readme.md for @push.rocks/smartmail

A unified format for representing and dealing with emails, with support for attachments, email validation, dynamic templating, and wire format serialization for microservice communication.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
pnpm add @push.rocks/smartmail
```

Features

- □□ **Advanced Email Address Validation** - Validate email format, check for disposable/free domains, verify MX records, and detect role accounts
- □□ **Rich Email Representation** - Create emails with multiple recipients (to/cc/bcc), attachments, HTML content, and custom headers
- □□ **Template Support** - Use mustache templates for dynamic content in subject, body, and HTML
- □□ **MIME Formatting** - Convert emails to standard MIME format compatible with nodemailer and other sending libraries
- ⚡ **Caching & Performance** - Smart caching of DNS lookups for better performance
- □□ **Creation Object Reference** - Associate arbitrary typed data with emails for tracking and context preservation
- □□ **Fluent Chainable API** - All methods return `this` for elegant method chaining

- □□ **Wire Format Serialization** - JSON serialization with base64 attachments for microservice communication
- □□ **Wire Protocol Classes** - `WireTarget` and `WireParser` for client-server email service architecture

Usage

Importing the Module

```
import {
  Smartmail,
  EmailAddressValidator,
  WireTarget,
  WireParser,
  createMessageId,
  createTimestamp
} from '@push.rocks/smartmail';
```

□□ Fluent Chainable API

All mutation methods return `this`, enabling elegant method chaining:

```
const response = await new Smartmail({
  from: 'sender@example.com',
  subject: 'Hello {{name}}',
  body: 'Welcome to our service!'
})
  .addRecipient('user@example.com')
  .addRecipient('manager@example.com', 'cc')
  .addRecipients(['team1@example.com', 'team2@example.com'], 'bcc')
  .setReplyTo('support@example.com')
  .setPriority('high')
  .addHeader('X-Campaign-ID', '12345')
  .applyVariables({ name: 'John' })
  .sendTo(wireTarget);  // Send directly to a WireTarget
```

Email Address Validation

Basic Email Validation

```
const emailValidator = new EmailAddressValidator();

const result = await emailValidator.validate('user@example.com');
console.log(result);
/*
{
  valid: true,           // Overall validity
  formatValid: true,     // Email format is valid
  localPartValid: true,  // Local part (before @) is valid
  domainPartValid: true, // Domain part (after @) is valid
  mxValid: true,         // Domain has valid MX records
  disposable: false,     // Not a disposable email domain
  freemail: false,       // Not a free email provider
  reason: 'Email is valid'
}
*/
```

Advanced Validation Options

```
const validator = new EmailAddressValidator({
  skipOnlineDomainFetch: true,  // Use only local domain list
  cacheDnsResults: true,        // Cache DNS lookups
  cacheExpiryMs: 7200000        // Cache expires after 2 hours
});

// Validate each part separately
const isValidFormat = validator.isValidEmailFormat('user@example.com');
const isValidLocalPart = validator.isValidLocalPart('user');
const isValidDomain = validator.isValidDomainPart('example.com');

// Check for disposable or free email providers
const result = await validator.validate('user@gmail.com');
```

```
if (result.freemail) {
  console.log('This is a free email provider');
}
```

Checking for Disposable Emails and Role Accounts

```
const validator = new EmailAddressValidator();

// Check if an email is from a disposable domain
const isDisposable = await validator.isDisposableEmail('user@tempmail.com');

// Check if an email is a role account (info@, support@, admin@, etc.)
const isRole = validator.isRoleAccount('support@example.com');

// Get MX records for a domain
const mxRecords = await validator.getMxRecords('example.com');
// ['mx1.example.com', 'mx2.example.com']
```

□□ Creating and Using Smartmail Objects

Basic Email Creation

```
const email = new Smartmail({
  from: 'sender@example.com',
  to: ['recipient@example.com'],
  subject: 'Hello from SmartMail',
  body: 'This is a plain text email'
});
```

Using Creation Object Reference

Associate any typed data with your email for tracking and context preservation:

```
interface OrderNotification {
  orderId: string;
  customerName: string;
  items: string[];
  total: number;
}

const orderEmail = new Smartmail<OrderNotification>({
  from: 'orders@example.com',
  to: ['customer@example.com'],
  subject: 'Your Order #{{orderId}} Confirmation',
  body: 'Thank you for your order, {{customerName}}!',
  creationObjectRef: {
    orderId: '12345',
    customerName: 'John Smith',
    items: ['Product A', 'Product B'],
    total: 99.95
  }
});

// Later, retrieve the original reference data
const orderData = orderEmail.getCreationObject();
console.log(`Processing email for order ${orderData.orderId}`);

// Use the reference data for templating
const subject = orderEmail.getSubject(orderData);  // "Your Order #12345 Confirmation"
```

Template Variables in Subject and Body

```
const template = new Smartmail({
  from: 'notifications@example.com',
  subject: 'Welcome, {{name}}!',
  body: 'Hello {{name}},\n\nYour account ({{email}}) has been activated.',
  htmlBody: '<h1>Welcome, {{name}}!</h1><p>Your account (<strong>{{email}}</strong>) has been
activated.</p>'
});
```

```
// Apply template variables when retrieving content
const subject = template.getSubject({ name: 'John Doe' });
const plainBody = template.getBody({ name: 'John Doe', email: 'john@example.com' });

// Or apply variables directly (modifies in place, chainable)
template.applyVariables({ name: 'John Doe', email: 'john@example.com' });
```

Adding Attachments

```
import { SmartFile } from '@push.rocks/smartfile';

const email = new Smartmail({
  from: 'sender@example.com',
  to: ['recipient@example.com'],
  subject: 'Report Attached',
  body: 'Please find the attached report.'
});

const report = await SmartFile.fromFilePath('/path/to/report.pdf');
const image = await SmartFile.fromFilePath('/path/to/image.png');

email
  .addAttachment(report)
  .addAttachment(image);
```

Converting to MIME Format

```
const email = new Smartmail({
  from: 'sender@example.com',
  to: ['recipient@example.com'],
  subject: 'Hello {{name}}',
  body: 'Text version: Hello {{name}}',
  htmlBody: '<p>HTML version: Hello <strong>{{name}}</strong></p>',
  validateEmails: true
});

const mimeObj = await email.toMimeFormat({ name: 'John' });
```

```
// Use with nodemailer or other email sending libraries
```

□□ Wire Format Serialization

Smartmail provides JSON serialization for transmitting emails between microservices. Attachments are automatically encoded as base64.

Serializing to JSON

```
const email = new Smartmail({
  from: 'sender@example.com',
  to: ['recipient@example.com'],
  subject: 'Wire Format Test',
  body: 'This email can be serialized!'
})
  .addAttachment(someFile)
  .setPriority('high');

// Serialize to JSON object
const jsonObject = email.toObject();

// Serialize to JSON string
const jsonString = email.toJson();
```

Deserializing from JSON

```
// From JSON object
const email1 = Smartmail.fromObject(jsonObject);

// From JSON string
const email2 = Smartmail.fromJson(jsonString);

// Attachments are automatically reconstructed from base64
console.log(email2.attachments[0].contentBuffer);
```

□□ Wire Protocol Classes

For microservice architectures, smartmail provides `WireTarget` and `WireParser` classes to handle client-server communication.

WireTarget (Client/SaaS Side)

The `WireTarget` is used by your SaaS application to communicate with an SMTP service:

```
import { WireTarget, Smartmail } from '@push.rocks/smartmail';

// Configure the target endpoint
const smtpTarget = new WireTarget({
  endpoint: 'https://smtp-service.example.com/api/wire',
  authToken: 'your-secret-token'
});

// Submit SMTP settings (extensible - add any custom settings)
await smtpTarget.updateSettings({
  smtp: {
    host: 'smtp.example.com',
    port: 587,
    secure: true,
    username: 'user',
    password: 'pass'
  },
  defaultFrom: 'noreply@example.com',
  customSetting: 'custom-value'  // Extensible!
});

// Send an email using fluent API
const response = await new Smartmail({
  from: 'sender@example.com',
  subject: 'Hello {{name}}',
  body: 'Welcome!'
})
  .addRecipient('user@example.com')
  .setPriority('high')
```

```
  .applyVariables({ name: 'John' })
  .sendTo(smtpTarget);

console.log(`Sent! Delivery ID: ${response.deliveryId}`);

// Check delivery status
const status = await smtpTarget.getStatus(response.deliveryId);
console.log(`Status: ${status.status}`);  // 'queued' | 'sending' | 'sent' | 'failed'

// List emails in a mailbox
const inbox = await smtpTarget.listMailbox('INBOX', { limit: 10, offset: 0 });
console.log(`Found ${inbox.total} emails`);

// Fetch a specific email
const fetchedEmail = await smtpTarget.fetchEmail('INBOX', 'email-id-123');
```

WireParser (Server/SMTP Side)

The `WireParser` is used by your SMTP service to handle incoming wire messages:

```
import { WireParser, createMessageId, createTimestamp } from '@push.rocks/smartmail';

const parser = new WireParser({
  // Handle email send requests
  async onMailSend(email, options) {
    const mimeFormat = await email.toMimeFormat();
    await transporter.sendMail(mimeFormat);

    return {
      type: 'mail.send.response',
      messageId: createMessageId(),
      timestamp: createTimestamp(),
      success: true,
      deliveryId: generateDeliveryId()
    };
  },

  // Handle settings updates
  async onSettingsUpdate(settings) {
```

```
  if (settings.smtp) {
    configureSmtpTransport(settings.smtp);
  }
  // Handle custom settings
  if (settings.customSetting) {
    handleCustomSetting(settings.customSetting);
  }

  return {
    type: 'settings.update.response',
    messageId: createMessageId(),
    timestamp: createTimestamp(),
    success: true
  };
},

// Handle mailbox list requests
async onMailboxList(mailbox, options) {
  const emails = await getEmailsFromMailbox(mailbox, options);
  return {
    type: 'mailbox.list.response',
    messageId: createMessageId(),
    timestamp: createTimestamp(),
    mailbox,
    emails: emails.map(e => e.toObject()),
    total: emails.length
  };
},

// Handle email fetch requests
async onMailFetch(mailbox, emailId) {
  const email = await getEmailById(mailbox, emailId);
  return {
    type: 'mail.fetch.response',
    messageId: createMessageId(),
    timestamp: createTimestamp(),
    email: email ? email.toObject() : null
  };
},
```

```
  // Handle status check requests
  async onMailStatus(deliveryId) {
    const status = await getDeliveryStatus(deliveryId);
    return {
      type: 'mail.status.response',
      messageId: createMessageId(),
      timestamp: createTimestamp(),
      deliveryId,
      status: status.state
    };
  }
});

// Express route handler example
app.post('/api/wire', async (req, res) => {
  const responseJson = await parser.parseAndHandle(JSON.stringify(req.body));
  res.json(JSON.parse(responseJson));
});
```

Wire Protocol Message Types

```
// All available message type interfaces
import type {
  IWireMessage,
  IMailSendRequest,
  IMailSendResponse,
  IMailboxListRequest,
  IMailboxListResponse,
  IMailFetchRequest,
  IMailFetchResponse,
  IMailStatusRequest,
  IMailStatusResponse,
  ISettingsUpdateRequest,
  ISettingsUpdateResponse,
  IWireSettings,
  ISmtpSettings,
  TWireMessage  // Union type for type discrimination
```

```
} from '@push.rocks/smartmail';

// Helper functions
import { createMessageId, createTimestamp } from '@push.rocks/smartmail';
```

API Reference

EmailAddressValidator

Constructor Options

Option	Type	Default	Description
`skipOnlineDomainFetch`	boolean	`false`	Skip fetching domain list from online source
`cacheDnsResults`	boolean	`true`	Cache DNS lookup results
`cacheExpiryMs`	number	`3600000`	Cache expiry in milliseconds (1 hour)

Methods

Method	Returns	Description
`validate(email)`	`Promise<IEmailValidationResult>`	Full validation with format, MX, and domain checks
`isValidEmailFormat(email)`	`boolean`	Check if email format is valid
`isValidLocalPart(localPart)`	`boolean`	Validate local part (before @)
`isValidDomainPart(domainPart)`	`boolean`	Validate domain part (after @)
`checkMxRecords(domain)`	`Promise<any>`	Check MX records for a domain
`getMxRecords(domain)`	`Promise<string[]>`	Get MX record hostnames for a domain
`isDisposableEmail(email)`	`Promise<boolean>`	Check if email is from a disposable domain
`isRoleAccount(email)`	`boolean`	Check if email is a role account

Smartmail

Constructor Options

Option	Type	Default	Description
`from`	string	*required*	Email address of sender
`to`	string[]	`[]`	Array of primary recipient email addresses
`cc`	string[]	`[]`	Array of CC recipient email addresses
`bcc`	string[]	`[]`	Array of BCC recipient email addresses
`subject`	string	*required*	Email subject line (supports templates)
`body`	string	*required*	Plain text email body (supports templates)
`htmlBody`	string	`undefined`	HTML version of email body (supports templates)
`replyTo`	string	`undefined`	Reply-to email address
`headers`	Record<string, string>	`{}`	Custom email headers
`priority`	`'high' | 'normal' | 'low'`	`'normal'`	Email priority level
`validateEmails`	boolean	`false`	Validate all emails before MIME conversion
`creationObjectRef`	T	`undefined`	Reference data of any type (generic)

Methods (All chainable methods return `this`)

Method	Returns	Description
`addRecipient(email, type?)`	`this`	Add a single recipient (to/cc/bcc)
`addRecipients(emails, type?)`	`this`	Add multiple recipients
`setReplyTo(email)`	`this`	Set reply-to address
`setPriority(priority)`	`this`	Set email priority
`addHeader(name, value)`	`this`	Add custom header
`addAttachment(smartfile)`	`this`	Add a file attachment
`applyVariables(variables)`	`this`	Apply template variables in place
`getSubject(data?)`	`string`	Get processed subject with template variables
`getBody(data?)`	`string`	Get processed plain text body
`getHtmlBody(data?)`	`string | null`	Get processed HTML body

Method	Returns	Description
`getCreationObject()`	`T`	Get the stored reference data
`validateAllEmails()`	`Promise<Record<string, boolean>>`	Validate all email addresses
`toMimeFormat(data?)`	`Promise<object>`	Convert to MIME format object
`toObject()`	`ISmartmailJson<T>`	Serialize to JSON object
`toJson()`	`string`	Serialize to JSON string
`sendTo(target)`	`Promise<IMailSendResponse>`	Send to a WireTarget

Static Methods

Method	Returns	Description
`Smartmail.fromObject(obj)`	`Smartmail<T>`	Create from JSON object
`Smartmail.fromJson(json)`	`Smartmail<T>`	Create from JSON string

WireTarget

Constructor Options

Option	Type	Description
`endpoint`	string	URL of the SMTP service endpoint
`authToken`	string	Optional authentication token

Methods

Method	Returns	Description
`sendEmail(email)`	`Promise<IMailSendResponse>`	Send an email through this target
`updateSettings(settings)`	`Promise<ISettingsUpdateResponse>`	Update settings on the target
`listMailbox(mailbox, options?)`	`Promise<IMailboxListResponse>`	List emails in a mailbox
`fetchEmail(mailbox, emailId)`	`Promise<Smartmail | null>`	Fetch a specific email
`getStatus(deliveryId)`	`Promise<IMailStatusResponse>`	Check delivery status

WireParser

Constructor

```
new WireParser(handlers: IWireHandlers)
```

Handler Interface

```
interface IWireHandlers {
  onMailSend?: (email: Smartmail, options?) => Promise<IMailSendResponse>;
  onMailboxList?: (mailbox: string, options?) => Promise<IMailboxListResponse>;
  onMailFetch?: (mailbox: string, emailId: string) => Promise<IMailFetchResponse>;
  onMailStatus?: (deliveryId: string) => Promise<IMailStatusResponse>;
  onSettingsUpdate?: (settings: IWireSettings) => Promise<ISettingsUpdateResponse>;
}
```

Methods

Method	Returns	Description
`parse(json)`	`TWireMessage`	Parse a wire message from JSON string
`handle(message)`	`Promise<IWireMessage>`	Handle a wire message and return response
`parseAndHandle(json)`	`Promise<string>`	Parse and handle in one step

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:11:28 UTC by foss.global Team
Updated 2026-03-28 12:18:18 UTC by foss.global Team