Embedded Systems KOREA

CANLIB 사용 설명서

㈜ 임베디드시스템코리아

예제 ("Hello, CAN!")

예제 1. CAN 메시지를 전송하는 간단한 프로그램

```
#include <stdlib.h>
#include <canlib.h>

void main(void)
    {
        canHandle h;

        canInitializeLibrary();
        h = canOpenChannel(0, canWANT_EXCLUSIVE);
        if (h != canOK) {
            char msg[64];
            canGetErrorText(h, msg, sizeof(msg));
            fprintf(stderr,    canOpenChannel failed (%s)₩n   , msg);
            exit(1);
    }
    canSetBusParams(h, BAUD_250K, 0, 0, 0, 0, 0);
    canSetOutputControl(h, canDRIVER_NORMAL);
    canBusOn(h);
    canWrite(h, 123,    HELLO!   , 6, 0);
    canWriteSync(h, 500);
    canBusOff(h);
    canClose(h);
}
```

위 내용은 무엇을 한 것입니까?

1. canInitializeLibrary 를 호출하여 CANLIB 라이브러리를 초기화시킵니다.
2. CAN 회로의 한 채널이 open 되었습니다. 여기서 우리는 CAN 인터페이스에서 첫 번째 채널이 되어야 하는 채널 0 을 open 합니다. canWANT_EXCLUSIVE 는 우리가 이 채널을 다른 현재 실행 프로그램과 공유하기를 원하지 않는다는 것을 뜻합니다.
3. 사전정의 버스 파라매터 세트를 사용하여, CAN bus 비트율은 250 kBit/s 로 설정됩니다.
4. CAN 버스 드라이버 유형이 설정됩니다.
5. CAN 칩이 활성화됩니다.
6. (11-bit) CAN id = 123, length 6 과 contents (decimal) 72, 69, 76, 76, 79, 31 를 가진 메시지가 전송 됩니다.

7. 메시지가 전송될 때 까지 또는 적어도 500 ms 를 기다립니다.
8. CAN 칩을 불활성화합니다.
9. 채널을 close 합니다.

위 내용에 들어있지 않은 것은 무엇입니까?

거의 모든 오류 검사가 생략되어 있습니다.

개요

CANLIB API 는 전적으로 함수들로 구성됩니다. 함수들의 대부분은 status 코드 유형, canStatus 를 회신하며, 이것은 호출이 실패한 경우 음수, 호출이 성공한 경우 canOK(zero)가 됩니다.
다양한 함수들이 포인터 인자를 수용합니다. 한 인자가 한 포인터인 대부분의 경우들에서, NULL 이 통과될 수 있습니다.

라이브러리에 대한 첫 번째 호출은 **canInitializeLibrary** 호출이 되어야 합니다. 이 함수는 라이브러리를 초기화하는 것입니다. 그 다음 호출은 **canOpenChannel** 호출이 되기 쉬운데, 이것은 특정 CAN 회로에 대한 처리를 회신합니다. 그런 후 이 처리는 라이브러리의 모든 연이은 호출들에 사용됩니다.
canOpenChannel 은 여러 가지 서로 다른 오류 코드를 보낼 수 있는데, 그 중 한 가지는 canERR_NOTFOUND 입니다.
이것은 첫 번째 파라매터에서 지정된 채널이 발견되지 않았다는 것을 뜻합니다. 이것은 흔히 하드웨어 또는 설치 문제에 기인합니다. 이 문제가 발생한다면 troubleshooting guide 를 참고하십시오.
그 다음은, 적절한 버스 파라매터가 설정되어야 합니다. 이것은 **canSetBusParams** 호출로 이루어집니다.
CANLIB 는 대부분의 공통 버스 속도 (예, 1M, 500k, 250k and 125k)에 관한 여러 가지 기본 파라매터
설정들을 제공합니다. 이것이 이루어지면, 우리는 안전하게 **canBusOn** 을 호출하여 버스로 갈 수 있습니다. 사용자는 **canWrite** 를 호출하여 메시지를 전송합니다.
수신된 메시지는 CANLIB 에 의해 버퍼되며, **canRead** 또는 이와 관련된 호출로 읽혀질 수 있습니다.
윈도우 프로그래머들은 CAN 메시지가 도착했을 때 통보받기를 원할 수 있는데, 이것은 **canSetNotify** 를 호출하여 이루어집니다. 사용자는 **canBusOff** 를 호출하여 일시적으로 회로를 버스 밖으로 가져갈 수 있습니다. 최종적인 회로 정돈은 **canClose** 를 호출하여 실행합니다.

라이브러리 초기화

canInitializeLibrary 를 호출하여 CANLIB 라이브러리를 초기화합니다. 이 루틴을 한 번 호출하는 것이 효율적입니다. 한 번 이상 그것을 부르는 것은 효과를 가지지 않을 것입니다.

칩과 채널들

하나의 CAN 인터페이스 카드는 종종 한 개 이상의 CAN 칩을 갖습니다. 사용자는 **canOpenChannel** 을 호출하여 하나의 처리를 하나의 특정 CNA 칩에 할당합니다. 이 루틴은 두 개의 인자들을 가져가며, 이들 중 첫 번째는 대상 채널의 번호입니다. 채널 번호설정은 사용자가 이용하는 하드웨어에 의존합니다.
예를 들어 사용자가 한 개의 LAPcan 카드를 갖고 있다면, 채널은 0 과 1 로 번호가 매겨집니다.

사용자가 한 개의 PCcan-Q 카드를 갖고 있다면, 채널들은 0 (첫 번째 Intel), 1 (두 번째 Intel), 2 (첫 번째 Philips), 3 (두 번째 Philips)으로 번호가 매겨집니다. 하드웨어 설명서를 참고하시기 바랍니다.

사용자는 **canGetChannelDat**a 를 사용하여 시스템에서 CAN 채널들을 셀 수 있습니다.
첫 번째 호출 **canGetNumberOfChannels** 시스템에서 채널들의 수를 획득합니다.
그 다음 호출 **canGetChannelData** 은 0, 1, 2, , n-1 채널 번호를 위한 것이며 여기서 n 은 **canGetNumberOfChannel**s 에 의해 회신된 수입니다.
두 개 또는 그 이상의 애플리케이션들이 동일한 CAN 컨트롤러를 공유할 수 있습니다.
예를 들어 사용자는 버스 상에서 메시지들을 전송할 하나의 애플리케이션과 버스를 단지 모니터할 또 다른 애플리케이션을 가질 수 있습니다. 이것을 원치 않는다면 (성능 또는 다른 이유로) 사용자는 칩에 **exclusive** 처리를 개시할 수 있습니다.이것은 동일한 칩에 처리를 개시할 수 있는 다른 애플리케이션이 없다는 것을 뜻합니다. 이것은 플래그 인자에 있는 canWANT_EXCLUSIVE 플래그를 **canOpenChanne**l 에 전달하여 실행합니다.

버스 파라매터

버스 파라매터들은 비트율, 샘플링 포인트의 위치 등을 포함합니다. 이것들은 대부분의 CAN 컨트롤러 데이터 쉬트에서 설명되어 있습니다.
버스 파라매터들을 설정하기 위해 **canSetBusParams** 를 사용합니다.
CANLIB 는 대부분의 공통 버스 속도를 위한 몇 가지 파라매터 기본 설정을 제공합니다; 여러 분은 **canSetBusParams** 호출에서 BAUD_xxx 상수들 (canlib.h 에서 정의된)중 어느 것이나 지정할 수 있으며 다른 파라매터들을 0 으로 설정할 수 있습니다.

다음의 BAUD_xxx 상수들이 현재 정의되어 있습니다.

BAUD_1M
BAUD_500K
BAUD_250K
BAUD_125K
BAUD_100K
BAUD_62K
BAUD_50K

예제 2. 버스 속도를 500 kbit/s 로 설정하기:

```
stat = canSetBusParams(hnd, BAUD_500K, 0, 0, 0, 0, 0);
```

비트율을 설정하는 것은 사전-정의되지 않습니다. 사용자는 스스로 적절한 버스 파라매터들을 계산해야만 합니다. 사용자가 **canSetBusParam**s 로 전달할 수 있는 서로 다른 파라매터들에 관한 상세한 설명은 온-라인 도움말을 보시기 바랍니다.

예제 3. 속도를 111111 kbit/s, 샘플링 포인트를 75%, SJW 를 2, 그리고 샘플 번호를 1 로 설정하기:

```
stat = canSetBusParams(hnd, 111111, 5, 2, 2, 1, 0);
```

여러 분이 82c200 CAN 컨트롤러의 레지스터 레이아웃을 사용하여 버스 파라매터를 지정하는 준-표준
방식을 사용하는 것이 더 편하다고 여겨진다면, **canSetBusParamsC200** 을 대신 호출할 수 있습니다.

예제 4. 버스 속도를 33.333 kbit/s 에 설정하기:

```
stat = canSetBusParamsC200(hnd, 0x5D, 0x05);
```

이것은 또한 샘플링 포인트를 데이터의 87.5%, SJW 를 2 quanta 에 설정하고 있습니다.

CAN 드라이버 모드 (CAN Driver Modes)

버스 드라이버 모드를 설정하는데 **canSetOutputControl** 을 사용합니다. 이것은 보통 드라이버의 표준 push-pull 타입을 얻기 위해 canDRIVER_NORMAL 로 설정됩니다. 어떤 컨트롤러들은 전송이 아닌 수신만이 가능한 canDRIVER_SILENT 를 지원하기도 합니다. 이것은 가령 자동 비트율 결정에는 편리할 수 있습니다.

On Bus / Off Bus

컨트롤러가 **on bus** 에 있을 때, 이것은 메시지를 수신하며 정확히 수신된 메시지에 대한 응답으로 acknowledge bits 를 전송합니다.
off bus 에 있는 컨트롤러는 버스 통신에 전혀 참여하지 않게 됩니다.
on bus 로 가려면 **canBusOn** 를 off bus 로 가려면 **canBusOff** 를 사용하십시오.

메시지 읽기

들어오는 메시지들은 드라이버 대기열에 놓여집니다. 마찬가지로 대부분의 경우 하드웨어는 메시지 버퍼링을 합니다. 사용자는 **canRead** 를 호출하여 대기열에서 첫 번째 메시지를 읽을 수 있습니다; 이용할 수 있는 메시지가 없다면 canERR_NOMSG 가 회신됩니다.
canRead 의 *flag* 파라매터는 canMSG_xxx 플래그들의 조합을 포함하며 메시지에 대한 더 많은 정보를 사용자에게 제공합니다;
예를 들어 29-bit 식별자를 가진 프레임은 canMSG_EXT 비트 세트를 가지며, remote 프레임은 canMSG_RTR 비트 세트를 갖습니다.

flag 인자는 canMSG_xxx 플래그들의 조합이며, 따라서 1 비트 이상이 설정될 수 있다는 것에 유의하시기 바랍니다.

때때로 대기열의 더 많은 원격 파트들을 들여다보고 싶을 수도 있습니다. 예를 들어, 특정 식별자를 가진 것을 기다리는 메시지가 있습니까? 사용자는 그 메시지를 읽기 위해서 **canReadSpecific** 를 호출할 수 있습니다.
특정 식별자에 맞지 않는 메시지는 대기열에 머무르게 되며 그 다음 호출시 canRead 로 되돌려집니다.

만약 사용자가 특정 식별자를 가진 메시지만 읽고 모든 다른 것들은 던져 버리고 싶다면, **canReadSpecificSki**p 을 호출하면 됩니다. 이 루틴은 해당 메시지의 앞에 있는 모든 다른 메시지들을 버리고, 특정 식별자를 가진 첫 번째 메시지를 복귀시킵니다.
사용자가 메시지가 도착할 때까지 (또는 타임아웃이 발생할 때까지) 기다려서 그것을 읽고 싶다면 ,**canReadWai**t 를 호출하십시오.

만약 사용자가 중재 메시지가 도착하는 것을 단지 기다릴 뿐, 그 메시지를 읽고 싶지 않다면 ,**canReadSyn**c 를 호출하십시오.
대기열에서 특정 식별자를 가진 최소 한 개의 메시지가 있을 때까지 기다리지만, 그것을 읽고 싶지 않다면, **canReadSyncSpecifi**c 를 호출하십시오.
사용자는 메시지를 기다리는 이러한 루틴들의 타임아웃(miliseconds 단위)을 지정할 수 있습니다.

예제 5. Input Queue Handling
위에서 언급한 루틴들의 예시로, 인풋 대기열에 도착된 다음의 식별자들을 가진 CAN 메시지의 경우를 가정해봅시다.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10
1. canRead(hnd, &id, ...) // canOK 와 id == 1 회신.
2. canRead(hnd, &id, ...) // canOK 와 id == 2 회신.
3. canReadSpecific(hnd, 7, ...) // canOK 와 id 7 을 가진 메시지 회신.
4. canReadSpecific(hnd, 7, ...) // canERR_NOMSG 회신.
5. canRead(hnd, &id, ...) // canOK 와 id == 3 회신.
6. canReadSpecificSkip(hnd, 5, ...) // canOK 와 id 5 를 가진 메시지 회신.
7. canRead(hnd, &id, ...) // canOK 와 id == 6 회신.
8. canReadSyncSpecific(hnd, 7, ..., 500) // id=7 (또는 500ms 경과)을 가진 그 다음 메시지를
// 기다리며 canOK 또는 canERR_TIMEOUT 회신.
9. canRead(hnd, &id, ...) // canOK 와 id == 8 회신.
10. canReadSpecific(hnd, 7, ...) // canReadSyncSpecific 에 대한 이전의 호출이 canOK 로
// 회신된 경우, canOK 와 id = 7 을 가진 메시지 회신,

Messages 보내기

나가는 CAN 메시지들은 전송 대기열에 저장되며 선입-선출 원칙에 따라 전송됩니다.
버스 상에 메시지를 보내기 위해서는 **canWrite** 를 사용합니다.

대기열의 메시지가 전송될 때까지 기다리려면 **canWriteSync** 를 사용하면 됩니다.

예제 6. CAN 메시지 전송하기

```
char msg[8];
...
stat = canWrite(hnd, 234, msg, 8, 0);
```

확장 CAN (CAN 2.0B) 사용

'Standard' CAN 은 0 - 2047 범위에서 11 비트 식별자를 갖습니다.
CAN 2.0B 으로도 불리는, 'Extended' CAN 은 29 비트 식별자를 갖습니다.
사용자는 **canWrite** 호출에서 사용하고 싶은 식별자의 종류를 지정합니다; 만약 사용자가 *flag* 인자에서 canMSG_EXT 플래그를 설정한다면, 메시지는 29 비트 식별자를 갖고 전송되게 됩니다.
반대로, 수신된 29 비트 식별자 메시지는 canMSG_EXT 플래그 세트를 갖습니다.

주의 : 모든 CAN 컨트롤러들이 CAN 2.0B 를 지원하지는 않습니다.
하드웨어 설명서를 검토하십시오, 대체로, 새로운 모든 CAN 보드들은 2.0B 를 지원하는 CAN 컨트롤러를 갖습니다. 만약 사용자가 예전(1999 또는 그 이전) PCcan 보드를 갖고 있다면, 살펴보시기 바랍니다.

예제 7. 29-비트 식별자를 가진 CAN 메시지 전송하기

```
char msg[8];
...
stat = canWrite(hnd, 23456, msg, 8, canMSG_EXT);
```

메시지 [Mailboxes]

시중에 있는 몇 가지 다른 CAN 드라이버 라이브러리들은 흔히 "mailboxes", "message objects", "user buffers" 등으로 불리는 기능이 있습니다.
반면 CANLIB 는 메시지 대기열에 초점을 두고 있는데, 왜냐하면 우리는 이것이 CAN 시스템에서 보다 적은 제한을 둔다고 여기기 때문입니다. 예를 들어, 동일한 식별자를 가진 CAN 메시지 순차를 사용하는 상위층 프로토콜은 mailboxes 를 사용하여 구현하는 것이 어렵습니다: 이것은 그 다음 메시지가 도착했을 때 이전 것을 overwrite 합니다.
사용자는, 어쨌든, **canReadSpecific** 을 사용하여 mailboxes 의 행동을 실험(simulate) 할 수 있습니다.
사용자는 **canReadSpecific** 에 의해 읽히지 않은 메시지들을 정리하기 위해 주기적으로 **canRead** 를 호출해야 할 것입니다.

예제 8. mailbox-style CAN 인터페이스를 시뮬레이트 하는 방법

사용자가 식별자 530, 540, 550 을 가진 CAN 메시지에 특별히 관심이 있다고 가정합시다. 그러면 사용자의 메일-처리 루프는 다음과 같은 어떤 것으로 보여질 것입니다:

```
    while (..) {
    if (canReadSpecific(h, 530, buf, ...) == canOK)    // Process msg 530

    if (canReadSpecific(h, 540, buf, ...) == canOK)    // Process msg 540

    if (canReadSpecific(h, 550, buf, ...) == canOK)    // Process msg 550

    while (canRead(h, &id, buf, ...) == canOK) {       // Handle other messages
    }
}
```

사용자는 자신의 WM_CANLIB 핸들러에서 이 메시지-처리 루틴을 호출하거나 이것을 사용자 프로그램의 메인 루틴으로 정할 수 있습니다.

Acceptance Filter 셋팅

사용자는 수신 메시지들의 수를 줄이는 필터를 설정할 수 있습니다. 보드상에서 하드웨어 필터 설정을 지원합니다. CANLIB 는 CAN 인터페이스 이것은 **canAccept** 함수로 실행됩니다. 사용자는 어떤 CAN 식별자들이 수신되는지 또는 거절되는지를 같이 결정하는 acceptance code 와 acceptance mask 를 설정합니다.

예제 9. acceptance filters 설정.

```
    stat = canAccept(hnd, 0xE7, canFILTER_SET_CODE_STD);
    stat = canAccept(hnd, 0xFF, canFILTER_SET_MASK_STD);
```

이 코드 부분은 0xE7 과 같지 않은 lower 8 bits 를 가진 11 비트 식별자를 포함하는 모든 메시지들이 거절(reject) 되게 만듭니다.

Bus 와/또는 Circuit 의 상태 얻기

CAN 컨트롤러의 오류 카운터들을 읽기 위해서는 **canReadErrorCounters** 를 사용합니다.
하나의 CAN 컨트롤러에는 이러한 두 개의 카운터들이 있습니다. (이것들은 protocol definition 이 필요로 하는 것들입니다). 모든 CAN 컨트롤러들이 오류 카운터의 액세스를 허용하는 것은 아니므로 CANLIB 는 사용자들에게 대신 'educated guess' 를 제공할 수 있다는 것을 알아두시기 바랍니다.
bus status 를 얻기 위해서는 **canReadStatus** 를 사용합니다. (error active, error passive, bus off; CAN standard 에 의해 정의된 것).

특수 정보 얻기

Use **canIoCtl** 를 사용하여 특수 handle 을 위한 다양한 정보를 얻거나, 또는 특수 기능들을 실행합니다.
더 자세한 정보는 help 파일을 보시기 바랍니다.
특정 채널의 데이터를 얻기 위해 canGetChannelData 를 사용합니다, 예를 들어 CAN 인터페이스의 하드웨어 타입.

비동기 통지 (Asynchronous Notification) 수신하기

특정 이벤트가 CANLIB 에서 발생할 때, 예를 들면 어떤 메시지가 도착할 때 또는 CAN 컨트롤러가 버스 오류로 인하여 'error passive'로 갈 때, 사용자는 비동기 통지를 수신할 수 있습니다.
canSetNotify 를 사용하여 사용자가 어떤 이벤트들에 관한 통지를 원하는지 CANLIB 에 알려 주십시오.
이벤트들의 여러 가지 유형들은 아래에 서술되어 있습니다.
지정된 유형의 이벤트가 발생할 때, WM_CANLIB 메시지가 **canSetNotify** 의 호출에서 지정된 handle 을
가진 윈도우에 붙여집니다. 모든 윈도우 메시지들처럼, 이 특정 메시지는 두 개의 파라매터, WPARAM 와 LPARAM 를 전달합니다; 아래에 기술된 이벤트 유형에 따라 그들의 내용은 다양해집니다.

Receive Event

receive event 는 CAN 메시지가 사전에 비어 있는 대기열에 도착할 때 발생합니다. 다시 말해, 대기열이 서비스를 필요로 할 때 receive event 가 있긴 하지만 사용자는 모든 수신된 CAN 메시지에 한 이벤트를 수신하지 않게 됩니다. WPARAM 파라매터는 메시지를 수신한 회로의 handle 에 설정되어집니다.
LPARAM 파라매터의 가장 하위 2 byte 들은 canEVENT_RX 에 설정되고 두 개의 most significant byte 는 0 이 됩니다.

Transmit Event

transmit event 는 CAN 메시지가 전송될 때마다 발생합니다. WPARAM 파라매터는 메시지를 전송했던 회로의 handle 에 설정되어집니다.
LPARAM 파라매터의 가장 하위 2 byte 들은 canEVENT_TX 에 설정되고 두 개의 most significant byte 는 0 이 됩니다.

Status Event

status event 는 CAN 컨트롤러의 버스 상태가 변경될 때, 예를 들어, 만약 컨트롤러가 버스 오류로 인하여 'error passive'로 갈 때, 발생합니다.

WPARAM 파라매터는 상태가 변경된 회로의 handle 에 설정되어집니다. LPARAM 파라매터의 가장 하위 2 byte 들은 canEVENT_STATUS 에 설정되고 두 개의 most significant byte 는 0 이 됩니다.

Error Event

error 프레임이 수신 되었습니다. WPARAM 매개 변수(parameter)는 error 프레임이 수신된 회로의 핸들에 설정됩니다. LPARAM 매개 변수(parameter)의 최하 2 바이트는 canEVENT_ERROR 에 설정되며, 2 개의 가장 중요한 바이트는 0 이 될 것입니다.

Event Handle 을 사용하여 비동기 통지 (Asynchronous Notification) 수신하기

사용자는 Win32 API 함수들 **WaitForSingleObject** 또는 **WaitForMultipleObject**s 로 보낼 수 있는 이벤트에 대한 처리를 할 수 있는 **canIoCtl** 를 호출할 수 있습니다.

이 이벤트는 'something' 이 발생할 때 설정되게 됩니다. .
주의: 이 handle 을 **WaitForXXX** 로 전달하는 것이 지원되는 유일한 기능입니다. 이 이벤트를 스스로 설정하거나 리셋하려고 하지 마십시오.
사용자는 또한 **canWaitForEven**t 를 호출할 수 있습니다; 사용자의 thread 는 CANLIB 안에서 sleep 되게 되고 'something'이 발생했을 때 return 되게 됩니다.
실제로 무엇이 발생했는가에 관해 이용할 수 있는 더 이상의 정보는 없습니다.

Thread safety

CANLIB 는 한가지 중요 조건이 충족되는 한 다중 스레드(multiple threads)를 가진 프로그램을 지원합니다:

한 CAN 회로에 대한 하나의 특수 handle 은 오 직 하나의 thread 에서 사용될 수 있습니다.
이것은 **canOpenChannel** 의 호출에 의해 한 스레드에서 획득된 하나의 handle 은 다른 어떤 스레드에서도 사용될 수 없다는 것을 의미합니다. 각각의 스레드는 자신의 고유한 handle 을 회로에 open 해야만 합니다.

Message Time Stamp

메시지들은 도착할 때 시간이 기록됩니다. 이 타입 스탬프 작업은 사용자의 하드웨어 플랫폼에 따라, CAN 인터페이스 또는 CANLIB 에 의해 이루어집니다. 전자의 경우, 10 microseconds (us) 단위로 **정확성**이 매우 뛰어납니다; CANLIB 이 이것을 할 때는, 정확성은 100 us 에서 10 ms 정도이며 사용자는 다소 큰 jitter(신호 불규칙으로 인한 떨림)를 경험할 수 있습니다. 이것은 윈도우가 실시간 운영 시스템이 아니기 때문입니다. 타임 스탬프의 **resolution** 은 디폴트에 의해, 1ms 입니다. 현재 시간을 읽으려면 **canReadTimer** 을 사용합니다.

오픈 채널의 handle 을 이 API call 로 넘겨야 합니다; 이 값은 그 채널의 클럭을 사용하는 현재 시간입니다. 원한다면 canIOCTL_SETTIMER_SCALE 의 함수 코드와 함께 canIoCtl 을 사용하여 타임 스탬프의 분해력을 변경할 수 있습니다. 이것은 타임 스탬프의 정확성에는 영향을 주지 않습니다.

Wakeup Frame

사용자의 하드웨어가 지원한다면 (LAPcan 과 DRVcan S 가 필요합니다) wakeup frames 을 전송할 수 있습니다. **canWrit**e 를 호출할 때 단지 canMSG_WAKEUP 플래그를 설정하면 됩니다.

Remote Request

canMSG_RTR 플래그를 **canWrit**e 로 전달하여 remote requests 를 전송할 수 있습니다. 수신된 remote frames 은 같은 플래그를 사용하여 **canRead** et al 에 의해 보고됩니다.

Error Frame

몇몇 하드웨어 플랫폼들은 오류 메시지 검출을 지원합니다. 오류 프레임이 도착한다면, 플래그 canMSG_ERROR_FRAME 이 **canRea**d 의 플래그 인자에 설정됩니다. 오류 프레임이 수신된 경우 식별자는 쓰레기가 되지만 LAPcan 의 경우 우연히도 2048 더하기 SJA1000 의 오류 코드가 됩니다.

어떤 플랫폼들은 (다시 말하면 LAPcan) 오류 프레임 전송도 지원합니다. 단지 플래그 인자에 있는 canMSG_ERROR_FRAME 플래그를 **canWrit**e 에 설정하면 됩니다.

Overload Frame

이들은 요즘 사용되지 않습니다. 일부 오래된 CAN 컨트롤러들(인텔 82526)은 어떤 경우에는 delay 프레임을 처리하도록 사용됩니다.

흥미로운 다른 프레임 유형

LAPcan + 1053/1054 type DRVcans 의 경우, 프레임이 "fault-tolerant" 모드에서 수신되면 canMSG_NERR 플래그가 설치됩니다.

Overruns

CAN 인터페이스 또는 드라이버가 버퍼 영역 외부에서 실행하거나, 또는 CAN 컨트롤러가 소통량을 따라갈 수 없을 정도로 버스 로드가 과중하다면, overload 상태가 애플리케이션에 신호되어집니다. 드라이버는 **canRead** 의 플래그 인자와 이것과 관련된 것에 canMSGERR_HW_OVERRUN 그리고/또는 canMSGERR_SW_OVERRUN 플래그를 설치하게 됩니다. 플래그들은 overrun 또는 overload 조건이 발생한 후 드라이버에서 읽혀진 최초 메시지에 설정되어집니다. 사용자 애플리케이션은 두 가지 이러한 조건들에 대해 테스트되어야 합니다. 이것을 하기 위해 사용자는 상수 canMSGERR_OVERRUN 을 사용할 수 있습니다.

가상 채널

CANLIB 는 아직 하드웨어가 인스톨되지 않았을 때, 사용자가 테스트 혹은 데모용으로 사용할 수 있는 **가상 채널들**을 지원합니다.

가상 채널을 열기 위해서는, 올바른 채널 번호 (사용자는 **canGetChannelData** 와 **canGetNumberOfChannels** 을 사용하여 컴퓨터에서 현재 이용할 수 있는 채널들을 열거시킬 수 있습니다) 와 함께 **canOpenChannel** 을 호출하고 플래그 인자에 있는 canWANT_VIRTUAL 을 **canOpenChannel** 에 명시합니다.

Note:

몇몇 디바이스 드라이버들의 경우, 가상 채널들이 일반 드라이버에 포함되어 있으며 동시에 인스톨되기도 합니다. 새로운 디바이스 드라이버들은 가상 채널들이 별도의 디바이스 드라이버로써 인스톨되며 자동적으로 하드웨어 드라이버들과 같이 이용될 수 없습니다.

사용자 코드의 컴파일과 링크하기

CANLIB 는 Windows 95, Windows 98, Windows ME, Windows NT 4.0, Windows 2000, Windows XP 그리고 Widnows Server 2003 에서 실행하는 32-비트 Win32 프로그램들을 지원합니다.

C, C++ (Borland and/or Microsoft)

- Include canlib.h.
- 중요한 라이브러리 canlib32.lib 와 함께 링크됩니다.

Borland 와 Microsoft 는 중요한 라이브러리들을 위한 서로 다른 포맷을 갖습니다 - 정확한 것을 선택해야 합니다; CANLIB SDK 는 두 가지 버전들을 포함하여 선적됩니다.

Borland Delphi, 2.0 또는 그 이상

사용자가 CANLIB API 를 직접 사용하고 싶다면 CAN 유닛을 사용하십시오.

이 유닛은 적절한 DLL, 32-비트 프로그램을 위한 canlib32.dll 을 매우 신속하게 (Windows API function LoadLibrary 에 대한 호출로) 로드합니다. 사용자를 좀더 편하게 해 줄 수 있는 VCL 컴포넌트도 있습니다; TCanChannelEx.

Microsoft Visual Basic, 5.0 또는 6.0

사용자의 프로젝트에 canlib32.bas 를 포함시키십시오. VB5 로 테스트된 한 개의 예제 프로그램이 포함되어 있습니다. 약간의 삭제-붙이기 작업으로 사용자가 프로그램 코드(.FRM 파일에 있는)를 사용할 수 있는 VB4.

그 외 언어들

CANLIB32.DLL 는 표준 윈도우 DLL 이어서 모든 entry 포인트들이 관습대로 호출하는 Windows **stdcall** 을 사용하고 있습니다. 이것은 어떤 다른 언어에서도 호출될 수 있다는 것을 뜻합니다.

표준 윈도우 DLL 로 어떻게 연결하는가에 대한 자료는 사용자의 language 매뉴얼을 참조하시기 바랍니다.

드라이버 설치

드라이버들은 일반적으로 하드웨어가 인스톨될 때 (또는 설치 프로그램이 플러그-앤-플레이가 아닌 OS 를 위해 실행될 때) 설치됩니다. 어떤 것도 수작업으로 설치할 필요는 없습니다.
우리의 여러 가지 CAN 보드들 각각의 설치 과정이 설명되어 있는 별도의 문서들을 (우리의 웹 사이트에서) 이용하실 수 있습니다. 또한 문제 해결 지침도 같이 수록되어 있습니다.

자신의 애플리케이션 사용하기

애플리케이션 프로그래머로서, 사용자는 CANLIB32.DLL 을 자신의 애플리케이션에 분배하고 싶을 수 있습니다. 이렇게 선택하면 응용 프로그램의 디렉터리에 배치해야합니다. 이러한 방식으로 사용자는 컴퓨터의 어떤 다른 애플리케이션들도 끊지 않게 됩니다. 사용자의 애플리케이션에 어떤 다른 CANLIB DLL 또는 디바이스 드라이버를 분배하려 시도하지 마십시오.
사용자가 타겟 시스템에 CANLIB32.DLL 을 인스톨하고, 디바이스 드라이버들의 나머지가 인스톨되지 않으면, CANLIB32. DLL 에 대한 사용자의 어떤 호출도 오류 코드로 응답할 것입니다.
이 상황을 숙지하고, 자신의 애플리케이션에서 추론 방식으로 오류 코드를 처리해야 합니다.

디바이스 드라이버 사전 –인스톨

애플리케이션 프로그래머로서 사용자는 필요한 디바이스 드라이버를 사용자 개입 없이 프로그램에 입각하여 인스톨해보고 싶을 수 있습니다. 불행하게도, 마이크로소프트는 윈도우 2000, XP 그리고 그 이상의 운영 체제에서는 이것을 지원하지 않습니다.

Windows 2000/XP

SDK 에서 Samples₩PreInstall 서브 디렉토리에 있는 preinstall 예제를 보십시오. 사용자는 반드시 모든 드라이버 파일들을 타겟의 하드 디스크에 있는 디렉토리에 복사한 후 .inf 파일을 복사하도록 SetupCopyOEMInf 를 호출해야 합니다.

Debug DLL

CANLIB SDK 는 몇 개의 debug DLL 들과 같이 선적됩니다; CANLIB SDK 의 ..₩sys₩debug 디렉토리에서 찾아보십시오. debug DLL 은 어느 순간 대화 상자를 띄워서 사용자의 애플리케이션 또는 부분적 인스톨 문제를 해결하도록 도와줄 것입니다.
debug DLLs 을 사용자의 애플리케이션에 배치하려고 하지 마십시오.
최종 사용자들이 그것을 인식하지 못할 것입니다.