

WINSOCK 2

WINDOWS SOCKET: PROGRAM EXAMPLES PART

9

Machine's OS is standalone Windows XP Pro with SP2 except whenever mentioned. Compiler used was Visual C++ 2003 .Net 1.1. Beware the codes that span more than one line. Program examples have been tested for Non Destructive Test. All information compiled for Windows 2000 (NT5.0) above and...

1. The story was discussed at [Winsock introduction story](#).
2. Related functions, structures and macros used in the program examples have been dumped at [Winsock structure & function 1](#) and [Winsock structure & function 2](#).
3. Other related and required information (if any) not available in no. 2 can be found at [MSDN & Visual C++ online reference](#).

Abilities

- Able to understand Winsock implementation and operations through the APIs and program examples.
- Able to gather, understand and use the Winsock [functions](#), [structures](#) and [macros](#) in your programs.
- Able to build programs that use Microsoft C/Standard C programming language and Winsock APIs.

getnameinfo()

Item	Description
Function	getnameinfo()
Use	Provides name resolution from an address to the host name.
Prototype	<pre>int getnameinfo(const struct sockaddr* sa, socklen_t salen, char* host, DWORD hostlen, char* serv, DWORD servlen, int flags);</pre>
Parameters	<p>sa - [in] Pointer to a socket address structure containing the address and port number of the socket. For IPv4, the sa parameter points to a sockaddr_in structure; for IPv6, the sa parameter points to a sockaddr_in6 structure.</p> <p>salen - [in] Length of the structure pointed to in the sa parameter, in bytes.</p> <p>host - [out] Pointer to the host name. The host name is returned as a Fully Qualified Domain Name (FQDN) by default.</p>

	<p>hostlen - [in] Length of the buffer pointed to by the host parameter, in bytes. The caller must provide a buffer large enough to hold the host name, including terminating NULL characters. A value of zero indicates the caller does not want to receive the string provided in host.</p> <p>serv - [out] Pointer to the service name associated with the port number.</p> <p>servlen - [in] Length of the buffer pointed to by the serv parameter, in bytes. The caller must provide a buffer large enough to hold the service name, including terminating null characters. A value of zero indicates the caller does not want to receive the string provided in serv.</p> <p>flags - [in] Flags used to customize processing of the getnameinfo() function.</p>
Return value	Success returns zero. Any nonzero return value indicates failure. Use the WSAGetLastError() function to retrieve error information.
Include file	<winsock2.h >, <ws2tcpip.h>
Library	-
Remark	Declared in ws2tcpip.h; include <wsapi.h> for Windows 95/98/Me, Windows 2000 and Windows NT.

Table 1

Remarks

To simplify determining buffer requirements for the host and serv parameters, the following values for maximum host name length and maximum service name are defined in the ws2tcpip.h header file:

```
#define NI_MAXHOST 1025
#define NI_MAXSERV 32
```

The flags parameter can be used to customize processing of the getnameinfo function. The following flags are available:

- NI_NOFQDN
- NI_NUMERICHOST
- NI_NAMEREQD
- NI_NUMERICSERV
- NI_DGRAM

When the NI_NAMEREQD flag is set, host names that cannot be resolved by Domain Name System (DNS) result in an error. Setting the NI_NOFQDN flag results in local hosts having only their Relative Distinguished Name (RDN) returned in the host parameter. Setting the NI_NUMERICHOST flag returns the numeric form of the host name instead of its name. The numeric form of the host name is also returned if the host name cannot be resolved by DNS.

Setting the NI_NUMERICSERV flag returns the port number of the service instead of its name. If NI_NUMERICSERV is not specified and the port number contained in sa does not resolve to a well known service, the getnameinfo() function fails. When NI_NUMERICSERV is specified, the port number is returned as a numeric string.

Setting the NI_DGRAM flag indicates that the service is a datagram service. This flag is necessary for the few services that provide different port numbers for UDP and TCP service. The capability to perform reverse lookups using the getnameinfo() function is convenient, but such lookups are considered inherently unreliable, and should be used only as a hint. Macros in the Winsock header file define the mixed-case function name GetNameInfo() to getnameinfo(); either spelling is acceptable.

Program Example

The following example demonstrates the use of the getnameinfo() function.

```
// Run on Windows Xp Pro, Visual C++ .Net 1.1
#include <stdio.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#define NI_MAXHOST 1025
#define NI_MAXSERV 32

int main()
{
    // Declare variables
    WSADATA wsaData;

    // Initialize Winsock, request the Winsock 2.2
    int iResult = WSASStartup(MAKEWORD(2, 2), &wsaData);
    if(iResult != NO_ERROR)
        printf("Error at WSASStartup().\n");
    else
        printf("WSASStartup() is OK.\n");

    // Declare and initialize variables
    hostent* remoteHost;
    char* ip;
    struct sockaddr_in saGNI;
    char *hostName;
    char servInfo[256];
    int retVal;
    u_short port;
    port = 55555;

    // Input name of host
    printf("Input name of host: ");
```

```

hostName = (char*) malloc(sizeof(char*)*128);
fgets(hostName, 128, stdin);

printf("\nUsing gethostbyname()...\n");
hostName[strlen(hostName)-1] = '\0';
remoteHost = gethostbyname(hostName);
ip = inet_ntoa(*(struct in_addr *)*remoteHost->h_addr_list);
printf("IP address is: %s.\n", ip);
printf("Address type: %i.\n\n", remoteHost->h_addrtype);

// Set up sockaddr_in structure which is passed
// to the getnameinfo() function
saGNI.sin_family = AF_INET;
// saGNI.sin_addr.s_addr = inet_addr(ip);
saGNI.sin_addr.s_addr = inet_addr(ip);
saGNI.sin_port = htons(port);

// Call getnameinfo
if ((retVal = getnameinfo((SOCKADDR *)&saGNI,
    sizeof(sockaddr),
    hostName,
    256,
    servInfo,
    256,
    NI_NUMERICSERV)) != 0)
{
    printf("getnameinfo() failed.\n");
    printf("Error #: %ld.\n", WSAGetLastError());
}
else
    printf("getnameinfo() is OK.\n");

WSACleanup();
return 0;
}

```

```

C:\ "f:\myproject\mywinsock\Deb...
WSAStartup() is OK.
Input name of host: www.yahoo.com

Using gethostbyname()...
IP address is: 66.94.230.35.
Address type: 2.

getnameinfo() is OK.
Press any key to continue

```

Figure 1

getaddrinfo()

Item	Description
Function	getaddrinfo()
Use	Provides protocol-independent translation from host name to address.
Prototype	<pre>int getaddrinfo(const char* nodename, const char* servname, const struct addrinfo* hints, struct addrinfo** res);</pre>
Parameters	<p>nodename - [in] Pointer to a null-terminated string containing a host (node) name or a numeric host address string. The numeric host address string is a dotted-decimal IPv4 address or an IPv6 hex address.</p> <p>servname - [in] Pointer to a null-terminated string containing either a service name or port number.</p> <p>hints - [in] Pointer to an addrinfo structure that provides hints about the type of socket the caller supports.</p> <p>res - [out] Pointer to a linked list of one or more addrinfo structures containing response information about the host.</p>
Return value	see below
Include file	<winsock2.h>, <ws2tcpip.h>
Library	ws2_32.lib.
Remark	Include <wsapi.h> for Windows 95/98/Me, Windows 2000 and Windows NT.

Table 2

Return Values

Success returns zero. Failure returns a nonzero Windows Sockets error code, as found in the Windows Sockets Error Codes. Nonzero error codes returned by the `getaddrinfo()` function also map to the set of errors outlined by IETF recommendations. The following table shows these error codes and their WSA* equivalents. It is recommended that the WSA* error codes be used, as they offer familiar and comprehensive error information for Winsock programmers.

Error value	WSA* equivalent	Description
EAI_AGAIN	WSATRY_AGAIN	Temporary failure in name resolution.
EAI_BADFLAGS	WSAEINVAL	Invalid value for <code>ai_flags</code> .
EAI_FAIL	WSANO_RECOVERY	Non-recoverable failure in name resolution.

EAI_FAMILY	WSAEAFNOSUPPORT	The ai_family member is not supported.
EAI_MEMORY	WSA_NOT_ENOUGH_MEMORY	Memory allocation failure.
EAI_NODATA	WSANO_DATA	No address associated with nodename.
EAI_NONAME	WSAHOST_NOT_FOUND	Neither nodename nor servname provided, or not known.
EAI_SERVICE	WSATYPE_NOT_FOUND	The servname parameter is not supported for ai_socktype.
EAI_SOCKTYPE	WSAESOCKTNOSUPPORT	The ai_socktype member is not supported.

Table 3

You can use the `gai_strerror()` function to print error messages based on the `EAI_*` codes returned by the `getaddrinfo()` function. The `gai_strerror()` function is provided for compliance with [IETF](#) recommendations, but it is not thread safe. Therefore, use of traditional Windows Sockets functions such as `WSAGetLastError()` is recommended.

Remarks

One or both of the `nodename` or `servname` parameters must point to a NULL-terminated string; generally both are provided. Upon success, a linked list of `addrinfo` structures is returned in the `res` parameter; the list can be processed by following the pointer provided in the `ai_next` member of each returned `addrinfo` structure until a null pointer is encountered. In each returned `addrinfo` structure, the `ai_family`, `ai_socktype`, and `ai_protocol` members correspond to respective

arguments in a `socket()` function call. Also, the `ai_addr` member in each returned `addrinfo` structure points to a filled-in socket address structure, the length of which is specified in its `ai_addrlen` member.

If `nodename` is a machine name, machine permanent addresses are returned. If `nodename` contains the string `"..localmachine"`, all registered addresses are returned. If `nodename` refers to a cluster virtual server name, only virtual server addresses are returned. Callers of the `getaddrinfo()` function can provide hints about the type of socket supported through an `addrinfo` structure pointed to by the `hints` parameter. When the `hints` parameter is used, the following rules apply to its associated `addrinfo` structure:

- A value of `PF_UNSPEC` for `ai_family` indicates the caller will accept any protocol family.
- A value of zero for `ai_socktype` indicates the caller will accept any socket type.
- A value of zero for `ai_protocol` indicates the caller will accept any protocol.
- `ai_addrlen` must be zero or a null pointer.
- `ai_canonname` must be zero or a null pointer.
- `ai_addr` must be zero or a null pointer.
- `ai_next` must be zero or a null pointer.

Other values in the `addrinfo` structure provided in the `hints` parameter indicate specific requirements. For example, if the caller handles only TCP and does not handle UDP, the `ai_socktype` member should be set to `SOCK_STREAM`. For another example, if the caller handles only IPv4 and does not handle IPv6, the `ai_family` member should be set to `PF_INET`. If the `hints` parameter is a null pointer, the `getaddrinfo()` function treats it as if the `addrinfo` structure in `hints` were initialized with its `ai_family` member set to `PF_UNSPEC` and all other members set to zero. Macros in the Winsock header file define the mixed-case function name `GetAddrInfo()` to `getaddrinfo()`; either spelling is acceptable.

Program Example

The following example demonstrates the use of the `getaddrinfo()` function.

```
// Run on Windows Xp Pro, Visual C++ .Net 1.1
#include <stdio.h>
#include <winsock2.h>
#include <ws2tcpip.h>

int main()
{
    // Declare variables
    WSADATA wsaData;

    // Initialize Winsock, request the Winsock 2.2
```

```

int iResult = WSASStartup(MAKEWORD(2, 2), &wsaData);
if(iResult != NO_ERROR)
    printf("Error at WSASStartup().\n");
else
    printf("WSASStartup() is OK.\n");

// Declare and initialize variables.
char* ip = "127.0.0.1";
char* port = "55555";
struct addrinfo aiHints;
struct addrinfo *aiList = NULL;
int retVal;

// Setup the hints address info structure
// which is passed to the getaddrinfo() function
memset(&aiHints, 0, sizeof(aiHints));
aiHints.ai_family = AF_INET;
aiHints.ai_socktype = SOCK_STREAM;
aiHints.ai_protocol = IPPROTO_TCP;

// Call getaddrinfo(). If the call succeeds,
// the aiList variable will hold a linked list
// of addrinfo structures containing response
// information about the host
if ((retVal = getaddrinfo(ip, port, &aiHints, &aiList)) != 0)
{
    printf("getaddrinfo() failed.\n");
}
else
    printf("getaddrinfo() is OK.\n");

freeaddrinfo(aiList);
WSACleanup();
return 0;
}

```

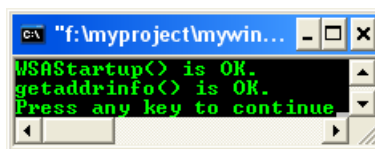


Figure 2

Ensure that the development environment targets the newest version of ws2tcpip.h which includes structure and function definitions for `addrinfo()` and `getaddrinfo()`, respectively.

Use of `ai_flags` in the hints parameter

Flags in the `ai_flags` member of the optional `addrinfo` structure provided in the `hints` parameter are as follows:

- `AI_PASSIVE`
- `AI_CANONNAME`
- `AI_NUMERICHOST`

Setting the `AI_PASSIVE` flag indicates the caller intends to use the returned socket address structure in a call to the `bind()` function. When the `AI_PASSIVE` flag is set and `nodename` is a null pointer, the IP address portion of the socket address structure is set to `INADDR_ANY` for IPv4 addresses and `IN6ADDR_ANY_INIT` for IPv6 addresses.

When the `AI_PASSIVE` flag is not set, the returned socket address structure is ready for a call to the `connect()` function for a connection-oriented protocol, or ready for a call to either the `connect()`, `sendto()`, or `send()` functions for a connectionless protocol. If the `nodename` parameter is a **null** pointer in this case, the IP address portion of the socket address structure is set to the loopback address. If neither `AI_CANONNAME` nor `AI_NUMERICHOST` is used, the `getaddrinfo()` function attempts resolution. If a literal string is passed `getaddrinfo()` attempts to convert the string, and if a host name is passed the `getaddrinfo()` function attempts to resolve it.

When the `AI_CANONNAME` bit is set and the `getaddrinfo()` function returns success, the `ai_canonname` member in the `hints` parameter points to a NULL-terminated string that contains the canonical name of the specified node. The `getaddrinfo()` function can return success when the `AI_CANONNAME` flag is set, yet the `ai_canonname` member in the associated `addrinfo` structure is null. Therefore, the recommended use of the `AI_CANONNAME` flag includes testing whether the `ai_canonname` member in the associated `addrinfo` structure is null. When the `AI_NUMERICHOST` bit is set, the `nodename` parameter must contain a non-NULL numeric host address string, otherwise the `EAI_NONAME` error is returned. This prevents a name resolution service from being called; all information returned by the `getaddrinfo()` function is dynamically allocated.

Freeing Address Information from Dynamic Allocation

All information returned by the `getaddrinfo()` function is dynamically allocated, including all `addrinfo` structures, socket address structures, and canonical host name strings pointed to by `addrinfo` structures. To return that information to the system, call the `freeaddrinfo()` function.