Contacts, Calendar & Labels

- Contacts
- Calendar
- Labels & Addresses

Contacts

Contacts

CRUD operations for contacts and contact emails via `/contacts/v4`.

Operations

Get Contacts

```
// Single contact by ID
contact, err := c.GetContact(ctx, contactID)

// Count contacts
count, err := c.CountContacts(ctx)

// Paginated listing
contacts, err := c.GetContacts(ctx, page, pageSize)

// Auto-paginating - get all
allContacts, err := c.GetAllContacts(ctx)

// Auto-paginating with custom page size
allContacts, err := c.GetAllContactsPaged(ctx, pageSize)
```

Search by Email

```
// Count matches
count, err := c.CountContactEmails(ctx, "user@example.com")

// Paginated search
```

```
emails, err := c.GetContactEmails(ctx, "user@example.com", page, pageSize)

// Get all matching
allEmails, err := c.GetAllContactEmails(ctx, "user@example.com")
```

Create / Update / Delete

```
// Bulk create with overwrite and label options
results, err := c.CreateContacts(ctx, proton.CreateContactsReq{
    Contacts: []proton.Contact{
        {
            Cards: []proton.ContactCard{...},
            Email: "user@example.com",
        },
    },
})

// Update a contact
updated, err := c.UpdateContact(ctx, contactID, proton.UpdateContactReq{
    Cards: []proton.ContactCard{...},
})

// Bulk delete
err := c.DeleteContacts(ctx, proton.DeleteContactsReq{
    IDs: []string{id1, id2},
})
```

Per-Contact Encryption Preferences

Contact settings are stored as VCard-embedded X-PM-* fields:

```
// Get encryption settings for a contact
settings := contact.GetSettings()
// Returns: MIME type, PGP scheme (inline/MIME), sign/encrypt flags, PGP keys
```

```
// Set encryption settings
contact.SetSettings(proton.ContactSettings{
    MIMEType:       "text/plain",
    Scheme:         proton.PGPMIMEScheme,
    Sign:           true,
    Encrypt:        true,
    EncryptUntrusted: true,
    PGPKeys:        []proton.PGPKey{...},
})
```

Key Types

Type	Description
`Contact`	Composite of ContactMetadata and ContactCards
`ContactMetadata`	ID, name, UID, size, timestamps, emails, labels
`ContactCards`	VCard data for the contact
`ContactEmail`	Email entry with ID, name, email string, type tags, contactID, labels
`ContactSettings`	Per-contact encryption preferences (MIME type, scheme, sign/encrypt flags, PGP keys)
`RecipientType`	Internal vs external recipient classification

Calendar

Calendar

Calendar and calendar event retrieval via `/calendar/v1`. Includes key management for encrypted calendar data.

Operations

Calendars

```
// List all calendars
calendars, err := c.GetCalendars(ctx)

// Get a single calendar
cal, err := c.GetCalendar(ctx, calendarID)

// Get encryption keys for a calendar
keys, err := c.GetCalendarKeys(ctx, calendarID)

// Get shared calendar members
members, err := c.GetCalendarMembers(ctx, calendarID)

// Get passphrase for calendar key decryption
passphrase, err := c.GetCalendarPassphrase(ctx, calendarID)
```

Calendar Events

```
// Count events
count, err := c.CountCalendarEvents(ctx, calendarID)
```

```
// Paginated listing with optional filters
events, err := c.GetCalendarEvents(ctx, calendarID, page, pageSize, filter)

// Auto-paginating - get all events
allEvents, err := c.GetAllCalendarEvents(ctx, calendarID, filter)

// Single event by ID
event, err := c.GetCalendarEvent(ctx, calendarID, eventID)
```

Decryption

Calendar events use **two-layer encryption**:

1. Calendar keys (passphrase-locked)
2. Shared events add a second layer of sharing key packets

```
// Unlock calendar keys with passphrase
keyRing, err := keys.Unlock(passphrase)

// Decrypt individual event parts
decoded, err := eventPart.Decode(calKR, addrKR, keyPacket)
// Handles both two-layer decryption and PGP signature verification
```

Key Types

Type	Description
`Calendar`	ID, name, description, color, display flag, type (normal/subscribed), flags
`CalendarKey`	Encrypted private key with `Unlock(passphrase)` returning `*crypto.Key`
`CalendarKeys`	Slice of keys with `Unlock(passphrase)` returning `*crypto.KeyRing`
`CalendarMember`	ID, permissions, email, color, display, calendarID
`CalendarPassphrase`	Encrypted passphrase with `Decrypt(memberID, addrKR)` for decryption

Type	Description
`CalendarEvent`	UID, start/end times, timezone, full-day flag, author, attendees, encrypted parts
`CalendarEventPart`	Individual part with type (clear/encrypted/signed), data, signature, author
`EventAction`	Delete, Create, Update, UpdateFlags

PGP Signature Verification

Encrypted calendar parts can also be signed. The `Decode` method verifies signatures using the address keyring.

Labels & Addresses

Labels

Label (category/folder) management via `/core/v4/labels`.

Operations

```
// Fetch labels filtered by type(s)
labels, err := c.GetLabels(ctx, proton.LabelLabel, proton.LabelFolder)

// Find a specific label by ID
label, err := c.GetLabel(ctx, labelID, proton.LabelLabel)

// Create a label
label, err := c.CreateLabel(ctx, proton.CreateLabelReq{
    Name:   "Personal",
    Color:  "#ff0000",
    Type:   proton.LabelLabel,
    Parent: parentID,
})

// Update a label
err := c.UpdateLabel(ctx, labelID, proton.UpdateLabelReq{
    Name: "Updated Name",
    Color: "#00ff00",
})

// Delete a label
err := c.DeleteLabel(ctx, labelID)
```

System Label Constants

Constant	ID	Description
`InboxLabel`	"0"	Inbox
`AllSentLabel`	"2"	All Sent
`TrashLabel`	"3"	Trash
`SpamLabel`	"4"	Spam
`AllMailLabel`	"5"	All Mail
`StarredLabel`	"10"	Starred

Email Addresses

Email address management via `/core/v4/addresses`.

Operations

```
// List addresses (sorted by order)
addresses, err := c.GetAddresses(ctx)

// Get a single address
addr, err := c.GetAddress(ctx, addressID)

// Reorder addresses
err := c.OrderAddresses(ctx, proton.OrderAddressesReq{...})

// Enable/disable address
err := c.EnableAddress(ctx, addressID)
err := c.DisableAddress(ctx, addressID)

// Delete address
err := c.DeleteAddress(ctx, addressID)
```

Address Types

Type	Description
`Original`	Original Proton address

Type	Description
`Alias`	Proton alias
`Custom`	Custom domain address
`Premium`	Premium address
`External`	Bring Your Own Email (BYOE)

BYOE Detection

```
if addr.IsBYOEAddress() {
    // External type with sending enabled
}
```

Key Types

Type	Description
`Label`	ID, parentID, name, path ([]string), color, type
`LabelType`	Label, ContactGroup, Folder, System
`Address`	ID, email, send/receive flags, status, type, order, display name, keys
`AddressStatus`	Disabled, enabled, deleting
`AddressType`	Original, alias, custom, premium, external

Path Handling

The API sends label path as a string (e.g., `"Inbox/SubFolder"`), which is split into a `[]string` slice for Go use, with custom `MarshalJSON`/`UnmarshalJSON`.