



Löschen ohne Lesen

Reden mit dem Mailserver

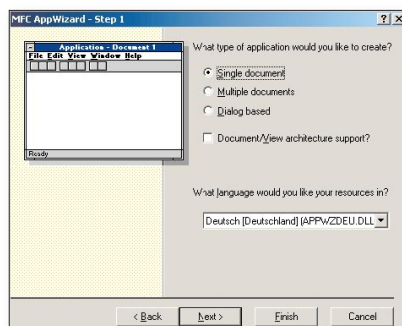


Die Kommunikation zwischen einem E-Mail Client und dem Mail-Server findet über das Post Office Protocol (POP) statt. Man kann darüber auch Mails auf dem Server löschen – ohne sie herunterzuladen.

THOMAS WÖLFER

Einigen eigenen E-Mail Client mit allem drum und dran zu programmieren ist ein eher aufwändiger Schritt. Zusätzlich zum reinen POP3-Protokoll müsste ein solches Programm noch Möglichkeiten zum Lesen und Schreiben von E-Mails bieten und auch eine Server-Verwaltung für verschiedene Mail-Konten wäre vermutlich notwendig.

Gar so weit geht es in diesem Beispiel-Programm nicht. Entwickelt werden zwei Klassen, die das POP3-Protokoll unterstützen. Das sind die Klassen *XPop3Connection*, die die Verbindung kapselt sowie die Klasse *XPop3Socket*, das die tatsächliche Arbeit mit den Windows Sockets für die POP3-Verbindung besorgt. Die eigene Socket-Klasse dient dabei in erster Linie als An-



DER MAIL-CLIENT ist eine Single-Dokument-Anwendung, aber ohne Document-View-Support.

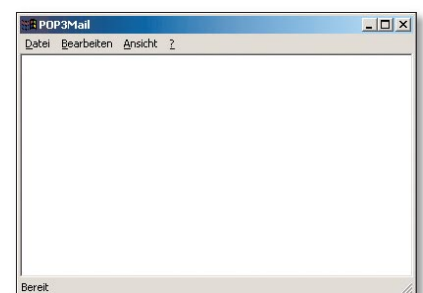
haltspunkt für die Arbeit mit Windows Sockets. Die müssen mit VC++ nicht unbedingt „von Hand“ versorgt werden, denn die MFC-Bibliothek bietet für die Arbeit mit Sockets extra eigene Socket-Klassen. Wie Sie diese effizient verwenden erfahren Sie im Beitrag „Der eigene Chat mit C++“ in diesem Sonderheft.

Mit Hilfe der beiden Klassen entwickeln Sie einen einfachen E-Mail Client. Dieser kann von dem *Subject*, den Absender und die Größe aller auf einem POP3-Server wartenden E-Mails anzeigen. In dieser Liste der Mails können eine oder mehrere ausgewählt werden. Die ausgewählten können Sie dann per Menü-Befehl löschen.

Das Grundgerüst

Beim Grundgerüst handelt es sich um eine normale MFC „Single Document“-Anwendung, allerdings ohne *Document/View*-Support. Der ist deshalb nicht notwendig, weil das Programm vollständig ohne Dokument-Daten auskommt. Wenn Sie das Projekt anlegen, ist es wichtig, dass Sie die Unterstützung für Windows Sockets aktivieren, denn diese Unterstützung ist für den Betrieb des POP3-Protokolls lebensnotwendig. (Sie müssen übrigens nicht den ganzen Quell-Code abtippen. Stattdessen können Sie das fertige Programm oder den zugehörigen Quell-Code von der Heft-CD verwenden.)

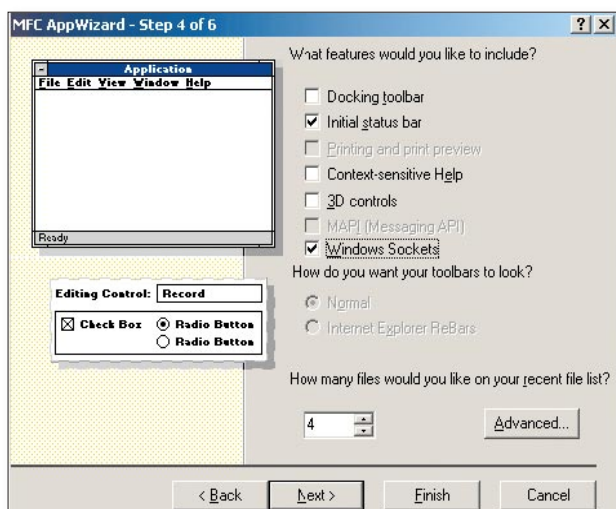
Die einzigen Änderungen, die Sie am vom AppWizard erzeugten Projekt vornehmen müssen, betreffen das Menü



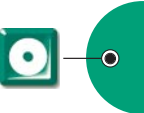
DER MAIL-CLIENT im Rohzustand. So sieht die Anwendung aus, bevor Sie auch nur eine Zeile Code geschrieben haben.

und die *CChildView*-Klasse. Keine der anderen Dateien müssen verändert werden.

Die Änderungen im Menü betreffen nur das *Bearbeiten*-Menü. Hier müssen die vom AppWizard erzeugten Menüpunkte entfernt werden, da sie keinerlei Bedeutung für den eigenen POP3-Mail-



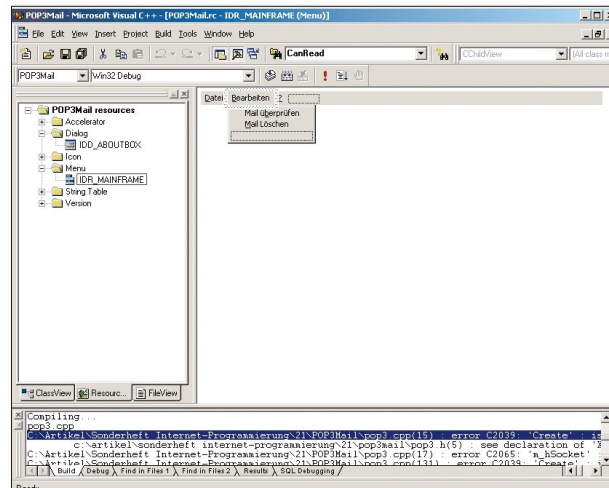
DIE WINDOWS SOCKETS brauchen Sie auf jeden Fall. Ob Sie gern eine Status-Zeile hätten oder nicht ist Geschmackssache.



Client haben. Statt dessen fügen Sie zwei neue Menübefehle ein: *Mail überprüfen* und *Mail löschen*.

Beide Befehle behandeln Sie in der *CChildView*-Klasse.

Diese Klasse kapselt den Arbeitsbereich der Anwendung. Dieser soll im Wesentlichen durch eine Liste der auf dem Mail-Server anliegenden Mails ausgefüllt werden. Dazu verwenden Sie eine Instanz vom Typ *CListCtrl*, die Sie als Member-Variable in der View-Klasse einbetten. Nachdem das Hinzufügen von Einträgen in diese Listen ein wenig aufwändig ist, deklarieren Sie auch noch



MIT DEM MENÜ-EDITOR von VC++ ist die Änderung des Menüs ein Kinderspiel.

„dows“, das zum Objekt wird im Konstruktor ist als Schema eben noch nicht angelegt. Dazu muss immer in einem zweiten Schritt eine *Create()*-Funktion aufgerufen werden. Der andere Grund liegt darin, dass das List-Control den gesamten Bereich der *CChildView*-Klasse ausfüllen soll. Das geschieht auch dann, wenn das Fenster der POP3-Anwendung in seiner Größe verändert wird, und das passiert nicht automatisch, sondern muss von Ihnen programmiert werden.

Um beide benötigten Vorgänge auszuführen, müssen Sie zwei Message-Handler in der *CChildView*-

Klasse einfügen. Einen für *WM_CREATE* und einen für *WM_SIZE*. Das führt dazu, dass die Klasse um zwei Funktionen erweitert wird: *OnCreate()* und *OnSize()*. (Siehe dazu Listing 1 im Kasten auf der nächsten Seite.)

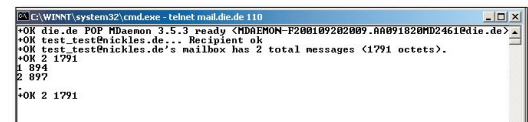
In *OnCreate()* rufen Sie zunächst die *Create()*-Funktion der Basisklasse auf. Dies stellt sicher, dass die grundlegend benötigten Dinge zum Erzeugen des Fensters erledigt werden. Dann erzeugen Sie die eigentliche Lis-

te mit *m_list.Create()*. Beim Erzeugen der Listen können Sie eine ganze Reihe von Parametern verwenden, mit denen im Wesentlichen das Aussehen der Liste festgelegt wird. Beim Beispiel wird die Liste so angelegt, dass sie im *Report*-Modus läuft.

Ist das erledigt muss das *CListCtrl* noch um die einzelnen Spalte und deren Überschriften erweitert werden. Dabei soll der Mail-Client 5 Spalten erhalten: Eine für den Namen des Mails-Servers, eine für die Nummer der Mail auf diesem Server, eine für das „Subject“ der Mail, sowie eine für den Absender und die Größe der Mail.

Beim Anlegen der Spalten geben Sie auch an, wie die Spalten formatiert sein sollen (linksbündig, rechtsbündig...) und welche Breite die Spalten initial haben sollen.

Damit sind alle benötigten Elemente erzeugt. Nun müssen Sie sich, wie bereits erwähnt, noch darum kümmern, dass das List-Control immer die richtige Größe hat. Dies passiert im *OnSize()*-Handler.

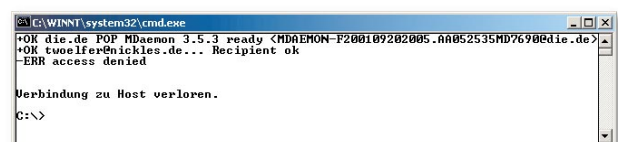


EINE TELNET-SITZUNG mit angemeldetem User. Das Bild zeigt die Antworten auf eine *LIST* und ein *STAT*-Kommando.

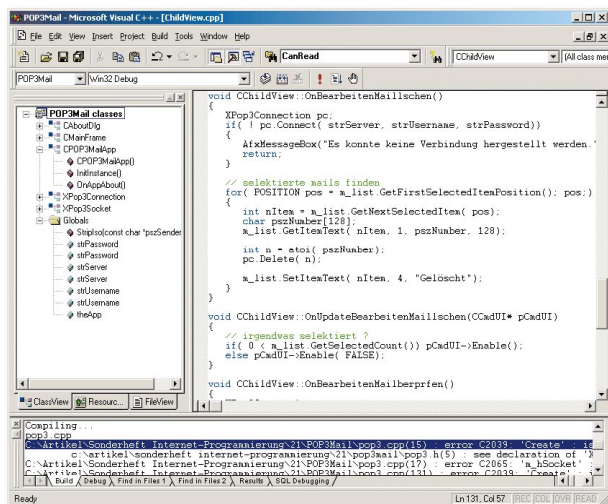
```
void CChildView::OnSize(UINT nType, int cx, int cy)
{
    CWnd::OnSize(nType, cx, cy);
    CRect rect;
    GetClientRect(&rect);
    m_list.MoveWindow(&rect);
}
```

Dieser ist relativ einfach zu implementieren. Nachdem es keine weiteren Kontroll-Elemente in der View gibt, können Sie mit *GetClientRect()* das Viereck der View ermitteln und die Liste mit *MoveWindow()* über die komplette Fläche dieses Rechtecks ausdehnen.

Bevor Sie die Bedienungsfunktionalität in den Mail-Client einbauen müssen Sie die POP3-Klassen implementieren.



AUCH MIT "TELNET" können Sie eine Verbindung zu einem POP3-Server herstellen. Mails bekommen Sie nur mit dem richtigen Passwort.



DIE CCHILDVIEW-KLASSE ist die einzige der erzeugten Klassen, in denen Sie Änderungen vornehmen müssen.

direkt eine *AddItem()*-Methode als Hilfsfunktion.

```
// ... anderer code
private:
void AddItem(const char* psz
    Server, const char* pszNumber,
    const char* pszSubject, const
    char* pszSender, const char*
    pszSize);
CListCtrl m_list;
// ... anderer code
```

Das Einbetten der *CListCtrl*-Instanz in die View-Klasse ist noch nicht ausreichend. Das hat zwei Gründe. Zum einen verwenden die MFC immer einen zweistufigen Erzeugungsprozess für Objekte. Der Aufruf des Konstruktors erzeugt zwar das Objekt, aber eben nicht vollständig, denn das „Windows Win-



POP3 für MFC

Die Pop3-Implementierung finden Sie auf der Heft-CD in den Dateien pop3.h und pop3.cpp. Im Header-File werden zwei Klassen deklariert, die in der Implementierungs-Datei ausprogrammiert werden.

Bei den beiden Klassen handelt es sich um die XPop3Socket und die XPop3

Connection. Die beiden Klassen implementieren nicht das komplette POP3-Protokoll, sind aber sicherlich ausführlich genug, um es Ihnen einfach zu machen, die wenigen fehlenden Teile selbst zu implementieren. Für einen vollständigen E-Mail Client wäre hier vermutlich noch eine XPop3Mail-Klasse hilfreich, die sich um die Belange ei-

ner eigenen POP3-Mail kümmert. Dazu würde auch die Verarbeitung des Mail-Headers zählen, die aus Gründen der Übersichtlichkeit im Beispiel-Code direkt in der Pop3Connection abgehandelt wird.

Der XPop3Socket kümmert sich um die Kommunikation mit den Windows Sockets und hat mit POP3 als solchen nicht viel zu tun. Die Klasse kann eine Socket-Verbindung auf einem beliebigen Port aufbauen sowie Daten über

LISTING 1

```
int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;

    m_list.Create( WS_VISIBLE | LVS_REPORT | LVS_SHOWSELALWAYS |
        WS_CHILDWINDOW, CRect( 0,0, 200, 200), this, 1);

    m_list.InsertColumn( 0, "Server", LVCFMT_LEFT, 100);
    m_list.InsertColumn( 1, "Nummer", LVCFMT_LEFT, 70);
    m_list.InsertColumn( 2, "Subjekt", LVCFMT_LEFT, 100);
    m_list.InsertColumn( 3, "Absender", LVCFMT_LEFT, 100);
    m_list.InsertColumn( 4, "Größe", LVCFMT_LEFT, 70);
    return 0;
}
```

LISTING 2

```
class XPop3Socket
{
public:
    XPop3Socket();
    ~XPop3Socket();

    BOOL Create();
    BOOL Connect( const char* pszHost, int nPort = 110);
    BOOL Send( const char* pszBuf, int cbBuf);
    void Close();
    int Receive( char* pszBuf, int cbBuf);
    BOOL CanRead( BOOL* pRead);

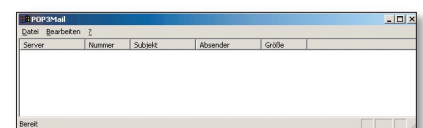
protected:
    BOOL Connect( const SOCKADDR* pSockAddr, int nSockAddrLen);
private:
    SOCKET m_hSocket;
};
```

LISTING 3

```
BOOL
XPop3Socket::Connect(const char* pszHost, int nPort)
{
    SOCKADDR_IN sockAddr;
    ZeroMemory(&sockAddr, sizeof(sockAddr));
    sockAddr.sin_family = AF_INET;
    sockAddr.sin_port = htons((u_short)nPort);
    LPHOSTENT lphost;
    lphost = gethostbyname( pszHost);
    sockAddr.sin_addr.s_addr = ((LPIN_ADDR)lphost->h_addr)->s_addr;
    return (connect( m_hSocket, (SOCKADDR*)&sockAddr, sizeof( sock
        &Addr)) != SOCKET_ERROR);
}
```

LISTING 4

```
BOOL XPop3Socket::CanRead( BOOL* pRead)
{
    timeval timeout = {0, 0};
    fd_set fds;
    FD_ZERO( &fds);
    FD_SET( m_hSocket, &fds);
    int idStatus = select(0, &fds, NULL, NULL, &timeout);
    if (idStatus == SOCKET_ERROR) return FALSE;
    *pRead = !( idStatus == 0);
    return TRUE;
}
```



DER MAIL-CLIENT im Einsatz. Noch wurden keine Informationen über Mails abgeholt.

diese Verbindung senden und empfangen. Außerdem kümmert sich die Klasse um den Status des Sockets. Das wird benötigt, um eventuelle Timeout oder sonstige Fehler behandeln zu können. Ähnlich wie andere MFC-Klassen verwendet dabei auch der XPop3Socket eine zweistufige Konstruktionsphase. (Siehe dazu Listing 2 im Kasten links.)

Der Konstruktor und Destruktor des XPop3Sockets sind übersichtlich. Der Konstruktor initialisiert das Socket-Handle m_hSocket einfach nur mit einem bekannten Wert, der eine Fehlerbe-



HIER IST ETWAS SCHIEF GEGANGEN. Vermutlich stimmt das Passwort nicht.

dingung signalisiert, während der Destruktor die Socket-Verbindung schließt:

```
XPop3Socket::~XPop3Socket()
{
    m_hSocket = INVALID_SOCKET;
}

XPop3Socket::~~XPop3Socket()
{
    Close();
}
```

Der zweite Schritt im Erzeugungsprozess eines XPop3Sockets ist der Aufruf der dazugehörigen *Create()*-Methode. Diese verwendet die *socket()*-Methode aus dem SDK, um das zum XPop3Socket-Objekte gehörende Windows Socket zu erzeugen:

```
BOOL XPop3Socket::Create()
{
    ...
}
```




```
m_hSocket = socket(AF_INET,
↳SOCK_STREAM, 0);
return (m_hSocket !=
↳INVALID_SOCKET);
}
```

Der tatsächliche Verbindungsaufbau wird in der Methode *Connect()* durchgeführt. Diese Methode erhält zwei Parameter: Den Namen eines Host-Rechners sowie den Port, auf dem die Verbindung aufgebaut werden soll. Diese Port-Nummer ist als Default-Para-

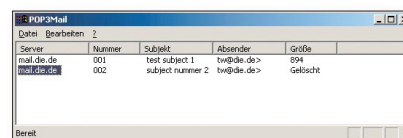
meist nicht bekannt. Statt dessen wird der Anwender immer den Namen des Rechners kennen, etwa „mail.pc-magazin.de“ und nicht 152.188.31.202.

Genau dieser Name „mail.pc-magazin.de“ wird der XPop3Socket-Klasse übergeben. Damit die Socket-Verbindung hergestellt werden kann, muss dieser Name in seine numerische IP-Adresse aufgelöst werden. Das geht mit der SDK-Funktion *gethostbyname()*. Kann der zugehörige Server ermittelt werden, liefert *gethostbyname()* einen Zeiger auf eine Struktur vom Typ *HOSTENT* zurück und die enthält unter anderem die benötigte Information.

Mit diesem fertig zusammengesetzten *SOCKADDR_IN* kann per *Socket-Connect* die Verbindung aufgebaut werden. Das Handle für diese Socket-Verbindung wird bei der Gelegenheit im *m_hSocket* der Klasse gespeichert. Mit Hilfe dieses Handles kann später über den Socket kommuniziert werden.

■ Senden und empfangen

Mit den Methoden *Send()* und *Receive()* kann eine XPop3Socket-Instanz Daten zum anderen Rechner senden bzw. Daten von dort empfangen. Nachdem die ganze Funktionalität dazu bereits von den SDK-Funktionen erledigt wird, sind die Methoden des XPop3Sockets entsprechend wenig umfangreich:



WURDE EINE E-MAIL GELOESCHT, wird dies in der Anzeige des Mail-Clients entsprechend angezeigt.

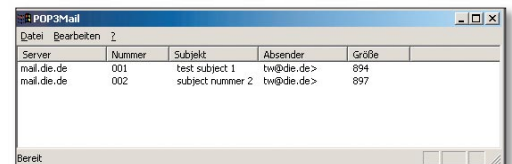
```
BOOL XPop3Socket::Send( const
↳char* pszBuf, int cbBuf)
{
    ASSERT( m_hSocket !=
↳INVALID_SOCKET);
    return ( send( m_hSocket, psz
↳Buf, cbBuf, 0) != SOCKET_
↳ERROR);
}
```

Zuletzt hat die XPop3Socket-Klasse noch eine besondere Funktionalität. Bei

einer Socket-Verbindung kann alles Mögliche schief gehen. Der Rechner auf der Gegenseite kann zum Beispiel plötzlich abstürzen und nicht mehr antworten. Oder die Antworten können sich plötzlich verzögern, zum Beispiel, weil irgendwo eine Störung im Netz eingetreten ist, oder einer der beteiligten Rechner oder Router kurzfristig mit anderen Dingen beschäftigt ist.

Das führt dazu, dass Sie bei Verbindungen über Sockets grundsätzlich damit rechnen müssen, dass Timeouts eintreten. Andererseits müssen Sie auch davon ausgehen, dass eine Verzögerung nicht unbedingt einen Fehlerfall signalisiert.

Bei den Windows Sockets kann der Zustand einer Socket-Verbindung mit der *select()*-Funktion getestet werden. Entweder scheitert diese, oder sie signalisiert den Zustand der Verbindung.

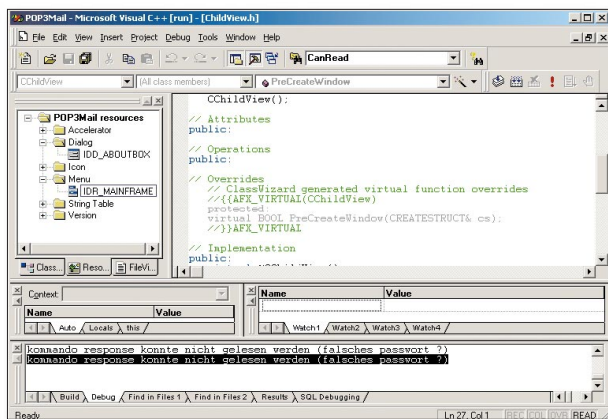


IN AKTION: Auf dem Server liegen zwei E-Mails. Der Client gibt darüber Aufschluss, wie groß diese sind und wer der Sender der Mail war.

Auch dieser kann natürlich „fehlerhaft“ sein. Ist er das nicht, kann der Socket weiter zur Kommunikation verwendet werden, wenn auch, unter Umständen, sehr langsam.

Die XPop3Socket-Klasse wird von der XPop3Connection-Klasse an verschiedenen Stellen verwendet. Die XPop3Connection-Klasse muss über den Zustand der Verbindung informiert sein. Sonst kann es leicht passieren, dass vergeblich auf Daten gewartet wird, die schlicht und ergreifend nicht kommen werden. Andersherum ist es auch möglich, dass die Verbindung per Timeout beendet wird, obwohl die Sockets in einem gültigen Zustand sind. Sogar passiert dann, wenn das Timeout, so wie im Beispiel, implementiert in der darüberliegenden Schicht ausgelöst wird. Damit solche unschönen Zustände nicht eintreten, überprüft die XPop3Connection den Zustand der verwendeten Socket-Instanz. Und das tut sie mit der Funktion *XPop3Socket::CanRead()*. (Siehe dazu Listing 4 im Kasten auf der vorherigen Seite.)

Die Ihnen bisher vorliegende Implementierung ermöglicht die Verbin-



FEHLGESCHLAGENE BEFEHLE protokolliert der Mail-Client im Output-Fenster von VC++.

meter mit dem Wert 110 belegt. Dies ist der bei POP3 normalerweise verwendete Port. (Siehe dazu Listing 3 im Kasten auf der vorherigen Seite.)

Nachdem hier eine Verbindung aufgebaut wird, mag es Sie verwundern, dass die Methode keinen Benutzernamen und kein Passwort als Parameter erhält. Das liegt daran, dass solche Parameter für eine Authentifizierung erst auf einer höheren Ebene benötigt werden. Die Socket-Verbindung, falls physisch möglich, kann immer durchgeführt werden. Ob das Protokoll, das über diese Verbindung verwendet wird, Authentifizierung benutzt, ist eine andere Sache. Bei POP3 geschieht das so, und Benutzername und Passwort kommen bei der *XPop3Connection*-Klasse ins Spiel.

Um eine Socket-Verbindung aufzubauen, benötigen Sie eine richtig initialisierte Variable vom Typ *SOCKADDR_IN*. Diese muss ausgenullt werden. Danach ist die „Address Family“ zu setzen, die immer *AF_INET* lautet. Ferner muss die Port-Nummer für die Verbindung angegeben werden. Diese erhält *Connect()* als Parameter.

Schließlich brauchen sie noch die IP-Adresse des Rechners, der kontaktiert werden soll. Das ist ein bisschen problematisch, denn diese Adresse ist dem Anwender eines solchen Programmes



dungsaufnahme per Socket. Was noch fehlt, ist die Protokoll-Schicht, die das eigentliche POP3-Protokoll implementiert. Das ist Aufgabe der XPop3 Connection-Klasse, die Sie im Folgenden erläutern finden.

Die XPop3Connection implementiert dabei die elementaren Teile des POP3-Protokolls. Zu Pop3 muss man wissen, dass dieses Protokoll anders als

Anmelden gibt es das *PASS* und das *USER* Kommando.

Ebenso gibt es Kommandos zum Löschen von E-Mails und zum Herunterladen (anfordern) von Mails. Schließlich kann der Mail-Header mit dem *TOP*-Kommando abgerufen werden, ohne sonst den Zustand auf dem Mail-Server zu verändern.

Die vorliegende XPop3Connection-Klasse implementiert nicht alle POP3-Kommandos. Allerdings können Sie die fehlenden mit den vorhandenen Hilfsfunktionen leicht nachrüsten. Im Wesentlichen unterstützt die vorliegende Implementierung Folgendes:

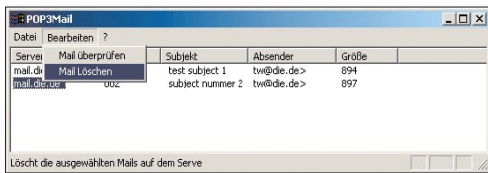
- Verbindungsaufnahme mit dem POP3-Server
- Authentifizierung am Server mit User-Name und Passwort (*USER* und *PASS*)
- Erfragen der Anzahl der Mails und Auflistung der Größen der Mails (*STAT* und *LIST*)
- Erfragen eines Message-Headers (*TOP*)
- Löschen einer Nachricht auf dem Server (*DELETE*)
- Beenden der Verbindung (*QUIT*)

Der Konstruktor der XPop3Connection initialisiert im Wesentlichen nur seine eigenen Member-Variablen. Der Verbindungsaufbau findet mit der *Connect()*-Methode statt. Auch der De-

struktor der Klasse ist nicht gerade überlastet. Er ruft die *Disconnect()*-Methode auf, mit der die Verbindung beendet wird.

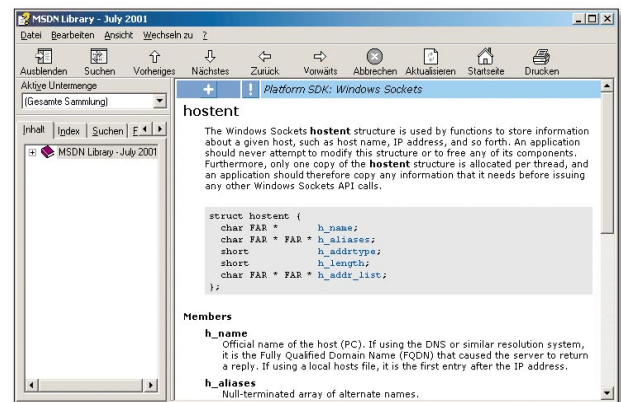
Die *Connect*-Methode hat – unter Auslassung der Fehlerbehandlung – folgenden Aufbau: siehe Listing 5 im Kasten auf der nächsten Seite.

Zunächst wird die eingebettete Instanz des XPop3Sockets erzeugt. Dann wird mit diesem Socket eine Verbindung zum angegebenen Server hergestellt. Dieser Zustand wird in einer Member-Variable gemerkt. Im vollständigen Quell-Code findet an dieser Stelle auch eine Fehlerbehandlung statt. Es kann durchaus sein, dass der angegebene Ser-



AUSGEWÄHLTE E-MAILS können direkt auf dem Server gelöscht werden, ohne dass sie zuvor heruntergeladen werden müssen.

zum Beispiel FTP funktioniert. (Einen Beitrag über einen eigenen FTP-Client finden Sie an anderer Stelle in diesem Sonderheft.). Das FTP-Protokoll war nie dafür gedacht „automatisiert“ zwischen zwei Rechnern eingesetzt zu werden. Bei Pop3 ist das anders. Trotzdem ist das Protokoll eher rudimentär. Um genau zu sein, ist es so einfach aufgebaut, dass Sie auch mit dem bei Windows mitgelieferten Telnet-Programm, mit einem POP3-Server kommunizieren können, denn alle Kommandos und Daten, die per POP3 versendet werden, sind klar lesbar. So gibt es zum Beispiel das *LIST*-Kommando, das den POP3-Server dazu veranlasst, alle Mails des angemeldeten Users aufzulisten. Zum

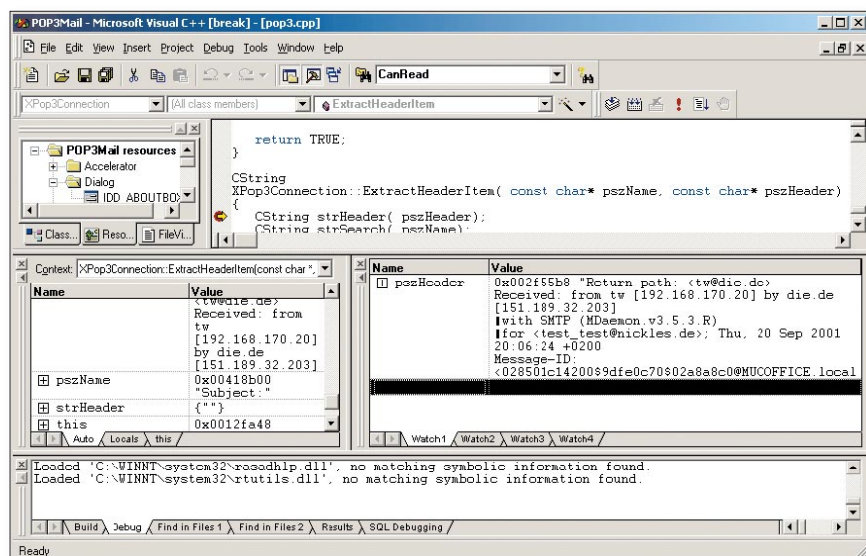


MSDN IST IMMER EINEN BLICK WERT. Hier findet sich auch die Dokumentation des Socket SDKs.

ver nicht erreicht werden kann. Das ist sicherlich ein Zustand, der von Interesse ist und daher auch behandelt wird. Das würde den abgedruckten Quell-Code unnötig aufblähen, daher wird im Folgenden nur erläutert, wo welche Behandlung stattfindet, statt den kompletten Code mit abzudrucken.

Ist die Verbindung hergestellt, rufen Sie *ReadCommandResponse()* auf. Diese Funktion (und Ähnliche, wie zum Beispiel *ReadListResponse()* oder *ReadTopResponse()*) werden Sie im Folgenden noch des öfteren zu Gesicht bekommen. Die Funktion dient dazu, die letzte Antwort des POP3-Servers von dort abzuholen und auszuwerten. Gelingt dies nicht, handelt es sich immer um einen Fehlerfall.

Nun beginnt der Authentifizierungsschritt. Dazu muss der Username an den Server geschickt werden. Das geschieht mit dem *USER*-Kommando, das vom Namen des Users gefolgt sein muss. Der dazu vorbereitete String wird mit *m_pop3.Send()* an den Server übermittelt. Direkt anschließend wird



DER MAIL-HEADER enthält eine ganze Menge Informationen. Welche das sind, kann man sich im Debugger leicht anzeigen lassen.



die Command-Response, also die Server-Antwort, ausgewertet. Tritt hier ein Fehler auf, wurde vermutlich der

Name eines auf dem Server unbekannten Accounts an diesen übermittelt. Im folgende Schritterfolgen nochmals

LISTING 5

```

BOOL XPop3Connection::Connect( const char* pszServer, const char*
    pszUser, const char* pszPassword, int nPort)
{
    m_pop3.Create();
    m_pop3.Connect( pszServer, nPort);
    m_fIsConnected = true;
    ReadCommandResponse();
    CString strUSER = CString("USER ") + pszUser + CString("\r\n");
    m_pop3.Send( strUSER, strUSER.GetLength());
    ReadCommandResponse();
    CString strPW = CString("PASS ") + pszPassword +
        CString("\r\n");
    m_pop3.Send( strPW, strPW.GetLength());
    ReadCommandResponse();
    return TRUE;
}

```

LISTING 6

```

while( ! fFoundTerminator)
{
    if(( ::GetTickCount() - dwStartTimeout) > m_dwTimeout)
    {
        TRACE("TIMEOUT!\n");
        return FALSE;
    }
    BOOL fReady;
    if( ! m_pop3.CanRead( &fReady))
    {
        TRACE("TIMEOUT(2)\n");
        return FALSE;
    }
    // world wide wait ...
    if( ! fReady)
    {
        Sleep(300);
        continue;
    }
}
// ... restlicher Code.

```

LISTING 7

```

void CChildView::OnBearbeitenMailberprfen()
{
    XPop3Connection pc;
    if( ! pc.Connect( strServer, strUsername, strPassword))
    {
        AfxMessageBox("Es konnte keine Verbindung hergestellt
            werden.");
        return;
    }
    int nMails, cbMail;
    if( ! pc.Stats( nMails, cbMail))
    {
        AfxMessageBox("STATS konnte nicht abgerufen werden.");
        return;
    }
    for( int i=1; i<=nMails; i++)
    {
        DWORD dwSize;
        pc.GetMessageSize( i, dwSize);
        char szNumber[16];
        sprintf( szNumber, "%03d", i);
        char szSize[16];
        sprintf( szSize, "%d", dwSize);
        CString strFrom, strSubject;
        pc.GetMessageHeader( i, strFrom, strSubject);
        AddItem( strServer, szNumber, strSubject, StripIso(str
            From), szSize);
    }
    if( ! nMails)
        AddItem( strServer, "000", "-keine Mails-", "-", "-");
}

```

die gleichen Schritte, diesmal aber mit dem zum User-Namen gehörenden Passwort. Auch hier kann *ReadCommandResponse()* fehlschlagen. Ein Fehlschlag bedeutet mit recht großer Sicherheit, dass das angegebene Passwort nicht korrekt war. War alles in Ordnung liefert die Methode *true*. Der Anwender ist nun angemeldet.

Statistiken erfragen

Im angemeldeten Zustand können Statistiken vom POP3-Server erfragt werden. Mit *ReadStatResponse()* wird dabei die Antwort auf ein *STAT*-Kommando ausgelesen, *ReadListResponse()* wertet die Antwort aufs *LIST*-Kommando aus. Mit dem *List*-Kommando kann die Größe aller Mails auf dem Server ermittelt werden, das *STAT*-Kommando liefert zuvor die Anzahl der POP3-Mails für den angemeldeten Account. Nachdem beide Funktionen relativ ähnlich funktionieren, hier nur der generelle Ablauf anhand von *ReadListResponse()*, auch hier ohne Fehlerbehandlung und Ähnlichem.

Das Problem bei diesen Kommandos ist, dass die Größe des Speicherblocks, der für das Auslesen der Antwort benötigt wird, nicht im Vorhinein klar ist. Darum wird ein „Overflow“-Puffer verwendet. Reicht der normale Speicherblock nicht aus, wird statt dessen zusätzlicher Speicher alloziert und dessen Adresse in „pszOverflow“ zurückgeliefert. Ansonsten enthält „pszBuf“ die Antwort:

```

ReadResponse( pszBuf, cbSize,
    pszOverflow);
if( pszOverflow)
    pszMessageBuf = pszOverflow;

```

Diese Antwort muss zunächst überprüft werden. War alles o.k., müssen die ersten Paar Bytes der Antwort, den Text *OK* beinhalten. Tun sie das nicht, hat der POP3-Server eine Antwort geliefert, mit der man programmatisch nicht viel anfangen kann. Darauf müssen Sie reagieren, schließlich haben Sie keine gültige Antwort erhalten.

```

if( strncmp( pszMessageBuf, "+OK",
    3) != 0)
{
    delete [] pszBuf;
    if(pszOverflow)
        delete [] pszOverflow;
    return FALSE;
}

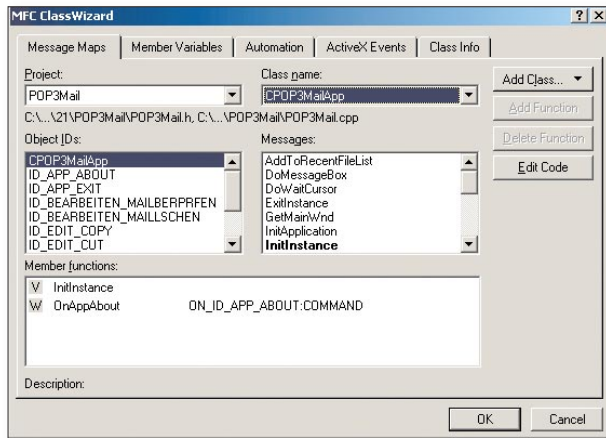
```

Ansonsten ist die Antwort gültig und enthält, durch Tabs oder Leerzeichen getrennt, die Größen der Nachrichten (natürlich nur beim *List*-Kommando). Das Ende der Nachricht ist mit einem einzelnen Punkt markiert:



```
for( ; *pszSize != '.';
    *pszSize++)
    if( *pszSize == '\t' || *psz
        Size == ' ')
        m_arrMsgSizes.Add( atoi(psz
            *Size));
```

Schließlich müssen Sie die Texte nur noch in Integer umwandelt. Das geht am einfachsten mit `atoi()` aus der C-Run-time-Bibliothek.



ZUM EINFÜGEN DER MESSAGE-HANDLER verwenden Sie am besten den ClassWizard, damit ist es am einfachsten.

■ Nachrichten-Header lesen

Schließlich können Sie mit der Methode `ReadTopResponse()` die Header der Nachrichten abrufen. `ReadTopResponse` gehört zum `TOP` Kommando. Auf dieses Kommando reagiert der Server mit der Übertragung des Message-Headers. Jeder Eintrag im Header steht

dabei in einer eigenen Zeile. So hat etwa das *Subjekt* eine separate Zeile, genau wie der *From*-Eintrag und alle anderen.

Für den POP3 Client brauchen Sie davon nur die Einträge *From* und *Subject*, denn die Größe der Nachricht haben Sie ja bereits auf anderem Wege ermittelt. Dazu gibt es im Beispiel-Code eine einfache Funktion namens `ExtractHeader-`

`Item()`, das einen beliebigen Eintrag aus dem zuvor gelesenen Header extrahieren und als separaten String zurückliefern kann. Wenn Sie also den Mail-Client um weitere Header Informationen erweitern möchten, ist das damit möglich.

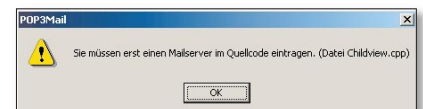
Trickreich ist schließlich noch die `ReadResponse()`-Methode. Diese Methode wird überall aufgerufen und ist für die eigentliche Kommunikation per XPop3Socket zuständig. Das Trickreiche daran ist, dass diese Methode nicht nur die Daten abholen muss. Sie muss sich auch darum kümmern, dass keine Timeouts eintreten oder Unsinn geliefert wird, weil der Socket nicht mehr ordnungsgemäß zur Verfügung steht.

Das erledigen Sie auf zwei Arten. Zum einen wird jedesmal vor dem Lesen aus dem Socket, die bereits erläuterte `Can-`

`Read()`-Funktion aufgerufen, zum anderen verwenden Sie die Windows `GetTickCount()` API um einen rudimentären Schutz vor Timeouts sicherzustellen. Kann aus dem Socket zwar gelesen werden und stehen auch Daten an, nur sind diese noch nicht verfügbar, legen Sie das Programm für ein Paar Millisekunden schlafen, bevor Sie den nächsten Versuch starten: siehe dazu Listing 6 im Kasten auf der vorherigen Seite.

■ POP3-Klassen verwenden

Im Client selbst sind die POP3-Klassen einfach zu verwenden. Der Handler zum überprüfen auf neue E-Mails hat dabei den folgenden Aufbau: siehe da-



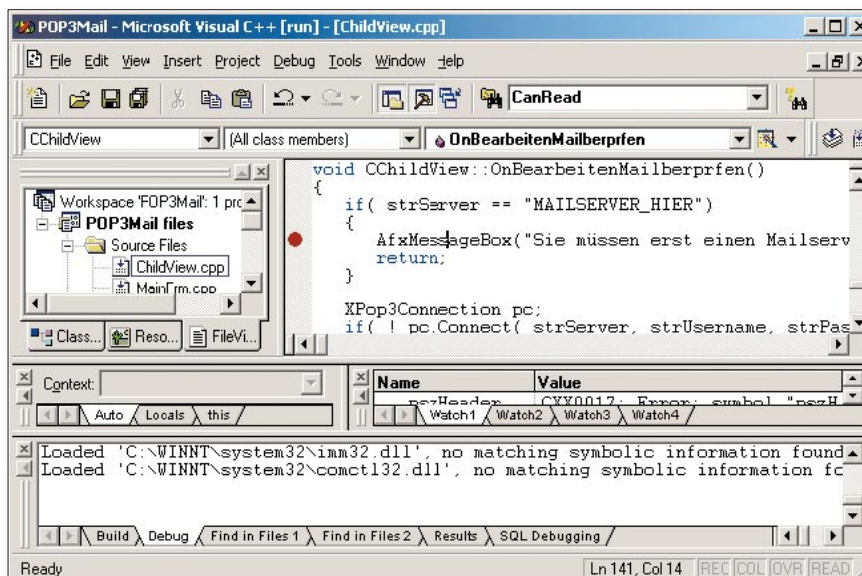
WENN SIE DAS PROGRAMM nicht neu übersetzt haben, weiß es nicht, welchen Mail-Server es kontaktieren soll.

zu Listing 7 im Kasten auf der vorherigen Seite.

Zunächst wird die POP3 Verbindung aufgebaut. Das Programm verwendet dazu fest einkompilierte Werte. Für die tägliche Benutzung wäre es eine mögliche Erweiterung, wenn Sie ein Benutzer-Interface für die Eingabe von Server, Benutzername und Passwort hinzufügen. Konnte die Verbindung hergestellt werden, wird mit `GetStats()` ermittelt, wie viele Mails auf dem Server liegen, und dann für jede einzelne Nachricht mit `GetMessageHeader()` die gewünschten Header-Informationen erfragt.

Alle erfragten Informationen werden mit der bereits erläuterten `AddItem()`-Hilfsmethode in der Liste eingetragen. Dabei wird auch die Nummer der E-Mail auf dem Server mit gespeichert. Das ist wichtig, denn diese Nummer brauchen Sie später, wenn Sie mit `Delete()` eine oder mehrere Mails auf dem Server löschen möchten. Beim Beispielprogramm geht das so, dass Sie die zu löschenden Mails in der Liste markieren und dann den Befehl zum Löschen auswählen.

In diesem Beitrag haben Sie erfahren, wie Sie das POP3-Protokoll mit eigenen Programmen nutzen können. Die dabei ganz nebenbei angefallenen XPop3-Klassen können Sie ohne Weiteres weiterverwenden. Dem vollständigen eigenen E-Mail Client steht also nichts mehr im Wege. UR



UM DEN MAIL-CLIENT verwenden zu können, müssen Sie im Quell-Code erst einen Mail-Server eintragen.