24.5.3 POP3-Parser

Für den POP3-Client im Kapitel 24.5.4 wird die Komponente *gb.net.pop3client* von Sebastian Kulesz eingesetzt. Der Vorteil der Komponente liegt m.E. darin, dass alle POP3-Kommandos nach RFC 1939 in Methoden gekapselt sind. Da die Klasse Pop3Client unter anderem die Methode Exec(...) mitbringt, können Sie alle POP3-Kommandos auch direkt ausführen. Als Erweiterung der Komponente wurde durch den Buch-Autor Hans Lehmann das Authentifikationsverfahren APOP implementiert. Das Verfahren stellt sicher, dass das POP3-Passwort verschlüsselt vom POP3-Client zum POP3-Server übertragen wird. Eine Beschreibung des Authentifikationsverfahren APOP können Sie im → Kapitel 24.5.3 nachlesen. Die Implementation erfolgte so, dass *automatisch* auf das Verfahren APOP gewechselt wird, wenn der POP3-Client erkennt, dass der POP3-Server auch APOP als Authentifikationsverfahren anbietet. Schalten Sie in der Erprobung des POP3-Clients die Eigenschaft 'Debug' auf den Wert *True*, so können Sie die vollständige Kommunikation zwischen POP3-Client und POP3-Server verfolgen:

```
gb.net.pop3: Connecting to mx.freenet.de
gb.net.pop3: Connecting to Port 995
gb.net.pop3: Encryption: Net.SSL
gb.net.pop3: Change to APOP
gb.net.pop3: Authenticating...
gb.net.pop3: Sending: APOP wer@ist.da 1eeddd29088529659e3bc76e84c4d44c
gb.net.pop3: +OK 2 messages (66766 octets).
gb.net.pop3: APOP Authentification OK
gb.net.pop3: Refreshing inbox cache
gb.net.pop3: Sending: STAT
gb.net.pop3: +OK 2 66766
gb.net.pop3: Sending: UIDL
gb.net.pop3: +OK...
gb.net.pop3: Sending: UIDL
gb.net.pop3: +OK...
gb.net.pop3: Creating new message instance for 0
gb.net.pop3: Sending: RETR 1
gb.net.pop3: +OK 1879 octets...
gb.net.pop3: Creating new message instance for 1
gb.net.pop3: Sending: RETR 2
gb.net.pop3: +OK 64887 octets...
gb.net.pop3: Sending: QUIT
gb.net.pop3: +OK
gb.net.pop3: Disconnecting...
```

Der POP3-Client nutzt für die Einstellungen des erzeugten POP3-Client-Objekts die Festlegungen, die über den Manager für EMail-Konten in einer Konfigurationsdatei hinterlegt worden sind → Kapitel 24.3.4 'Manager für EMail-Konten'. Besteht beim ersten Programm-Start noch keine Konfigurationsdatei, dann startet der Manager automatisch. Diese sechs Einstellungen sind für POP3 mit geeigneten Werten zu belegen:

- POP3-Server
- POP3-Port (110, 995)
- Verbindungssicherheit
  POP3Encryption: Net.SSL, Net.TLS oder Net.NONE
- Authentifikationsverfahren
  POP3Authentication: Basic (Passwort normal) oder APOP (Passwort verschlüsselt)
- POP3-User
- POP3-Passwort

und werden in der folgenden Prozedur benötigt:

```
Public Sub GeneratePOP3Client(_Debug As Boolean)

  If hPOP3Client <> Null Then
     If hPOP3Client.Status = Net.Connected Then
        If hPOP3Client.Close() = False Then
           hPOP3Client = Null
        Endif
     Else
        hPOP3Client = Null
     Endif
  Endif

  hPOP3Client = New Pop3Client
  hPOP3Client.Host = acSettings[sCurAccount & "/POP3Server"]
  hPOP3Client.Port = CInt(acSettings[sCurAccount & "/POP3Port"])
```

```
  If Upper(acSettings[sCurAccount & "/POP3Encryption"]) = "NET.SSL" Then hPOP3Client.Encrypt = 1
  If Upper(acSettings[sCurAccount & "/POP3Encryption"]) = "NET.NONE" Then hPOP3Client.Encrypt = 0

  If acSettings[sCurAccount &/ "POP3Authentication"] = ("Password normal") Then \
                            hPOP3Client.Authent = Net.AuthBasic ' 0
  If acSettings[sCurAccount &/ "POP3Authentication"] = ("Password encrypted (APOP)") Then \
                            hPOP3Client.Authent = Net.AuthAPOP ' 3

  hPOP3Client.User = acSettings[sCurAccount & "/POP3UserName"]
  hPOP3Client.Password = M3.E4(acSettings[sCurAccount & "/POP3UserPassword"]) ' Decrypted password

  hPOP3Client.Debug = _Debug

End ' GeneratePOP3Client(...)

Public Function ConnectToPOP3Server() As String

  Try hPOP3Client.Open()
  If Not Error Then Return "NoError"

End ' ConnectToServer()
```

Wenn eine Verbindung vom POP3-Client zum POP3-Server besteht, dann können Sie sich informieren:

- ob EMails im Postfach in Ihrem Konto liegen und wie viel es insgesamt sind. Entweder lesen Sie die Eigenschaft *Pop3Client.Count* aus oder lassen sich über die Methode Stat mit *Stat[0]* die Anzahl der EMails und mit *Stat[1]* die Gesamtgröße aller EMails im Postfach ausgeben,
- über die Größe jeder einzelnen EMail in einer Liste (Methode List()),
- über die Nummer und die eindeutige ID jeder EMail in einer Liste (Methode ListUniqueID()).

Außerdem können Sie einzelne EMails im Postfach auf dem POP3-Server löschen, indem Sie über die Nummer der EMail die EMail als zu löschende EMail markieren. Sie können diese Markierung durch die Methode Reset() für alle betroffenen EMails zurücknehmen. Die markierten EMails werden erst dann tatsächlich gelöscht, wenn Sie die TCP-IP-Verbindung beenden.

Mit diesen Informationen sind Sie in der Lage, sich entweder gezielt eine bestimmte EMail aus der Postbox auf dem POP3-Server auf den heimischen PC in eine lokale Postbox zu einem bestimmten EMail-Konto zu laden oder das für alle EMails zu tun. Der POP3-Client schiebt stets den Download für alle EMails im Postfach an – aber nur dann, wenn eine EMail noch nicht im lokalen Postfach liegt. In den Kontoeinstellungen können Sie auch vorgeben, dass alle EMails nach dem Download im Postfach auf dem POP3-Server gelöscht werden sollen.

Im folgenden Absatz wird beschrieben, wie der Download aller im ausgewählten Postfach auf dem POP3-Server vorhandenen EMails implementiert wurde. Die Speicherung einer EMail im lokalen Postfach zu einem bestimmten EMail-Konto erfolgt durch die Speicherung des EMail-Quelltextes in einer Datei. Der Dateiname für den Quelltext wird aus der eindeutigen ID jeder EMail erzeugt.

Es wird der vollständige Quelltext für die Prozedur angegeben, der anschließend kommentiert wird.

Quelltext für Download und Speicherung von EMails:

```
[1]  Public Sub RetrieveEMails()
[2]
[3]    Dim i, iCount, iIndex As Integer
[4]    Dim sFileName, sBaseName, sMessage, sMailBoxDir, sMimeMessageText, sSavePath, sUniqueIDText, sHash
       As String
[5]    Dim dMMDate As Date
[6]    Dim iUID As Integer
[7]    Dim aValue, aElement As New Variant[]
[8]    Dim aDownloadList As New Variant[][]
[9]    Dim aBaseNameList As New String[]
[10]   Dim hMimeMessageHeaders As MimeMessage
[11]
[12]   If Not MMMP.SystemOnline() Then Return
[13]
[14]   sMailBoxDir = sBaseDir &/ sCurAccount
[15]
[16] '-------------------------------------------------------------------------
[17]   GeneratePOP3Client(True) ' True for debugging for tests
[18] '-------------------------------------------------------------------------
[19]
[20]   If ConnectToPOP3Server() = "NoError" Then
```

```
[21]      sMessage = ("Connected to ") & acSettings[sCurAccount & "/POP3Server"]
[22]      lblStatus.Text = sMessage
[23]    Else
[24]      sMessage = ("Error connecting to the POP3 server!")
[25]      lblStatus.Text = sMessage
[26]      Return
[27]    Endif
[28]    Wait 0.2
[29]
[30]    If hPOP3Client.Count > 0 Then
[31]      For Each sFileName In Dir(sMailBoxDir, "*.*", gb.File) ' Lokales Postfach
[32]        aBaseNameList.Add(Scan(sFileName, "*.*.*")[1])
[33]      Next
[34]
[35]      For i = 0 To hPOP3Client.Count - 1
[36]      ' Muster ListUniqueID: 4 000008394ef05e55
[37]        sUniqueIDText = Scan(hPOP3Client.ListUniqueID()[i], "* *")[1]
[38]        sHash = Split(Base64(Digest["MD5"].Hash(sUniqueIDText)), "=")[0]
[39]        sHash = Replace(sHash, "+", "A")
[40]        sHash = Replace(sHash, "/", "B")
[41]
[42]        sBaseName = sHash
[43]
[44]      ' Wenn eine EMail auf dem POP3-Server noch NICHT lokal gespeichert wurde, dann wird ihre
[45]      ' EMail-Nummer (UID-Zahl) auf dem Server und ihr konvertierter UID-Text als sBaseName = sHash
[46]      ' als Basis-Dateiname in die Download-Liste eingetragen.
[47]        If Not aBaseNameList.Exist(sBaseName) Then
[48]          aValue = New Variant[]
[49]          iUID = CInt(Scan(hPOP3Client.ListUniqueID()[i], "* *")[0])
[50]          aValue.Add(iUID) ' EMail-Nummer
[51]          aValue.Add(sBaseName & ".txt") ' Dateiname als konvertierter UID-Text
[52]          aDownloadList.Add(aValue)
[53]        Endif
[54]      Next
[55]    Endif
[56]
[57]    If aDownloadList.Count = 0
[58]      lblStatus.Text = ("You have no new messages!")
[59]      Wait 0.5
[60]      lblStatus.Text = ""
[61]      If Connected() Then DisconnectFromPOP3Server()
[62]      Return
[63]    Else
[64]      lblStatus.Text = Subst$(("You have &1 new &2."), aDownloadList.Count, \
[65]                              IIf(aDownloadList.Count = 1, ("message"), ("messages")))
[66]      Wait 0.3
[67]    Endif
[68]
[69]    lblStatus.Text = ""
[70]    iCount = 0
[71]    wevBody.Url = Application.Path &/ "download.html"
[72]
[73]    For Each aElement In aDownloadList
[74]      Inc iCount
[75]      lblStatus.Text = ("Download ") & Str$(iCount) & (" of ") & Str$(aDownloadList.Count)
[76]      iIndex = aElement[0]
[77]
[78]    ' ------------------------------------------------------------------------------------
[79]    ' Download einer EMail (EMail-Quelltext vom Typ MimeMessageText)
[80]      sMimeMessageText = hPOP3Client[iIndex - 1].Text
[81]      Wait
[82]    ' ------------------------------------------------------------------------------------
[83]
[84]      hMimeMessageHeaders = New MimeMessage(MMMP.GetMMHeaderText(sMimeMessageText))
[85]
[86]      dMMDate = MMMP.ReadDate(hMimeMessageHeaders.Headers["Received"])
[87]      sSavePath = sMailBoxDir &/ Format(dMMDate, "yyyy-mm-dd-hh-nn-ss") & "." & aElement[1]
[88]      File.Save(sSavePath, sMimeMessageText)
[89]      Wait
[90]    ' ------------------------------------------------------------------------------------------------
[91]    ' Löschen der EMail *auf dem Server* nach dem Download, wenn das Löschen der EMails vorgesehen ist!
[92]      If bDeleteMailAllowed = True Then hPOP3Client.Remove(iIndex - 1)
[93]
[94]      UpdateListHeaders(sBaseDir &/ sCurAccount)
[95]    ' Anzeige einer Übersicht der lokal gespeicherten EMails
[96]    ' (Anhang-Symbol(optional), Betreff, Absender, Datum und Größe)
[97]      ShowMailHeaders(aListHeaders)
[98]
[99]    Next ' EMail
[100]
[101]     If aListHeaders.Count > 0 Then
[102]       If Dir(sBaseDir &/ sCurAccount, "*.txt", gb.File).Count > 0 Then
[103]         ShowMailData() ' Anzeige der EMail im internen Browser (WebView)
[104]       Endif
```

```
[105]    Endif
[106]
[107]    lblStatus.Text = ("Download completed!")
[108]    Wait 0.3
[109]
[110]  '-----------------------------------------------
[111]    If Connected() Then DisconnectFromPOP3Server()
[112]    DestroyPOP3Client()
[113]  '-----------------------------------------------
[114]
[115]  End ' RetrieveEMails()
```

Kommentar:

- In der Zeile 12 wird zum Programm-Start geprüft, ob eine aktive Verbindung zum Internet existiert.
- Die Auswahl des aktuellen Postfaches auf dem Server wird in der Zeile 14 vorgenommen.
- Ein POP3-Client-Objekt wird in der Zeile 17 angelegt. Die Eigenschaften werden aus einer Konfigurationsdatei ausgelesen. Für Erprobungen ist es unerlässlich, das Debugging einzuschalten.
- In den Zeilen 31 bis 33 wird eine Liste (aBaseNameList) erzeugt, in die von allen Dateien im aktuellen lokalen Postfach der zweite Teil des Dateinamens eingetragen wird, die das Muster *.*.* im Dateinamen wie zum Beispiel 2016-03-07-09-17-35.XanBcUYCsp6RltyFsocqiA.txt besitzen. Die Scan()-Funktion bietet – nicht nur an dieser Stelle – eine sehr gute Funktionalität!
- In den Zeilen 37 bis 42 wird aus dem Text-Teil der Unique-ID über mehrere Konvertierungen (MD5, Base64 mit Ersetzungen) ein Hash-Wert erzeugt, der als Basis-Datei-Name dient.
- Damit eine EMail auf dem POP3-Server in das lokale Postfach gespeichert wird, muss sichergestellt werden, dass diese EMail noch nicht gespeichert wurde. Dazu wird in der Zeile 47 geprüft, ob es im lokalen Postfach noch keine Datei mit dem konvertierten UID-Text als Basis-Datei-Name gibt. In diesem Fall werden die UID-Nummer → Zeile 50 und der Basis-Datei-Name → Zeile 51 in einer weiteren Liste (aValue) gespeichert, die dann als Download-Liste genutzt und abgearbeitet wird.
- In der Zeile 80 wird der Quelltext der EMail vom Typ MimeMessageText der Variablen *sMimeMessageText* zugewiesen – das ist der eigentliche Download.
- Dieser Text wird in der Zeile 88 in einer Datei gespeichert. Der Dateiname hat folgendes Format: "Konvertiertes Datum ('Received')" PUNKT Basis-Datei-Name PUNKT "txt".
- Wenn in der Konfiguration das Löschen der EMail nach dem Download vom POP3-Server angegeben wurde, dann wird die EMail → Zeile 92 zum Löschen *markiert*.
- In den Zeilen ab 94 werden die EMail-Übersicht neu angezeigt und die erste EMail im Browser.
- Abschließend werden die bestehende Verbindung von POP3-Client und POP3-Server getrennt sowie der POP3-Client zerstört.

Nach dem erfolgreichen Download liegen die EMails im lokalen Postfach. Wenn in den folgenden Absätzen von EMails gesprochen wird, so ist stets der EMail-Quelltext vom Typ MimeMessage gemeint, für dessen Bearbeitung die Klassen der Komponente *gb.mime* eingesetzt werden → Kapitel 24.3.0 Komponente *gb.mime*.

Die für die Anzeige der EMails notwendigen Algorithmen basieren auf der Kenntnis und dem Verständnis, wie EMails intern strukturiert sind. Aus diesem Grunde wurde zuerst ein Parser entwickelt, der für einen vorgegebenen EMail-Quelltext vom Typ MimeMessage die Struktur ermittelt und sie als HTML-Datei speichert und auf den Parser im → Kapitel 24.3.2 Klasse MimePart zurückgreift. Dabei fällt auf, dass diese Struktur einen (rekursiv) verschachtelten Charakter hat, die sich so auch im Projekt-Quelltext (recursive loop) des Struktur-Parsers für EMails widerspiegelt:

```
':::::  MIMEMESSAGE-STRUKTUR GENERIEREN UND ALS HTML-DATEI SPEICHERN  ::::::::::::::::::::::::::::::::::

' Funktion:         Structure(...)
' Parameter:        MimeMessageText Typ: String
' Funktionswert:    Datei-Pfad zur Struktur-Datei Typ: String
Public Function Structure(MimeMessageText As String) As String

  Dim sHTMLData As String

  $sMM = New MimeMessage(MimeMessageText)

  sHTMLData = "<!DOCTYPE html>" & gb.NewLine
  sHTMLData &= "<html lang=\"de\">" & gb.NewLine
  sHTMLData &= " <head>" & gb.NewLine
  sHTMLData &= "  <meta charset=\"utf-8\">" & gb.NewLine
```

```
  sHTMLData &= "  <style>" & gb.NewLine
  sHTMLData &= "    body{font-family:Arial,Verdana;color:darkgreen;font-size:16px;}" & gb.NewLine
  sHTMLData &= "  </style>" & gb.NewLine
  sHTMLData &= " </head>" & gb.NewLine
  sHTMLData &= " <body>" & gb.NewLine
  sHTMLData &= Replace(GetMimeMessageStructure($sMM.Part), gb.NewLine, "<br>\n")
  sHTMLData &= " </body>" & gb.NewLine
  sHTMLData &= "</html>"

' Struktur-Datei temporär speichern
  File.Save($sBasePath &/ _STRUCTURFILENAME, sHTMLData)
  If $bDebug Then _PrintDebug(("Show EMail-Structure"))

  Return $sBasePath &/ _STRUCTURFILENAME

End ' Structure(...)

' Funktion:        GetMimeMessageStructure(...)
' Parameter:       Part Typ: MimePart
' Funktionswert:   Text der Struktur-Datei Typ: String
Private Function GetMimeMessageStructure(Part As MimePart) As String

  Dim sBodyType As String

  If $bDebug Then _PrintDebug(("Create EMail-Structure"))
  If Not Part Then Error.Raise("MimePart not set")

  sBodyType = Scan(Part.Headers["Content-Type"], "*;*")[0]

  $sStructure = ""

  $sStructureH = ("STRUCTURE MIME-MESSAGE")
  $sStructureH &= gb.NewLine
  $sStructureH &= String$(String.Len($sStructureH), "-") & gb.NewLine
  $sStructureH &= gb.NewLine
  $sStructureH &= "+ " & sBodyType & gb.NewLine

  $sStructureB = ""
  ParseStructureBody($sMM.Body)    ' Parse Structure: Body

  $sStructureA = ""
  $iAttachmentCount = 0
  $iFlag = 0
  ParseStructureAttachments(Part) ' Parse Structure: Attachments

  $sStructure = $sStructureH & $sStructureB & $sStructureA

  Return $sStructure

End ' GetMimeMessageStructure(...)


' Prozedur:       ParseStructureBody(...)
' Parameter:      Part Typ: MimePart
' Aktion:         Parsen der Struktur des MimeMessage-Bodys.
'                 Das Ergebnis wird in der Variablen $sStructureB gespeichert
Private Sub ParseStructureBody(Part As MimePart)

  Dim hChild As MimePart
  Dim sLine As String

  sLine &= String$($iLevel, "| ") & "| " & gb.NewLine
  sLine &= String$($iLevel, "| ") & "+-" & If(Part.Count, "+ ", "- ")
  sLine &= Part.ContentType & " " & IIf(Part.FileName, Part.FileName & " ", "") & Part.Count & gb.NewLine
  $sStructureB &= sLine

  Inc $iLevel
    For Each hChild In Part
        ParseStructureBody(hChild) ' Recursive loop
    Next
  Dec $iLevel

End ' ParseStructureBody(...)


' Prozedur:       ParseStructureAttachments(Part As MimePart)
' Parameter:      Part Typ: MimePart
' Aktion:         Parsen der Struktur des MM-Anhangs.
'                 Das Ergebnis wird in der (globalen) Variablen $sStructureA gespeichert
Private Sub ParseStructureAttachments(Part As MimePart)

  Dim hChild As MimePart
  Dim sLine As String
```

```
  If Part.Disposition = "attachment" And Str(Part.Count) = 0 Then
     If $iFlag = 0 Then
        sLine &= "|" & gb.NewLine
        Inc $iFlag
     Endif
     sLine &= ("+ Attachment ") & Str($iAttachmentCount + 1) & ": " & " " & Part.ContentType
     sLine &= " " & Part.FileName & gb.NewLine
     Inc $iAttachmentCount
     $sStructureA &= sLine
  Endif

  Inc $iLevel
    For Each hChild In Part
        ParseStructureAttachments(hChild) ' Recursive loop
    Next
  Dec $iLevel

End ' ParseStructureAttachments(..)

':::::  ENDE MIMEMESSAGE-STRUKTUR GENERIEREN UND ALS HTML-DATEI  SPEICHERN ::::::::::::::::::::::::::::::
```

Als Ergebnis erhalten Sie für den als Argument übergebenen EMail-Quelltext vom Parser zum Beispiel folgende Anzeige der Struktur einer EMail im Browser:

```
STRUKTUR MIME-MESSAGE
---------------------

+ multipart/mixed
|
+-+ multipart/alternative 2
| |
| +-- text/plain 0
| |
| +-+ multipart/related 3
| | |
| | +-- text/html 0
| | |
| | +-- image/png bild1.png 0
| | |
| | +-- image/png bild2.png 0
|
+ Anhang 1: text/plain body.txt
+ Anhang 2: image/png chart.png
```

Interpretation:

- Die EMail besteht aus mehreren, unterschiedlichen Teilen (multipart/mixed) – dem Nachrichten-Teil (body) und einem Anhang-Teil (attachment). Die Nachricht und der Anhang haben unterschiedliche Formate.
- Der Nachrichten-Teil besteht aus zwei Teilen (multipart/alternative), die jeweils den gleichen Nachrichten-Text in reinem Text (Format text/plain) und alternativ als HTML-Quelltext (Format text/html) enthalten.
- Der HTML-Quelltext besteht aus drei Teilen (multipart/related), denn im HTML-Teil sind neben dem Text-Teil noch zwei Bilder enthalten, die in zwei Bild-Dateien gespeichert sind.
- Der Anhang besteht aus 2 unterschiedlichen Teilen – aus einer Text-Datei (text/plain) und einer png-Bild-Datei.

Der nächste Schritt besteht darin, mit Hilfe der Methoden der Klasse 'MimeMessageParser' die einzelnen Teile einer EMail für die Anzeige entsprechend ihrer rekursiven Struktur zu isolieren, zu dekodieren und in geeigneter Weise *temporär* zu speichern. Das Isolieren der Teile erfolgt getrennt nach Body und Attachment. Nutzt man die Klassen der Komponente gb.mime, dann braucht man sich zum Beispiel um die Dekodierung eines Bildes, das ja im EMail-Quelltext base64-kodiert vorliegt, nicht weiter zu kümmern. Die Dekodierung übernimmt die Data-Eigenschaft für jedes Teil *automatisch*, so dass Sie sich nur der Speicherung der dekodierten Teile zuwenden müssen. Die Text-Teile einer Nachricht werden in einer HTML-Datei abgespeichert. Multimediale Objekte wie zum Beispiel Bilder oder Videos werden in einzelnen Dateien abgespeichert und der Datei-Pfad wird jeweils als Link in die HTML-Datei eingefügt. Der Pfad zur HTML-Datei wird abschließend in einer Variable gespeichert und der URL-Eigenschaft einer WebView als Wert übergeben. Die Anhänge (optional) werden dekodiert unter ihrem originalen Datei-Namen temporär in einem speziellen Verzeichnis gespeichert. Für jeden Anhang wird ein Button generiert, dessen Tag-Eigenschaft den Pfad zum Anhang enthält. Sie können sich den Inhalt eines Anhangs (bei geeignetem Typ) ansehen oder den Anhang im Dialog speichern.

So präsentiert sich die Anzeige einer EMail (Format text/plain) mit 2 Anhängen im Browser, nachdem der EMail-Quelltext geparst wurde:

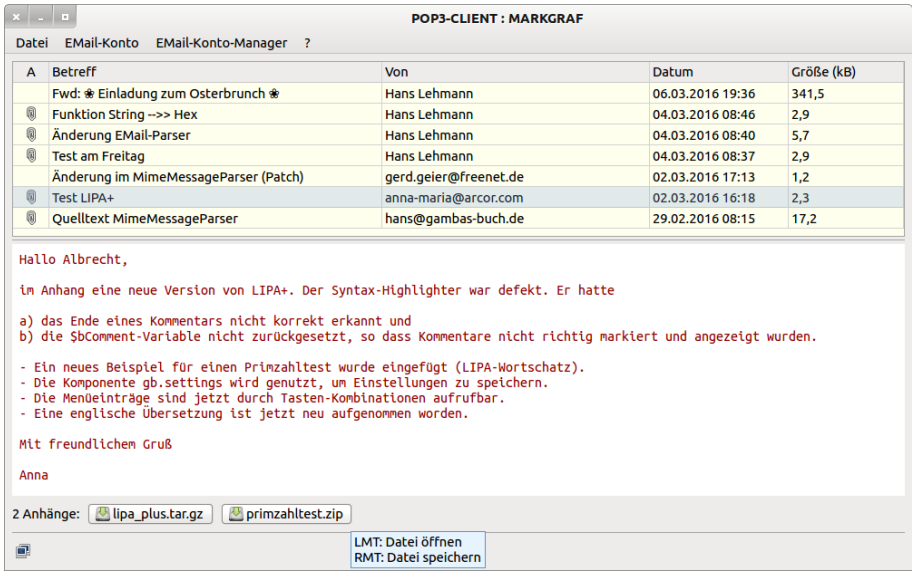

Abbildung 24.5.3.1: GUI POP3-Client

Der Quelltext für die wichtigsten Prozeduren des MimeMessageParsers ist nur auf den ersten Blick kompliziert. Das liegt m.E. vor Allem an den Abschnitten, in den *Rekursionen* den Programm-Ablauf bestimmen (→ recursive loop):

```
'::::::  BEGINN BODY PARSEN  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
' Funktion:         MessageBody(...)
' Parameter:        MimeMessageText Typ: String
' Funktionswert:    Datei-Pfad zum Text-Teil - Typ: String
Public Function MessageBody(MimeMessageText As String) As String

  $cInlineCID.Clear
  $bIsText = False
  $bIsHTML = False

  MMMP.DeleteTempFiles($sBasePath &/ $sInlineDir)
  MMMP.DeleteTempFiles($sBasePath)

  $sMM = New MimeMessage(MimeMessageText)
  ParseBody($sMM.Body)

  If $cInlineCID.Count > 0 Then
     ReplaceCID()
  Endif
  If $bIsHtml Or ($bIsHtml And $bIsText) Then
     Return $sBasePath &/ _HTMLFILENAME
  Endif
  If $bIsText Then
     Return $sBasePath &/ _TEXTFILENAME
  Endif

 End

' Prozedur:  ReplaceCID() - CID steht für Content-ID
' Parameter: -
' Aktion:    In der HTML-Datei wird jede CID durch den Pfad zur Inline-Datei ersetzt
Private Sub ReplaceCID()

  Dim sHTMLText As String
  Dim vElement As Variant

  sHTMLText = File.Load($sBasePath &/ _HTMLFILENAME)
  For Each vElement In $cInlineCID
    sHTMLText = Replace(sHTMLText, "cid:" & $cInlineCID.Key, $cInlineCID[$cInlineCID.Key])
  Next

' Speichern der geänderten HTML-Datei
  File.Save($sBasePath &/ _HTMLFILENAME, sHTMLText)

End
```

```
' Prozedur:        ParseBody()
' Parameter:       Part Typ: MimePart
' Parsen des MimeMessage-Bodys (Inline-Dateien, Nachricht)
' Die in einer HTML-Nachricht liegenden multimedialen Objekte (image, audio, video, application)
' werden in der Variablen '$cInlineCID' (Typ: Collection) mit dem originalen Datei-Namen gespeichert.
' Die Pfade zu den Objekten werden in der Variablen $cInlineCID (Typ: Collection) gespeichert.
' Eine Nachricht kann eine Text-Nachricht oder eine HTML-Nachricht oder beides sein.
' Jede Nachricht wird in einer Datei mit den Datei-Namen "html.part.html" oder "text.part.html"
' im Basis-Verzeichnis temporär gespeichert
Private Sub ParseBody(Part As MimePart)

  Dim hChild As MimePart
  Dim sTextData As String

  If Part.Data Then
     If Part.Disposition = "inline" And
        (Part.ContentType Like "image/*" Or
        Part.ContentType Like "audio/*" Or
        Part.ContentType Like "application/*" Or
        Part.ContentType Like "video/*") Then
           File.Save($sBasePath &/ $sInlineDir &/ Part.FileName, Part.Data)
           $cInlineCID[Part.ContentId] = $sBasePath &/ $sInlineDir &/ Part.FileName
     Endif

     If Part.ContentType = "text/html" Then
        $bIsHtml = True
        File.Save($sBasePath &/ _HTMLFILENAME, Part.Data)
     Endif

     If Part.ContentType = "text/plain" Then
        sTextData = "<!DOCTYPE html>" & gb.NewLine
        sTEXTData &= "<html lang=\"de\">" & gb.NewLine
        sTextData &= " <head>" & gb.NewLine
        sTextData &= "  <meta charset=\"utf-8\">" & gb.NewLine
        sTextData &= "  <style>" & gb.NewLine
        sTextData &= "    body{font-family:Verdana,sans-serif;color:darkred;font-size:16px;}" & gb.NewLine
        sTextData &= "  </style>" & gb.NewLine
        sTextData &= " </head>" & gb.NewLine
        sTextData &= " <body>" & gb.NewLine
        sTextData &= Replace(Part.Data, gb.NewLine, "<br>\n")
        sTextData &= " </body>" & gb.NewLine
        sTextData &= "</html>"
        $bIsText = True
        File.Save($sBasePath &/ _TEXTFILENAME, sTextData)
     Endif
  Endif ' Part.Data ?

  For Each hChild In Part
    ParseBody(hChild) ' Recursive loop
  Next

End

':::::  ENDE BODY PARSEN  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
':::::  ANHÄNGE TEMPORÄR SPEICHERN (DATEI)  ::::::::::::::::::::::::::::::::::::::::::::::::::::
' Funktion:        Attachments(...)
' Parameter:       MimeMessageText Typ: String
' Funktionswert:   Array mit den Datei-Pfaden zu den Anhängen - Typ: String-Array
Public Function Attachments(MimeMessageText As String) As String[]

  $aAttachmentPaths.Clear

  $sMM = New MimeMessage(MimeMessageText)
  ParseA($sMM.Part)

' Funktionswert: String-Array mit den Pfaden zu den Anhängen
  Return $aAttachmentPaths

End ' ParseAttachments(..)

' Prozedur:       ParseA(...)
' Parameter:      Part Typ: MimePart
' Aktion:         Parsen des MM-Anhangs.
'                 Jeder Anhang wird in einer Datei mit dem originalen Datei-Namen des Anhangs gespeichert
'                 Das Array $aAttachmentPaths wird mit den Datei-Pfaden zu den Anhängen gefüllt
Private Sub ParseA(Part As MimePart)

  Dim hChild As MimePart

  If Part.Disposition = "attachment" And Part.Count = 0 Then
   ' Anhang (temporär) in einer Datei mit dem Namen der Original-Datei speichern
     File.Save($sBasePath &/ $sAttachmentsDir &/ Part.FileName, Part.Data)
   ' Aktuellen Datei-Pfad zum String-Array hinzufügen
     $aAttachmentPaths.Add($sBasePath &/ $sAttachmentsDir &/ Part.FileName)
```

```
  Endif
  For Each hChild In Part
    ParseA(hChild) ' Recursive loop
  Next
End ' ParseA(..)

':::::  ENDE ANHÄNGE TEMPORÄR  SPEICHERN ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Achtung: Das Ersetzen der CID (content id) in der Prozedur ReplaceCID() funktioniert nur sicher, wenn multimediale Objekte über eine CID eingebunden worden sind.