Einführung

In diesem kurzen Tutorial erfahren Sie wie sich die lokale IP-Adresse samt CNAME mit C++ und Windows Sockets herausfinden lässt. Darüberhinaus lernen Sie wie sich ein Domainname mithilfe des so genannten *Domain name system* in die zugehörige IP-Adresse mappen lässt. In WinSock existiert eine Funktion mit dem Namen getaddrinfo um beispielsweise an die eigene IP-Adresse zu gelangen. Diese Funktion empfängt alle Details zur IP-Adresse in einer Variable vom Typ "structure addrinfo".

Der erste Schritt beim Erstellen des Programms ist sicher zu stellen, dass die Socket Library initialisiert wurde, falls diese nirgends anders schon initialisiert wurde. Im nächsten Schritt wird die Funktion getaddrinfo aufgerufen die einen Zeiger auf eine Liste mit addrinfo Strukturen zurückliefert. Dieser Zeiger kann dazu verwendet werden um die IP-Adresse in einen sockaddr_in Typ zu konvertieren. Tatsächlich ist es auch möglich die IP-Adresse ohne die Verwendung einer sockaddr_in Struktur zu erhalten. Aber mit sockaddr_in lässt sich diese Aufgabe viel sauberer und schneller erledigen.

Falls Sie bisher mit der Funktion gethostbyname gearbeitet haben, sei hiermit erwähnt das nach Angaben von Microsoft die Funktion gethostbyname für Anwendungen die auf Windows Sockets 2 aufbauen nicht mehr verwendet werden sollte. An die Stelle der alten Funktion tritt eine neue mit dem Namen getaddrinfo
die Teil der POSIX Standard API ist und daher eine plattformunabhängige Verwendung zulässt. Die Funktion ist folgendermaßen deklariert.

[code xml:lang="cpp"]int WSAAPI getaddrinfo(const char* nodename, const char* servname, const struct addrinfo* hints, struct addrinfo** res);[/code]

Der erste Parameter mit der Richtung [in] ist ein Zeiger auf einen NULL-terminierten ANSI string der einen Hostnamen (node) oder aber einen numerischen string beinhaltet. Der string hat die bekannte hexadezimale Notation (127.0.0.1) und kann eine IPv4 oder auch eine IPv6 Adresse sein. Der zweite Parameter, ebenfalls [in], ist ein Zeiger auf einen NULL-terminierten ANSI string der entweder einen Servicenamen oder eine Portnummer darstellt. Der dritte Parameter *h ints*
[in] ist ein Zeiger auf eine addrinfo Struktur. Diese Struktur stellt hintergründige Informationen über den Sockettyp den der Caller, also der Aufrufer, unterstützt bereit. Der letzte Parameter namens res ist die [out] Variable die alle unsere von getaddrinfo gesammelten Resultate über den Host enthält. Es handelt sich dabei um einen Zeiger auf eine verkettete Liste mit einer oder mehreren addrinfo Strukturen. Nachfolgend sehen sie den Aufbau dieser Struktur.

[code xml:lang="cpp"]typedef struct addrinfo { int ai_flags; int ai_family; int

ai_socktype; int ai_protocol; size_t ai_addrlen; char* ai_canonname; struct sockaddr* ai_addr; struct addrinfo* ai_next; } ADDRINFOA, *PADDRINFOA;[/code] Die Tatsache das es sich bei der Variable res um einen Zeiger auf eine verkettete Liste handelt impliziert das alle Strukturen dieser Liste dynamisch zur Laufzeit von der Funktion getaddrinfo allokiert werden. Deshalb muss ein erfolgreicher Aufruf dieser Funktion mit einem abschließenden Ruf nach **freeaddrinfo** enden um den allokierten Speicher wieder freizugeben.
[code xml:lang="cpp"]void freeaddrinfo(struct addrinfo* ai);[/code]

Ein Rückgabetyp ist sinngemäß beim Freigeben von Speicher nicht erforderlich.

Das Programm

Die Funktion getaddrinfo liefert im Gegensatz zu freeaddrinfo sehr wohl einen Wert zurück. So wird 0 zurückgegeben falls sich kein Fehler ereignet hat. Eine spezifischer Fehlercode kann mit dem Aufruf von WSAGetLastError() abgerufen werden. Als Beispiel habe ich in der Hilfsfunktion *PrintWSAErrorCodes* eine switch Anweisung implementiert die die Details zu einigen möglichen Fehlertypen ausgibt, welche sich bei Aufruf diverser Winsock Funktionen ereignen können.

Das Programm verwendet die Funktion PrintHostInfo um die Host Informationen aufzulisten. Als Parameter werden jeweils der Hostname und der Port übernommen. Anschließend führt die Funktion getaddrinfo mit diesen Informationen einen DNS Aufruf aus und speichert das Resultat in der Variable aiList. Die Funktion selbst wird ein zweites Mal verwendet um alle lokalen IP-Adressen des lokalen Hosts aufzulisten dessen Name zuvor mithilfe der Funktion gethostname ermittelt wurde. Anhand der Tatsache das wir durch die verkettete Liste iterieren kann man erkennen das es sich bei der Variable aiList um einen Zeiger handelt. Ein Member der Struktur addrinfo ist ai_next mit dessen Hilfe es möglich ist durch die verkettete Liste zu navigieren. Vor der Ausgabe muss noch eine Konvertierung in ein entsprechend lesbares Format vorgenommen werden, da das Feld ai_addr einfach eine hexadezimale Zahl zurückgibt die in eine uns bekannte Formatierung für IPv4-Adressen konvertiert werden muss. Diese Funktion erfüllt inet_ntoa() die jedoch eine Struktur vom Typ *in_addr* entgegennimmt. Dies erfordert einen Cast in den entsprechend korrekten Typ.

Zusätzlich nutzen wir zu Demonstrationszwecken bei der zweiten Ausgabe der IP-Adressen eine STL-Liste. Der überladene ostream Operator