



Ein Port-Scanner im Eigenbau

# Wo bin ich?

Script-Kiddies benutzen sie gerne, Administratoren brauchen sie fast täglich: Port-Scanner.

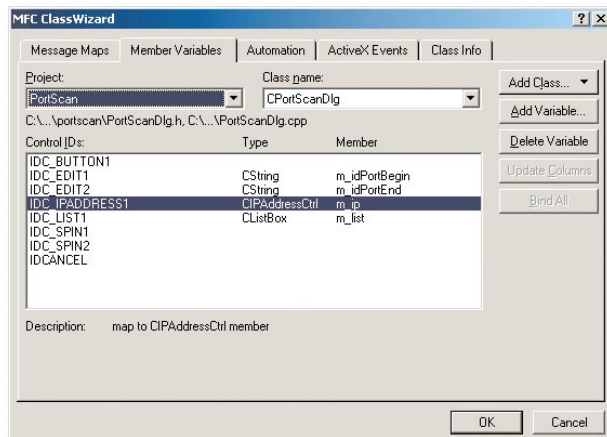
THOMAS WÖLFER

Mit seinen ausgewachsenen Konkurrenten wird Ihr eigener Scanner zunächst nicht mithalten können – an diesen Programmen wird schon lange gearbeitet, und sie werden verfeinert. Für Sie ist es interessant, die Grundzüge eines solchen Programms zu verstehen. Diese können dann in deutlich mächtigeren Programmen eingebaut und verwertet werden. So ist zum Beispiel ein Scanner mit multiplen Threads, der deutlich schneller wäre, auch nicht wesentlich aufwändiger zu programmieren, als das hier vorliegende Beispielprogramm.

## Ein paar Grundlagen

Nachdem das Internet als solches auf Grund von einfach zu bedienenden Browsern und relativ leicht konfigurierbaren Zugängen für jedermann erreichbar ist, ist es an vielen Stellen für die Masse der Internet-Benutzer weiterhin völlig unverständlich geblieben. TCP/IP, Ports und Sicherheitsfragen sind und bleiben für die meisten Surfer unverständliche Konzepte. Daher rührt wohl auch das etwas sagenumwobene Image von Port-Scannern. Viele Anwender fühlen sich regelrecht angegriffen, wenn die eigene „Personal Firewall“ einen Scan anzeigt. Dabei handelt es sich meist nur um einen Spieler, der einen Game-Server für Half-Life oder Ähnliches sucht und nicht um den bösen Hacker.

Port-Scans sind in der Praxis eine ganz simple Angelegenheit. Dabei versucht ein Programm über eines der zu TCP/IP gehörenden Protokolle mit einem oder mehreren Ports auf einem anderen Rechner Kontakt aufzunehmen. Sinn



**DIE BENÖTIGTEN VARIABLEN** legen Sie am einfachsten mit dem ClassWizard an.

der Sache ist es, nicht nur einen offenen Port zu finden, sondern Programme, die auf diesem offenen Port auf Anfragen warten und mit denen kommuniziert werden kann. Bei vielen Ports ist das zu erwartende Programm von vornherein bekannt, weil die Belegungen von einer ganzen Reihe von Ports standardisiert sind. „Professionelle“ Scanner versuchen auf anderen Ports, das auf der Gegenseite laufende Programm zu ermitteln. Das ist in vielen Fällen einfach, weil sich bestimmte Programme bei Anfragen mit festgelegten Antworten melden. Anders wäre es auch nicht möglich, denn wie sollte etwa ein „Half-Life“-Client sonst feststellen, dass er sich mit einem Half-Life-Server unterhält, und nicht mit irgendeiner anderen Software.

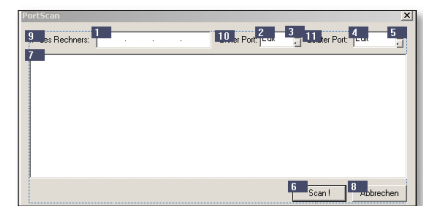
Nun aber zum eigenen Scanner. Den werden Sie mit VC++ implementieren. Der Scanner wird als auf MFC basierendes Projekt vom Typ *Dialogbox* an-

gelegt. Dabei sollten Sie darauf achten, dass Sie beim Erzeugen des Projektes die Option zum Support von Windows-Sockets einschalten, denn diese Sockets sind für die Kommunikation über TCP notwendig.

Wenn Sie aber vergessen, diese Option einzuschalten, ist das nicht so schlimm, denn im Wesentlichen führt die Option dazu, dass wenige Zeilen Code vom AppWizard eingefügt werden. Das können Sie bei Bedarf später selbst nachholen. Alles was benötigt wird, ist ein Funktionsaufruf in der *OnInitInstance()*-Methode Ihrer Anwendungs-klasse:

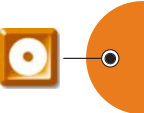
```
if (!AfxSocketInit())
{
    AfxMessageBox("Windows
    Sockets konnten nicht
    initialisiert werden.");
    return FALSE;
}
```

Die Dialogbox selbst wurde bereits vom AppWizard erzeugt, was Sie nun tun müssen ist diese mit den benötigten Elementen auszustatten. Sie brauchen folgende Elemente:



**DAMIT DIE UP-DOWN CONTROLS FUNKTIONIEREN**, müssen Sie in der Tab-Order direkt hinter den Edit-Controls stehen, die sie kontrollieren sollen.

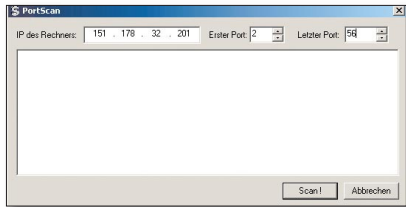
- eine IP-Control zur Eingabe des zu scannenden Rechners,
- ein Feld zur Eingabe der ersten zu scannenden Nummer. Aus Gründen der Bequemlichkeit kann es nicht schaden, ein Up-Down-Control mit einzufügen. Bei diesem setzen Sie noch die Style-Bits *AutoBuddy* und *SetBuddyIntegers*. Das



führt dazu, dass Sie das Control ohne weitere Maßnahmen zum Hoch- und Runterzählen des Wertes im Eingabefeld verwenden können.

- Ein weiteres Feld und Up-Down- Control für die Eingabe des letzten zu scannenden Ports,
- eine Listbox, in der die Ergebnisse des Scans angezeigt werden können,
- ein Button, der den Scan-Vorgang auslösen wird.

Nun fügen Sie mit dem ClassWizard ein paar Variable hinzu und binden diese an die Eingabefelder. Für das IP- Control verwenden Sie ein CIPAddress-



**GLEICH GEHT'S LOS:** Alle benötigten Angaben wurden bereits gemacht.

Control mit dem Namen `m_ip`. Für die beiden Port-Felder nehmen Sie je einen CString und nennen diese `m_idPortBegin` und `m_idPortEnd`. Das List-Control binden Sie an eine CListBox mit dem Namen `m_list`.

Das einzige Control, das Sie nun explizit behandeln müssen, ist der Button, der den Scan-Vorgang auslösen soll. Dafür definieren Sie einen `OnClick()`-Handler. Von dort aus wird dann die ganze Arbeit erledigt.

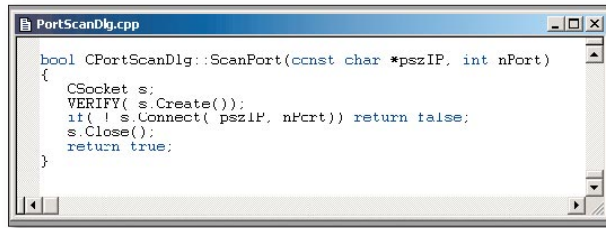
Zunächst stellen Sie darin sicher, dass der MFC-Data-Transfer auch stattgefunden hat, indem Sie `UpdateData()` aufrufen.

Lesen Sie nun die eingegebenen Port-Nummer aus und konvertieren diese mit `atoi()` in Integer:

```
UINT p1 = atoi( m_idPortBegin);
UINT p2 = atoi( m_idPortEnd);
```

Im vollständigen Beispiel-Code, den Sie auf der Heft-CD finden, werden ein Paar Fehlertests durchgeführt. Die sind hier nicht weiter von Interesse und

werden daher auch nicht erwähnt. Als nächstes lesen Sie die eingegebene



**DIE EIGENTLICHE SCAN-FUNKTION** könnte einfacher nicht sein. Dank MFC reicht im Wesentlichen ein einziger Funktionsaufruf.

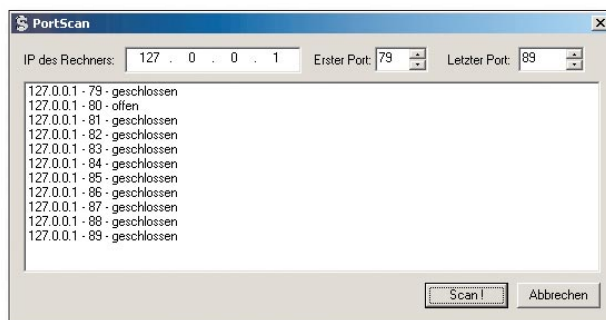
ID-Adresse aus. Das CIPAddressCtrl liefert diese als 4 Bytes zurück:

```
BYTE b0, b1, b2, b3;
m_ip.GetAddress( b0, b1, b2, b3);
```

Für den weiteren Ablauf brauchen Sie die Adresse als String, daher müssen Sie die 4 Bytes mit `sprintf()` zu einem solchen zusammensetzen:

```
char szIP[16];
sprintf( szIP, "%d.%d.%d.%d",
        b0, b1, b2, b3);
```

Der Scan kann beginnen. Dazu iterieren Sie über den angegebenen Port-Bereich. Für jeden Port rufen Sie die Funktion `ScanPort()` auf, die weiter unten beschrieben wird. In Abhängigkeit vom `Return`-Wert dieser Funktion bauen Sie erneut einen String zusammen und kopieren diesen an das Ende der Listbox: Diese enthält am En-



**SO SIEHT DAS RESULTAT AUS:** Der Scanner liefert den Zustand der Ports in der Listbox zurück.

de für jeden gescannten Port einen informativen Text.

```
for( UINT nPort = p1; nPort <=
    p2; nPort++)
{
    char sz[1024];
    if( ScanPort( szIP,
        nPort)) sprintf( sz,
        "%s - %d - offen",
        szIP, nPort);
```

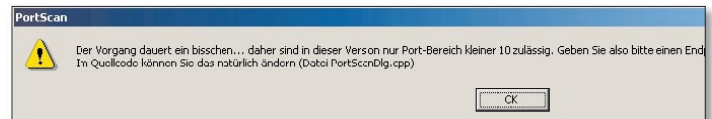
```
else sprintf( sz, "%s -
    %d - geschlossen", szIP,
    nPort);
    m_list.AddString( sz);
}
```

Nun kommt das eigentliche „Scannen“. Der komplette Scan-Vorgang kommt mit einer (!) Zeile Code und ein bisschen Infrastruktur außen herum aus:

```
bool CPortScanDlg::ScanPort
( const char *pszIP, int nPort)
{
    CSocket s;
    s.Create();
    if( ! s.Connect( pszIP,
        nPort)) return false;
    s.Close();
    return true;
}
```

Zunächst erzeugen Sie einen CSocket aus der MFC-Bibliothek. Verwenden Sie die `Connect()`-Methode dieses Sockets und versuchen sich mit dem angegebenen Rechner und dem angegebenen Port zu verbinden. Entweder geht das nun, oder es geht nicht. Damit ist unser Scan fertig.

Bei einem „echten“ Scanner würden Sie nun weitermachen. Wenn es sich



Wenn Sie größere Bereiche als einen Zehner-Bereich testen wollen, müssen Sie Änderungen am Quell-Code vornehmen.

beim Port um einen bekannten Port eines Protokolls handelt, könnten Sie zum Beispiel überprüfen, ob auf der anderen Seite auch ein Programm läuft, das dieses Protokoll unterstützt. Wäre das der Fall, könnten Sie sich mit dem Programm unterhalten. Wie so etwas funktioniert, finden Sie an einer anderen Stelle im Heft. Im Beitrag „Reden mit dem Mailserver“ wird dieser Vorgang zwischen einem POP3-Protokoll und einem entsprechenden Server beschrieben. Dort finden Sie übrigens auch eine „eigene“ Socket-Klasse implementiert, mit der Sie sehen können, wie die Kommunikation per Socket „intern“ abzuwickeln ist.

Bevor Sie nun anfangen, bedenkenlos Ports zu scannen, beachten Sie bitte, dass Sie damit nicht nur Bandbreite verschwenden, sondern unter Umständen auch Anwender erschrecken. Vielleicht beanspruchen Sie die Geduld von Administratoren gewaltig, zumindest dann, wenn Sie unentwegt mit Ihren Scans fortfahren. Zum Testen nehmen Sie daher besser einen eigenen Rechner. UR