Document Type: Tutorial
NI Supported: Yes
Publish Date: Sep 06, 2006

Overview of NI-CAN API

NI-CAN Design Philosophy

The NI-CAN Application Programming Interface (API), like most National Instruments APIs, is largely independent of operating system and programming language. You can use NI-CAN in a wide variety of programming environments, including LabVIEW and C programming environments such as LabWindows®/CVI. Applications written for NI-CAN are also portable across different operating systems, such as Windows 95 and Windows NT.

In addition to being independent of operating system and programming language, NI-CAN is designed to be largely independent of a specific device network protocol. Device network independence means that where possible, terminology specific to CAN alone is avoided so that the API can be expanded later to support higher level protocols based on CAN. Examples of such protocols include DeviceNet, Smart Distributed System (SDS), and CANopen. Device network independence largely applies to terminology such as function names, and in no way limits access to the CAN network. For example, the function provided to read data from a CAN frame is called ncRead, as opposed to a name specific to CAN, such as ncReadCanFrame.

Furthermore, NI-CAN often uses object-oriented terminology and concepts. Object-oriented terminology provides an excellent model for describing device networks in terms that are easy to understand.

In object-oriented terminology, the term class describes a classification of an object, and the term instance refers to a specific instance of that object. The term object is generally used as a synonym for instance. For example, NI-CAN defines a class called the CAN Network Interface Object, which encapsulates any network interface port on a National Instruments CAN hardware product. Specific instances of the CAN Network Interface Object are referenced with names like CAN0 and CAN1. Each instance of a particular class has attributes that define its externally visible qualities, as well as methods that are used to perform actions. For example, each instance of the CAN Network Interface Object has an attribute for the baud rate (bits per second) used for communication, as well as a method used to transmit CAN frames onto the network.

For more information on object-oriented and CAN terminology, refer to the Glossary.

NI-CAN Object Hierarchy

The basic model of the NI-CAN software architecture is a hierarchical collection of objects (instances), each of which has attributes and methods. The hierarchy shows relationships between various objects. In general, a given object in the hierarchy has an "is used to access" relationship with all objects above it in the hierarchy.

As an example, consider a CAN device network in which the network interface of a host computer is physically connected to two devices, a pushbutton and an LED, as shown in Figure 1-3.

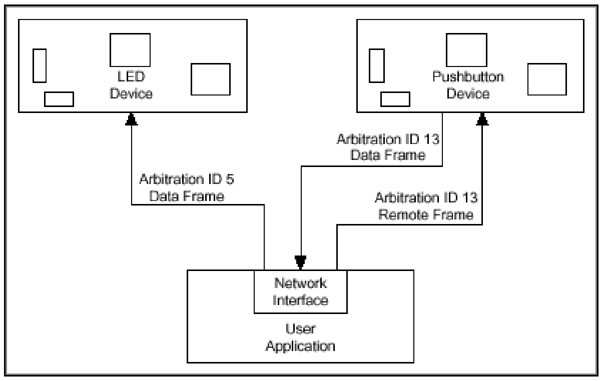
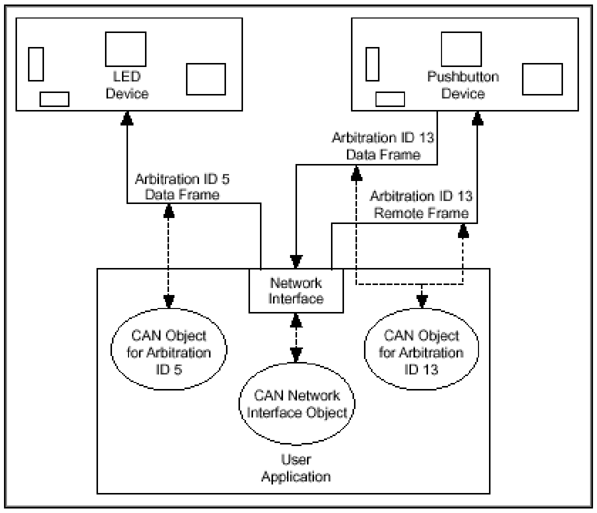

LED Device
Pushbutton Device
Arbitration ID 13 Data Frame
Arbitration ID 5 Data Frame
Arbitration ID 13 Remote Frame
Network Interface
User Application

Figure 1-3. Simple CAN Device Network Application

The pushbutton device transmits the state of the button in a CAN data frame with standard arbitration ID 13. The frame data consists of a single byte—zero if the button is off, one if the button is on. In order for an NI-CAN application to obtain the current state of the pushbutton, it transmits a CAN remote frame with standard arbitration ID 13. The pushbutton device responds to this remote transmission request by transmitting the button state in its CAN data frame.

The LED device expects to obtain the state of the LED from a CAN data frame with standard arbitration ID 5. It expects the frame data to consist of a single byte -- zero to turn the light off, one to turn the light on.

Figure 1-4 shows how NI-CAN objects encapsulate access to this CAN device network. The ovals in Figure 1-4 indicate NI-CAN objects, and the dotted lines indicate what each object encapsulates.



LED Device
Pushbutton Device
Arbitration ID 13 Data Frame
Arbitration ID 5 Data Frame
Arbitration ID 13 Remote Frame
Network Interface
CAN Object for Arbitration ID 5
CAN Object for Arbitration ID 13
CAN Network Interface Object
User Application

Figure 1-4. Applying NI-CAN Objects to the Example in Figure 1-1

The CAN Network Interface Object encapsulates the entire CAN network interface. Its attributes are used to configure settings that apply to the network interface as a whole. For example, the CAN Network Interface Object contains an attribute you can use to set the baud rate that the network interface hardware uses for communication. You can also use the CAN Network Interface Object to communicate on the CAN device network. For example, you can use the write function to transmit a CAN remote frame to the pushbutton device, then use the read function to retrieve the resulting CAN data frame. Because the CAN Network Interface Object provides direct access to the network interface, the write and read functions require all information about the CAN frame to be specified, including arbitration ID, whether the frame is a CAN data frame or a CAN remote frame, the number of data bytes, and the frame data (assuming a CAN data frame).

The CAN Object encapsulates a specific arbitration ID, along with its data. In addition to providing the ability to transmit and receive frames for a specific arbitration ID, CAN Objects also provide various forms of background access. For example, you can configure a CAN Object for arbitration ID 13 (the pushbutton) to automatically transmit a CAN remote frame every 500 ms, and to store the data of the resulting CAN data frame for later retrieval. After the CAN Object is configured in this manner, you can use the read function to obtain a single data byte that holds the most recent state of the pushbutton.