Network byte order and host byte order

Ports and addresses are always specified in calls to the socket functions using the network byte order convention. This convention is a method of sorting bytes that is independent of specific machine architectures. Host byte order, on the other hand, sorts bytes in the manner which is most natural to the host software and hardware. There are two common host byte order methods:

- **Little-endian** byte ordering places the least significant byte first. This method is used in Intel microprocessors, for example.
- **Big-endian** byte ordering places the most significant byte first. This method is used in IBM® z/Architecture® and S/390® mainframes and Motorola microprocessors, for example.

The network byte order is defined to always be big-endian, which may differ from the host byte order on a particular machine. Using network byte ordering for data exchanged between hosts allows hosts using different architectures to exchange address information without confusion because of byte ordering. The following C functions allow the application program to switch numbers easily back and forth between the host byte order and network byte order without having to first know what method is used for the host byte order:

- htonl() translates an unsigned long integer into network byte order.
- htons() translates an unsigned short integer into network byte order.
- ntohl() translates an unsigned long integer into host byte order.
- ntohs() translates an unsigned short integer into host byte order.

See Figure 2, Figure 4, and Figure 5 for examples of using the htons() call to put port numbers into network byte order.

The C functions inet_ntop() and inet_pton() are used to manipulate IPv6 addresses. For more information on these functions, see XL C/C++ for z/VM: Runtime Library Reference.