```
/* file: $RCSfile: xmsgCodec.c,v $
** rcsid: $Id: xmsgCodec.c,v 1.5 2003/06/05 22:18:23 jwp Exp $
** Copyright Jeffrey W Percival
**  *********************************************************************
** Space Astronomy Laboratory
** University of Wisconsin
** 1150 University Avenue
** Madison, WI 53706 USA
**  *********************************************************************
** Do not use this software without permission.
** Do not use this software without attribution.
** Do not remove or alter any of the lines above.
**  *********************************************************************
*/
static char *rcsid = "$Id";

/*
**  *********************************************************************
** $RCSfile: xmsgCodec.c,v $ - encode and decode numbers
** The effect of these routines matches that of XDR,
** used in Unix remote procedure calls (RPCs).
** We are interoperable with XDR.
**  *********************************************************************
*/

#include <string.h>
#include "xmsg.h"

/*---------------------------------------------------------------------*/

/* encode an unsigned 1-byte value */
int
xmsgEncodeU1(BYTE *xmsg, int ix, int xmax, int ival)
{

    if (ix < xmax) {
        ival = RANGE(ival, 0, 0xff);
        xmsg[ix++] = (BYTE)(ival & 0xff);
    }

    return(ix);
}

int
xmsgDecodeU1(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int ival = 0;

    if (ix <= xlen-1) {
        ival = xmsg[ix++];
    }

    *pival = ival;

    return(ix);
}

/*---------------------------------------------------------------------*/

/* encode an unsigned 2-byte value */
int
xmsgEncodeU2(BYTE *xmsg, int ix, int xmax, int ival)
{
```

```
    if (ix <= xmax-2) {
        ival = RANGE(ival, 0, 0xffff);
        xmsg[ix++] = (BYTE)((ival >>  8) & 0xff);
        xmsg[ix++] = (BYTE)(ival & 0xff);
    }

    return(ix);
}

int
xmsgDecodeU2(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int ival = 0;

    if (ix <= xlen-2) {
        ival = xmsg[ix++] <<  8;
        ival |= xmsg[ix++];
    }

    *pival = ival;

    return(ix);
}

/*------------------------------------------------------------------------*/

/* encode an unsigned 4-byte value */
int
xmsgEncodeU4(BYTE *xmsg, int ix, int xmax, int ival)
{

    if (ix <= xmax-4) {
        xmsg[ix++] = (BYTE)((ival >> 24) & 0xff);
        xmsg[ix++] = (BYTE)((ival >> 16) & 0xff);
        xmsg[ix++] = (BYTE)((ival >>  8) & 0xff);
        xmsg[ix++] = (BYTE)(ival & 0xff);
    }

    return(ix);
}

int
xmsgDecodeU4(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int ival = 0;

    if (ix <= xlen-4) {
        ival = xmsg[ix++] << 24;
        ival |= xmsg[ix++] << 16;
        ival |= xmsg[ix++] <<  8;
        ival |= xmsg[ix++];
    }

    *pival = ival;

    return(ix);
}

/*------------------------------------------------------------------------*/

/* encode an unsigned 4-byte value using a variable number of bytes */
int
xmsgEncodeUV(BYTE *xmsg, int ix, int xmax, int ival)
{
```

```
    int x;

    if (ix <= xmax-4) {

        /* we use up to 3 signalling bits, leaving a 29-bit limit */
        ival = RANGE(ival, 0, 0x1fffffff);

        x = ival & 0x7f;
        if (ival > 127) {
            x |= 0x80;
        }
        ix = xmsgEncodeU1(xmsg, ix, xmax, x);
        ival >>= 7;
        if (ival > 0) {
            x = ival & 0x7f;
            if (ival > 127) {
                x |= 0x80;
            }
            ix = xmsgEncodeU1(xmsg, ix, xmax, x);
            ival >>= 7;
            if (ival > 0) {
                x = ival & 0x7f;
                if (ival > 127) {
                    x |= 0x80;
                }
                ix = xmsgEncodeU1(xmsg, ix, xmax, x);
                ival >>= 7;
                if (ival > 0) {
                    ix = xmsgEncodeU1(xmsg, ix, xmax, ival);
                }
            }
        }
    }


    return(ix);
}

int
xmsgDecodeUV(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int x;
    int ival = 0;

    ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
    ival = x & 0x7f;

    if (x & 0x80) {
        ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
        ival |= (x & 0x7f) << 7;
        if (x & 0x80) {
            ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
            ival |= (x & 0x7f) << 14;
            if (x & 0x80) {
                ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
                ival |= x << 21;
            }
        }
    }

    *pival = ival;

    return(ix);
}
```

```
/*------------------------------------------------------------------------*/

/* encode a signed 1-byte value */
int
xmsgEncodeS1(BYTE *xmsg, int ix, int xmax, int ival)
{

    if (ix <= xmax-1) {
        ival = RANGE(ival, 0x80, 0x7f);
        xmsg[ix++] = (BYTE)(ival & 0xff);
    }

    return(ix);
}

int
xmsgDecodeS1(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int ival = 0;

    if (ix <= xlen-1) {
        ival = xmsg[ix++];
        if (ival & 0x80) {
            ival -= 0x100;
        }
    }

    *pival = ival;

    return(ix);
}

/*------------------------------------------------------------------------*/

/* encode a signed 2-byte value */
int
xmsgEncodeS2(BYTE *xmsg, int ix, int xmax, int ival)
{

    if (ix <= xmax-2) {
        ival = RANGE(ival, 0x8000, 0x7fff);
        xmsg[ix++] = (BYTE)((ival >>  8) & 0xff);
        xmsg[ix++] = (BYTE)(ival & 0xff);
    }

    return(ix);
}

int
xmsgDecodeS2(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int ival = 0;

    if (ix <= xlen-2) {
        ival = xmsg[ix++] <<  8;
        ival |= xmsg[ix++];
        if (ival & 0x8000) {
            ival -= 0x10000;
        }
    }

    *pival = ival;
```

```
    return(ix);
}

/*------------------------------------------------------------------------*/

/* encode a signed 4-byte value */
int
xmsgEncodeS4(BYTE *xmsg, int ix, int xmax, int ival)
{
    if (ix <= xmax-4) {
        xmsg[ix++] = (BYTE)((ival >> 24) & 0xff);
        xmsg[ix++] = (BYTE)((ival >> 16) & 0xff);
        xmsg[ix++] = (BYTE)((ival >>  8) & 0xff);
        xmsg[ix++] = (BYTE)(ival & 0xff);
    }

    return(ix);
}

int
xmsgDecodeS4(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int ival = 0;

    if (ix <= xlen-4) {
        ival = xmsg[ix++] << 24;
        ival |= xmsg[ix++] << 16;
        ival |= xmsg[ix++] <<  8;
        ival |= xmsg[ix++];
    }

    *pival = ival;

    return(ix);
}

/*------------------------------------------------------------------------*/

/* encode a signed 4-byte value using a variable number of bytes */
int
xmsgEncodeSV(BYTE *xmsg, int ix, int xmax, int ival)
{
    int x = 0;

    if (ix <= xmax-4) {

        /* we use up to 4 signalling bits, leaving a 28-bit limit */
        ival = RANGE(ival, 0x08000000, 0x07ffffff);

        if (ival < 0) {
            x |= 0x40;
            ival *= -1;
        }

        x |= ival & 0x3f;
        if (ival > 63) {
            x |= 0x80;
        }
        ix = xmsgEncodeU1(xmsg, ix, xmax, x);
        ival >>= 6;
        if (ival > 0) {
            x = ival & 0x7f;
            if (ival > 127) {
                x |= 0x80;
```

```
            }
            ix = xmsgEncodeU1(xmsg, ix, xmax, x);
            ival >>= 7;
            if (ival > 0) {
                x = ival & 0x7f;
                if (ival > 127) {
                    x |= 0x80;
                }
                ix = xmsgEncodeU1(xmsg, ix, xmax, x);
                ival >>= 7;
                if (ival > 0) {
                    ix = xmsgEncodeU1(xmsg, ix, xmax, ival);
                }
            }
        }
    }

    return(ix);
}

int
xmsgDecodeSV(int *pival, BYTE *xmsg, int ix, int xlen)
{
    int x;
    int negative = 0;
    int ival = 0;

    ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
    ival = x & 0x3f;
    if (x & 0x40) {
        negative++;
    }

    if (x & 0x80) {
        ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
        ival |= (x & 0x7f) << 6;
        if (x & 0x80) {
            ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
            ival |= (x & 0x7f) << 13;
            if (x & 0x80) {
                ix = xmsgDecodeU1(&x, xmsg, ix, xlen);
                ival |= x << 20;
            }
        }
    }

    if (negative) {
        ival *= -1;
    }

    *pival = ival;

    return(ix);
}

/*---------------------------------------------------------------------*/

int
xmsgEncodeFloat(BYTE *xmsg, int ix, int xmax, float fval)
{
    char *p;
    int i;

    /* determine endian-ness */
```

```
    i = 1;
    p = (char *)&i;
    if (*p) {
        /* little endian */
        p = (char *)&fval;
        xmsg[ix++] = *(p+3) & 0xff;
        xmsg[ix++] = *(p+2) & 0xff;
        xmsg[ix++] = *(p+1) & 0xff;
        xmsg[ix++] = *(p+0) & 0xff;
    } else {
        /* big endian */
        p = (char *)&fval;
        xmsg[ix++] = *(p+0) & 0xff;
        xmsg[ix++] = *(p+1) & 0xff;
        xmsg[ix++] = *(p+2) & 0xff;
        xmsg[ix++] = *(p+3) & 0xff;
    }

    return(ix);
}

int
xmsgDecodeFloat(float *pfval, BYTE *xmsg, int ix, int xlen)
{
    char *p;
    int i;

    /* determine endian-ness */
    i = 1;
    p = (char *)&i;
    if (*p) {
        /* little endian */
        p = (char *)pfval;
        *(p+3) = xmsg[ix++] & 0xff;
        *(p+2) = xmsg[ix++] & 0xff;
        *(p+1) = xmsg[ix++] & 0xff;
        *(p+0) = xmsg[ix++] & 0xff;
    } else {
        /* big endian */
        p = (char *)pfval;
        *(p+0) = xmsg[ix++] & 0xff;
        *(p+1) = xmsg[ix++] & 0xff;
        *(p+2) = xmsg[ix++] & 0xff;
        *(p+3) = xmsg[ix++] & 0xff;
    }

    return(ix);
}

/*-------------------------------------------------------------------------*/

int
xmsgEncodeDouble(BYTE *xmsg, int ix, int xmax, double dval)
{
    char *p;
    int i;

    ix = ((ix+3)>>2) << 2;      /* align to next 4-byte boundary */

    /* determine endian-ness */
    i = 1;
    p = (char *)&i;
    if (*p) {
        /* little endian */
```

```
        p = (char *)&dval;
        xmsg[ix++] = *(p+7) & 0xff;
        xmsg[ix++] = *(p+6) & 0xff;
        xmsg[ix++] = *(p+5) & 0xff;
        xmsg[ix++] = *(p+4) & 0xff;
        xmsg[ix++] = *(p+3) & 0xff;
        xmsg[ix++] = *(p+2) & 0xff;
        xmsg[ix++] = *(p+1) & 0xff;
        xmsg[ix++] = *(p+0) & 0xff;
    } else {
        /* big endian */
        p = (char *)&dval;
        xmsg[ix++] = *(p+0) & 0xff;
        xmsg[ix++] = *(p+1) & 0xff;
        xmsg[ix++] = *(p+2) & 0xff;
        xmsg[ix++] = *(p+3) & 0xff;
        xmsg[ix++] = *(p+4) & 0xff;
        xmsg[ix++] = *(p+5) & 0xff;
        xmsg[ix++] = *(p+6) & 0xff;
        xmsg[ix++] = *(p+7) & 0xff;
    }

    return(ix);

}

int
xmsgDecodeDouble(double *pdval, BYTE *xmsg, int ix, int xlen)
{
    char *p;
    int i;

    ix = ((ix+3)>>2) << 2;      /* align to next 4-byte boundary */

    /* determine endian-ness */
    i = 1;
    p = (char *)&i;
    if (*p) {
        /* little endian */
        p = (char *)pdval;
        *(p+7) = xmsg[ix++] & 0xff;
        *(p+6) = xmsg[ix++] & 0xff;
        *(p+5) = xmsg[ix++] & 0xff;
        *(p+4) = xmsg[ix++] & 0xff;
        *(p+3) = xmsg[ix++] & 0xff;
        *(p+2) = xmsg[ix++] & 0xff;
        *(p+1) = xmsg[ix++] & 0xff;
        *(p+0) = xmsg[ix++] & 0xff;
    } else {
        /* big endian */
        p = (char *)pdval;
        *(p+0) = xmsg[ix++] & 0xff;
        *(p+1) = xmsg[ix++] & 0xff;
        *(p+2) = xmsg[ix++] & 0xff;
        *(p+3) = xmsg[ix++] & 0xff;
        *(p+4) = xmsg[ix++] & 0xff;
        *(p+5) = xmsg[ix++] & 0xff;
        *(p+6) = xmsg[ix++] & 0xff;
        *(p+7) = xmsg[ix++] & 0xff;
    }

    return(ix);

}
```

```
/*------------------------------------------------------------------------*/

/* the XDR string format:
1. aligned to next 4-byte boundary
2. embed 4-byte count
3. embed that many characters
4. zero-fill to next 4-byte boundary
*/

int
xmsgEncodeString(BYTE *xmsg, int ix, int xmax, char *sval)
{
    int n = strlen(sval);

    ix = ((ix+3)>>2) << 2;      /* align to next 4-byte boundary */

    /* protect against overrunning */
    if (ix+n > xmax) {
        n = xmax - ix;
    }

    /* embed the count */
    ix = xmsgEncodeU4(xmsg, ix, xmax, n);

    /* copy the string data */
    (void)memcpy(xmsg+ix, sval, n);
    ix += n;

    ix = ((ix+3)>>2) << 2;      /* align to next 4-byte boundary */

    return(ix);
}

int
xmsgDecodeString(char *sval, BYTE *xmsg, int ix, int xlen)
{
    int n;      /* string length */

    ix = ((ix+3)>>2) << 2;      /* align to next 4-byte boundary */

    /* extract the count */
    ix = xmsgDecodeU4(&n, xmsg, ix, xlen);

    /* copy the string data */
    (void)memcpy(sval, xmsg+ix, n);
    ix += n;

    /* null-terminate the result */
    sval[n] = '\0';

    ix = ((ix+3)>>2) << 2;      /* align to next 4-byte boundary */

    return(ix);
}

/*------------------------------------------------------------------------*/
```