Annex D (Programming Examples)

The following section explains step by step how to program EB200 as seen from the controller's end. The examples are written in the programming language C and may be used as a basis for control programs. They are based on Microsoft Windows Sockets. Importing to other operating systems should be possible with little effort, as all socket calls used are defined by the Berkeley Institute.

This section deals exclusively with the programming of EB200. The basics of socket programming and network engineering are not explained. For information on these topics, consult the comprehensive relevant literature, eg TCP/IP Illustrated, volume1, by W. Richard Stevens.

1. Setting up a connection

Before remote-controlling the EB200, the PPP connection must be available, as described in annex A (not required if the LAN option is used). If a TCP connection is to be set up, the IP address of the TCP server and its service number (also referred to as port number) must be known. They can be set in the EB200 menu SETUP – REMOTE. The default address for the PPP connection is 192.0.0.2 with port number 5555. When the PPP connection is set up, the host computer is assigned an IP address consisting of EB200 IP address + 1.

Example:

EB200 IP address: 192.0.0.2 -> host IP address: 192.0.0.3

The assigned IP address is only valid for the PPP channel. The IP address of a built-in network card is a different one.

If the EB200 is connected to a LAN, the IP network numbers (including sub-network numbers) of the host computer and the EB200 must be the same. Some examples:

IP host computer	Sub-network mask of host computer	IP EB200	Sub-network mask of EB200	Network class
89.10.6.53	255.0.0.0	89.17.11.23	255.0.0.0	Class A
89.10.6.53	255.255.0.0	89.10.11.23	255.255.0.0	Class A
89.10.6.53	255.255.255.0	89.10.6.23	255.255.255.0	Class A
132.2.3.4	255.255.0.0	132.2.20.21	255.255.0.0	Class B
132.2.3.4	255.255.255.0	132.2.3.21	255.255.255.0	Class B
192.3.4.1	255.255.255.0	192.3.4.2	255.255.255.0	Class C

For the EB200 to be controlled from outside the local sub-network, a suitable gateway must be made known to it. The relevant gateway IP address can be set in the SETUP – REMOTE menu (only for LAN option; for PPP, the PPP partner is always used as gateway).

The following should always be observed: **Each IP address must be unambiguous.** If an IP address is used for more than one component, the possible consequences are highly unpredictable and may bring a network down entirely. For this reason, the IP addresses are normally assigned for all components by the network administrator. The network administrator knows which IP addresses have already been assigned and how to make the remaining network settings (sub-network mask and gateway).

The following programming example describes the making of a socket connection to an EB200 unit with IP address 192.0.0.2 and port number 5555.

```
struct sockaddr_in  addr;
int err;
SOCKET nSocketID;

/* create a new socket descriptor */
nSocketID = socket(AF_INET, SOCK_STREAM, 0);
if (nSocketID != -1)
{
        /* we have got a valid socket descriptor.
        now setup a connection request to EB200 */
        memset(&addr, 0, sizeof(addr));
        addr.sin_family      = AF_INET;
        /* fill out IP-Address */
        addr.sin_addr.s_addr = inet_addr("192.0.0.2");
        addr.sin_port        = htons(5555);

        /* now do the connection */
        err = connect(nSocketID, (struct sockaddr *)&addr, sizeof(addr));
        if (!err)
        {
                /* Connection has been accepted by EB200.
                Now do some initialisations */

                /* Disable nagle algorithm to get better realtime responses */
                int i=1;
                setsockopt(nSocketID, IPPROTO_TCP, TCP_NODELAY, (char*)&i, sizeof(i));
                /* Do something with EB200 */
        }
```

The routine following the connect call serves for improving the response time. This is done by switching off the Nagle algorithm. Normally, the Nagle algorithm groups smaller data packets to a larger packet, which results in a higher data throughput. However in remote-control applications this may lead to undesirable delays, as there is usually interaction between commands and polls.

2. Initializing the unit

At first, the device should be brought into a defined status. The command *CLS deletes the status reporting system, while the command *RST loads default values for all setting parameters.

3. Transmitting device setting commands

The following example illustrates the setting of receive frequency, bandwidth and demodulation type.

```
send(nSocketID, "FREQ 98.5E6\n", 12, 0);
send(nSocketID, "BAND 150 khz\n", 13, 0);
send(nSocketID, "DEM FM\n", 7, 0);
```

4. Reading out device settings

The parameters set in the examples under 3 are read out again. This is done by sending three polling commands in an SCPI string. The response is then read in and printed out.

```
char cBuffer[100];    /* Buffer for device response */
int len;
send(nSocketID, "FREQ?;:BAND?;:DEM?\n", 19, 0);
len = recv(nSocketID, cBuffer, sizeof(cBuffer)-1, 0);
cBuffer[len] = 0;
printf("frequency;bandwidth;demodulation: %s\n", cBuffer);
```

When device responses are to be read in, it should be noted that the response packets to recv calls can be smaller than expected. In this case, the remaining data has to be read in via a renewed call of this function. The delimiter (linefeed) may be used as a criterion. The above example is extended as follows:

```
char cBuffer[100];    /* Buffer for device response */
int len;
int totallen = 0;
send(nSocketID, "FREQ?;:BAND?;:DEM?\n", 19, 0);
do
{
       len = recv(nSocketID, &cBuffer[totallen], sizeof(cBuffer)-1-totallen, 0);
       totallen += len;
} while (cBuffer[totallen-1] != ’\n’);
cBuffer[totallen] = 0;
printf("frequency;bandwidth;demodulation: %s\n", cBuffer);
```

5. Processing SRQs

SRQs serve for signalling asynchronous events (error messages, results, etc.). For this purpose, IEEE488 systems (IEC-625, IEC/IEEE bus) are equipped with a hardware line connecting the unit with the controller. EB200 simulates these activities by sending the string &SRQ<CR><LF> via the socket. This string can be sent completely asynchronously to a device response. It therefore has to be taken into account that the host computer may receive SRQ messages in the middle of a response string. Bearing this in mind, the above example can be extended as follows:

```
int bSrq = 0;
char *pSRQ;
do
{
        len = recv(nSocketID, &cBuffer[totallen], sizeof(cBuffer)-1-totallen, 0);
        totallen += len;
} while (cBuffer[totallen-1] != ’\n’);
cBuffer[totallen] = 0;
/* Look for SRQ message in string */
do
{
        pSRQ = strstr(cBuffer, "&SRQ\r\n");
        if (pSRQ != NULL)
        {
                /* SRQ message encountered */
                bSrq = 1;
                /* delete SRQ message from received string  */
                memmove(pSRQ, pSRQ+6, strlen(pSRQ)-5);
        }
} while (pSRQ != NULL);
```

Once the string has been read in including the delimiter, it is examined for SRQs. If the hardware line is simulated, only the changing of edges 0->1 is signalled, which means that SRQ messages must not be lost, as otherwise any SRQ-driven communication will be halted.

When an SRQ occurs, it will therefore be stored under Flag bSrq. This flag requires further processing. For this purpose, a serial poll (&POL) is transmitted to the device. The device response is &xyz<CR><LF>, xyz representing the value of the status byte from the status reporting system. It contains the cause of the SRQ in encoded form.

6. Program example TCP/IP

The supplied disk (Utility Disk) contains a short program for controlling the EB200 (CExample.c). It may be used by the user for generating programs of his own. The example is written in ANSI C and was tested by using Visual C 5.0. When configuring the project in Visual C the “wsock32.lib“ library must be linked up to. A further program was made with programming language JAVA . It is also contained on the disk as source code (eb200.java and eb200example.java).

Both programs initialize a search between 118 MHz and 136 MHz with a stepwidth of 25 kHz. The search is then started and the measurement results are on the screen. To make the progress of the scan visible, the change in frequency is also displayed.

7. Program example UDP

If configured accordingly, the EB200 is able to send datagrams (UDP data). See Annex F.

The supplied Utility Disk contains a short UDP program (UDPExample.exe and the associated C-sources. This program shows how to configure EB200 for the transmission of datagrams. Irrespective of the operating mode (CW, FSCAN, MSCAN, DSCAN, FASTLEVCW, LIST, IFPAN, AUDIO), the program receives datagrams from EB200, evaluates the data and continuously displays status information.

When for the call parameters in the UDPExample an audio mode ≠ 0 is selected (see also the table describing the command `SYSTem:AUDio:REMote:MODe`), the AF is transmitted via the remote-control interface in the data format selected and then reproduced via the sound card of the PC. Uninterrupted AF signal transfer via the RS232 remote-control interface is only possible at high baud rates and in audio mode 12 or 13. The LAN remote-control interface is able to handle AF signals in any data format or audio mode without any gaps. This application may be started in parallel to any remote-control application.