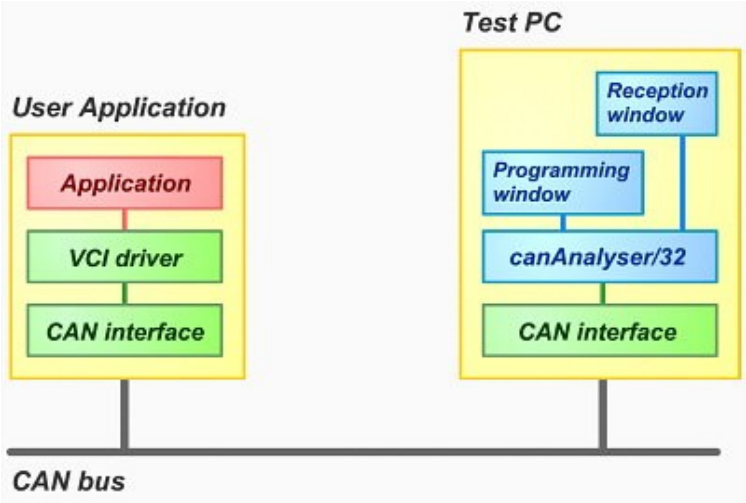
CAN Analyser의 기능 확장

상용화된 CAN Analyser가 이미 많은 경우들에서 매우 효과적으로 사용될 수 있다 할지라도, 상당히 더 높은 수준의 기능을 요구하는 작업들로 인하여, 테스트에 필요한 주요 기능을 이용할 수 없는 경우가 종종 생깁니다. 이것이 canAnalyser/32에 "Programming Client"가 있는 이유입니다.
이 특수한 canAnalyser 애플리케이션은 C와 유사한 구문과 이벤트-지향 프로그램 구조를 바탕으로 CAN node의 간단한 시뮬레이션을 가능하게 합니다. 이 경우에서의 이벤트들은, 예를 들면, 특정 CAN 메시지의 도착 ("on message"), 시간 간격의 만료 ("on timer") 또는 특정한 키의 누름 ("on key")이 될 수 있습니다.
응용 프로그래밍 클라이언트의 한 예로서, 다음에서는 프로그래밍 클라이언트가 네트워크 노드에서 CANopen SDO "domain download" 프로토콜의 구현을 테스트하는데 어떻게 사용될 수 있는지를 보여 줍니다.



위의 그림은 이것을 위한 기본 시스템 구성을 보여줍니다. 이것은 test-PC 상에서, 테스트될 애플리케이션이 실행되는 테스트될 네트워크 노드로 구성되며, 그 곳에서 원격 스테이션이 canAnalyser/32와 프로그래밍 클라이언트에 의하여 시뮬레이트됩니다. 이 개발의 목적은 IXXAT layer-2 VCI 드라이버를 기본으로, SDO 프로토콜을 구현하는 것입니다. 이 방식을 테스트하기 위해서는 프로토콜 기능, 원격 스테이션이 필요하며, 이것이 SDO 서버의 기능을 시뮬레이트합니다.
다음 예는 SDO domain 다운로드 프로토콜 (server side)의 시뮬레이션을 위한 프로그래밍 클라이언트 프로그램의 부분을 보여줍니다. 전송된 데이터는 이 예에서 생략되었습니다.
프로그램의 첫 번째 부분에서, SDO 특정 파라메터들이 우선 정의됩니다. 수신된 CAN 메시지들이 처리 함수 "on message (CLIENT_COBID, RequestMsg)"에서 처리됩니다. 이 함수는 정의된 식별자 "CLIENT_COBID"를 가진 메시지가 수신된 경우에만 실행될 뿐입니다. 수신된 메시지는 파라메터 "Request-Msg"를 통해 액세스 될 수 있습니다. 데이터 필드의 첫 번째 바이트에 포함되어 있는, 클라이언트 명령 작성자로 불리는, client command specifier (CCS)에 따라, 분기 (branching)가 배정된 특정 프로토콜 순차에 발생합니다. 그 곳에서 해당 응답 메시지들이 컴파일됩니다. 프로그램의 끝에서, 응답 메시지는 곧 SendCANObj (ResponseMsg)를 사용하여 전송됩니다.

```
// Program for simulation of a SDO download protocol server
enum
{
    CLIENT_COBID = 0x600,
    SERVER_COBID = 0x580
};

// SDO protocol parameters
enum
{
    SDO_INIT_DOM_DOWN_REQ = 0x20,
    SDO_INIT_DOM_DOWN_RSP = 0x60,
    SDO_DOWN_DOM_SEG_REQ = 0x00,
    SDO_DOWN_DOM_SEG_RSP = 0x20,
    SDO_INIT_DOM_UP_REQ = 0x40,
    SDO_INIT_DOM_UP_RSP = 0x40,
    SDO_UP_DOM_SEG_REQ = 0x60,
    SDO_UP_DOM_SEG_RSP = 0x00,
    SDO_ABORT_TRANSFER = 0x80,
    SDO_CCS_MASK = 0xE0,
    SDO_TOGGLE_BIT = 0x10,
    SDO_EXPEDITED_BIT = 0x02,
    SDO_FINISHED_BIT = 0x01
};

// Processes SDO domain download requests
on message(CLIENT_COBID, RequestMsg)
{
    CANObj ResponseMsg;
    byte ClientCommandSpecifier;

    // Initialize response message
    ResponseMsg.id = SERVER_COBID;
    ResponseMsg.len4_rtr_cyc_res2 = 8;
    ResponseMsg.data = " \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0";

// Extract CCS from request telegram
    ClientCommandSpecifier = (RequestMsg.data[0] & SDO_CCS_MASK );
```

```
switch (ClientCommandSpecifier)
{
    case SDO_INIT_DOM_DOWN_REQ:
    {
        write("InitDownloadReq₩n");
        ResponseMsg.data[0] = SDO_INIT_DOM_DOWN_RSP;
        // copy rest of information from request message
        ResponseMsg.data[1] = RequestMsg.data[1]; // Index lowbyte
        ResponseMsg.data[2] = RequestMsg.data[2]; // Index highbyte
        ResponseMsg.data[3] = RequestMsg.data[3]; // SubIndex
    }
    break;

    case SDO_DOWN_DOM_SEG_REQ:
    {
        write("DownloadSegReq₩n");
        ResponseMsg.data[0] = SDO_DOWN_DOM_SEG_RSP;
        // copy toggle flag from request telegram
        ResponseMsg.data[0] |= (RequestMsg.data[0] & 0x10);
    }
    break;

    case SDO_ABORT_TRANSFER:
    {
        write("Abort₩n");
        return; // Abort: do not send a response
    }
    break;

    default:
    {
        write("Unknown request₩n");
        return; // Unknown request: do not send a response
    }
    break;
}
SendCANObj(ResponseMsg);
```