```
/*
 *  Copyright (C) 1997, 1998 Olivetti & Oracle Research Laboratory
 *
 *  This is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This software is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this software; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307,
 *  USA.
 */

/*
 * rfbproto.h - header file for the RFB protocol version 3.3
 *
 * Uses types CARD<n> for an n-bit unsigned integer, INT<n> for an n-bit signed
 * integer (for n = 8, 16 and 32).
 *
 * All multiple byte integers are in big endian (network) order (most
 * significant byte first).  Unless noted otherwise there is no special
 * alignment of protocol structures.
 *
 *
 * Once the initial handshaking is done, all messages start with a type byte,
 * (usually) followed by message-specific data.  The order of definitions in
 * this file is as follows:
 *
 *  (1) Structures used in several types of message.
 *  (2) Structures used in the initial handshaking.
 *  (3) Message types.
 *  (4) Encoding types.
 *  (5) For each message type, the form of the data following the type byte.
 *      Sometimes this is defined by a single structure but the more complex
 *      messages have to be explained by comments.
 */


/*****************************************************************************
 *
 * Structures used in several messages
 *
 *****************************************************************************/

/*-----------------------------------------------------------------------------
 * Structure used to specify a rectangle.  This structure is a multiple of 4
 * bytes so that it can be interspersed with 32-bit pixel data without
 * affecting alignment.
 */

typedef struct {
    CARD16 x;
    CARD16 y;
```

```
    CARD16 w;
    CARD16 h;
} rfbRectangle;

#define sz_rfbRectangle 8


/*-----------------------------------------------------------------------------
 * Structure used to specify pixel format.
 */

typedef struct {

    CARD8 bitsPerPixel;         /* 8,16,32 only */

    CARD8 depth;                /* 8 to 32 */

    CARD8 bigEndian;            /* True if multi-byte pixels are interpreted
                                   as big endian, or if single-bit-per-pixel
                                   has most significant bit of the byte
                                   corresponding to first (leftmost) pixel. Of
                                   course this is meaningless for 8 bits/pix */

    CARD8 trueColour;           /* If false then we need a "colour map" to
                                   convert pixels to RGB.  If true, xxxMax and
                                   xxxShift specify bits used for red, green
                                   and blue */

    /* the following fields are only meaningful if trueColour is true */

    CARD16 redMax;              /* maximum red value (= 2^n - 1 where n is the
                                   number of bits used for red). Note this
                                   value is always in big endian order. */

    CARD16 greenMax;            /* similar for green */

    CARD16 blueMax;             /* and blue */

    CARD8 redShift;             /* number of shifts needed to get the red
                                   value in a pixel to the least significant
                                   bit. To find the red value from a given
                                   pixel, do the following:
                                   1) Swap pixel value according to bigEndian
                                      (e.g. if bigEndian is false and host byte
                                      order is big endian, then swap).
                                   2) Shift right by redShift.
                                   3) AND with redMax (in host byte order).
                                   4) You now have the red value between 0 and
                                      redMax. */

    CARD8 greenShift;           /* similar for green */

    CARD8 blueShift;            /* and blue */

    CARD8 pad1;
    CARD16 pad2;

} rfbPixelFormat;
```

```
#define sz_rfbPixelFormat 16



/*****************************************************************************
 *
 * Initial handshaking messages
 *
 *****************************************************************************/

/*-----------------------------------------------------------------------------
 * Protocol Version
 *
 * The server always sends 12 bytes to start which identifies the latest RFB
 * protocol version number which it supports.  These bytes are interpreted
 * as a string of 12 ASCII characters in the format "RFB xxx.yyy\n" where
 * xxx and yyy are the major and minor version numbers (for version 3.3
 * this is "RFB 003.003\n").
 *
 * The client then replies with a similar 12-byte message giving the version
 * number of the protocol which should actually be used (which may be different
 * to that quoted by the server).
 *
 * It is intended that both clients and servers may provide some level of
 * backwards compatibility by this mechanism.  Servers in particular should
 * attempt to provide backwards compatibility, and even forwards compatibility
 * to some extent.  For example if a client demands version 3.1 of the
 * protocol, a 3.0 server can probably assume that by ignoring requests for
 * encoding types it doesn't understand, everything will still work OK.  This
 * will probably not be the case for changes in the major version number.
 *
 * The format string below can be used in sprintf or sscanf to generate or
 * decode the version string respectively.
 */

#define rfbProtocolVersionFormat "RFB %03d.%03d\n"
#define rfbProtocolMajorVersion 3
#define rfbProtocolMinorVersion 3

typedef char rfbProtocolVersionMsg[13]; /* allow extra byte for null */

#define sz_rfbProtocolVersionMsg 12


/*-----------------------------------------------------------------------------
 * Authentication
 *
 * Once the protocol version has been decided, the server then sends a 32-bit
 * word indicating whether any authentication is needed on the connection.
 * The value of this word determines the authentication scheme in use.  For
 * version 3.0 of the protocol this may have one of the following values:
 */

#define rfbConnFailed 0
#define rfbNoAuth 1
#define rfbVncAuth 2

/*
 * rfbConnFailed:       For some reason the connection failed (e.g. the server
```

```
 *                      cannot support the desired protocol version).  This is
 *                      followed by a string describing the reason (where a
 *                      string is specified as a 32-bit length followed by that
 *                      many ASCII characters).
 *
 * rfbNoAuth:           No authentication is needed.
 *
 * rfbVncAuth:          The VNC authentication scheme is to be used.  A 16-byte
 *                      challenge follows, which the client encrypts as
 *                      appropriate using the password and sends the resulting
 *                      16-byte response.  If the response is correct, the
 *                      server sends the 32-bit word rfbVncAuthOK.  If a simple
 *                      failure happens, the server sends rfbVncAuthFailed and
 *                      closes the connection. If the server decides that too
 *                      many failures have occurred, it sends rfbVncAuthTooMany
 *                      and closes the connection.  In the latter case, the
 *                      server should not allow an immediate reconnection by
 *                      the client.
 */

#define rfbVncAuthOK 0
#define rfbVncAuthFailed 1
#define rfbVncAuthTooMany 2


/*-----------------------------------------------------------------------------
 * Client Initialisation Message
 *
 * Once the client and server are sure that they're happy to talk to one
 * another, the client sends an initialisation message.  At present this
 * message only consists of a boolean indicating whether the server should try
 * to share the desktop by leaving other clients connected, or give exclusive
 * access to this client by disconnecting all other clients.
 */

typedef struct {
    CARD8 shared;
} rfbClientInitMsg;

#define sz_rfbClientInitMsg 1


/*-----------------------------------------------------------------------------
 * Server Initialisation Message
 *
 * After the client initialisation message, the server sends one of its own.
 * This tells the client the width and height of the server's framebuffer,
 * its pixel format and the name associated with the desktop.
 */

typedef struct {
    CARD16 framebufferWidth;
    CARD16 framebufferHeight;
    rfbPixelFormat format;      /* the server's preferred pixel format */
    CARD32 nameLength;
    /* followed by char name[nameLength] */
} rfbServerInitMsg;

#define sz_rfbServerInitMsg (8 + sz_rfbPixelFormat)
```

```
/*
 * Following the server initialisation message it's up to the client to send
 * whichever protocol messages it wants.  Typically it will send a
 * SetPixelFormat message and a SetEncodings message, followed by a
 * FramebufferUpdateRequest.  From then on the server will send
 * FramebufferUpdate messages in response to the client's
 * FramebufferUpdateRequest messages.  The client should send
 * FramebufferUpdateRequest messages with incremental set to true when it has
 * finished processing one FramebufferUpdate and is ready to process another.
 * With a fast client, the rate at which FramebufferUpdateRequests are sent
 * should be regulated to avoid hogging the network.
 */



/*****************************************************************************
 *
 * Message types
 *
 *****************************************************************************/

/* server -> client */

#define rfbFramebufferUpdate 0
#define rfbSetColourMapEntries 1
#define rfbBell 2
#define rfbServerCutText 3


/* client -> server */

#define rfbSetPixelFormat 0
#define rfbFixColourMapEntries 1        /* not currently supported */
#define rfbSetEncodings 2
#define rfbFramebufferUpdateRequest 3
#define rfbKeyEvent 4
#define rfbPointerEvent 5
#define rfbClientCutText 6



/*****************************************************************************
 *
 * Encoding types
 *
 *****************************************************************************/

#define rfbEncodingRaw 0
#define rfbEncodingCopyRect 1
#define rfbEncodingRRE 2
#define rfbEncodingCoRRE 4
#define rfbEncodingHextile 5


/*****************************************************************************
```

```
 *
 * Server -> client message definitions
 *
 ****************************************************************************/


/*-----------------------------------------------------------------------------
 * FramebufferUpdate - a block of rectangles to be copied to the framebuffer.
 *
 * This message consists of a header giving the number of rectangles of pixel
 * data followed by the rectangles themselves.  The header is padded so that
 * together with the type byte it is an exact multiple of 4 bytes (to help
 * with alignment of 32-bit pixels):
 */

typedef struct {
    CARD8 type;                 /* always rfbFramebufferUpdate */
    CARD8 pad;
    CARD16 nRects;
    /* followed by nRects rectangles */
} rfbFramebufferUpdateMsg;

#define sz_rfbFramebufferUpdateMsg 4

/*
 * Each rectangle of pixel data consists of a header describing the position
 * and size of the rectangle and a type word describing the encoding of the
 * pixel data, followed finally by the pixel data.  Note that if the client has
 * not sent a SetEncodings message then it will only receive raw pixel data.
 * Also note again that this structure is a multiple of 4 bytes.
 */

typedef struct {
    rfbRectangle r;
    CARD32 encoding;    /* one of the encoding types rfbEncoding... */
} rfbFramebufferUpdateRectHeader;

#define sz_rfbFramebufferUpdateRectHeader (sz_rfbRectangle + 4)


/*-----------------------------------------------------------------------------
 * Raw Encoding.  Pixels are sent in top-to-bottom scanline order,
 * left-to-right within a scanline with no padding in between.
 */


/*-----------------------------------------------------------------------------
 * CopyRect Encoding.  The pixels are specified simply by the x and y position
 * of the source rectangle.
 */

typedef struct {
    CARD16 srcX;
    CARD16 srcY;
} rfbCopyRect;

#define sz_rfbCopyRect 4
```

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * RRE - Rise-and-Run-length Encoding.  We have an rfbRREHeader structure
 * giving the number of subrectangles following.  Finally the data follows in
 * the form [<bgpixel><subrect><subrect>...] where each <subrect> is
 * [<pixel><rfbRectangle>].
 */

typedef struct {
    CARD32 nSubrects;
} rfbRREHeader;

#define sz_rfbRREHeader 4


/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * CoRRE - Compact RRE Encoding.  We have an rfbRREHeader structure giving
 * the number of subrectangles following.  Finally the data follows in the form
 * [<bgpixel><subrect><subrect>...] where each <subrect> is
 * [<pixel><rfbCoRRERectangle>].  This means that
 * the whole rectangle must be at most 255x255 pixels.
 */

typedef struct {
    CARD8 x;
    CARD8 y;
    CARD8 w;
    CARD8 h;
} rfbCoRRERectangle;

#define sz_rfbCoRRERectangle 4


/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * Hextile Encoding.  The rectangle is divided up into "tiles" of 16x16 pixels,
 * starting at the top left going in left-to-right, top-to-bottom order.  If
 * the width of the rectangle is not an exact multiple of 16 then the width of
 * the last tile in each row will be correspondingly smaller.  Similarly if the
 * height is not an exact multiple of 16 then the height of each tile in the
 * final row will also be smaller.  Each tile begins with a "subencoding" type
 * byte, which is a mask made up of a number of bits.  If the Raw bit is set
 * then the other bits are irrelevant; w*h pixel values follow (where w and h
 * are the width and height of the tile).  Otherwise the tile is encoded in a
 * similar way to RRE, except that the position and size of each subrectangle
 * can be specified in just two bytes.  The other bits in the mask are as
 * follows:
 *
 * BackgroundSpecified - if set, a pixel value follows which specifies
 *    the background colour for this tile.  The first non-raw tile in a
 *    rectangle must have this bit set.  If this bit isn't set then the
 *    background is the same as the last tile.
 *
 * ForegroundSpecified - if set, a pixel value follows which specifies
 *    the foreground colour to be used for all subrectangles in this tile.
 *    If this bit is set then the SubrectsColoured bit must be zero.
 *
 * AnySubrects - if set, a single byte follows giving the number of
 *    subrectangles following.  If not set, there are no subrectangles (i.e.
 *    the whole tile is just solid background colour).
 *
```

```
 * SubrectsColoured - if set then each subrectangle is preceded by a pixel
 *    value giving the colour of that subrectangle.  If not set, all
 *    subrectangles are the same colour, the foreground colour;  if the
 *    ForegroundSpecified bit wasn't set then the foreground is the same as
 *    the last tile.
 *
 * The position and size of each subrectangle is specified in two bytes.  The
 * Pack macros below can be used to generate the two bytes from x, y, w, h,
 * and the Extract macros can be used to extract the x, y, w, h values from
 * the two bytes.
 */

#define rfbHextileRaw                   (1 << 0)
#define rfbHextileBackgroundSpecified   (1 << 1)
#define rfbHextileForegroundSpecified   (1 << 2)
#define rfbHextileAnySubrects           (1 << 3)
#define rfbHextileSubrectsColoured      (1 << 4)

#define rfbHextilePackXY(x,y) (((x) << 4) | (y))
#define rfbHextilePackWH(w,h) ((((w)-1) << 4) | ((h)-1))
#define rfbHextileExtractX(byte) ((byte) >> 4)
#define rfbHextileExtractY(byte) ((byte) & 0xf)
#define rfbHextileExtractW(byte) (((byte) >> 4) + 1)
#define rfbHextileExtractH(byte) (((byte) & 0xf) + 1)


/*-----------------------------------------------------------------------------
 * SetColourMapEntries - these messages are only sent if the pixel
 * format uses a "colour map" (i.e. trueColour false) and the client has not
 * fixed the entire colour map using FixColourMapEntries.  In addition they
 * will only start being sent after the client has sent its first
 * FramebufferUpdateRequest.  So if the client always tells the server to use
 * trueColour then it never needs to process this type of message.
 */

typedef struct {
    CARD8 type;                 /* always rfbSetColourMapEntries */
    CARD8 pad;
    CARD16 firstColour;
    CARD16 nColours;

    /* Followed by nColours * 3 * CARD16
       r1, g1, b1, r2, g2, b2, r3, g3, b3, ..., rn, bn, gn */

} rfbSetColourMapEntriesMsg;

#define sz_rfbSetColourMapEntriesMsg 6



/*-----------------------------------------------------------------------------
 * Bell - ring a bell on the client if it has one.
 */

typedef struct {
    CARD8 type;                 /* always rfbBell */
} rfbBellMsg;

#define sz_rfbBellMsg 1
```

```
/*-----------------------------------------------------------------------------
 * ServerCutText - the server has new text in its cut buffer.
 */

typedef struct {
    CARD8 type;                 /* always rfbServerCutText */
    CARD8 pad1;
    CARD16 pad2;
    CARD32 length;
    /* followed by char text[length] */
} rfbServerCutTextMsg;

#define sz_rfbServerCutTextMsg 8


/*-----------------------------------------------------------------------------
 * Union of all server->client messages.
 */

typedef union {
    CARD8 type;
    rfbFramebufferUpdateMsg fu;
    rfbSetColourMapEntriesMsg scme;
    rfbBellMsg b;
    rfbServerCutTextMsg sct;
} rfbServerToClientMsg;



/*****************************************************************************
 *
 * Message definitions (client -> server)
 *
 *****************************************************************************/


/*-----------------------------------------------------------------------------
 * SetPixelFormat - tell the RFB server the format in which the client wants
 * pixels sent.
 */

typedef struct {
    CARD8 type;                 /* always rfbSetPixelFormat */
    CARD8 pad1;
    CARD16 pad2;
    rfbPixelFormat format;
} rfbSetPixelFormatMsg;

#define sz_rfbSetPixelFormatMsg (sz_rfbPixelFormat + 4)


/*-----------------------------------------------------------------------------
 * FixColourMapEntries - when the pixel format uses a "colour map", fix
 * read-only colour map entries.
 *
 *    ***************** NOT CURRENTLY SUPPORTED ****************
```

```
 */

typedef struct {
    CARD8 type;                 /* always rfbFixColourMapEntries */
    CARD8 pad;
    CARD16 firstColour;
    CARD16 nColours;

    /* Followed by nColours * 3 * CARD16
       r1, g1, b1, r2, g2, b2, r3, g3, b3, ..., rn, bn, gn */

} rfbFixColourMapEntriesMsg;

#define sz_rfbFixColourMapEntriesMsg 6


/*-----------------------------------------------------------------------------
 * SetEncodings - tell the RFB server which encoding types we accept.  Put them
 * in order of preference, if we have any.  We may always receive raw
 * encoding, even if we don't specify it here.
 */

typedef struct {
    CARD8 type;                 /* always rfbSetEncodings */
    CARD8 pad;
    CARD16 nEncodings;
    /* followed by nEncodings * CARD32 encoding types */
} rfbSetEncodingsMsg;

#define sz_rfbSetEncodingsMsg 4


/*-----------------------------------------------------------------------------
 * FramebufferUpdateRequest - request for a framebuffer update.  If incremental
 * is true then the client just wants the changes since the last update.  If
 * false then it wants the whole of the specified rectangle.
 */

typedef struct {
    CARD8 type;                 /* always rfbFramebufferUpdateRequest */
    CARD8 incremental;
    CARD16 x;
    CARD16 y;
    CARD16 w;
    CARD16 h;
} rfbFramebufferUpdateRequestMsg;

#define sz_rfbFramebufferUpdateRequestMsg 10


/*-----------------------------------------------------------------------------
 * KeyEvent - key press or release
 *
 * Keys are specified using the "keysym" values defined by the X Window System.
 * For most ordinary keys, the keysym is the same as the corresponding ASCII
 * value.  Other common keys are:
 *
 * BackSpace            0xff08
 * Tab                  0xff09
```

```
 * Return or Enter      0xff0d
 * Escape               0xff1b
 * Insert               0xff63
 * Delete               0xffff
 * Home                 0xff50
 * End                  0xff57
 * Page Up              0xff55
 * Page Down            0xff56
 * Left                 0xff51
 * Up                   0xff52
 * Right                0xff53
 * Down                 0xff54
 * F1                   0xffbe
 * F2                   0xffbf
 * ...                  ...
 * F12                  0xffc9
 * Shift                0xffe1
 * Control              0xffe3
 * Meta                 0xffe7
 * Alt                  0xffe9
 */

typedef struct {
    CARD8 type;                 /* always rfbKeyEvent */
    CARD8 down;                 /* true if down (press), false if up */
    CARD16 pad;
    CARD32 key;                 /* key is specified as an X keysym */
} rfbKeyEventMsg;

#define sz_rfbKeyEventMsg 8


/*-----------------------------------------------------------------------------
 * PointerEvent - mouse/pen move and/or button press.
 */

typedef struct {
    CARD8 type;                 /* always rfbPointerEvent */
    CARD8 buttonMask;           /* bits 0-7 are buttons 1-8, 0=up, 1=down */
    CARD16 x;
    CARD16 y;
} rfbPointerEventMsg;

#define rfbButton1Mask 1
#define rfbButton2Mask 2
#define rfbButton3Mask 4

#define sz_rfbPointerEventMsg 6



/*-----------------------------------------------------------------------------
 * ClientCutText - the client has new text in its cut buffer.
 */

typedef struct {
    CARD8 type;                 /* always rfbClientCutText */
    CARD8 pad1;
    CARD16 pad2;
```

```
    CARD32 length;
    /* followed by char text[length] */
} rfbClientCutTextMsg;

#define sz_rfbClientCutTextMsg 8



/*-----------------------------------------------------------------------------
 * Union of all client->server messages.
 */

typedef union {
    CARD8 type;
    rfbSetPixelFormatMsg spf;
    rfbFixColourMapEntriesMsg fcme;
    rfbSetEncodingsMsg se;
    rfbFramebufferUpdateRequestMsg fur;
    rfbKeyEventMsg ke;
    rfbPointerEventMsg pe;
    rfbClientCutTextMsg cct;
} rfbClientToServerMsg;
```