Custom Receiver Strategy

Creating a Custom Receiver Strategy for Derafu Mail

Anonymous · 09/09/2026

Creating a Custom Receiver Strategy for Derafu Mail

This guide explains how to create custom receiver strategies for Derafu Mail. Receiver strategies allow you to implement different methods of retrieving emails beyond the default IMAP implementation.

Understanding Receiver Strategies

A receiver strategy in Derafu Mail is responsible for connecting to mail sources and retrieving messages. The library comes with an IMAP implementation, but you might want to add support for:

- API-based email services (Gmail API, Microsoft Graph, etc.).
- Custom email storage systems.
- Database-stored emails.
- Webhook receivers for incoming emails.
- Testing/mock implementations.

Option 1: Extending AbstractMailboxStrategy

The simplest approach is to extend the `AbstractMailboxStrategy` class, which provides reusable functionality for email retrieval scenarios that use a mailbox-like interface.

When to Use This Approach

- When your email source follows a mailbox paradigm.
- When you can use the PHP-IMAP library or similar interfaces.
- When you need to reuse common mailbox operations.
- When your strategy follows a similar workflow to the IMAP strategy.

Implementation Steps

1. Create a new class that extends `AbstractMailboxStrategy`:

```
<?php

declare(strict_types=1);

namespace YourNamespace\Strategy;

use Derafu\Backbone\Attribute\Strategy;
use Derafu\Config\Contract\OptionsInterface;
use
Derafu\Mail\Component\Exchange\Worker\Receiver\Strategy\Abstract\AbstractMailboxStrate
gy;
use
Derafu\Mail\Component\Exchange\Worker\Receiver\Strategy\Contract\ReceiverStrategyInter
face;
use Derafu\Mail\Model\Mailbox;
use Derafu\Mail\Model\Contract\MailboxInterface;

#[Strategy(name: 'gmail-api', worker: 'receiver', component: 'exchange', package:
'mail')]
class GmailApiStrategy extends AbstractMailboxStrategy implements
ReceiverStrategyInterface
{
    /**
     * Schema of the options.
     *
     * @var array<string,array>
     */
    protected array $optionsSchema = [
        'strategy' => [
            'types' => 'string',
            'default' => 'gmail-api',
        ],
        'transport' => [
            'types' => 'array',
            'schema' => [
                'client_id' => [
                    'types' => 'string',
                    'required' => true,
                ],
                'client_secret' => [
                    'types' => 'string',
                    'required' => true,
                ],
                'refresh_token' => [
                    'types' => 'string',
                    'required' => true,
                ],
                'user_email' => [
                    'types' => 'string',
                    'required' => true,
```

```
                ],
                'label' => [
                    'types' => 'string',
                    'default' => 'INBOX',
                ],
                'dsn' => [
                    'types' => 'string',
                ],
                'endpoint' => [
                    'types' => 'string',
                ],
                'search' => [
                    'types' => 'array',
                    'schema' => [
                        'query' => [
                            'types' => 'string',
                            'default' => 'is:unread',
                        ],
                        'markAsSeen' => [
                            'types' => 'bool',
                            'default' => false,
                        ],
                        'attachmentFilters' => [
                            'types' => 'array',
                            'default' => [],
                        ],
                    ],
                ],
            ],
        ],
    ];

    /**
     * {@inheritDoc}
     */
    protected function createMailbox(OptionsInterface $options): MailboxInterface
    {
        // Instead of using the standard Mailbox, create a specialized Gmail API
mailbox.
        // This could be a custom class that implements MailboxInterface.

        $transportOptions = $options->get('transport');

        // This would be a custom implementation for Gmail API.
        return new GmailApiMailbox(
            $transportOptions['client_id'],
            $transportOptions['client_secret'],
            $transportOptions['refresh_token'],
            $transportOptions['user_email'],
            $transportOptions['label'] ?? 'INBOX'
        );
```

```
    }

    /**
     * {@inheritDoc}
     */
    protected function resolveDsn(OptionsInterface $options): string
    {
        $transportOptions = $options->get('transport');

        if (!empty($transportOptions['dsn'])) {
            return $transportOptions['dsn'];
        }

        // Construct a representative DSN for Gmail API.
        $dsn = sprintf(
            'gmail-api://%s',
            $transportOptions['user_email']
        );

        $options->set('transport.dsn', $dsn);

        return $dsn;
    }

    /**
     * {@inheritDoc}
     */
    protected function resolveEndpoint(OptionsInterface $options): string
    {
        $transportOptions = $options->get('transport');

        if (!empty($transportOptions['endpoint'])) {
            return $transportOptions['endpoint'];
        }

        $endpoint = sprintf(
            'https://gmail.googleapis.com/gmail/v1/users/%s',
            $transportOptions['user_email']
        );

        $options->set('transport.endpoint', $endpoint);

        return $endpoint;
    }
}

/**
 * Custom Mailbox implementation for Gmail API.
 * This class would need to implement all methods from MailboxInterface
 * but would use the Gmail API instead of IMAP.
 */
```

```
class GmailApiMailbox implements MailboxInterface
{
    // Implement all required methods from MailboxInterface.
    // This would use Google API Client or similar to fetch emails.
}
```

2. Define your options schema to specify the required configuration.

3. Override the `createMailbox()` method to return your custom mailbox implementation.

4. Implement the `resolveDsn()` and `resolveEndpoint()` methods.

5. Create a custom mailbox class that implements `MailboxInterface` if needed.

Key Benefits

- Leverages existing workflow and error handling.
- Preserves compatibility with the rest of the library.
- Reuses attachment filtering and other common functionality.
- Maintains consistent behavior across strategies.

Option 2: Implementing ReceiverStrategyInterface

For more specialized use cases, you can implement the `ReceiverStrategyInterface` directly.

When to Use This Approach

- When your receiving mechanism is fundamentally different from a mailbox model.
- When you need complete control over the receiving process.
- When you're creating a strategy for webhooks or other non-polling mechanisms.
- For custom integrations with proprietary systems.

Implementation Steps

1. Create a new class that implements `ReceiverStrategyInterface`:

```
<?php

declare(strict_types=1);

namespace YourNamespace\Strategy;

use Derafu\Backbone\Abstract\AbstractStrategy;
```

```
use Derafu\Backbone\Attribute\Strategy;
use
Derafu\Mail\Component\Exchange\Worker\Receiver\Strategy\Contract\ReceiverStrategyInter
face;
use Derafu\Mail\Exception\MailException;
use Derafu\Mail\Model\Contract\EnvelopeInterface;
use Derafu\Mail\Model\Contract\MessageInterface;
use Derafu\Mail\Model\Contract\PostmanInterface;
use Derafu\Mail\Model\Envelope;
use Derafu\Mail\Model\Message;
use Symfony\Component\Mime\Address;
use Throwable;

#[Strategy(name: 'webhook-receiver', worker: 'receiver', component: 'exchange',
package: 'mail')]
class WebhookReceiverStrategy extends AbstractStrategy implements
ReceiverStrategyInterface
{
    /**
     * Schema of the options.
     *
     * @var array<string,array>
     */
    protected array $optionsSchema = [
        'strategy' => [
            'types' => 'string',
            'default' => 'webhook-receiver',
        ],
        'transport' => [
            'types' => 'array',
            'schema' => [
                'webhook_data' => [
                    'types' => 'array',
                    'required' => true,
                ],
                'secret_key' => [
                    'types' => 'string',
                    'default' => '',
                ],
                // Add any other configuration options needed.
            ],
        ],
    ];

    /**
     * {@inheritDoc}
     */
    public function receive(PostmanInterface $postman): array
    {
        $options = $this->resolveOptions($postman->getOptions());
        $transportOptions = $options->get('transport');
```

```
        $webhookData = $transportOptions['webhook_data'];
        $secretKey = $transportOptions['secret_key'] ?? '';

        try {
            // Validate webhook data if a secret is configured.
            if ($secretKey && !$this->validateWebhookSignature($webhookData,
$secretKey)) {
                throw new MailException('Invalid webhook signature');
            }

            // Process the webhook data to extract email information.
            $emails = $this->processWebhookData($webhookData);

            // Create envelopes and add them to the postman.
            foreach ($emails as $emailData) {
                $envelope = $this->createEnvelope($emailData);
                $postman->addEnvelope($envelope);
            }

            // Optionally acknowledge receipt to the webhook source.
            $this->acknowledgeReceipt($webhookData);

        } catch (Throwable $e) {
            throw new MailException(
                sprintf(
                    'An error occurred while processing webhook data: %s',
                    $e->getMessage()
                ),
                0,
                $e
            );
        }

        return $postman->getEnvelopes();
    }

    /**
     * Validates the webhook signature to ensure authenticity.
     *
     * @param array $webhookData
     * @param string $secretKey
     * @return bool
     */
    private function validateWebhookSignature(array $webhookData, string $secretKey):
bool
    {
        // Implement signature validation logic.
        // The exact implementation depends on how your webhook source signs requests.
        return true; // Placeholder.
    }
```

```
    /**
     * Processes the webhook data to extract email information.
     *
     * @param array $webhookData
     * @return array
     */
    private function processWebhookData(array $webhookData): array
    {
        // Convert webhook data to a standardized email format.
        // This will depend entirely on the webhook format you're receiving.

        // Placeholder implementation - extract email data from webhook.
        $emails = [];

        if (isset($webhookData['emails']) && is_array($webhookData['emails'])) {
            foreach ($webhookData['emails'] as $email) {
                $emails[] = [
                    'from' => $email['sender'] ?? '',
                    'from_name' => $email['sender_name'] ?? '',
                    'to' => $email['recipient'] ?? '',
                    'to_name' => $email['recipient_name'] ?? '',
                    'subject' => $email['subject'] ?? '',
                    'text_body' => $email['plain_text'] ?? '',
                    'html_body' => $email['html'] ?? '',
                    'attachments' => $email['attachments'] ?? [],
                ];
            }
        }

        return $emails;
    }

    /**
     * Creates an envelope from email data.
     *
     * @param array $emailData
     * @return EnvelopeInterface
     */
    private function createEnvelope(array $emailData): EnvelopeInterface
    {
        // Create a sender address.
        $sender = new Address(
            $emailData['from'],
            $emailData['from_name'] ?? ''
        );

        // Create recipient addresses.
        $recipients = [
            new Address(
                $emailData['to'],
                $emailData['to_name'] ?? ''
```

```
            )
        ];

        // Create the envelope.
        $envelope = new Envelope($sender, $recipients);

        // Create and add the message.
        $message = $this->createMessage($emailData);
        $envelope->addMessage($message);

        return $envelope;
    }

    /**
     * Creates a message from email data.
     *
     * @param array $emailData
     * @return MessageInterface
     */
    private function createMessage(array $emailData): MessageInterface
    {
        // Create the message.
        $message = new Message();

        // Set basic properties
        $message->subject($emailData['subject'] ?? '');

        if (!empty($emailData['text_body'])) {
            $message->text($emailData['text_body']);
        }

        if (!empty($emailData['html_body'])) {
            $message->html($emailData['html_body']);
        }

        $message->from(new Address(
            $emailData['from'],
            $emailData['from_name'] ?? ''
        ));

        $message->to(new Address(
            $emailData['to'],
            $emailData['to_name'] ?? ''
        ));

        // Process attachments if any.
        if (!empty($emailData['attachments']) && is_array($emailData['attachments']))
{
            foreach ($emailData['attachments'] as $attachment) {
                if (isset($attachment['content'], $attachment['name'],
$attachment['type'])) {
```

```
                    $content = base64_decode($attachment['content']);
                    $message->attach(
                        $content,
                        $attachment['name'],
                        $attachment['type']
                    );
                }
            }
        }

        return $message;
    }

    /**
     * Acknowledges receipt to the webhook source if needed.
     *
     * @param array $webhookData
     * @return void
     */
    private function acknowledgeReceipt(array $webhookData): void
    {
        // Some webhook providers require an acknowledgement.
        // Implement if needed for your specific case.
    }
}
```

2. Define your options schema to specify the required configuration.

3. Implement the `receive()` method to handle the entire receiving process.

4. Add helper methods for your specific implementation needs.

Key Considerations

When implementing from scratch:

- **Error Handling**: Properly catch and handle all exceptions.
- **Data Transformation**: Carefully map external data to Derafu Mail models.
- **Security**: Validate webhook signatures or implement other security measures.
- **Consistency**: Ensure your implementation behaves consistently with other strategies.
- **Testing**: Create comprehensive tests for your implementation.

Usage

Once you've created your custom strategy, you can use it by specifying its name in the Postman configuration:

```
$postman = new Postman([
    'strategy' => 'gmail-api', // Or 'webhook-receiver'
    'transport' => [
        // Strategy-specific configuration options.
        'client_id' => 'your-client-id',
        'client_secret' => 'your-client-secret',
        'refresh_token' => 'your-refresh-token',
        'user_email' => 'user@example.com',
        // Other options...
    ],
]);

$receiverWorker = $exchangeComponent->getReceiverWorker();
$envelopes = $receiverWorker->receive($postman);
```

Best Practices

1. **Error Handling**: Always catch and handle exceptions appropriately.
2. **Logging**: Add detailed logging to help troubleshoot issues.
3. **Configuration Validation**: Use the options schema to validate configuration.
4. **Rate Limiting**: Consider rate limits for API-based strategies.
5. **Pagination**: Implement proper pagination for retrieving large volumes of emails.
6. **Performance**: Be mindful of memory usage when dealing with attachments.
7. **Documentation**: Document your strategy's specific requirements and limitations.

Conclusion

Creating custom receiver strategies allows you to extend Derafu Mail to work with any email source or system. Whether you extend the abstract class or implement the interface directly depends on your specific needs and how much you want to leverage the existing infrastructure.

For sources that follow a mailbox-like model, extending `AbstractMailboxStrategy` is recommended. For completely different mechanisms like webhooks, implementing `ReceiverStrategyInterface` directly gives you maximum flexibility.

Last updated on 09/09/2026