Protocol Specifications:

Baud Rate: 9600
Date Bits: 8
Parity: None
Stop Bits: 1

Packet Definition:

The proposed Daktronics time stamp packet provided to IDS is thirty-one (31) bytes long. It consists of a start character, data, checksum, and end character. The definition of the data packet is illustrated below:

Field	Description	Length	Justification	Type
1	Start Character – SOH (0x01)	1		Start Char
2	Game Clock Time – “MM:SS.T”	7	Right	Data
3	Game Clock Status	1		Data
4	Shot Clock Time – “SS“	2	Right	Data
5	Home Team Score	3	Right	Data
6	Guest Team Score	3	Right	Data
7	Home Team Fouls	2	Right	Data
8	Guest Team Fouls	2	Right	Data
9	Home Time Outs Left – Full	1		Data
10	Home Time Outs Left – Partial	1		Data
11	Home Time Outs Left – Total	1		Data
12	Guest Time Outs Left – Full	1		Data
13	Guest Time Outs Left – Partial	1		Data
14	Guest Time Outs Left – Total	1		Data
15	Period	1		Data
16	Checksum	2	MS, LS	Checksum
17	End Character – EOT (0x04)	1		End Char

Programming Notes:

- All data characters and the checksum are ASCII printable characters. Leading characters and fields that are blank will be filled with ASCII spaces (0x20).
- The data will be transmitted every time that any data field changes. If nothing is changing or the game clock is stopped, the data will be retransmitted approximately once a second.
- The J2 port on the back of the ProSport 6000 will be the default port.

Field 2:

The decimal point and tenths of a second value will only be transmitted for the last minute of the period. The field will be padded with ASCII spaces (0x20) for the remainder of the period.

Field 3:

An ASCII space (0x20) will be transmitted to indicate that the game clock is running. An ASCII ‘s’ (0x73) will be transmitted to indicate that the game clock is stopped.

Field 15:

Regulation and overtime periods will be transmitted with the corresponding ASCII value for the period number.

Field 16:

The checksum includes only the data bytes and is calculated by adding all of the data bytes together. A corresponding ASCII character for the value in each nibble is then transmitted to make the two (2) byte checksum. The Most Significant byte is sent first and then followed by the Least Significant. Please use the following ‘C’ function as an example for calculating the checksum.

```
void near calc_checksum(char *buf,int start,int stop,char &high,char &low)
{
        int             iIndex;
        unsigned int        uiChecksum;
        const unsigned char         ucaHexChar[17] = { "0123456789ABCDEF" };

        // calculate checksum
        for (iIndex=start,uiChecksum=0; iIndex<=stop; iIndex++)
                uiChecksum += (unsigned int)buf[iIndex];

        // AND total with FF to isolate low byte
        uiChecksum &= 0xff;

        // convert low nibble to hex
        low = ucaHexChar[uiChecksum & 0x0f];

        // convert high nibble to hex
        high = ucaHexChar[uiChecksum >> 4];

        return;
}       // end of calc_checksum()
```

<u>Examples: (Note valid checksums are not shown here)</u>

```
<SOH>12:34__s22_87_76_7_93142134CS<EOT>
<SOH>12:33__s21_87_76_7_93142134CS<EOT>
<SOH>12:32__s20_87_76_7_93142134CS<EOT>
<SOH>12:31__s19_87_76_7_93142134CS<EOT>

<SOH>12:34__s15_87_76_7_93142134CS<EOT>
<SOH>12:33__s14_87_76_7_93142134CS<EOT>
<SOH>12:32__s24_90_76_7_93142134CS<EOT>
<SOH>12:31__s23_90_76_7_93142134CS<EOT>

<SOH>12:34__s11_87_76_7_93142134CS<EOT>
<SOH>12:33__s10_87_76_7_93142134CS<EOT>
<SOH>12:32__ 24_89_76_7103142134CS<EOT>
<SOH>12:32__ 24_89_76_7103142134CS<EOT>
```