoLetterCodes = ValidTwoLetterCodes();

private string br = "\r\n";

private string name, streetOrAddressline1, houseNumberOrAddressline2, zipCode, city, country;

private AddressType adrType;

/// <summary>

/// Contact type. Can be used for payee, ultimate payee, etc. with address in structured mode (S).

/// </summary>

/// <param name="name">Last name or company (optional first name)</param>

/// <param name="zipCode">Zip-/Postcode</param>

/// <param name="city">City name</param>

/// <param name="country">Two-letter country code as defined in ISO 3166-1</param>

/// <param name="street">Streetname without house number</param>

/// <param name="houseNumber">House number</param>

[Obsolete("This constructor is deprecated. Use WithStructuredAddress instead.")]

public Contact(string name, string zipCode, string city, string country, string street = null, string houseNumber = null) : this (name, zipCode, city, country, street, houseNumber, AddressType.StructuredAddress)

{

}

/// <summary>

/// Contact type. Can be used for payee, ultimate payee, etc. with address in combined mode (K).

/// </summary>

/// <param name="name">Last name or company (optional first name)</param>

/// <param name="country">Two-letter country code as defined in ISO 3166-1</param>

/// <param name="addressLine1">Adress line 1</param>

/// <param name="addressLine2">Adress line 2</param>

[Obsolete("This constructor is deprecated. Use WithCombinedAddress instead.")]

public Contact(string name, string country, string addressLine1, string addressLine2) : this(name, null, null, country, addressLine1, addressLine2, AddressType.CombinedAddress)

{

}

public static Contact WithStructuredAddress(string name, string zipCode, string city, string country, string street = null, string houseNumber = null)

{

return new Contact(name, zipCode, city, country, street, houseNumber, AddressType.StructuredAddress);

}

public static Contact WithCombinedAddress(string name, string country, string addressLine1, string addressLine2)

{

return new Contact(name, null, null, country, addressLine1, addressLine2, AddressType.CombinedAddress);

}

private Contact(string name, string zipCode, string city, string country, string streetOrAddressline1, string houseNumberOrAddressline2, AddressType addressType)

{

//Pattern extracted from https://qr-validation.iso-payments.ch as explained in https://github.com/codebude/QRCoder/issues/97

var charsetPattern = @"^([a-zA-Z0-9\.,;:'\ \+\-/\(\)?\\*\[\]\{\}\\`´~ ]|[!""#%&<>÷=@\_$£]|[àáâäçèéêëìíîïñòóôöùúûüýßÀÁÂÄÇÈÉÊËÌÍÎÏÒÓÔÖÙÚÛÜÑ])\*$";

this.adrType = addressType;

if (string.IsNullOrEmpty(name))

throw new SwissQrCodeContactException("Name must not be empty.");

if (name.Length > 70)

throw new SwissQrCodeContactException("Name must be shorter than 71 chars.");

if (!Regex.IsMatch(name, charsetPattern))

throw new SwissQrCodeContactException($"Name must match the following pattern as defined in pain.001: {charsetPattern}");

this.name = name;

if (AddressType.StructuredAddress == this.adrType)

{

if (!string.IsNullOrEmpty(streetOrAddressline1) && (streetOrAddressline1.Length > 70))

throw new SwissQrCodeContactException("Street must be shorter than 71 chars.");

if (!string.IsNullOrEmpty(streetOrAddressline1) && !Regex.IsMatch(streetOrAddressline1, charsetPattern))

throw new SwissQrCodeContactException($"Street must match the following pattern as defined in pain.001: {charsetPattern}");

this.streetOrAddressline1 = streetOrAddressline1;

if (!string.IsNullOrEmpty(houseNumberOrAddressline2) && houseNumberOrAddressline2.Length > 16)

throw new SwissQrCodeContactException("House number must be shorter than 17 chars.");

this.houseNumberOrAddressline2 = houseNumberOrAddressline2;

}

else

{

if (!string.IsNullOrEmpty(streetOrAddressline1) && (streetOrAddressline1.Length > 70))

throw new SwissQrCodeContactException("Address line 1 must be shorter than 71 chars.");

if (!string.IsNullOrEmpty(streetOrAddressline1) && !Regex.IsMatch(streetOrAddressline1, charsetPattern))

throw new SwissQrCodeContactException($"Address line 1 must match the following pattern as defined in pain.001: {charsetPattern}");

this.streetOrAddressline1 = streetOrAddressline1;

if (string.IsNullOrEmpty(houseNumberOrAddressline2))

throw new SwissQrCodeContactException("Address line 2 must be provided for combined addresses (address line-based addresses).");

if (!string.IsNullOrEmpty(houseNumberOrAddressline2) && (houseNumberOrAddressline2.Length > 70))

throw new SwissQrCodeContactException("Address line 2 must be shorter than 71 chars.");

if (!string.IsNullOrEmpty(houseNumberOrAddressline2) && !Regex.IsMatch(houseNumberOrAddressline2, charsetPattern))

throw new SwissQrCodeContactException($"Address line 2 must match the following pattern as defined in pain.001: {charsetPattern}");

this.houseNumberOrAddressline2 = houseNumberOrAddressline2;

}

if (AddressType.StructuredAddress == this.adrType) {

if (string.IsNullOrEmpty(zipCode))

throw new SwissQrCodeContactException("Zip code must not be empty.");

if (zipCode.Length > 16)

throw new SwissQrCodeContactException("Zip code must be shorter than 17 chars.");

if (!Regex.IsMatch(zipCode, charsetPattern))

throw new SwissQrCodeContactException($"Zip code must match the following pattern as defined in pain.001: {charsetPattern}");

this.zipCode = zipCode;

if (string.IsNullOrEmpty(city))

throw new SwissQrCodeContactException("City must not be empty.");

if (city.Length > 35)

throw new SwissQrCodeContactException("City name must be shorter than 36 chars.");

if (!Regex.IsMatch(city, charsetPattern))

throw new SwissQrCodeContactException($"City name must match the following pattern as defined in pain.001: {charsetPattern}");

this.city = city;

}

else

{

this.zipCode = this.city = string.Empty;

}

if (!IsValidTwoLetterCode(country))

throw new SwissQrCodeContactException("Country must be a valid \"two letter\" country code as defined by ISO 3166-1, but it isn't.");

this.country = country;

}

private static bool IsValidTwoLetterCode(string code) => twoLetterCodes.Contains(code);

private static HashSet<string> ValidTwoLetterCodes()

{

string[] codes = new string[]{ "AF", "AL", "DZ", "AS", "AD", "AO", "AI", "AQ", "AG", "AR", "AM", "AW", "AU", "AT", "AZ", "BS", "BH", "BD", "BB", "BY", "BE", "BZ", "BJ", "BM", "BT", "BO", "BQ", "BA", "BW", "BV", "BR", "IO", "BN", "BG", "BF", "BI", "CV", "KH", "CM", "CA", "KY", "CF", "TD", "CL", "CN", "CX", "CC", "CO", "KM", "CG", "CD", "CK", "CR", "CI", "HR", "CU", "CW", "CY", "CZ", "DK", "DJ", "DM", "DO", "EC", "EG", "SV", "GQ", "ER", "EE", "SZ", "ET", "FK", "FO", "FJ", "FI", "FR", "GF", "PF", "TF", "GA", "GM", "GE", "DE", "GH", "GI", "GR", "GL", "GD", "GP", "GU", "GT", "GG", "GN", "GW", "GY", "HT", "HM", "VA", "HN", "HK", "HU", "IS", "IN", "ID", "IR", "IQ", "IE", "IM", "IL", "IT", "JM", "JP", "JE", "JO", "KZ", "KE", "KI", "KP", "KR", "KW", "KG", "LA", "LV", "LB", "LS", "LR", "LY", "LI", "LT", "LU", "MO", "MG", "MW", "MY", "MV", "ML", "MT", "MH", "MQ", "MR", "MU", "YT", "MX", "FM", "MD", "MC", "MN", "ME", "MS", "MA", "MZ", "MM", "NA", "NR", "NP", "NL", "NC", "NZ", "NI", "NE", "NG", "NU", "NF", "MP", "MK", "NO", "OM", "PK", "PW", "PS", "PA", "PG", "PY", "PE", "PH", "PN", "PL", "PT", "PR", "QA", "RE", "RO", "RU", "RW", "BL", "SH", "KN", "LC", "MF", "PM", "VC", "WS", "SM", "ST", "SA", "SN", "RS", "SC", "SL", "SG", "SX", "SK", "SI", "SB", "SO", "ZA", "GS", "SS", "ES", "LK", "SD", "SR", "SJ", "SE", "CH", "SY", "TW", "TJ", "TZ", "TH", "TL", "TG", "TK", "TO", "TT", "TN", "TR", "TM", "TC", "TV", "UG", "UA", "AE", "GB", "US", "UM", "UY", "UZ", "VU", "VE", "VN", "VG", "VI", "WF", "EH", "YE", "ZM", "ZW", "AX" };

return new HashSet<string>(codes, StringComparer.OrdinalIgnoreCase);

}

public override string ToString()

{

string contactData = $"{(AddressType.StructuredAddress == adrType ? "S" : "K")}{br}"; //AdrTp

contactData += name.Replace("\n", "") + br; //Name

contactData += (!string.IsNullOrEmpty(streetOrAddressline1) ? streetOrAddressline1.Replace("\n","") : string.Empty) + br; //StrtNmOrAdrLine1

contactData += (!string.IsNullOrEmpty(houseNumberOrAddressline2) ? houseNumberOrAddressline2.Replace("\n", "") : string.Empty) + br; //BldgNbOrAdrLine2

contactData += zipCode.Replace("\n", "") + br; //PstCd

contactData += city.Replace("\n", "") + br; //TwnNm

contactData += country + br; //Ctry

return contactData;

}

public enum AddressType

{

StructuredAddress,

CombinedAddress

}

public class SwissQrCodeContactException : Exception

{

public SwissQrCodeContactException()

{

}

public SwissQrCodeContactException(string message)

: base(message)

{

}

public SwissQrCodeContactException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public override string ToString()

{

//Header "logical" element

var SwissQrCodePayload = "SPC" + br; //QRType

SwissQrCodePayload += "0200" + br; //Version

SwissQrCodePayload += "1" + br; //Coding

//CdtrInf "logical" element

SwissQrCodePayload += iban.ToString() + br; //IBAN

//Cdtr "logical" element

SwissQrCodePayload += creditor.ToString();

//UltmtCdtr "logical" element

//Since version 2.0 ultimate creditor was marked as "for future use" and has to be delivered empty in any case!

SwissQrCodePayload += string.Concat(Enumerable.Repeat(br, 7).ToArray());

//CcyAmtDate "logical" element

//Amoutn has to use . as decimal seperator in any case. See https://www.paymentstandards.ch/dam/downloads/ig-qr-bill-en.pdf page 27.

SwissQrCodePayload += (amount != null ? $"{amount:0.00}".Replace(",", ".") : string.Empty) + br; //Amt

SwissQrCodePayload += currency + br; //Ccy

//Removed in S-QR version 2.0

//SwissQrCodePayload += (requestedDateOfPayment != null ? ((DateTime)requestedDateOfPayment).ToString("yyyy-MM-dd") : string.Empty) + br; //ReqdExctnDt

//UltmtDbtr "logical" element

if (debitor != null)

SwissQrCodePayload += debitor.ToString();

else

SwissQrCodePayload += string.Concat(Enumerable.Repeat(br, 7).ToArray());

//RmtInf "logical" element

SwissQrCodePayload += reference.RefType.ToString() + br; //Tp

SwissQrCodePayload += (!string.IsNullOrEmpty(reference.ReferenceText) ? reference.ReferenceText : string.Empty) + br; //Ref

//AddInf "logical" element

SwissQrCodePayload += (!string.IsNullOrEmpty(additionalInformation.UnstructureMessage) ? additionalInformation.UnstructureMessage : string.Empty) + br; //Ustrd

SwissQrCodePayload += additionalInformation.Trailer + br; //Trailer

SwissQrCodePayload += (!string.IsNullOrEmpty(additionalInformation.BillInformation) ? additionalInformation.BillInformation : string.Empty) + br; //StrdBkgInf

//AltPmtInf "logical" element

if (!string.IsNullOrEmpty(alternativeProcedure1))

SwissQrCodePayload += alternativeProcedure1.Replace("\n", "") + br; //AltPmt

if (!string.IsNullOrEmpty(alternativeProcedure2))

SwissQrCodePayload += alternativeProcedure2.Replace("\n", "") + br; //AltPmt

//S-QR specification 2.0, chapter 4.2.3

if (SwissQrCodePayload.EndsWith(br))

SwissQrCodePayload = SwissQrCodePayload.Remove(SwissQrCodePayload.Length - br.Length);

return SwissQrCodePayload;

}

/// <summary>

/// ISO 4217 currency codes

/// </summary>

public enum Currency

{

CHF = 756,

EUR = 978

}

public class SwissQrCodeException : Exception

{

public SwissQrCodeException()

{

}

public SwissQrCodeException(string message)

: base(message)

{

}

public SwissQrCodeException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class Girocode : Payload

{

//Keep in mind, that the ECC level has to be set to "M" when generating a Girocode!

//Girocode specification: http://www.europeanpaymentscouncil.eu/index.cfm/knowledge-bank/epc-documents/quick-response-code-guidelines-to-enable-data-capture-for-the-initiation-of-a-sepa-credit-transfer/epc069-12-quick-response-code-guidelines-to-enable-data-capture-for-the-initiation-of-a-sepa-credit-transfer1/

private string br = "\n";

private readonly string iban, bic, name, purposeOfCreditTransfer, remittanceInformation, messageToGirocodeUser;

private readonly decimal amount;

private readonly GirocodeVersion version;

private readonly GirocodeEncoding encoding;

private readonly TypeOfRemittance typeOfRemittance;

/// <summary>

/// Generates the payload for a Girocode (QR-Code with credit transfer information).

/// Attention: When using Girocode payload, QR code must be generated with ECC level M!

/// </summary>

/// <param name="iban">Account number of the Beneficiary. Only IBAN is allowed.</param>

/// <param name="bic">BIC of the Beneficiary Bank.</param>

/// <param name="name">Name of the Beneficiary.</param>

/// <param name="amount">Amount of the Credit Transfer in Euro.

/// (Amount must be more than 0.01 and less than 999999999.99)</param>

/// <param name="remittanceInformation">Remittance Information (Purpose-/reference text). (optional)</param>

/// <param name="typeOfRemittance">Type of remittance information. Either structured (e.g. ISO 11649 RF Creditor Reference) and max. 35 chars or unstructured and max. 140 chars.</param>

/// <param name="purposeOfCreditTransfer">Purpose of the Credit Transfer (optional)</param>

/// <param name="messageToGirocodeUser">Beneficiary to originator information. (optional)</param>

/// <param name="version">Girocode version. Either 001 or 002. Default: 001.</param>

/// <param name="encoding">Encoding of the Girocode payload. Default: ISO-8859-1</param>

public Girocode(string iban, string bic, string name, decimal amount, string remittanceInformation = "", TypeOfRemittance typeOfRemittance = TypeOfRemittance.Unstructured, string purposeOfCreditTransfer = "", string messageToGirocodeUser = "", GirocodeVersion version = GirocodeVersion.Version1, GirocodeEncoding encoding = GirocodeEncoding.ISO\_8859\_1)

{

this.version = version;

this.encoding = encoding;

if (!IsValidIban(iban))

throw new GirocodeException("The IBAN entered isn't valid.");

this.iban = iban.Replace(" ","").ToUpper();

if (!IsValidBic(bic))

throw new GirocodeException("The BIC entered isn't valid.");

this.bic = bic.Replace(" ", "").ToUpper();

if (name.Length > 70)

throw new GirocodeException("(Payee-)Name must be shorter than 71 chars.");

this.name = name;

if (amount.ToString().Replace(",", ".").Contains(".") && amount.ToString().Replace(",",".").Split('.')[1].TrimEnd('0').Length > 2)

throw new GirocodeException("Amount must have less than 3 digits after decimal point.");

if (amount < 0.01m || amount > 999999999.99m)

throw new GirocodeException("Amount has to at least 0.01 and must be smaller or equal to 999999999.99.");

this.amount = amount;

if (purposeOfCreditTransfer.Length > 4)

throw new GirocodeException("Purpose of credit transfer can only have 4 chars at maximum.");

this.purposeOfCreditTransfer = purposeOfCreditTransfer;

if (typeOfRemittance == TypeOfRemittance.Unstructured && remittanceInformation.Length > 140)

throw new GirocodeException("Unstructured reference texts have to shorter than 141 chars.");

if (typeOfRemittance == TypeOfRemittance.Structured && remittanceInformation.Length > 35)

throw new GirocodeException("Structured reference texts have to shorter than 36 chars.");

this.typeOfRemittance = typeOfRemittance;

this.remittanceInformation = remittanceInformation;

if (messageToGirocodeUser.Length > 70)

throw new GirocodeException("Message to the Girocode-User reader texts have to shorter than 71 chars.");

this.messageToGirocodeUser = messageToGirocodeUser;

}

public override string ToString()

{

var girocodePayload = "BCD" + br;

girocodePayload += ((version == GirocodeVersion.Version1) ? "001" : "002") + br;

girocodePayload += (int)encoding + 1 + br;

girocodePayload += "SCT" + br;

girocodePayload += bic + br;

girocodePayload += name + br;

girocodePayload += iban + br;

girocodePayload += $"EUR{amount:0.00}".Replace(",",".") + br;

girocodePayload += purposeOfCreditTransfer + br;

girocodePayload += ((typeOfRemittance == TypeOfRemittance.Structured)

? remittanceInformation

: string.Empty) + br;

girocodePayload += ((typeOfRemittance == TypeOfRemittance.Unstructured)

? remittanceInformation

: string.Empty) + br;

girocodePayload += messageToGirocodeUser;

return ConvertStringToEncoding(girocodePayload, encoding.ToString().Replace("\_","-"));

}

public enum GirocodeVersion

{

Version1,

Version2

}

public enum TypeOfRemittance

{

Structured,

Unstructured

}

public enum GirocodeEncoding

{

UTF\_8,

ISO\_8859\_1,

ISO\_8859\_2,

ISO\_8859\_4,

ISO\_8859\_5,

ISO\_8859\_7,

ISO\_8859\_10,

ISO\_8859\_15

}

public class GirocodeException : Exception

{

public GirocodeException()

{

}

public GirocodeException(string message)

: base(message)

{

}

public GirocodeException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class BezahlCode : Payload

{

//BezahlCode specification: http://www.bezahlcode.de/wp-content/uploads/BezahlCode\_TechDok.pdf

private readonly string name, iban, bic, account, bnc, sepaReference, reason, creditorId, mandateId, periodicTimeunit;

private readonly decimal amount;

private readonly int postingKey, periodicTimeunitRotation;

private readonly Currency currency;

private readonly AuthorityType authority;

private readonly DateTime executionDate, dateOfSignature, periodicFirstExecutionDate, periodicLastExecutionDate;

/// <summary>

/// Constructor for contact data

/// </summary>

/// <param name="authority">Type of the bank transfer</param>

/// <param name="name">Name of the receiver (Empfänger)</param>

/// <param name="account">Bank account (Kontonummer)</param>

/// <param name="bnc">Bank institute (Bankleitzahl)</param>

/// <param name="iban">IBAN</param>

/// <param name="bic">BIC</param>

/// <param name="reason">Reason (Verwendungszweck)</param>

public BezahlCode(AuthorityType authority, string name, string account = "", string bnc = "", string iban = "", string bic = "", string reason = "") : this(authority, name, account, bnc, iban, bic, 0, string.Empty, 0, null, null, string.Empty, string.Empty, null, reason, 0, string.Empty, Currency.EUR, null, 1)

{

}

/// <summary>

/// Constructor for non-SEPA payments

/// </summary>

/// <param name="authority">Type of the bank transfer</param>

/// <param name="name">Name of the receiver (Empfänger)</param>

/// <param name="account">Bank account (Kontonummer)</param>

/// <param name="bnc">Bank institute (Bankleitzahl)</param>

/// <param name="amount">Amount (Betrag)</param>

/// <param name="periodicTimeunit">Unit of intervall for payment ('M' = monthly, 'W' = weekly)</param>

/// <param name="periodicTimeunitRotation">Intervall for payment. This value is combined with 'periodicTimeunit'</param>

/// <param name="periodicFirstExecutionDate">Date of first periodic execution</param>

/// <param name="periodicLastExecutionDate">Date of last periodic execution</param>

/// <param name="reason">Reason (Verwendungszweck)</param>

/// <param name="postingKey">Transfer Key (Textschlüssel, z.B. Spendenzahlung = 69)</param>

/// <param name="currency">Currency (Währung)</param>

/// <param name="executionDate">Execution date (Ausführungsdatum)</param>

public BezahlCode(AuthorityType authority, string name, string account, string bnc, decimal amount, string periodicTimeunit = "", int periodicTimeunitRotation = 0, DateTime? periodicFirstExecutionDate = null, DateTime? periodicLastExecutionDate = null, string reason = "", int postingKey = 0, Currency currency = Currency.EUR, DateTime? executionDate = null) : this(authority, name, account, bnc, string.Empty, string.Empty, amount, periodicTimeunit, periodicTimeunitRotation, periodicFirstExecutionDate, periodicLastExecutionDate, string.Empty, string.Empty, null, reason, postingKey, string.Empty, currency, executionDate, 2)

{

}

/// <summary>

/// Constructor for SEPA payments

/// </summary>

/// <param name="authority">Type of the bank transfer</param>

/// <param name="name">Name of the receiver (Empfänger)</param>

/// <param name="iban">IBAN</param>

/// <param name="bic">BIC</param>

/// <param name="amount">Amount (Betrag)</param>

/// <param name="periodicTimeunit">Unit of intervall for payment ('M' = monthly, 'W' = weekly)</param>

/// <param name="periodicTimeunitRotation">Intervall for payment. This value is combined with 'periodicTimeunit'</param>

/// <param name="periodicFirstExecutionDate">Date of first periodic execution</param>

/// <param name="periodicLastExecutionDate">Date of last periodic execution</param>

/// <param name="creditorId">Creditor id (Gläubiger ID)</param>

/// <param name="mandateId">Manadate id (Mandatsreferenz)</param>

/// <param name="dateOfSignature">Signature date (Erteilungsdatum des Mandats)</param>

/// <param name="reason">Reason (Verwendungszweck)</param>

/// <param name="postingKey">Transfer Key (Textschlüssel, z.B. Spendenzahlung = 69)</param>

/// <param name="sepaReference">SEPA reference (SEPA-Referenz)</param>

/// <param name="currency">Currency (Währung)</param>

/// <param name="executionDate">Execution date (Ausführungsdatum)</param>

public BezahlCode(AuthorityType authority, string name, string iban, string bic, decimal amount, string periodicTimeunit = "", int periodicTimeunitRotation = 0, DateTime? periodicFirstExecutionDate = null, DateTime? periodicLastExecutionDate = null, string creditorId = "", string mandateId = "", DateTime? dateOfSignature = null, string reason = "", string sepaReference = "", Currency currency = Currency.EUR, DateTime? executionDate = null) : this(authority, name, string.Empty, string.Empty, iban, bic, amount, periodicTimeunit, periodicTimeunitRotation, periodicFirstExecutionDate, periodicLastExecutionDate, creditorId, mandateId, dateOfSignature, reason, 0, sepaReference, currency, executionDate, 3)

{

}

/// <summary>

/// Generic constructor. Please use specific (non-SEPA or SEPA) constructor

/// </summary>

/// <param name="authority">Type of the bank transfer</param>

/// <param name="name">Name of the receiver (Empfänger)</param>

/// <param name="account">Bank account (Kontonummer)</param>

/// <param name="bnc">Bank institute (Bankleitzahl)</param>

/// <param name="iban">IBAN</param>

/// <param name="bic">BIC</param>

/// <param name="amount">Amount (Betrag)</param>

/// <param name="periodicTimeunit">Unit of intervall for payment ('M' = monthly, 'W' = weekly)</param>

/// <param name="periodicTimeunitRotation">Intervall for payment. This value is combined with 'periodicTimeunit'</param>

/// <param name="periodicFirstExecutionDate">Date of first periodic execution</param>

/// <param name="periodicLastExecutionDate">Date of last periodic execution</param>

/// <param name="creditorId">Creditor id (Gläubiger ID)</param>

/// <param name="mandateId">Manadate id (Mandatsreferenz)</param>

/// <param name="dateOfSignature">Signature date (Erteilungsdatum des Mandats)</param>

/// <param name="reason">Reason (Verwendungszweck)</param>

/// <param name="postingKey">Transfer Key (Textschlüssel, z.B. Spendenzahlung = 69)</param>

/// <param name="sepaReference">SEPA reference (SEPA-Referenz)</param>

/// <param name="currency">Currency (Währung)</param>

/// <param name="executionDate">Execution date (Ausführungsdatum)</param>

/// <param name="internalMode">Only used for internal state handdling</param>

public BezahlCode(AuthorityType authority, string name, string account, string bnc, string iban, string bic, decimal amount, string periodicTimeunit = "", int periodicTimeunitRotation = 0, DateTime? periodicFirstExecutionDate = null, DateTime? periodicLastExecutionDate = null, string creditorId = "", string mandateId = "", DateTime? dateOfSignature = null, string reason = "", int postingKey = 0, string sepaReference = "", Currency currency = Currency.EUR, DateTime? executionDate = null, int internalMode = 0)

{

//Loaded via "contact-constructor"

if (internalMode == 1)

{

if (authority != AuthorityType.contact && authority != AuthorityType.contact\_v2)

throw new BezahlCodeException("The constructor without an amount may only ne used with authority types 'contact' and 'contact\_v2'.");

if (authority == AuthorityType.contact && (string.IsNullOrEmpty(account) || string.IsNullOrEmpty(bnc)))

throw new BezahlCodeException("When using authority type 'contact' the parameters 'account' and 'bnc' must be set.");

if (authority != AuthorityType.contact\_v2)

{

var oldFilled = (!string.IsNullOrEmpty(account) && !string.IsNullOrEmpty(bnc));

var newFilled = (!string.IsNullOrEmpty(iban) && !string.IsNullOrEmpty(bic));

if ((!oldFilled && !newFilled) || (oldFilled && newFilled))

throw new BezahlCodeException("When using authority type 'contact\_v2' either the parameters 'account' and 'bnc' or the parameters 'iban' and 'bic' must be set. Leave the other parameter pair empty.");

}

}

else if (internalMode == 2)

{

#pragma warning disable CS0612

if (authority != AuthorityType.periodicsinglepayment && authority != AuthorityType.singledirectdebit && authority != AuthorityType.singlepayment)

throw new BezahlCodeException("The constructor with 'account' and 'bnc' may only be used with 'non SEPA' authority types. Either choose another authority type or switch constructor.");

if (authority == AuthorityType.periodicsinglepayment && (string.IsNullOrEmpty(periodicTimeunit) || periodicTimeunitRotation == 0))

throw new BezahlCodeException("When using 'periodicsinglepayment' as authority type, the parameters 'periodicTimeunit' and 'periodicTimeunitRotation' must be set.");

#pragma warning restore CS0612

}

else if (internalMode == 3)

{

if (authority != AuthorityType.periodicsinglepaymentsepa && authority != AuthorityType.singledirectdebitsepa && authority != AuthorityType.singlepaymentsepa)

throw new BezahlCodeException("The constructor with 'iban' and 'bic' may only be used with 'SEPA' authority types. Either choose another authority type or switch constructor.");

if (authority == AuthorityType.periodicsinglepaymentsepa && (string.IsNullOrEmpty(periodicTimeunit) || periodicTimeunitRotation == 0))

throw new BezahlCodeException("When using 'periodicsinglepaymentsepa' as authority type, the parameters 'periodicTimeunit' and 'periodicTimeunitRotation' must be set.");

}

this.authority = authority;

if (name.Length > 70)

throw new BezahlCodeException("(Payee-)Name must be shorter than 71 chars.");

this.name = name;

if (reason.Length > 27)

throw new BezahlCodeException("Reasons texts have to be shorter than 28 chars.");

this.reason = reason;

var oldWayFilled = (!string.IsNullOrEmpty(account) && !string.IsNullOrEmpty(bnc));

var newWayFilled = (!string.IsNullOrEmpty(iban) && !string.IsNullOrEmpty(bic));

//Non-SEPA payment types

#pragma warning disable CS0612

if (authority == AuthorityType.periodicsinglepayment || authority == AuthorityType.singledirectdebit || authority == AuthorityType.singlepayment || authority == AuthorityType.contact || (authority == AuthorityType.contact\_v2 && oldWayFilled))

{

#pragma warning restore CS0612

if (!Regex.IsMatch(account.Replace(" ", ""), @"^[0-9]{1,9}$"))

throw new BezahlCodeException("The account entered isn't valid.");

this.account = account.Replace(" ", "").ToUpper();

if(!Regex.IsMatch(bnc.Replace(" ", ""), @"^[0-9]{1,9}$"))

throw new BezahlCodeException("The bnc entered isn't valid.");

this.bnc = bnc.Replace(" ", "").ToUpper();

if (authority != AuthorityType.contact && authority != AuthorityType.contact\_v2)

{

if (postingKey < 0 || postingKey >= 100)

throw new BezahlCodeException("PostingKey must be within 0 and 99.");

this.postingKey = postingKey;

}

}

//SEPA payment types

if (authority == AuthorityType.periodicsinglepaymentsepa || authority == AuthorityType.singledirectdebitsepa || authority == AuthorityType.singlepaymentsepa || (authority == AuthorityType.contact\_v2 && newWayFilled))

{

if (!IsValidIban(iban))

throw new BezahlCodeException("The IBAN entered isn't valid.");

this.iban = iban.Replace(" ", "").ToUpper();

if (!IsValidBic(bic))

throw new BezahlCodeException("The BIC entered isn't valid.");

this.bic = bic.Replace(" ", "").ToUpper();

if (authority != AuthorityType.contact\_v2)

{

if (sepaReference.Length > 35)

throw new BezahlCodeException("SEPA reference texts have to be shorter than 36 chars.");

this.sepaReference = sepaReference;

if (!string.IsNullOrEmpty(creditorId) && !Regex.IsMatch(creditorId.Replace(" ", ""), @"^[a-zA-Z]{2,2}[0-9]{2,2}([A-Za-z0-9]|[\+|\?|/|\-|:|\(|\)|\.|,|']){3,3}([A-Za-z0-9]|[\+|\?|/|\-|:|\(|\)|\.|,|']){1,28}$"))

throw new BezahlCodeException("The creditorId entered isn't valid.");

this.creditorId = creditorId;

if (!string.IsNullOrEmpty(mandateId) && !Regex.IsMatch(mandateId.Replace(" ", ""), @"^([A-Za-z0-9]|[\+|\?|/|\-|:|\(|\)|\.|,|']){1,35}$"))

throw new BezahlCodeException("The mandateId entered isn't valid.");

this.mandateId = mandateId;

if (dateOfSignature != null)

this.dateOfSignature = (DateTime)dateOfSignature;

}

}

//Checks for all payment types

if (authority != AuthorityType.contact && authority != AuthorityType.contact\_v2)

{

if (amount.ToString().Replace(",", ".").Contains(".") && amount.ToString().Replace(",", ".").Split('.')[1].TrimEnd('0').Length > 2)

throw new BezahlCodeException("Amount must have less than 3 digits after decimal point.");

if (amount < 0.01m || amount > 999999999.99m)

throw new BezahlCodeException("Amount has to at least 0.01 and must be smaller or equal to 999999999.99.");

this.amount = amount;

this.currency = currency;

if (executionDate == null)

this.executionDate = DateTime.Now;

else

{

if (DateTime.Today.Ticks > executionDate.Value.Ticks)

throw new BezahlCodeException("Execution date must be today or in future.");

this.executionDate = (DateTime)executionDate;

}

#pragma warning disable CS0612

if (authority == AuthorityType.periodicsinglepayment || authority == AuthorityType.periodicsinglepaymentsepa)

#pragma warning restore CS0612

{

if (periodicTimeunit.ToUpper() != "M" && periodicTimeunit.ToUpper() != "W")

throw new BezahlCodeException("The periodicTimeunit must be either 'M' (monthly) or 'W' (weekly).");

this.periodicTimeunit = periodicTimeunit;

if (periodicTimeunitRotation < 1 || periodicTimeunitRotation > 52)

throw new BezahlCodeException("The periodicTimeunitRotation must be 1 or greater. (It means repeat the payment every 'periodicTimeunitRotation' weeks/months.");

this.periodicTimeunitRotation = periodicTimeunitRotation;

if (periodicFirstExecutionDate != null)

this.periodicFirstExecutionDate = (DateTime)periodicFirstExecutionDate;

if (periodicLastExecutionDate != null)

this.periodicLastExecutionDate = (DateTime)periodicLastExecutionDate;

}

}

}

public override string ToString()

{

var bezahlCodePayload = $"bank://{authority}?";

bezahlCodePayload += $"name={Uri.EscapeDataString(name)}&";

if (authority != AuthorityType.contact && authority != AuthorityType.contact\_v2)

{

//Handle what is same for all payments

#pragma warning disable CS0612

if (authority == AuthorityType.periodicsinglepayment || authority == AuthorityType.singledirectdebit || authority == AuthorityType.singlepayment)

#pragma warning restore CS0612

{

bezahlCodePayload += $"account={account}&";

bezahlCodePayload += $"bnc={bnc}&";

if (postingKey > 0)

bezahlCodePayload += $"postingkey={postingKey}&";

}

else

{

bezahlCodePayload += $"iban={iban}&";

bezahlCodePayload += $"bic={bic}&";

if (!string.IsNullOrEmpty(sepaReference))

bezahlCodePayload += $"separeference={ Uri.EscapeDataString(sepaReference)}&";

if (authority == AuthorityType.singledirectdebitsepa)

{

if (!string.IsNullOrEmpty(creditorId))

bezahlCodePayload += $"creditorid={ Uri.EscapeDataString(creditorId)}&";

if (!string.IsNullOrEmpty(mandateId))

bezahlCodePayload += $"mandateid={ Uri.EscapeDataString(mandateId)}&";

if (dateOfSignature != DateTime.MinValue)

bezahlCodePayload += $"dateofsignature={dateOfSignature.ToString("ddMMyyyy")}&";

}

}

bezahlCodePayload += $"amount={amount:0.00}&".Replace(".", ",");

if (!string.IsNullOrEmpty(reason))

bezahlCodePayload += $"reason={ Uri.EscapeDataString(reason)}&";

bezahlCodePayload += $"currency={currency}&";

bezahlCodePayload += $"executiondate={executionDate.ToString("ddMMyyyy")}&";

#pragma warning disable CS0612

if (authority == AuthorityType.periodicsinglepayment || authority == AuthorityType.periodicsinglepaymentsepa)

{

bezahlCodePayload += $"periodictimeunit={periodicTimeunit}&";

bezahlCodePayload += $"periodictimeunitrotation={periodicTimeunitRotation}&";

if (periodicFirstExecutionDate != DateTime.MinValue)

bezahlCodePayload += $"periodicfirstexecutiondate={periodicFirstExecutionDate.ToString("ddMMyyyy")}&";

if (periodicLastExecutionDate != DateTime.MinValue)

bezahlCodePayload += $"periodiclastexecutiondate={periodicLastExecutionDate.ToString("ddMMyyyy")}&";

}

#pragma warning restore CS0612

}

else

{

//Handle what is same for all contacts

if (authority == AuthorityType.contact)

{

bezahlCodePayload += $"account={account}&";

bezahlCodePayload += $"bnc={bnc}&";

}

else if (authority == AuthorityType.contact\_v2)

{

if (!string.IsNullOrEmpty(account) && !string.IsNullOrEmpty(bnc))

{

bezahlCodePayload += $"account={account}&";

bezahlCodePayload += $"bnc={bnc}&";

}

else

{

bezahlCodePayload += $"iban={iban}&";

bezahlCodePayload += $"bic={bic}&";

}

}

if (!string.IsNullOrEmpty(reason))

bezahlCodePayload += $"reason={ Uri.EscapeDataString(reason)}&";

}

return bezahlCodePayload.Trim('&');

}

/// <summary>

/// ISO 4217 currency codes

/// </summary>

public enum Currency

{

AED = 784,

AFN = 971,

ALL = 008,

AMD = 051,

ANG = 532,

AOA = 973,

ARS = 032,

AUD = 036,

AWG = 533,

AZN = 944,

BAM = 977,

BBD = 052,

BDT = 050,

BGN = 975,

BHD = 048,

BIF = 108,

BMD = 060,

BND = 096,

BOB = 068,

BOV = 984,

BRL = 986,

BSD = 044,

BTN = 064,

BWP = 072,

BYR = 974,

BZD = 084,

CAD = 124,

CDF = 976,

CHE = 947,

CHF = 756,

CHW = 948,

CLF = 990,

CLP = 152,

CNY = 156,

COP = 170,

COU = 970,

CRC = 188,

CUC = 931,

CUP = 192,

CVE = 132,

CZK = 203,

DJF = 262,

DKK = 208,

DOP = 214,

DZD = 012,

EGP = 818,

ERN = 232,

ETB = 230,

EUR = 978,

FJD = 242,

FKP = 238,

GBP = 826,

GEL = 981,

GHS = 936,

GIP = 292,

GMD = 270,

GNF = 324,

GTQ = 320,

GYD = 328,

HKD = 344,

HNL = 340,

HRK = 191,

HTG = 332,

HUF = 348,

IDR = 360,

ILS = 376,

INR = 356,

IQD = 368,

IRR = 364,

ISK = 352,

JMD = 388,

JOD = 400,

JPY = 392,

KES = 404,

KGS = 417,

KHR = 116,

KMF = 174,

KPW = 408,

KRW = 410,

KWD = 414,

KYD = 136,

KZT = 398,

LAK = 418,

LBP = 422,

LKR = 144,

LRD = 430,

LSL = 426,

LYD = 434,

MAD = 504,

MDL = 498,

MGA = 969,

MKD = 807,

MMK = 104,

MNT = 496,

MOP = 446,

MRO = 478,

MUR = 480,

MVR = 462,

MWK = 454,

MXN = 484,

MXV = 979,

MYR = 458,

MZN = 943,

NAD = 516,

NGN = 566,

NIO = 558,

NOK = 578,

NPR = 524,

NZD = 554,

OMR = 512,

PAB = 590,

PEN = 604,

PGK = 598,

PHP = 608,

PKR = 586,

PLN = 985,

PYG = 600,

QAR = 634,

RON = 946,

RSD = 941,

RUB = 643,

RWF = 646,

SAR = 682,

SBD = 090,

SCR = 690,

SDG = 938,

SEK = 752,

SGD = 702,

SHP = 654,

SLL = 694,

SOS = 706,

SRD = 968,

SSP = 728,

STD = 678,

SVC = 222,

SYP = 760,

SZL = 748,

THB = 764,

TJS = 972,

TMT = 934,

TND = 788,

TOP = 776,

TRY = 949,

TTD = 780,

TWD = 901,

TZS = 834,

UAH = 980,

UGX = 800,

USD = 840,

USN = 997,

UYI = 940,

UYU = 858,

UZS = 860,

VEF = 937,

VND = 704,

VUV = 548,

WST = 882,

XAF = 950,

XAG = 961,

XAU = 959,

XBA = 955,

XBB = 956,

XBC = 957,

XBD = 958,

XCD = 951,

XDR = 960,

XOF = 952,

XPD = 964,

XPF = 953,

XPT = 962,

XSU = 994,

XTS = 963,

XUA = 965,

XXX = 999,

YER = 886,

ZAR = 710,

ZMW = 967,

ZWL = 932

}

/// <summary>

/// Operation modes of the BezahlCode

/// </summary>

public enum AuthorityType

{

/// <summary>

/// Single payment (Überweisung)

/// </summary>

[Obsolete]

singlepayment,

/// <summary>

/// Single SEPA payment (SEPA-Überweisung)

/// </summary>

singlepaymentsepa,

/// <summary>

/// Single debit (Lastschrift)

/// </summary>

[Obsolete]

singledirectdebit,

/// <summary>

/// Single SEPA debit (SEPA-Lastschrift)

/// </summary>

singledirectdebitsepa,

/// <summary>

/// Periodic payment (Dauerauftrag)

/// </summary>

[Obsolete]

periodicsinglepayment,

/// <summary>

/// Periodic SEPA payment (SEPA-Dauerauftrag)

/// </summary>

periodicsinglepaymentsepa,

/// <summary>

/// Contact data

/// </summary>

contact,

/// <summary>

/// Contact data V2

/// </summary>

contact\_v2

}

public class BezahlCodeException : Exception

{

public BezahlCodeException()

{

}

public BezahlCodeException(string message)

: base(message)

{

}

public BezahlCodeException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class CalendarEvent : Payload

{

private readonly string subject, description, location, start, end;

private readonly EventEncoding encoding;

/// <summary>

/// Generates a calender entry/event payload.

/// </summary>

/// <param name="subject">Subject/title of the calender event</param>

/// <param name="description">Description of the event</param>

/// <param name="location">Location (lat:long or address) of the event</param>

/// <param name="start">Start time of the event</param>

/// <param name="end">End time of the event</param>

/// <param name="allDayEvent">Is it a full day event?</param>

/// <param name="encoding">Type of encoding (universal or iCal)</param>

public CalendarEvent(string subject, string description, string location, DateTime start, DateTime end, bool allDayEvent, EventEncoding encoding = EventEncoding.Universal)

{

this.subject = subject;

this.description = description;

this.location = location;

this.encoding = encoding;

string dtFormat = allDayEvent ? "yyyyMMdd" : "yyyyMMddTHHmmss";

this.start = start.ToString(dtFormat);

this.end = end.ToString(dtFormat);

}

public override string ToString()

{

var vEvent = $"BEGIN:VEVENT{Environment.NewLine}";

vEvent += $"SUMMARY:{this.subject}{Environment.NewLine}";

vEvent += !string.IsNullOrEmpty(this.description) ? $"DESCRIPTION:{this.description}{Environment.NewLine}" : "";

vEvent += !string.IsNullOrEmpty(this.location) ? $"LOCATION:{this.location}{Environment.NewLine}" : "";

vEvent += $"DTSTART:{this.start}{Environment.NewLine}";

vEvent += $"DTEND:{this.end}{Environment.NewLine}";

vEvent += "END:VEVENT";

if (this.encoding == EventEncoding.iCalComplete)

vEvent = $@"BEGIN:VCALENDAR{Environment.NewLine}VERSION:2.0{Environment.NewLine}{vEvent}{Environment.NewLine}END:VCALENDAR";

return vEvent;

}

public enum EventEncoding

{

iCalComplete,

Universal

}

}

public class OneTimePassword : Payload

{

//https://github.com/google/google-authenticator/wiki/Key-Uri-Format

public OneTimePasswordAuthType Type { get; set; } = OneTimePasswordAuthType.TOTP;

public string Secret { get; set; }

public OneTimePasswordAuthAlgorithm AuthAlgorithm { get; set; } = OneTimePasswordAuthAlgorithm.SHA1;

[Obsolete("This property is obsolete, use " + nameof(AuthAlgorithm) + " instead", false)]

public OoneTimePasswordAuthAlgorithm Algorithm

{

get { return (OoneTimePasswordAuthAlgorithm)Enum.Parse(typeof(OoneTimePasswordAuthAlgorithm), AuthAlgorithm.ToString()); }

set { AuthAlgorithm = (OneTimePasswordAuthAlgorithm)Enum.Parse(typeof(OneTimePasswordAuthAlgorithm), value.ToString()); }

}

public string Issuer { get; set; }

public string Label { get; set; }

public int Digits { get; set; } = 6;

public int? Counter { get; set; } = null;

public int? Period { get; set; } = 30;

public enum OneTimePasswordAuthType

{

TOTP,

HOTP,

}

public enum OneTimePasswordAuthAlgorithm

{

SHA1,

SHA256,

SHA512,

}

[Obsolete("This enum is obsolete, use " + nameof(OneTimePasswordAuthAlgorithm) + " instead", false)]

public enum OoneTimePasswordAuthAlgorithm

{

SHA1,

SHA256,

SHA512,

}

public override string ToString()

{

switch (Type)

{

case OneTimePasswordAuthType.TOTP:

return TimeToString();

case OneTimePasswordAuthType.HOTP:

return HMACToString();

default:

throw new ArgumentOutOfRangeException();

}

}

// Note: Issuer:Label must only contain 1 : if either of the Issuer or the Label has a : then it is invalid.

// Defaults are 6 digits and 30 for Period

private string HMACToString()

{

var sb = new StringBuilder("otpauth://hotp/");

ProcessCommonFields(sb);

var actualCounter = Counter ?? 1;

sb.Append("&counter=" + actualCounter);

return sb.ToString();

}

private string TimeToString()

{

if (Period == null)

{

throw new Exception("Period must be set when using OneTimePasswordAuthType.TOTP");

}

var sb = new StringBuilder("otpauth://totp/");

ProcessCommonFields(sb);

if (Period != 30)

{

sb.Append("&period=" + Period);

}

return sb.ToString();

}

private void ProcessCommonFields(StringBuilder sb)

{

if (String40Methods.IsNullOrWhiteSpace(Secret))

{

throw new Exception("Secret must be a filled out base32 encoded string");

}

string strippedSecret = Secret.Replace(" ", "");

string escapedIssuer = null;

string escapedLabel = null;

if (!String40Methods.IsNullOrWhiteSpace(Issuer))

{

if (Issuer.Contains(":"))

{

throw new Exception("Issuer must not have a ':'");

}

escapedIssuer = Uri.EscapeUriString(Issuer);

}

if (!String40Methods.IsNullOrWhiteSpace(Label))

{

if (Label.Contains(":"))

{

throw new Exception("Label must not have a ':'");

}

escapedLabel = Uri.EscapeUriString(Label);

}

if (escapedLabel != null)

{

if (escapedIssuer != null)

{

escapedLabel = escapedIssuer + ":" + escapedLabel;

}

}

else if (escapedIssuer != null)

{

escapedLabel = escapedIssuer;

}

if (escapedLabel != null)

{

sb.Append(escapedLabel);

}

sb.Append("?secret=" + strippedSecret);

if (escapedIssuer != null)

{

sb.Append("&issuer=" + escapedIssuer);

}

if (Digits != 6)

{

sb.Append("&digits=" + Digits);

}

}

}

public class ShadowSocksConfig : Payload

{

private readonly string hostname, password, tag, methodStr, parameter;

private readonly Method method;

private readonly int port;

private Dictionary<string, string> encryptionTexts = new Dictionary<string, string>() {

{ "Chacha20IetfPoly1305", "chacha20-ietf-poly1305" },

{ "Aes128Gcm", "aes-128-gcm" },

{ "Aes192Gcm", "aes-192-gcm" },

{ "Aes256Gcm", "aes-256-gcm" },

{ "XChacha20IetfPoly1305", "xchacha20-ietf-poly1305" },

{ "Aes128Cfb", "aes-128-cfb" },

{ "Aes192Cfb", "aes-192-cfb" },

{ "Aes256Cfb", "aes-256-cfb" },

{ "Aes128Ctr", "aes-128-ctr" },

{ "Aes192Ctr", "aes-192-ctr" },

{ "Aes256Ctr", "aes-256-ctr" },

{ "Camellia128Cfb", "camellia-128-cfb" },

{ "Camellia192Cfb", "camellia-192-cfb" },

{ "Camellia256Cfb", "camellia-256-cfb" },

{ "Chacha20Ietf", "chacha20-ietf" },

{ "Aes256Cb", "aes-256-cfb" },

{ "Aes128Ofb", "aes-128-ofb" },

{ "Aes192Ofb", "aes-192-ofb" },

{ "Aes256Ofb", "aes-256-ofb" },

{ "Aes128Cfb1", "aes-128-cfb1" },

{ "Aes192Cfb1", "aes-192-cfb1" },

{ "Aes256Cfb1", "aes-256-cfb1" },

{ "Aes128Cfb8", "aes-128-cfb8" },

{ "Aes192Cfb8", "aes-192-cfb8" },

{ "Aes256Cfb8", "aes-256-cfb8" },

{ "Chacha20", "chacha20" },

{ "BfCfb", "bf-cfb" },

{ "Rc4Md5", "rc4-md5" },

{ "Salsa20", "salsa20" },

{ "DesCfb", "des-cfb" },

{ "IdeaCfb", "idea-cfb" },

{ "Rc2Cfb", "rc2-cfb" },

{ "Cast5Cfb", "cast5-cfb" },

{ "Salsa20Ctr", "salsa20-ctr" },

{ "Rc4", "rc4" },

{ "SeedCfb", "seed-cfb" },

{ "Table", "table" }

};

/// <summary>

/// Generates a ShadowSocks proxy config payload.

/// </summary>

/// <param name="hostname">Hostname of the ShadowSocks proxy</param>

/// <param name="port">Port of the ShadowSocks proxy</param>

/// <param name="password">Password of the SS proxy</param>

/// <param name="method">Encryption type</param>

/// <param name="tag">Optional tag line</param>

public ShadowSocksConfig(string hostname, int port, string password, Method method, string tag = null) :

this(hostname, port, password, method, null, tag)

{ }

public ShadowSocksConfig(string hostname, int port, string password, Method method, string plugin, string pluginOption, string tag = null) :

this(hostname, port, password, method, new Dictionary<string, string>

{

["plugin"] = plugin + (

string.IsNullOrEmpty(pluginOption)

? ""

: $";{pluginOption}"

)

}, tag)

{ }

private Dictionary<string, string> UrlEncodeTable = new Dictionary<string, string>

{

[" "] = "+",

["\0"] = "%00",

["\t"] = "%09",

["\n"] = "%0a",

["\r"] = "%0d",

["\""] = "%22",

["#"] = "%23",

["$"] = "%24",

["%"] = "%25",

["&"] = "%26",

["'"] = "%27",

["+"] = "%2b",

[","] = "%2c",

["/"] = "%2f",

[":"] = "%3a",

[";"] = "%3b",

["<"] = "%3c",

["="] = "%3d",

[">"] = "%3e",

["?"] = "%3f",

["@"] = "%40",

["["] = "%5b",

["\\"] = "%5c",

["]"] = "%5d",

["^"] = "%5e",

["`"] = "%60",

["{"] = "%7b",

["|"] = "%7c",

["}"] = "%7d",

["~"] = "%7e",

};

private string UrlEncode(string i)

{

string j = i;

foreach (var kv in UrlEncodeTable)

{

j = j.Replace(kv.Key, kv.Value);

}

return j;

}

public ShadowSocksConfig(string hostname, int port, string password, Method method, Dictionary<string, string> parameters, string tag = null)

{

this.hostname = Uri.CheckHostName(hostname) == UriHostNameType.IPv6

? $"[{hostname}]"

: hostname;

if (port < 1 || port > 65535)

throw new ShadowSocksConfigException("Value of 'port' must be within 0 and 65535.");

this.port = port;

this.password = password;

this.method = method;

this.methodStr = encryptionTexts[method.ToString()];

this.tag = tag;

if (parameters != null)

this.parameter =

string.Join("&",

parameters.Select(

kv => $"{UrlEncode(kv.Key)}={UrlEncode(kv.Value)}"

).ToArray());

}

public override string ToString()

{

if (string.IsNullOrEmpty(parameter))

{

var connectionString = $"{methodStr}:{password}@{hostname}:{port}";

var connectionStringEncoded = Convert.ToBase64String(Encoding.UTF8.GetBytes(connectionString));

return $"ss://{connectionStringEncoded}{(!string.IsNullOrEmpty(tag) ? $"#{tag}" : string.Empty)}";

}

var authString = $"{methodStr}:{password}";

var authStringEncoded = Convert.ToBase64String(Encoding.UTF8.GetBytes(authString))

.Replace('+', '-')

.Replace('/', '\_')

.TrimEnd('=');

return $"ss://{authStringEncoded}@{hostname}:{port}/?{parameter}{(!string.IsNullOrEmpty(tag) ? $"#{tag}" : string.Empty)}";

}

public enum Method

{

// AEAD

Chacha20IetfPoly1305,

Aes128Gcm,

Aes192Gcm,

Aes256Gcm,

// AEAD, not standard

XChacha20IetfPoly1305,

// Stream cipher

Aes128Cfb,

Aes192Cfb,

Aes256Cfb,

Aes128Ctr,

Aes192Ctr,

Aes256Ctr,

Camellia128Cfb,

Camellia192Cfb,

Camellia256Cfb,

Chacha20Ietf,

// alias of Aes256Cfb

Aes256Cb,

// Stream cipher, not standard

Aes128Ofb,

Aes192Ofb,

Aes256Ofb,

Aes128Cfb1,

Aes192Cfb1,

Aes256Cfb1,

Aes128Cfb8,

Aes192Cfb8,

Aes256Cfb8,

// Stream cipher, deprecated

Chacha20,

BfCfb,

Rc4Md5,

Salsa20,

// Not standard and not in acitve use

DesCfb,

IdeaCfb,

Rc2Cfb,

Cast5Cfb,

Salsa20Ctr,

Rc4,

SeedCfb,

Table

}

public class ShadowSocksConfigException : Exception

{

public ShadowSocksConfigException()

{

}

public ShadowSocksConfigException(string message)

: base(message)

{

}

public ShadowSocksConfigException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class MoneroTransaction : Payload

{

private readonly string address, txPaymentId, recipientName, txDescription;

private readonly float? txAmount;

/// <summary>

/// Creates a monero transaction payload

/// </summary>

/// <param name="address">Receiver's monero address</param>

/// <param name="txAmount">Amount to transfer</param>

/// <param name="txPaymentId">Payment id</param>

/// <param name="recipientName">Receipient's name</param>

/// <param name="txDescription">Reference text / payment description</param>

public MoneroTransaction(string address, float? txAmount = null, string txPaymentId = null, string recipientName = null, string txDescription = null)

{

if (string.IsNullOrEmpty(address))

throw new MoneroTransactionException("The address is mandatory and has to be set.");

this.address = address;

if (txAmount != null && txAmount <= 0)

throw new MoneroTransactionException("Value of 'txAmount' must be greater than 0.");

this.txAmount = txAmount;

this.txPaymentId = txPaymentId;

this.recipientName = recipientName;

this.txDescription = txDescription;

}

public override string ToString()

{

var moneroUri = $"monero://{address}{(!string.IsNullOrEmpty(txPaymentId) || !string.IsNullOrEmpty(recipientName) || !string.IsNullOrEmpty(txDescription) || txAmount != null ? "?" : string.Empty)}";

moneroUri += (!string.IsNullOrEmpty(txPaymentId) ? $"tx\_payment\_id={Uri.EscapeDataString(txPaymentId)}&" : string.Empty);

moneroUri += (!string.IsNullOrEmpty(recipientName) ? $"recipient\_name={Uri.EscapeDataString(recipientName)}&" : string.Empty);

moneroUri += (txAmount != null ? $"tx\_amount={txAmount.ToString().Replace(",",".")}&" : string.Empty);

moneroUri += (!string.IsNullOrEmpty(txDescription) ? $"tx\_description={Uri.EscapeDataString(txDescription)}" : string.Empty);

return moneroUri.TrimEnd('&');

}

public class MoneroTransactionException : Exception

{

public MoneroTransactionException()

{

}

public MoneroTransactionException(string message)

: base(message)

{

}

public MoneroTransactionException(string message, Exception inner)

: base(message, inner)

{

}

}

}

public class SlovenianUpnQr : Payload

{

//Keep in mind, that the ECC level has to be set to "M", version to 15 and ECI to EciMode.Iso8859\_2 when generating a SlovenianUpnQr!

//SlovenianUpnQr specification: https://www.upn-qr.si/uploads/files/NavodilaZaProgramerjeUPNQR.pdf

private string \_payerName = "";

private string \_payerAddress = "";

private string \_payerPlace = "";

private string \_amount = "";

private string \_code = "";

private string \_purpose = "";

private string \_deadLine = "";

private string \_recipientIban = "";

private string \_recipientName = "";

private string \_recipientAddress = "";

private string \_recipientPlace = "";

private string \_recipientSiModel = "";

private string \_recipientSiReference = "";

public override int Version { get { return 15; } }

public override QRCodeGenerator.ECCLevel EccLevel { get { return QRCodeGenerator.ECCLevel.M; } }

public override QRCodeGenerator.EciMode EciMode { get { return QRCodeGenerator.EciMode.Iso8859\_2; } }

private string LimitLength(string value, int maxLength)

{

return (value.Length <= maxLength) ? value : value.Substring(0, maxLength);

}

public SlovenianUpnQr(string payerName, string payerAddress, string payerPlace, string recipientName, string recipientAddress, string recipientPlace, string recipientIban, string description, double amount, string recipientSiModel = "SI00", string recipientSiReference = "", string code = "OTHR") :

this(payerName, payerAddress, payerPlace, recipientName, recipientAddress, recipientPlace, recipientIban, description, amount, null, recipientSiModel, recipientSiReference, code)

{ }

public SlovenianUpnQr(string payerName, string payerAddress, string payerPlace, string recipientName, string recipientAddress, string recipientPlace, string recipientIban, string description, double amount, DateTime? deadline, string recipientSiModel = "SI99", string recipientSiReference = "", string code = "OTHR")

{

\_payerName = LimitLength(payerName.Trim(), 33);

\_payerAddress = LimitLength(payerAddress.Trim(), 33);

\_payerPlace = LimitLength(payerPlace.Trim(), 33);

\_amount = FormatAmount(amount);

\_code = LimitLength(code.Trim().ToUpper(), 4);

\_purpose = LimitLength(description.Trim(), 42);

\_deadLine = (deadline == null) ? "" : deadline?.ToString("dd.MM.yyyy");

\_recipientIban = LimitLength(recipientIban.Trim(), 34);

\_recipientName = LimitLength(recipientName.Trim(), 33);

\_recipientAddress = LimitLength(recipientAddress.Trim(), 33);

\_recipientPlace = LimitLength(recipientPlace.Trim(), 33);

\_recipientSiModel = LimitLength(recipientSiModel.Trim().ToUpper(), 4);

\_recipientSiReference = LimitLength(recipientSiReference.Trim(), 22);

}

private string FormatAmount(double amount)

{

int \_amt = (int)Math.Round(amount \* 100.0);

return String.Format("{0:00000000000}", \_amt);

}

private int CalculateChecksum()

{

int \_cs = 5 + \_payerName.Length; //5 = UPNQR constant Length

\_cs += \_payerAddress.Length;

\_cs += \_payerPlace.Length;

\_cs += \_amount.Length;

\_cs += \_code.Length;

\_cs += \_purpose.Length;

\_cs += \_deadLine.Length;

\_cs += \_recipientIban.Length;

\_cs += \_recipientName.Length;

\_cs += \_recipientAddress.Length;

\_cs += \_recipientPlace.Length;

\_cs += \_recipientSiModel.Length;

\_cs += \_recipientSiReference.Length;

\_cs += 19;

return \_cs;

}

public override string ToString()

{

var \_sb = new StringBuilder();

\_sb.Append("UPNQR");

\_sb.Append('\n').Append('\n').Append('\n').Append('\n').Append('\n');

\_sb.Append(\_payerName).Append('\n');

\_sb.Append(\_payerAddress).Append('\n');

\_sb.Append(\_payerPlace).Append('\n');

\_sb.Append(\_amount).Append('\n').Append('\n').Append('\n');

\_sb.Append(\_code.ToUpper()).Append('\n');

\_sb.Append(\_purpose).Append('\n');

\_sb.Append(\_deadLine).Append('\n');

\_sb.Append(\_recipientIban.ToUpper()).Append('\n');

\_sb.Append(\_recipientSiModel).Append(\_recipientSiReference).Append('\n');

\_sb.Append(\_recipientName).Append('\n');

\_sb.Append(\_recipientAddress).Append('\n');

\_sb.Append(\_recipientPlace).Append('\n');

\_sb.AppendFormat("{0:000}", CalculateChecksum()).Append('\n');

return \_sb.ToString();

}

}

private static bool IsValidIban(string iban)

{

//Clean IBAN

var ibanCleared = iban.ToUpper().Replace(" ", "").Replace("-", "");

//Check for general structure

var structurallyValid = Regex.IsMatch(ibanCleared, @"^[a-zA-Z]{2}[0-9]{2}([a-zA-Z0-9]?){16,30}$");

//Check IBAN checksum

var checksumValid = false;

var sum = $"{ibanCleared.Substring(4)}{ibanCleared.Substring(0, 4)}".ToCharArray().Aggregate("", (current, c) => current + (char.IsLetter(c) ? (c - 55).ToString() : c.ToString()));

int m = 0;

for (int i = 0; i < (int)Math.Ceiling((sum.Length - 2) / 7d); i++){

var offset = (i == 0 ? 0 : 2);

var start = i \* 7 + offset;

var n = (i == 0 ? "" : m.ToString()) + sum.Substring(start, Math.Min(9 - offset, sum.Length - start));

if (!int.TryParse(n, NumberStyles.Any, CultureInfo.InvariantCulture, out m))

break;

m = m % 97;

}

checksumValid = m == 1;

return structurallyValid && checksumValid;

}

private static bool IsValidQRIban(string iban)

{

var foundQrIid = false;

try

{

var ibanCleared = iban.ToUpper().Replace(" ", "").Replace("-", "");

var possibleQrIid = Convert.ToInt32(ibanCleared.Substring(4, 5));

foundQrIid = possibleQrIid >= 30000 && possibleQrIid <= 31999;

} catch { }

return IsValidIban(iban) && foundQrIid;

}

private static bool IsValidBic(string bic)

{

return Regex.IsMatch(bic.Replace(" ", ""), @"^([a-zA-Z]{4}[a-zA-Z]{2}[a-zA-Z0-9]{2}([a-zA-Z0-9]{3})?)$");

}

private static string ConvertStringToEncoding(string message, string encoding)

{

Encoding iso = Encoding.GetEncoding(encoding);

Encoding utf8 = Encoding.UTF8;

byte[] utfBytes = utf8.GetBytes(message);

byte[] isoBytes = Encoding.Convert(utf8, iso, utfBytes);

#if NET40

return iso.GetString(isoBytes);

#else

return iso.GetString(isoBytes,0, isoBytes.Length);

#endif

}

private static string EscapeInput(string inp, bool simple = false)

{

char[] forbiddenChars = {'\\', ';', ',', ':'};

if (simple)

{

forbiddenChars = new char[1] {':'};

}

foreach (var c in forbiddenChars)

{

inp = inp.Replace(c.ToString(), "\\" + c);

}

return inp;

}

public static bool ChecksumMod10(string digits)

{

if (string.IsNullOrEmpty(digits) || digits.Length < 2)

return false;

int[] mods = new int[] { 0, 9, 4, 6, 8, 2, 7, 1, 3, 5 };

int remainder = 0;

for (int i = 0; i < digits.Length - 1; i++)

{

var num = Convert.ToInt32(digits[i]) - 48;

remainder = mods[(num + remainder) % 10];

}

var checksum = (10 - remainder) % 10;

return checksum == Convert.ToInt32(digits[digits.Length - 1]) - 48;

}

private static bool isHexStyle(string inp)

{

return (System.Text.RegularExpressions.Regex.IsMatch(inp, @"\A\b[0-9a-fA-F]+\b\Z") || System.Text.RegularExpressions.Regex.IsMatch(inp, @"\A\b(0[xX])?[0-9a-fA-F]+\b\Z"));

}

}

}