

Attachments, Shares & Events

- [Attachments & Drive Shares](#)
- [Real-time Events & User Account](#)

Attachments & Drive Shares

Attachments

Attachment upload and download via `/mail/v4/attachments`.

Upload

```
attachment, err := c.UploadAttachment(ctx, addrKR, proton.CreateAttachmentReq{
    MessageID: "message-id",
    Filename:  "document.pdf",
    MIMETYPE: "application/pdf",
    Disposition: proton.AttachmentDisposition,
    Body:      fileBytes,
})
```

Upload is **encrypted with the address keyring** and **signed (detached)** before upload. The server stores the encrypted `KeyPacket`, `DataPacket`, and `Signature` separately.

Download

```
// Download as bytes
data, err := c.GetAttachment(ctx, attachmentID)

// Stream into io.ReaderFrom
err := c.GetAttachmentInto(ctx, attachmentID, reader)
```

Key Types

Type	Description
<code>Attachment</code>	ID, name, size, MIME type, disposition (inline/attachment), headers, key packets, signature

Type	Description
CreateAttachmentReq	MessageID, filename, MIME type, disposition, content ID, body bytes
Disposition	InlineDisposition, AttachmentDisposition

Drive Shares

Proton Drive share management via `/drive/shares`.

Operations

```
// List shares
shares, err := c.ListShares(ctx, all)

// Get a single share with full key material
share, err := c.GetShare(ctx, shareID)
```

Key Derivation

Shares use a two-step PGP decryption + passphrase unlock process:

```
// Decrypt passphrase with address keyring, verify signature, unlock share key
keyRing, err := share.GetKeyRing(addrKR)
// Returns *crypto.KeyRing for Drive access
```

Key Types

Type	Description
ShareMetadata	ShareID, linkID, volumeID, type, state, creation/modify times, creator email
Share	Extends metadata with addressID, addressKeyID, encrypted key, passphrase, passphrase signature
ShareType	Main (1), Standard (2), Device (3)
ShareState	Active (1), Deleted (2)

Real-time Events & User Account

Real-time Events

Event polling and streaming for real-time notifications via `/core/v4/events`.

Event Streaming

```
// Get the latest event ID
fromEventID, err := c.GetLatestEventID(context.Background())

// Create a new event streamer
for event := range c.NewEventStream(ctx, 20*time.Second, 20*time.Second, fromEventID) {
    fmt.Println(event.EventID)
    // Process event.User, event.Messages, event.Labels, etc.
}
```

The event stream uses a custom `NewTicker` with random jitter to avoid thundering herd.

Event Polling

```
// Fetch events starting after eventID
events, more, err := c.GetEvent(ctx, eventID)
// Returns up to 50 events per call with more indicating continuation
```

Event Types

The `Event` struct contains typed sub-events:

```

event.User           // User changes
event.UserSettings  // User settings changes
event.MailSettings  // Mail settings changes
event.Messages       // Message changes ([]MessageEvent)
event.Labels        // Label changes ([]LabelEvent)
event.Addresses     // Address changes ([]AddressEvent)
event.Notifications // Notifications ([]NotificationEvent)
event.UsedSpace     // Storage usage changes

```

Key Types

Type	Description
<code>Event</code>	Contains eventID, refresh flags, and typed sub-events
<code>RefreshFlag</code>	RefreshMail (1), RefreshAll (255)
<code>EventAction</code>	Delete, Create, Update, UpdateFlags
<code>MessageEvent</code>	EventItem (ID + Action) with full Message entity
<code>LabelEvent</code>	EventItem (ID + Action) with full Label entity
<code>AddressEvent</code>	EventItem (ID + Action) with full Address entity

User Account

User account operations via `/core/v4/users`.

Operations

```

// Fetch current user info
user, err := c.GetUser(ctx)

// With human verification token
user, err := c.GetUserWithHV(ctx, hvToken)

// Delete account (requires SRP proof re-authentication)
err := c.DeleteUser(ctx, password, proton.DeleteUserReq{

```

```
Reason: "no longer needed",  
Feedback: "test account",  
})
```

Key Types

Type	Description
User	ID, name, display name, email, keys, used/max/upload space limits, credit, currency
ProductUsedSpace	Space breakdown by calendar, contact, drive, mail, pass (password manager)
DeleteUserReq	Reason, feedback, email confirmation

Feature Flags

Fetch feature flag/toggle results from the Proton API:

```
features, err := c.GetFeatures(ctx, "sticky-key-uuid")  
// Returns FeatureFlagResult for a given UUID sticky key
```