**PROTOCOLO RFB**

Tristan Richardson

RealVNC Ltd

(Antes Olivetti Research Ltd / AT&T Labs Cambridge)*

Version 3.8

Ultima actualizacion 26 Noviembre 2010

Contenido

1 Introduccion 3

2 Protocolo de visualizacion 3

3 Protocolo de ingreso 4

4 Representacion de los datos de pixel 4

5 Extensiones del protocolo 5

6 Mensajes del protocolo 5

6.1 Mensajes de negociacion 7

6.1.1 Version de protocolo 8

6.1.2 Seguridad 9

6.1.3 Resultado de la seguridad 11

6.2 Tipos de seguridad 12

6.2.1 Ninguno 13

6.2.2 Autenticacion VNC 14

6.3 Mensajes de inicializacion 15

6.3.1 ClientInit 16

6.3.2 ServerInit 17

6.4 Mensajes del cliente al servidor 19

* James Weatherall, Andy Harter y Ken Wood tambien colaboraron el el diseño del protocolo RFB.

CONTENIDO 2

6.4.1 SetPixelFormat 20

6.4.2 SetEncodings 21

6.4.3 FramebufferUpdateRequest 22

6.4.4 KeyEvent 23

6.4.5 PointerEvent 25

6.4.6 ClientCutText 26

6.5 Mensajes del servidor al cliente 27

6.5.1 FramebufferUpdate 28

6.5.2 SetColourMapEntries 29

6.5.3 Bell 30

6.5.4 ServerCutText 31

6.6 Codificaciones 32

6.6.1 Codificacion bruta 33

6.6.2 Codificacion CopyRect 34

6.6.3 Codificacion RRE 35

6.6.4 Codificacion Hextile 36

6.6.5 Codificacion ZRLE 38

6.7 Pseudo-codificaciones 41

6.7.1 Pseudo-codificaciones del cursor 42

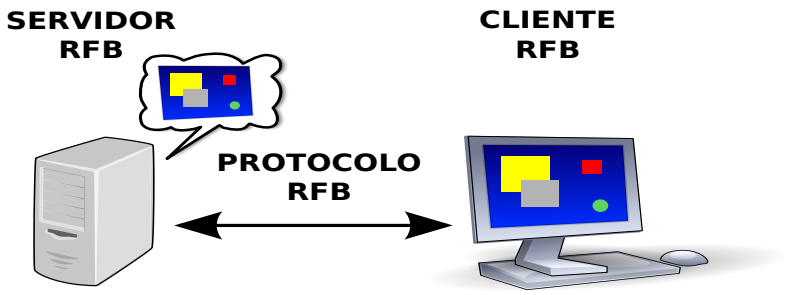
6.7.2 Pseudo-codificacion DesktopSize 43

## 1 Introduccion

RFB (“remote framebuffer”) es un protocolo simple para acceso remoto a interfaces graficas de usuario. Debido a que trabaja al nivel del framebuffer es aplicable a todos los sistemas de ventanas y aplicaciones, incluyendo X11, Windows y Macintosh. RFB es el protocolo usado en VNC (Virtual Network Computing).

El equipo remoto donde el usuario se encuentra (p.e. el monitor junto con el teclado y raton)

es llamado el cliente RFB o visualizador. El equipo donde los cambios al framebuffer se originan (p.e. el sistema de ventanas y aplicaciones) es conocido como el servidor RFB.




RFB es verdaderamente un protocolo de “cliente ligero” . El enfasis en el diseño del protocolo RFB es hacer que el cliente necesite de muy pocos requerimientos. De esta forma , los clientes pueden funcionar en un muy amplio rango de hardware, y la tarea de implementacion del cliente sea lo mas simple posible.

El protocolo tambien hace que el cliente no necesita memoria para recordar su estado. Si un cliente se desconecta de un determinado servidor, y posteriormente se reconecta al mismo servidor, el estado de la interfaz de usuario es preservado. Aun mas, se puede usar un equipo diferente para conectarse al servidor RFB. En el nuevo equipo, el usuario vera exactamente la misma interfaz grafica de usuario que el equipo original. En efecto, la interfaz hacia las aplicaciones del usuario se vuelve completamente movil. Siempre que exista algun nivel usable de conectividad, el usuario podra acceder a sus aplicaciones personales, y el estado de estas aplicaciones sera mantenido entre los diferentes accesos desde diferentes ubicaciones. Esto provee al usuario con una vista familiar y uniforme de la infraestructura de computacion a donde quiera que se va.

## 2 Protocolo de visualizacion

La parte de visualizacion del protocolo esta basada en una simple primitiva grafica: "dibujar un rectangulo relleno de pixeles en determinada posicion x,y". A primera vista esto parece una forma ineficiente de dibujar los multiples componentes de una interfaz de usuario. Sin embargo, permite diferentes niveles de codificacion de pixeles otorgando un alto grado de flexibilidad en como mediar con diversos parametros como ancho de banda, velocidad de graficacion y procesamiento del servidor.

Una secuencia de estos rectangulos hace una actualizacion de framebuffer (o simplemente una actualizacion). Una actualizacion representa un cambio de un estado valido de framebuffer a otro, de alguna forma similar a un cuadro de video. Los rectangulos generalmente no estan juntos pero no es exclusivamente el caso. El protocolo de actualizacion es basado en demanda por el cliente. Esto significa que una actualizacion solamente se envia desde el servidor al cliente como respuesta a una peticion explicita del cliente. Esto proporciona al protocolo una calidad adaptativa. Entre mas lento sea el cliente y la red, mas baja sera la velocidad de las actualizaciones. En aplicaciones tipicas, cambios en la misma area del framebuffer tienden a ocurrir una tras otra. Con un cliente lento y/o una red lenta, cambios intermedios del framebuffer pueden ser ignorados, produciendo menos trafico y menos redibujado en el cliente.

## 3 Protocolo de ingreso

La parte de ingreso del protocolo esta basada en un modelo estandar de teclado y raton con multiples botones. Eventos de ingreso son enviados simplemente al servidor desde el cliente ya sea por que el cliente pulsa una tecla o boton del raton, o cuando el raton es movido. Estos eventos de ingreso tambien pueden ser sintetizados desde otros dispositivos de entrada y salida no estandar. Por ejemplo, un dispositivo de reconocimiento de escritura manual puede generar eventos de teclado.

## 4 Representacion de los datos de pixel

La interaccion inicial entre el servidor RFB y el cliente implica una negociacion del formato y codificacion con que los datos de pixel sera enviados. Esta negociacion se ha diseñado para hacer el trabajo del cliente lo mas facil posible. La idea es que el servidor siempre debe enviar los datos de pixel en la forma en que el cliente los requiere. Sin embargo si el cliente esta en la capacidad de trabajar con diferentes formatos o codificaciones, se puede escoger la opcion que sea mas facil de generar para el servidor.

El formato de pixel se refiere a la representacion individual de colores mediante valores de pixel. Los formatos mas comunes de pixel son 24-bits o 16-bits "color verdadero", donde los campos de bits de los datos de pixel se traducen directamente en intensidades de rojo, verde y azul, y los "mapas de color" de 8 bits donde un mapeo arbitrario puede usarse par traducir los valores de pixel a intensidades RGB.

La codificacion se refiere a como un rectangulo relleno de pixeles se enviara por la red. Cada rectangulo relleno de pixeles es antecedido por una cabecera que indica la posicion X,Y del rectangulo en la pantalla, el ancho y alto del rectangulo, el tipo de codificacion que especifica la codificacion de los datos de pixels. Los datos de los pixels son enviados a continuacion usando la codificacion especificada. Los tipos de codificacion usados son Brutos (Raw), CopyRect, RRE, Hextile y ZRLE. En practica normalmente se usan solo ZRLE, Hextile y CopyRect ya que proveen la mejor compresion en computadores tipicos. Ver la seccion 6.6 para una descripcion de cada uno de los tipos de codificacion.

## 5 Extensiones del protocolo

Existen formas en las cuales el protocolo puede ser extendido:

**Nuevas codificaciones** Un nuevo tipo de codificacion puede puede ser añadido al protocolo relativamente facil manteniendo compatibilidad con los clientes y servidores existentes. Los servidores existentes simplemente ignoraran peticiones para una nueva codificacion que ellos no soporte. Los clientes existentes nunca haran peticiones de nuevas codificaciones asi que nunca veran los rectangulos codificados de esa forma.

**Pseudo codificaciones** En adicion a las codificaciones genuinas, un cliente puede hacer una peticion de "pseudo-codificacion" para declarar al servidor que soporta una determinada extension para el protocolo. Un servidor que no soporte dicha extension simplemente ignorara la pseudo-codificacion. Hay que tener en cuenta que esto significa que el cliente debe asumir que el servidor no soporta dicha extension hasta que obtenga una confirmacion especifica de la extension . Ver en la seccion 6.7 para una descripcion de las pseudo-codificaciones actuales.

**Nuevos tipos de seguridad** Añadir un nuevo tipo de seguridad otorga lo ultimo en flexibilidad al modificar el comportamiento del protocolo sin sacrificar la compatibilidad con clientes y servidores existentes. Un cliente y servidor que concuerden en un tipo nuevo de seguridad pueden efectivamente hablar cualquier otro tipo de protocolo que se quiere despues de ello - no necesariamente tiene que ser un protocolo relacionado con el RFB.

**Bajo ninguna circunstancia usted deberia usar un numero de version de protocolo diferente**. Las versiones de protocolo estan definidas por los mantenedores del protocolo RFB, RealVNC Ltd. Si usted utiliza un numero de version de protocolo diferente, entonces usted no sera RFB/VNC compatible. Para asegurar que usted siga siendo compatible con el protocolo RFB es importante que usted se contacte con RealVNC para estar seguro que sus tipos de codificacion y de seguridad no entren en conflicto. Por favor visite el sitio web de RealVNC en http://www.realvnc.com para detalles en como contactarnos; enviando un mensaje a la lista de correo de VNC sera la mejor via de estar en contacto y permitir que el resto de la comunidad VNC este al tanto.

## 6 Mensajes de protocolo

El protocolo RFB puede operar bajo cualquier transporte confiable, ya sea basado en flujo de bits u orientado a mensaje. Convencionalmente es usado sobre una conexion TCP/IP. Hay tres etapas en el protocolo. Primero es la fase de negociacion, su proposito es lograr un acuerdo comun en cuanto a version de protocolo y el tipo de seguridad a utilizar. La segunda etapa es una fase de inicializacion donde el cliente y servidor intercambian los mensajes ClientInit y ServerInit. la etapa filan es la interaccion normal del protocolo. El cliente puede enviar

La etapa final es la interaccion normal del protocolo. El cliente puede enviar cualquier mensaje que desee, y puede recibir mensajes del servidor como

resultado. Todos estos mensajes comienzan con un byte de tipo-de-mensaje, seguido por los datos especificos del mensaje.

Las siguientes descripciones de mensajes de protocolo usan los tipos basicos de datos U8, U16, U32, S8, S16, S32. Estos representan respectivamente enteros no signados de 8, 16 y 32 bits y enteros signados de 8,16 y 32 bits. Todo los enteros de multiples (diferentes a los valores de pixel ) estan en orden big endian (byte mas significativo primero).

El tipo d dato PIXEL se debe tomar como el valor de los pixeles de longitud bytesPerPixel bytes, donde 8 x bytesPerPixel es el numero de bits-por-pixel acordado entre el cliente y el servidor ya sea en el mensaje ServerInit (seccion 6.3.2) o mediante el mensaje SetPixelFormat (seccion 6.4.1)

### 6.1 Mensajes de negociacion

### 6.1.1 ProtocolVersion

La negociacion comienza con el servidor enviando al cliente el mensaje ProtocolVersion. Este permite al cliente conocer la mayor version del protocolo RFB soportada por el servidor. El cliente entonces responde con un mensaje similar enviando el numero de la version de protocolo que deberia ser usado (la cual puede ser diferente a la ofrecida por el servidor). Un cliente nunca debe pedir una version de protocolo mayor a la ofrecida por el servidor. La intencion es que tanto cliente como servidor tengan algun nivel de compatibilidad hacia atras mediante este mecanismo.

Las unicas versiones de protocolo publicadas hasta el momento son 3.3, 3.7, 3.8 (Version 3.5 fue anunciada equivocadamente por algunos clientes, pero esto debe ser interpretado por todos los servidores como 3.3). La adicion de un tipo nuevo de codificacion o pseudo-codificacion no requiere un cambio en la version del protocolo, puesto que un servidor simplemente puede ignorar codificaciones que no comprenda.

El mensaje ProtocolVersion consiste de 12 bytes interpretado como una cadena de caracteres ASCII en el formato "RFB xxx.yyy\n" donde xxx e yyy son los numeros de version mayore y menor, rellenados con ceros.

| No. de bytes | Valor |
|---|---|
| 12 | "RFB 003.003\n" (hex 52 46 42 20 30 30 33 2e 30 30 33 0a) |

o

| No. de bytes | Valor |
|---|---|
| 12 | "RFB 003.007\n" (hex 52 46 42 20 30 30 33 2e 30 30 37 0a) |

o

| No. de bytes | Valor |
|---|---|
| 12 | "RFB 003.008\n" (hex 52 46 42 20 30 30 33 2e 30 30 38 0a) |

### 6.1.2 Seguridad

Una vez la version del protocolo se ha decidido, el servidor y cliente deben acordar el tipo de seguridad que sera usada en la conexion

**Version 3.7 en adelante** El servidor lista los tipos de seguridad que soporta:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | numero-de-tipos-de-seguridad |
| numero-de-tipos-de-seguridad | U8 arreglo | tipos-de-seguridad |

Si el servidor lista al menos un tipo de seguridad valida soportada por el cliente, el cliente enviara de regreso un solo byte indicando el tipo de seguridad usado en la conexion:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | tipo-de-seguridad |

Si numeros-de-tipos-de-seguridad es cero, entonces por alguna razon la conexion ha fallado (p.e el servidor no soporta el la version de protocolo deseada). Esto seguido por una cadena describiendo la razon (donde la cadena se especifica como la longitud precedida por ese numero de caracteres ASCII)

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | U32 | longitud-de-la-razon |
| longitud-de-la-razon | U8 | cadena-razon |

El servidor cierra la conexion despues de enviar la cadena de razon

**Version 3.3** El servidor decide el tipo de seguridad y envia una sola palabra

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | U32 | tipo-de-seguridad |

El tipo de seguridad solo tomara los valores 0,1 o 2. Un valor de 0 significa que la conexion ha fallado y es seguida de una cadena informando la razon como se describio mas arriba.

Los tipos de seguridad definidos en este documento son:

| Numero | Nombre |
|---|---|
| 0 | Invalido |
| 1 | **Ninguno** |
| 2 | **Autenticacion VNC** |

Otros tipos de seguridad registrados son:

| Numero | Nombre |
|---|---|
| 5 | **RA2** |
| 6 | **RA2ne** |
| 16 | Tight |
| 17 | Ultra |
| 18 | TLS |
| 19 | VeNCrypt |
| 20 | GTK-VNC SASL |
| 21 | MD5 hash authentication |
| 22 | Colin Dean xvp |

Una vez que el tipo de seguridad se ha decidido, los datos especificos de dicho tipo de seguridad vendran despues (ver seccion 6.2 para detalles). Al final de fase de negociacion de la seguridad, el protocolo normalmente continua con el mensaje SecurityResul

Se debe tener en cuenta que despues de la fase de negociacion de la seguridad, es posible que exista informacion adicional sobre un canal encriptado o alterado de otra forma.

### 6.1.3 SecurityResult

El servidor envia una palabra para informar al cliente que tipo de seguridad se negocio exitosamente.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | U32 | estado: |
| | 0 | **OK** |
| | 1 | **Fallido** |

Si es exitoso, el protocolo pasa a la fase de inicializacion (seccion 6.3)

**Version 3.8 en adelante** si no es exitoso, el servidor envia una cadena describiendo la razon de la falla y luego cierra la conexion:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | U32 | longitud-de-la-razon |
| longitud-de-la-razon | U8 arreglo | cadena-razon |

**Version 3.3 y 3.7** Si no es exitoso, el servidor cierra la conexion.

### 6.2 Tipos de seguridad

### 6.2.1 Ninguno

No se requiere ninguna autenticacion, los datos seran enviados sin encripcion

**Version 3.8 en adelante** El protocolo continua con el mensaje SecurityResult.

**Version 3.3 y 3.7** El protocolo pasa a la fase de inicializacion (seccion 6.3)

### 6.2.2 Authenticacion VNC

La autenticacion VNC sera usada y los datos del protocolo seran enviados sin encripcion. El servidor envia un reto de 16 bytes:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 16 | U8 | reto |

El cliente encripta el reto mediante DES, usando una contraseña proporcionada por el usuario como la llave y se envia la respuesta de 16 bytes:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 16 | U8 | respuesta |

El protocolo continua con el mensaje SecurityResult

### 6.3 Mensajes de inicializacion

Una vez que el cliente y el servidor estan seguros que estan felices hablando el uno al otro usando el tipo de seguridad acordada, el protocolo pasa a la fase de inicializacion. El cliente envia un mensaje ClientInit seguido por el servidor enviando un mensaje ServerInit.

### 6.3.1 ClientInit

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | bandera-compartir |

bandera-compartir no cero  (verdadera) si el servidor deberia tratar de compartir el escritorio dejando otros clientes conectados, cero (falso) si deberia dar acceso exclusivo al cliente desconectando a los demas.

### 6.3.2 ServerInit

Despues de recibir el mensaje ClientInit, el servidor envia un mensaje ServerInit. Este indica al cliente el ancho y alto del framebuffer del servidor, su formato de pixel y el nombre asociado con el escritorio:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 2 | U16 | framebuffer-ancho |
| 2 | U16 | framebuffer-alto |
| 16 | PIXEL_FORMAT | formato-pixel-servidor |
| 4 | U32 | longitud-nombre |
| longitud-nombre | U8 arreglo | cadena-nombre |

donde PIXEL_FORMAT es

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | bits-por-pixel |
| 1 | U8 | profundidad |
| 1 | U8 | bandera-big-endian |
| 1 | U8 | bandera-color-verdadero |
| 2 | U16 | rojo-maximo |
| 2 | U16 | verde-maximo |
| 2 | U16 | azul-maximo |
| 1 | U8 | desplazamiento del rojo |
| 1 | U8 | desplazamiento del verde |
| 1 | U8 | desplazamiento del azul |
| 3 | | relleno |

formato-pixel-servidor especifica el formato natural de los pixel del servidor. Este formato de pixel sera usado a menos que el cliente pida usar uno diferente usando el mensaje SetPixelFormat. (seccion 6.4.1).

Bits-por-pixel es el numero de bits usado para cada pixel a transmitir por la red. Este valor debe ser mayor o igual que la profundidad que es el numero util de bits en el pixel. Actualmente bits-por-pixel debe ser 8,16 o 32 - menos de 8-bit por pixel no estan soportados. la bandera de big endian es diferente de cero (verdadero) si pixeles multi byte son interpretados como big endian. Por supuesto esto no tiene sentido en 8 bits por pixel.

Si la bandera de color-verdadero no es cero (verdadero) entonces los ultimos seis items especificaran como extraer las intensidades roja, verde y azul de los datos de pixel. Rojo maximo es el valor maximo de rojo (= 2 n-1 donde n es el numero de bits usado para el rojo). Tenga en cuenta que este valor siempre es en orden big endian. Desplazamiento del rojo es el numero de desplazamiento de bits usado para obtener el valor del rojo en un pixel en el bit menos significativo. Verde-maximo desplazamiento de verde, azul maximo, desplazamiento de azul son similares. Por ejemplo, para encontrar el valor del rojo (entre 0 y rojo-maximo) de un determinado pixel, hacer lo siguiente.

* Intercambiar el valor de pixel dependiendo de la bandera de big-endian (p.e. si la bandera big-endian es cero (falso) y el orden de bytes del servidor es big endian, entonces intercambiar )

* Desplazar a la derecha dezplazamiento-rojo veces.

* AND con rojo-maximo (en el orden de bytes del servidor ).

Si la bandera de color-verdadero es cero (falsa)  entonces el servidor usara los valores de pixel que no estan compuestos directamente de intensidades de rojo, verde y azul, pero sirven como indices de un mapa de colores. Las posiciones en el mapa de colores son configuradas por el servidor usando el mensaje SetColourMapEntries (seccion 6.5.2)

### 6.4 Mensajes del cliente al servidor

Los tipos de mensajes del servidor al cliente definidos en este documento son:

| Numero | Nombre |
|---|---|
| 0 | SetPixelFormat |
| 2 | SetEncodings |
| 3 | FramebufferUpdateRequest |
| 4 | KeyEvent |
| 5 | PointerEvent |
| 6 | ClientCutText |

Otros tipos de mensajes registrados son:

| Numero | Nombre |
|---|---|
| 255 | Anthony Liguori |
| 254, 127 | VMWare |
| 253 | gii |
| 252 | tight |
| 251 | Pierre Ossman SetDesktopSize |
| 250 | Colin Dean xvp |
| 249 | OLIVE Call Control |

hay que tener en cuenta que antes de enviar un mensaje no definido en este documento, el cliente debe determinar si el servidor soporta la extension requerida recibiendo algun tipo de confirmacion especifica de la extension desde el servidor.

### 6.4.1 SetPixelFormat

Define el formato por el cual los valores de pixel deberan ser enviados en mensajes FrameBufferUpdate. Si el cliente no envia un mensaje SetPixelFormat entonces el servidor envia los valores de pixel en su formato natural especificado en el mensaje ServerInit (seccion 6.3.2). Si la bandera de color-verdadero es cero (falso) entonces esto indica que un "mapa de colores" sera usado. El servidor puede definir cualquiera de las posiciones en el mapa de colores usando el mensaje SetColourMapEntries (seccion 6.5.2). Inmediatamente despues que el cliente ha enviado este mensaje, el mapa de colores se borrara, aunque el servido lo hubiese definido con anterioridad.

| No. de bytes | Tipo [Valor] | Description |
|---|---|---|
| 1 | U8 0 | tipo-de-mensaje |
| 3 | | relleno |
| 16 | PIXEL_FORMAT | formato-pixel |

Donde PIXEL_FORMAT es descrito in la seccion 6.3.2:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | bits-por-pixel |
| 1 | U8 | profundidad |
| 1 | U8 | bandera-big-endian |
| 1 | U8 | bandera-color-verdadero |
| 2 | U16 | rojo-maximo |
| 2 | U16 | verde-maximo |
| 2 | U16 | azul-maximo |
| 1 | U8 | desplazamiento-rojo |
| 1 | U8 | desplazamiento-verde |
| 1 | U8 | desplazamiento-azul |
| 3 | | relleno |

### 6.4.2 SetEncodings

Define el tipo de codificacion en el cual los datos de pixel seran enviados por el servidor. El orden de los tipos de codificacion dados por este mensaje dan una pista de lo que el cliente prefiere (el primer tipo de codificacion que aparece en el mensaje es el preferido). El servidor puede o no hacer uso de esta pista. Los datos de pixel pueden ser enviados en codificacion bruta (raw) aun sino se especifico explicitamente aqui.

En adicion a las codificaciones genuinas, un cliente puede hacer peticiones de "pseudo-codificaciones" para declarar al servidor que soporta ciertas extensiones al protocolo. Un servidor que no soporta la extension simplemente ignorara la "pseudo-codificacion". Hay que tener en cuenta que el cliente debe asumir que el servidor no soporta la extension hasta que se obtenga una confirmacion especifica de la extension desde el servidor.

Ver la seccion 6.6 para una descripcion de cada codificacion y la seccion 6.7 para el significado de las pseudo-codificaciones

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 2 | tipo-de-mensaje |
| 1 | | relleno |
| 2 | U16 | numero-de-codificaciones |

Seguido por numero-de-codificaciones repeticiones de lo siguiente:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | S32 | tipo-de-codificacion |

### 6.4.3 FramebufferUpdateRequest

Notifica al servidor que el cliente esta interesado en un area del framebuffer especificada por una posicion en x, posicion en y, ancho y alto. El servidor generalmente responde a un mensaje bufferUpdateRequest enviando un mensaje FramebufferUpdate. Hay que tener en cuenta que un solo mensaje FramebufferUpdate puede ser enviado en respuesta a multiples FramebufferUpdateRequests. El servidor asume que el cliente mantiene una copia de todas las partes del framebuffer en que esta interesado. Esto significa normalmente que el servidor solo necesita enviar actualizaciones incrementales al cliente.

Sin embargo, si por alguna razon el cliente ha perdido el contenido de un area particular que se necesita, entonces el cliente envia un mensaje FramebufferUpdateRequest con el incremental asignado en cero (falso). Esta peticion se hace al servidor para que envie el contenido completo del area especificada tan pronto como sea posible. El area no podra ser actualizada utilizando la codificacion CopyRect. Si el cliente no ha perdido ninguno del contenido del area en que esta interesado, entonces envia un mensaje FramebufferUpdateRequest con el incremental asignado en no-cero (verdadero). Cuando ocurran cambios al area especificada del framebuffer, el servidor enviara un mensaje FramebufferUpdate. Hay que tener en cuenta que puede ocurrir un periodo de tiempo indefinido entre la peticion bufferUpdateRequest y la respuesta de actualizacion FramebufferUpdate. En caso que un cliente sea muy rapido, este debe regular la velocidad con la cual hace peticiones incrementales FramebufferUpdate para evitar congestionar la red.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 3 | tipo-de-mensaje |
| 1 | U8 | incremental |
| 2 | U16 | posicion-x |
| 2 | U16 | posicion-y |
| 2 | U16 | ancho |
| 2 | U16 | alto |

### 6.4.4 KeyEvent

Se presiona o se libera una tecla. la bandera-abajo es diferente de cero (verdadero) si la tecla se presiono, cero (falso) si se acaba de liberar. La tecla en si misma se especifica usando el valor "keysym" definido por el sistema X Window.

| No. de bytes | Tipo [Valor] | Description |
|---|---|---|
| 1 | U8 4 | tipo-de-mensaje |
| 1 | U8 | bandera-abajo |
| 2 | | relleno |
| 4 | U32 | tecla |

Para la mayoria de teclas ordinarias, el valor "keysym" es el mismo que el valor ASCII.

para los detalles completos, ver el manual de referencia de Xlib, publicado por O'Reilly & Associates, o mirar el archivo de cabecera <X11/keysymdef.h> en cualquier instalacion de sistema X Window. Algunas otras teclas comunes son:

| Nombre de la tecla | Valor keysym | | Nombre de la tecla | Valor keysym |
|---|---|---|---|---|
| BackSpace | 0xff08 | | F1 | 0xffbe |
| Tab | 0xff09 | | F2 | 0xffbf |
| Return o Enter | 0xff0d | | F3 | 0xffc0 |
| Escape | 0xff1b | | F4 | 0xffc1 |
| Insert | 0xff63 | | ... ... | |
| Delete | 0xffff | | F12 | 0xffc9 |
| Home | 0xff50 | | Shift (izquierda) | 0xffe1 |
| End | 0xff57 | | Shift (derecha) | 0xffe2 |
| Page Up | 0xff55 | | Control (izquierda) | 0xffe3 |
| Page Down | 0xff56 | | Control (derecha) | 0xffe4 |
| Left | 0xff51 | | Meta (izquierda) | 0xffe7 |
| Up | 0xff52 | | Meta (derecha) | 0xffe8 |
| Right | 0xff53 | | Alt (izquierda) | 0xffe9 |
| Down | 0xff54 | | Alt (derecha) | 0xffea |

La interpretacion de los keysims es una tarea compleja. Para poder ser lo mas ampliamente interoperable como sea posible, se deberan seguir las siguientes recomendaciones.

* el estado de "shift" o teclas sostenidas solo debe ser usado como una indicacion. Por ejemplo en un teclado US (Estadounidense) el caracter "#" se escribe utilizando la tecla shift, pero en un teclado UK (Britanico) no lo es. Un servidor con un teclado Norteamericano recibiendo un caracter "#" desde un cliente Britanico no vera una tecla shift presionada. En este caso, es probable que el servidor deba generar una pulsacion "falsa" de shift en su sistema local, para obtener el caracter "#" y no , por ejemplo, un 3.

* La diferencia entre mayusuculas y minusculas es significante. Esto es diferente a como se procesa el teclado en sistemas X Window que las interpreta como una misma. Por ejemplo un servidor recibiendo una "A" mayuscula sin pulsacion de shift deberia ser interpretada como una "A" mayuscula. De nuevo esto requiere de una "falsa" pulsacion interna de shift.

* Los servidores deberian ignorar el estado de las teclas "aseguradas" como por ejemplo el CapsLock y el NumLock tanto como sea posible. En cambio deberian interpretar cada keysym acorde a su caso.

* A diferencia del shift, el estado modificador de algunas teclas como Control y Alt debe tomarse como una interpretacion modificada de otros keysyms. Hay que tener en cuenta que que no hay keysyms para caracteres de control ASCII como ctrl-a, estos deben ser generados por los visores enviando una pulsacion de la tecla Control seguido por una pulsacion de tecla "a".

* En visores donde los modificadores como Control y Alt pueden usarse para generar keysyms, el visor puede necesitar enviar evento extra de "liberacion" para que los keysyms puedan ser interpretados correctamente. Por ejemplo en un teclado Aleman, la combinacion ctrl-alt-q genera el caracter "@". En este caso, el visor necesita enviar "falsos" eventos de liberacion para Control y Alt para que el caracter "@" sea interpretado correctamente (ctr-alt-@ probablemente signifique algo completamente diferente en el servidor)

* No hay un estandar universal para el "tab invertido" en el sistema X Window. En algunos sistemas shift+tab genera el keysym "ISO tab izquierdo", en otros genera un keysym "tab inverso" y en otros genera un "Tab" y las aplicaciones informan mediante el estado del shift que se trata de un tab inverso en vez de un tab normal. En el protocolo RFB la ultima alternativa es la preferida. Los visores deberian generar un Tab sostenido en vez de un ISO tab izquierdo. Sin embargo para mantener compatibilidad hacia atras con visores existentes, los servidores deberian reconocer el tab izquiero ISO como un Tab sostenido.

### 6.4.5 PointerEvent

Indica tanto un movimiento del raton o una pulsacion o liberacion del boton del raton. El cursor del raton esta ahora en ( posicion-x, posicion-y ), y el estado actual de los botones 1 al 8 estan representados por los bits 0 al 7 de la mascara de botones respectivamente, 0 indica arriba (liberado), 1 indica abajo (presionado). En un raton convencional los botones 1,2 y 3 corresponde a los botones izquierdos, de la mitad y derecho del raton. En un raton con rueda de desplazamiento, cada paso de la rueda hacia arriba esta representado como presion o liberacion del boton 4, y cada paso de la rueda hacia abajo es representado como una pulsacion y liberacion del boton 5

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 5 | tipo-de-mensaje |
| 1 | U8 | mascara-de-botones |
| 2 | U16 | posicion-x |
| 2 | U16 | posicion-y |

### 6.4.6 ClientCutText

El cliente tiene un nuevo texto ISO 8859-1 (Latin-1) en su buffer. Los fines de lineas estan representados por el caracter de  fin de linea/nuevalinea (valor 10) solamente. No se necesita caracter de retorno de carro (valor 13). No hay actualmente una forma de transferir texto que no sean del conjunto de caracteres Latin-1.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 6 | tipo-de-mensajes |
| 3 | | relleno |
| 4 | U32 | longitud |
| longitud | U8 arreglo | texto |

### 6.5 Mensajes del servidor al cliente

Los tipos de mensajes del servidor al cliente definidos en este documento son:

| Numero | Nombre |
|---|---|
| 0 | FramebufferUpdate |
| 1 | SetColourMapEntries |
| 2 | Bell |
| 3 | ServerCutText |

Otros tipos de mensajes registrados son:

| Numero | Nombre |
|---|---|
| 255 | Anthony Liguori |
| 254, 127 | VMWare |
| 253 | gii |
| 252 | tight |
| 250 | Colin Dean xvp |
| 249 | OLIVE Call Control |

Hay que tener en cuenta que antes de enviar un mensaje no definido en este documento, el servidor debe determinar que el cliente soporta la extension respectiva recibiendo algun tipo de confirmacion especifica desde el cliente - usualmente una peticion para una determinada pseudo-codificacion.

### 6.5.1 FramebufferUpdate

Una actualizacion de framebuffer consiste en una secuencia de rectangulos rellenos de pixeles los cuales el cliente debe poner en su framebuffer. Son enviados en respuesta a un FramebufferUpdateRequest desde el cliente. Tenga en cuenta que puede existir un periodo de tiempo indefinido entre FramebufferUpdateRequest y la respuesta FramebufferUpdate.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 0 | tipo-de-mensaje |
| 1 | | relleno |
| 2 | U16 | numero-de-rectangulos |

Esto es seguido por numero-de-rectangulos rectangulos rellenos de pixels. Cada rectangulo consiste de:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 2 | U16 | posicion-x |
| 2 | U16 | posicion-y |
| 2 | U16 | ancho |
| 2 | U16 | alto |
| 4 | S32 | tipo-de-codificacion |

seguido por los datos de pixel en la codificacion especificada. Ver la seccion 6.6 para el formato de los datos para cada codificacion y la seccion 6.7 para el significado de las pseudo-codificaciones.

### 6.5.2 SetColourMapEntries

Cuando el formato de pixel usa un "mapa de colores", este mensaje le dice al cliente que los valores especificados de pixel deben ser mapeados a las intensidades RGB dadas.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 1 | tipo-de-mensaje |
| 1 | | relleno |
| 2 | U16 | primer-color |
| 2 | U16 | numero-de-colores |

Seguido por un numero-de-colores repeticiones de lo siguiente:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 2 | U16 | rojo |
| 2 | U16 | verde |
| 2 | U16 | azul |

### 6.5.3 Bell

Suena una campana en el cliente si tiene una,

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 2 | tipo-de-mensaje |

### 6.5.4 ServerCutText

El servidor tiene un texto 8859-1 (Latin-1) copiado en su buffer. Los fines de linea son representados por el caracter fin de linea / nueva linea (valor 10) unicamente. No se necesita retorno de carro (valor 13). Actualmente no existe la manera de transferir texto que no se encuentre en el conjunto de caracteres Latin-1.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 3 | tipo-de-mensaje |
| 3 | | relleno |
| 4 | U32 | longitud |
| longitud | U8 arreglo | texto |

### 6.6 Codificaciones

Las codificaciones definidas en este documento son:

| Numero | Nombre |
|---|---|
| 0 | Bruta (Raw) |
| 1 | CopyRect |
| 2 | RRE |
| 5 | Hextile |
| 16 | ZRLE |
| -239 | pseudo-codificacion de cursor |
| -223 | pseudo-codificacion DesktopSize |

Otras codificaciones registradas son:

| Numero | Nombre |
|---|---|
| 4 | CoRRE |
| 6 | zlib |
| 7 | tight |
| 8 | zlibhex |
| 15 | TRLE |
| 17 | Hitachi ZYWRLE |
| 18 | Adam Walling XZ |
| 19 | Adam Walling XZYW |
| -1 a -222 | |
| -224 a -238 | |
| -240 a -256 | tight options |
| -257 a -272 | Anthony Liguori |
| -273 a -304 | VMWare |
| -305 | gii |
| -306 | popa |
| -307 | Peter Astrand DesktopName |
| -308 | Pierre Ossman ExtendedDesktopSize |
| -309 | Colin Dean xvp |
| -310 | OLIVE Call Control |
| -311 | CursorWithAlpha |
| -412 a -512 | TurboVNC fine-grained quality level |
| -763 a -768 | TurboVNC subsampling level |
| 0x574d5600 a 0x574d56ff | VMWare |

### 6.6.1 Codificacion bruta (raw)

la codificacion mas simple son los datos de pixel brutos. En este caso los datos consisten en ancho x alto (donde ancho y alto se refieren al ancho y alto del rectangulo). Los valores simplemente representan cada pixel izquierda-a-derecha de arriba hacia abajo. Todos los clientes deben estar preparados para mediar con este tipo de codificacion bruta (raw) de pixels, y los servidores RFB deben producir unicamente codificacion bruta (raw) a menos que el cliente especificamente pregunte por algun otro tipo diferente.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| ancho × alto × bytesPorPixel | arreglo de PIXEL | pixels |

### 6.6.2 Codificacion CopyRect

La codificacion CopyRect (copiar rectangulo) es una forma muy simple y eficiente de codificacion la cual puede ser usada cuando el cliente ya tiene la misma informacion de pixel en algun otro sitio del framebuffer. La codificacion a transmitir es muy simple, consiste en una coordenada x,y. Esto proporciona una posicion en el framebuffer donde el cliente puede copiar un rectangulo de pixels. Esto puede ser usado en varias situaciones, la mas obvia es cuando un usuario arrastra una ventana por la pantalla y cuando se mueven las barra de desplazamiento en una ventana. Un uso menos obvio es para optimizar la graficacion de texto o algun otro patron repetitivo. Un servidor inteligente debe ser capaz de enviar un patron explicitamente una unica vez, y sabiendo la posicion anterior del patron en el framebuffer, enviar posteriormente ocurrencias del mismo patron usando la codificacion CopyRect

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 2 | U16 | posicion-x-fuente |
| 2 | U16 | posicion-y-fuente |

### 6.6.3 codificacion RRE

RRE son las iniciales en Ingles de rise-and-run-length como su nombre lo implica, es esencialmente una version en dos dimensiones de la codificacion run-length. Rectangulos codificados en RRE llegan al cliente de una forma que puede ser dibujado inmediata y eficientemente por el mas simple de los motores graficos. RRE no es apropiado para escritorios complejos, pero puede ser util en algunas situaciones.

La idea basica detras de RRE es el particionamiento del rectangulo relleno de pixeles en subregiones rectangulares (subrectangulos) cada uno de los cuales rellenos de pixeles de un unico valor y la union de ellos genera una gran region rectagular. La cuasi-optima particion de un determinado rectangulo en subrectangulos es relativamente facil de calcular. La codificacion consiste de los valores de fondo de los pixel, Vb (tipicamente el valor de pixel mas prevalente en el rectangulo) y un conteo N, seguido por una lista de N subrectangulos, cada uno de los cuales consiste en una tupla <v,x,y,w,h> donde v (diferente a Vb) es el valor de pixel, (x,y) son las coordenadas del subrectangulo relativo a la esquina superior izquierda del rectangulo, y (w,h) son el ancho y el alto del subrectangulo. El cliente puede dibujar el rectangulo original relleno del valor de los pixel y luego dibujar un rectangulo relleno correspondiente a cada subrectangulo. Al transmitir los datos comienzan con la cabecera:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | U32 | numero-de-subrectangulos |
| bytesPorPixel | PIXEL | valor-de-pixel-fondo |

Esto es seguido por un numero-de-subrectangulos instancias de la siguiente estructura:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| bytesPorPixel | PIXEL | valor-pixel-subrectangulo |
| 2 | U16 | x-posicion |
| 2 | U16 | y-posicion |
| 2 | U16 | ancho |
| 2 | U16 | alto |

### 6.6.4 **Codificacion Hextile**

Hextile es una variacion de la idea de RRE. Los rectangulos son son divididos en mosaicos de 16x16, permitiendo que las dimensiones de los subrectangulos se pueda especificar con 4 bits cada uno, 16 bits en total. El rectangulo se divide en mosaicos iniciando en la parte superior izquierda, el orden es de derecha a izquierda y de arriba a abajo. El contenido codificado de los mosaicos sigue el uno al otro en orden predeterminado. Si el ancho del rectangulo total no es multiplo exacto de 16 entonces el ancho del ultimo mosaico en cada columna sera mas pequeño. Similarmente si la altura del rectangulo no es multiplo exacto de 16 entonces la altura de cada mosaico en la ultima fila sera mas pequeño.

Cada mosaico es codificado como datos brutos de pixel (raw), o como una variacion en RRE. Cada mosaico tiene un valor de pixel como el anterior. El valor de pixel de fondo no necesita ser explicitamente definido para un determinado mosaico si es igual que el mosaico anterior. Sin embargo el valor de pixel de fondo no se puede llevar de un mosaico a otro si el mosaico anterior uso la codificacion bruta (raw). Si todos los subrectangulos de un mosaico tienen el mismo valor de pixel, esto puede ser especificado como el valor de primer plano del mosaico completo. Al igual que con el color de fondo, el valor de pixel de primer plano puede dejarse sin especificar, significando esto que se usara el del mosaico anterior. Esto no se podra llevar a cabo si el mosaico anterior tiene en verdadero el bit Raw o SubrectsColoured. Sin embargo si se puede utilizar con el anterior mosaico con el bit en falso AnySubrects, mientras que el mosaico en si mismo tenga un color valido de primer plano del anterior mosaico.

Los datos consisten en cada mosaico codificado en orden. Cada mosaico inicia con un byte de tipo de subcodificacion, el cual es una mascara hecha de un numero de bits:

| No. de bytes | Tipo [Valor] | Description |
|---|---|---|
| 1 | U8 | mascara-subcodificacion: |
| | 1 | **Bruto (Raw)** |
| | 2 | **BackgroundSpecified** |
| | 4 | **ForegroundSpecified** |
| | 8 | **AnySubrects** |
| | 16 | **SubrectsColoured** |

Si el bit raw esta en verdadero entonces los demas bits son irrelevantes; ancho x alto valores de pixel a continuacion (donde ancho y alto son los del mosaico). De otra forma los otros bits en las mascara son asi:

**BackgroundSpecified** - Si es verdadero , un valor de pixel a continuacion especificara el color de fondo para este mosaico.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| bytesPorPixel | PIXEL | valor-pixel-fondo |

El primer mosaico no bruto (raw) debe tener este bit en verdadero. Si este bit no esta en verdadero el color de fondo es igual al del ultimo mosaico

**ForegroundSpecified** - Si es verdadero , el valor de pixel seguira a continuacion para definir el color de primer plano de todos los subrectangulos en este mosaico

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| bytesPorPixel | PIXEL | valor-pixel-primer-plano |

Si este bit se asigna verdadero entonces el bit SubrectsColoured debera ser cero.

**AnySubrects** - Si es verdadero, un solo byte seguira a continuacion indicando el numero de rectangulos de la siguiente forma:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | numero-de-subrectangulos |

Si no esta en verdadero, entonces no hay subrectangulos (p.e. el mosaico completo solamente es un color de fondo solido)

**SubrectsColoured** - Si esta en verdadero, entonces cada subrectangulo sera precedido por un valor de pixel que da el valor de color de dicho subrectangulo, un subrectangulo es:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| bytesPorPixel | PIXEL | valor-pixel-subrectangulo |
| 1 | U8 | posicion-X-y-Y |
| 1 | U8 | ancho-y-alto |

Si no esta en verdadero, todos los subrectangulos son del mismo color, el color de primer plano; Si el bit ForegroundSpecified esta en falso el color de primer plano sera igual que el ultimo mosaico. Un subrectangulo es

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | posicion-X-y-Y |
| 1 | U8 | ancho-y-alto |

La posicion y tamaño de cada subrectangulo esta especificada en dos bytes, posicion-X-y-Y y ancho-y-alto. Los cuatro bits mas significativos de posicion-X-y-Y especifican la posicion en X, los menos significativos especifican la posicion en Y. Los cuatro bits mas significativos de ancho-y-alto especifican el ancho menos uno, los menos significativos especifican el alto menos uno.

### 6.6.5 **Codificacion ZRLE**

ZRLE son las siglas en Ingles para codificacion Zlib 1 Run-Length, y combina compresion zlib, mosaicos, paletas y codificacion run-length. Al transmitir, un rectangulo comienza con un campo de longitud de 4 bytes, y es seguido por esa cantidad de bytes de datos comprimidos en zlib. Un solo objeto “tipo flujo” zlib es usado para una determinada conexion del protocolo RFB, asi que los rectangulos ZRLE deben ser codificados y decodificados en estricto orden.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 4 | U32 | longitud |
| longitud | U8 arreglo | DatosZlib |

Los DatosZlib cuando se descomprimen representan mosaicos de 64x64 pixels en orden de izquierda-a-derecha y arriba-hacia-abajo, similar al hextile. Si el ancho del rectangulo no es un multiplo exacto de 64, entonces el ancho del ultimo mosaico en cada fila sera mas pequeño, y si el alto del rectangulo no es un multiplo exacto de 64, entonces el alto de cada mosaico en la ultima fila sera mas pequeño

ZRLE hace uso de un nuevo tipo de datos CPIXEL (pixel comprimido). Esto similar al tipo de datos PIXEL, excepto que cuando la bandera de color-verdadero no es cero (verdadero), bits-por-pixel es 32, la profundidad 24 o menos y todo los bits que componen las intensidades rojo, verde y azul estan contenidos dentro de los 3 bytes mas significativos o los 3 bytes menos significativos. En este caso la longitud de CPIXEL es unicamente de 3 bytes, y contiene los 3 bytes mas o menos significativos segun sea lo apropiado. bytesPorCpixel es el numero de bytes en un CPIXEL

Cada mosaico inicia con un byte de tipo de subcodificacion. El bit mas significativo de ese byte sera verdadero si el mosaico ha sido codificado mediante run-lenght, falso de cualquier otra forma. Los 7 bits menos significativos indicaran el tamaño de la paleta usada - cero significa que no hay paleta, uno significa que el mosaico es de un solo color, 2 a 127 indican la paleta para ese tamaño. Los valores posibles para subcodificacion son

**0** - Datos de pixel brutos (Raw). valores de pixel ancho × alto pixel seguiran a continuacion (donde ancho y alto son el ancho y el alto del mosaico):

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| ancho × alto × bytesPorCPixel | CPIXEL arreglo | pixels |

**1** - Un mosaico solido consistente en un solo color. Los valores de pixel seran asi:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| bytesPorCPixel | CPIXEL | valor pixel |

**2 a 16** - Tipos de paleta empaquetados. Seguidos por la paleta, que consiste en tamanoPaleta(=subcodificacion) valores de pixel. Luego siguen los pixeles empaquetados, cada pixel representado como un campo de bits dando un indice a la paleta (0 significa el primer indice

1 ver http://www.gzip.org/zlib/

en la paleta). Para tamanoPaleta 2, un campo de 1 bit sera usado, para tamanoPaleta 3 o 4 un campo de 2 bits sera usado y para tamanoPaleta de 5 a 16 un campo de 4 bits sera usado. Los campos de bits estan empaquetados en bytes, los bits mas significativos representan los pixels mas a la izquierda (p.e big endian). Para mosaicos que no son multiplos de 8, 4 o 2 pixeles de ancho ( segun sea apropiado), bits de relleno son usados para alinear cada fila a un numero exacto de bytes

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| tamanoPaleta × bytesPorCPixel | CPIXEL arreglo | paleta |
| m | U8 arreglo | pixels empaquetados |

Donde m es el numero de bytes que representan los pixeles empaquetados. Para tamanoPaleta 2 esto es f loor((ancho + 7)/8) × alto, para tamanoPaleta 3 o 4 es f loor ((ancho+3)/4)×alto, para tamanoPaleta 5 a 16 esto es f loor((ancho+1)/2) × alto.

17 a 127 - no usado (no hay una ventaja sobre la paleta RLE).

128 - RLE plano. Consiste en un numero de repeticiones hasta que el mosaico esta listo. Las repeticiones pueden continuar desde el fin de una fila hasta el comienzo de la siguiente. Cada repeticion es representada por un unico valor de pixel seguido por la longitud de la repeticion. La longitud es representada por uno o mas bytes. La longitud es calculada como uno mas de la suma de todos los bytes representando el largo. Cualquier otro valor de byte diferente a 255 indica el byte final. Por ejemplo longitud 1 es representada como [0], 255 como [254], 256 como [255,0], 257 como [255,1], 510 como [255,254], 511 como [255,255,0] y asi sucesivamente.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| bytesPorCPixel | CPIXEL | Valor cpixel |
| floor((runLength − 1)/255) | U8 arreglo 255 | |
| 1 | U8 | (runLength − 1)%255 |

129 - no usado

130 a 255 - Paleta RLE. Seguido por la paleta consistente en tamanoPaleta =

(subcodificacion − 128) valores de pixel:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| tamanoPaleta × bytesPorCPixel | CPIXEL arreglo | paleta |

Al igual que con RLE plano, consiste en un numero de repeticiones, ejecutadas hasta que el mosaico este finalizado. Una repeticion de longitud 1 es representada simplemente por un indice en la paleta:

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | indicePaleta |

Una repeticion de longitud mayor a uno esta representada por un indice en la paleta con el bit mas significativo en verdadero, seguido por la longitud de la repeticion igual que en RLE plano.

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| 1 | U8 | indicePaleta + 128 |
| floor((runLength − 1)/255) | U8 arreglo 255 | |
| 1 | U8 | (runLength − 1)%255 |

6.7 **Pseudo-codificaciones**

### 6.7.1 **Pseudo-codificacion de cursor**

Un cliente que hace una peticion de pseudo-codificacion de cursor esta declarando que esta en la posibilidad de dibujar un cursor de raton localmente. Esto puede mejorar significativamente la percepcion del desempeño sobre enlaces lentos. El servidor indica la forma del cursor enviando un pseudo-rectangulo con la pseudo-codificacion de cursor como parte de la actualizacion. La posicion en X y Y del pseudo-rectangulo indican el are activa del cursor, y el ancho y el alto indican el ancho y el alto del cursor en pixels. Los datos consisten de pixeles de ancho x alto seguidos por una mascara de bits. La mascara de bits consiste en lineas de barrido de izquierda-a-derecha y de arriba-hacia-abajo, donde cada linea de barrido esta rellena a un numero entero de bytes ((ancho +7)/8). El bit mas significativo de cada byte representa el bit mas a la izquierda, donde 1 significa que el correspondiente pixel en el cursor es valido

| No. de bytes | Tipo [Valor] | Descripcion |
|---|---|---|
| ancho × alto × bytesPorPixel | PIXEL arreglo | pixel cursor |
| floor((ancho + 7)/8) ∗ alto | U8 arreglo | mascara de bits |

### 6.7.2 **Pseudo-codificacion DesktopSize**

Un cliente que hace una peticion de pseudo-codificacion DesktopSize esta declarando que esta en la posibilidad de modificar el ancho y/o alto del framebuffer. El servidor cambia el tamaño del escritorio enviando un pseudo-rectangulo con pseudo-codificacion DesktopSize como el ultimo rectangulo en una actualizacion. La posicion X y Y del pseudo-rectangulo es ignorada, y el ancho y el alto indican el nuevo ancho y alto del framebuffer.

No ningun otro dato mas asociados con el pseudo-rectangulo