Донецкий Национальный Технический Университет

Лабораторная работа № 5

Использование протокола VNC при разработке систем удаленного администрирования

Выполнил:

ст. группы ИПЗ -13

Лысенко А. С.

Проверила:

Скрипник. Т. В.

Красноармейск 2016

rfbProto.java

**package** mindbright.vnc;  
**import** java.io.\*;  
**import** java.awt.\*;  
**import** java.net.Socket;  
  
**import** mindbright.ssh.\*;  
**import** mindbright.vnc.MindVNC;  
  
  
**class** myInputStream **extends** FilterInputStream {  
 **public** myInputStream(InputStream in) {  
 **super**(in);  
 }  
 **public int** read(**byte**[] b, **int** off, **int** len) **throws** IOException {  
 System.***out***.println(**"read(byte[] b, int off, int len) called"**);  
 **return super**.read(b, off, len);  
 }  
}  
  
**public class** rfbProto {  
  
 **final** String **versionMsg** = **"RFB 003.003\n"**;  
 **final int ConnFailed** = 0, **NoAuth** = 1, **VncAuth** = 2;  
 **final int VncAuthOK** = 0, **VncAuthFailed** = 1, **VncAuthTooMany** = 2;  
  
 **final int FramebufferUpdate** = 0, **SetColourMapEntries** = 1, **Bell** = 2,  
 **ServerCutText** = 3;  
  
 **final int SetPixelFormat** = 0, **FixColourMapEntries** = 1, **SetEncodings** = 2,  
 **FramebufferUpdateRequest** = 3, **KeyEvent** = 4, **PointerEvent** = 5,  
 **ClientCutText** = 6;  
  
 **final static int *EncodingRaw*** = 0, ***EncodingCopyRect*** = 1, ***EncodingRRE*** = 2,  
 ***EncodingCoRRE*** = 4, ***EncodingHextile*** = 5;  
  
 **final int HextileRaw** = (1 << 0);  
 **final int HextileBackgroundSpecified** = (1 << 1);  
 **final int HextileForegroundSpecified** = (1 << 2);  
 **final int HextileAnySubrects** = (1 << 3);  
 **final int HextileSubrectsColoured** = (1 << 4);  
  
 String **vncHost**;  
 **int vncPort**;  
  
 SSHSocket **sock**;  
 SSHSocketFactory **fact**;  
  
 DataInputStream **is**;  
 OutputStream **os**;  
 **public boolean inNormalProtocol** = **false**;  
 MindVNC **v**;  
  
   
 **public** rfbProto(String sshHost, **int** sshPort, String sshUser, String sshPasswd,  
 String vncHost, **int** vncPort, MindVNC v1) **throws** IOException {  
 **v** = v1;  
  
 **this**.**vncHost** = vncHost;  
 **this**.**vncPort** = vncPort;  
  
 **fact** = **new** SSHSocketFactory(sshHost, sshPort, **new** SSHPasswordAuthenticator(sshUser, sshPasswd));  
  
 **sock** = **fact**.createSocket(vncHost, vncPort);  
  
 **is** = **new** DataInputStream(**new** BufferedInputStream(**sock**.getInputStream(),  
 16384));  
 **os** = **sock**.getOutputStream();  
 }  
  
  
 **public void** close() {  
 **try** {  
 **sock**.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
   
 **int serverMajor**, **serverMinor**;  
  
 **void** readVersionMsg() **throws** IOException {  
  
 **byte**[] b = **new byte**[12];  
  
 **is**.readFully(b);  
  
 **if** ((b[0] != **'R'**) || (b[1] != **'F'**) || (b[2] != **'B'**) || (b[3] != **' '**)  
 || (b[4] < **'0'**) || (b[4] > **'9'**) || (b[5] < **'0'**) || (b[5] > **'9'**)  
 || (b[6] < **'0'**) || (b[6] > **'9'**) || (b[7] != **'.'**)  
 || (b[8] < **'0'**) || (b[8] > **'9'**) || (b[9] < **'0'**) || (b[9] > **'9'**)  
 || (b[10] < **'0'**) || (b[10] > **'9'**) || (b[11] != **'\n'**))  
 {  
 **throw new** IOException(**"Host "** + **vncHost** + **" port "** + **vncPort** +  
 **" is not an RFB server"**);  
 }  
  
 **serverMajor** = (b[4] - **'0'**) \* 100 + (b[5] - **'0'**) \* 10 + (b[6] - **'0'**);  
 **serverMinor** = (b[8] - **'0'**) \* 100 + (b[9] - **'0'**) \* 10 + (b[10] - **'0'**);  
 }  
  
 **public boolean** connectAndAuthenticate(authenticationPanel authenticator) **throws** IOException {  
 **boolean** authenticationDone = **false**;  
  
 readVersionMsg();  
  
 System.***out***.println(**"RFB server supports protocol version "** +  
 **serverMajor** + **"."** + **serverMinor**);  
  
 writeVersionMsg();  
  
 **switch** (readAuthScheme()) {  
  
 **case NoAuth**:  
 System.***out***.println(**"No authentication needed"**);  
 authenticationDone = **true**;  
 **break**;  
  
 **case VncAuth**:  
 **byte**[] challenge = **new byte**[16];  
 **is**.readFully(challenge);  
  
 String pw = authenticator.vncPassword.getText();  
 **if** (pw.length() > 8) pw = pw.substring(0,8); *// truncate to 8 chars* **byte**[] key = **new byte**[8];  
 pw.getBytes(0, pw.length(), key, 0);  
  
 **for** (**int** i = pw.length(); i < 8; i++) {  
 key[i] = (**byte**)0;  
 }  
  
 DesCipher des = **new** DesCipher(key);  
  
 des.encrypt(challenge,0,challenge,0);  
 des.encrypt(challenge,8,challenge,8);  
  
 **os**.write(challenge);  
  
 **int** authResult = **is**.readInt();  
  
 **switch** (authResult) {  
 **case VncAuthOK**:  
 System.***out***.println(**"VNC authentication succeeded"**);  
 authenticationDone = **true**;  
 **break**;  
 **case VncAuthFailed**:  
 System.***out***.println(**"VNC authentication failed"**);  
 authenticator.retry(**"VNC access denied"**);  
 **break**;  
 **case VncAuthTooMany**:  
 **throw new** IOException(**"VNC authentication failed - "** +  
 **"too many tries"**);  
 **default**:  
 **throw new** IOException(**"Unknown VNC authentication result "** +  
 authResult);  
 }  
 **break**;  
 }  
  
 **return** authenticationDone;  
 }  
 **void** writeVersionMsg() **throws** IOException {  
 **byte**[] b = **new byte**[12];  
 **versionMsg**.getBytes(0, 12, b, 0);  
 **os**.write(b);  
 }  
 **int** readAuthScheme() **throws** IOException {  
 **int** authScheme = **is**.readInt();  
  
 **switch** (authScheme) {  
  
 **case ConnFailed**:  
 **int** reasonLen = **is**.readInt();  
 **byte**[] reason = **new byte**[reasonLen];  
 **is**.readFully(reason);  
 **throw new** IOException(**new** String(reason, 0));  
  
 **case NoAuth**:  
 **case VncAuth**:  
 **return** authScheme;  
  
 **default**:  
 **throw new** IOException(**"Unknown authentication scheme from RFB "** +  
 **"server "** + authScheme);  
  
 }  
 }  
  
 **public void** doProtocolInitialisation(**int**[] encodings, **int** nEncodings) **throws** IOException {  
 System.***out***.println(**"sending client init"**);  
  
 writeClientInit();  
  
 readServerInit();  
  
 System.***out***.println(**"Desktop name is "** + **desktopName**);  
 System.***out***.println(**"Desktop size is "** + **framebufferWidth** + **" x "** +  
 **framebufferHeight**);  
  
 setEncodings(encodings, nEncodings);  
 }  
 **void** setEncodings(**int**[] encodings, **int** nEncodings) {  
 **try** {  
 **if** (**inNormalProtocol**) {  
 writeSetEncodings(encodings, nEncodings);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **void** writeClientInit() **throws** IOException {  
 **if** (**v**.**options**.shareDesktop) {  
 **os**.write(1);  
 } **else** {  
 **os**.write(0);  
 }  
 **v**.**options**.disableShareDesktop();  
 }  
  
 **public** String **desktopName**;  
 **public int framebufferWidth**, **framebufferHeight**;  
 **int bitsPerPixel**, **depth**;  
 **boolean bigEndian**, **trueColour**;  
 **int redMax**, **greenMax**, **blueMax**, **redShift**, **greenShift**, **blueShift**;  
  
 **void** readServerInit() **throws** IOException {  
 **framebufferWidth** = **is**.readUnsignedShort();  
 **framebufferHeight** = **is**.readUnsignedShort();  
 **bitsPerPixel** = **is**.readUnsignedByte();  
 **depth** = **is**.readUnsignedByte();  
 **bigEndian** = (**is**.readUnsignedByte() != 0);  
 **trueColour** = (**is**.readUnsignedByte() != 0);  
 **redMax** = **is**.readUnsignedShort();  
 **greenMax** = **is**.readUnsignedShort();  
 **blueMax** = **is**.readUnsignedShort();  
 **redShift** = **is**.readUnsignedByte();  
 **greenShift** = **is**.readUnsignedByte();  
 **blueShift** = **is**.readUnsignedByte();  
 **byte**[] pad = **new byte**[3];  
 **is**.read(pad);  
 **int** nameLength = **is**.readInt();  
 **byte**[] name = **new byte**[nameLength];  
 **is**.readFully(name);  
 **desktopName** = **new** String(name, 0);  
  
 **inNormalProtocol** = **true**;  
 }  
  
  
 **int** readServerMessageType() **throws** IOException {  
 **return is**.read();  
 }  
  
 **int updateNRects**;  
  
 **void** readFramebufferUpdate() **throws** IOException {  
 **is**.readByte();  
 **updateNRects** = **is**.readUnsignedShort();  
 }  
  
 **int updateRectX**, **updateRectY**, **updateRectW**, **updateRectH**, **updateRectEncoding**;  
  
 **void** readFramebufferUpdateRectHdr() **throws** IOException {  
 **updateRectX** = **is**.readUnsignedShort();  
 **updateRectY** = **is**.readUnsignedShort();  
 **updateRectW** = **is**.readUnsignedShort();  
 **updateRectH** = **is**.readUnsignedShort();  
 **updateRectEncoding** = **is**.readInt();  
  
 **if** ((**updateRectX** + **updateRectW** > **framebufferWidth**) ||  
 (**updateRectY** + **updateRectH** > **framebufferHeight**)) {  
 **throw new** IOException(**"Framebuffer update rectangle too large: "** +  
 **updateRectW** + **"x"** + **updateRectH** + **" at ("** +  
 **updateRectX** + **","** + **updateRectY** + **")"**);  
 }  
 }  
  
 **int copyRectSrcX**, **copyRectSrcY**;  
  
 **void** readCopyRect() **throws** IOException {  
 **copyRectSrcX** = **is**.readUnsignedShort();  
 **copyRectSrcY** = **is**.readUnsignedShort();  
 }  
  
 String readServerCutText() **throws** IOException {  
 **byte**[] pad = **new byte**[3];  
 **is**.read(pad);  
 **int** len = **is**.readInt();  
 **byte**[] text = **new byte**[len];  
 **is**.readFully(text);  
 **return new** String(text, 0);  
 }  
  
 **void** writeFramebufferUpdateRequest(**int** x, **int** y, **int** w, **int** h,  
 **boolean** incremental)  
 **throws** IOException  
 {  
 **byte**[] b = **new byte**[10];  
  
 b[0] = (**byte**) **FramebufferUpdateRequest**;  
 b[1] = (**byte**) (incremental ? 1 : 0);  
 b[2] = (**byte**) ((x >> 8) & 0xff);  
 b[3] = (**byte**) (x & 0xff);  
 b[4] = (**byte**) ((y >> 8) & 0xff);  
 b[5] = (**byte**) (y & 0xff);  
 b[6] = (**byte**) ((w >> 8) & 0xff);  
 b[7] = (**byte**) (w & 0xff);  
 b[8] = (**byte**) ((h >> 8) & 0xff);  
 b[9] = (**byte**) (h & 0xff);  
  
 **os**.write(b);  
 }  
   
 **void** writeSetPixelFormat(**int** bitsPerPixel, **int** depth, **boolean** bigEndian,  
 **boolean** trueColour,  
 **int** redMax, **int** greenMax, **int** blueMax,  
 **int** redShift, **int** greenShift, **int** blueShift)  
 **throws** IOException  
 {  
 **byte**[] b = **new byte**[20];  
  
 b[0] = (**byte**) **SetPixelFormat**;  
 b[4] = (**byte**) bitsPerPixel;  
 b[5] = (**byte**) depth;  
 b[6] = (**byte**) (bigEndian ? 1 : 0);  
 b[7] = (**byte**) (trueColour ? 1 : 0);  
 b[8] = (**byte**) ((redMax >> 8) & 0xff);  
 b[9] = (**byte**) (redMax & 0xff);  
 b[10] = (**byte**) ((greenMax >> 8) & 0xff);  
 b[11] = (**byte**) (greenMax & 0xff);  
 b[12] = (**byte**) ((blueMax >> 8) & 0xff);  
 b[13] = (**byte**) (blueMax & 0xff);  
 b[14] = (**byte**) redShift;  
 b[15] = (**byte**) greenShift;  
 b[16] = (**byte**) blueShift;  
  
 **os**.write(b);  
 }  
  
 **void** writeFixColourMapEntries(**int** firstColour, **int** nColours,  
 **int**[] red, **int**[] green, **int**[] blue)  
 **throws** IOException  
 {  
 **byte**[] b = **new byte**[6 + nColours \* 6];  
  
 b[0] = (**byte**) **FixColourMapEntries**;  
 b[2] = (**byte**) ((firstColour >> 8) & 0xff);  
 b[3] = (**byte**) (firstColour & 0xff);  
 b[4] = (**byte**) ((nColours >> 8) & 0xff);  
 b[5] = (**byte**) (nColours & 0xff);  
  
 **for** (**int** i = 0; i < nColours; i++) {  
 b[6 + i \* 6] = (**byte**) ((red[i] >> 8) & 0xff);  
 b[6 + i \* 6 + 1] = (**byte**) (red[i] & 0xff);  
 b[6 + i \* 6 + 2] = (**byte**) ((green[i] >> 8) & 0xff);  
 b[6 + i \* 6 + 3] = (**byte**) (green[i] & 0xff);  
 b[6 + i \* 6 + 4] = (**byte**) ((blue[i] >> 8) & 0xff);  
 b[6 + i \* 6 + 5] = (**byte**) (blue[i] & 0xff);  
 }  
  
 **os**.write(b);  
 }  
   
 **void** writeSetEncodings(**int**[] encs, **int** len) **throws** IOException {  
 **byte**[] b = **new byte**[4 + 4 \* len];  
  
 b[0] = (**byte**) **SetEncodings**;  
 b[2] = (**byte**) ((len >> 8) & 0xff);  
 b[3] = (**byte**) (len & 0xff);  
  
 **for** (**int** i = 0; i < len; i++) {  
 b[4 + 4 \* i] = (**byte**) ((encs[i] >> 24) & 0xff);  
 b[5 + 4 \* i] = (**byte**) ((encs[i] >> 16) & 0xff);  
 b[6 + 4 \* i] = (**byte**) ((encs[i] >> 8) & 0xff);  
 b[7 + 4 \* i] = (**byte**) (encs[i] & 0xff);  
 }  
  
 **os**.write(b);  
 }  
 **public void** writeClientCutText(String text) **throws** IOException {  
 **byte**[] b = **new byte**[8 + text.length()];  
  
 b[0] = (**byte**) **ClientCutText**;  
 b[4] = (**byte**) ((text.length() >> 24) & 0xff);  
 b[5] = (**byte**) ((text.length() >> 16) & 0xff);  
 b[6] = (**byte**) ((text.length() >> 8) & 0xff);  
 b[7] = (**byte**) (text.length() & 0xff);  
  
 text.getBytes(0, text.length(), b, 8);  
  
 **os**.write(b);  
 }  
 **byte**[] **eventBuf** = **new byte**[72];  
 **int eventBufLen**;  
   
 **int pointerMask** = 0;  
  
 **void** writePointerEvent(Event evt)  
 **throws** IOException  
 {  
 **byte**[] b = **new byte**[6];  
  
 **if** (evt.**id** == Event.***MOUSE\_DOWN***) {  
 **pointerMask** = 1;  
 **if** ((evt.**modifiers** & Event.***ALT\_MASK***) != 0) {  
 **if** (**v**.**options**.reverseMouseButtons2And3)  
 **pointerMask** = 4;  
 **else  
 pointerMask** = 2;  
 }  
 **if** ((evt.**modifiers** & Event.***META\_MASK***) != 0) {  
 **if** (**v**.**options**.reverseMouseButtons2And3)  
 **pointerMask** = 2;  
 **else  
 pointerMask** = 4;  
 }  
 } **else if** (evt.**id** == Event.***MOUSE\_UP***) {  
 **pointerMask** = 0;  
 }  
  
 evt.**modifiers** &= ~(Event.***ALT\_MASK***|Event.***META\_MASK***);  
  
 **eventBufLen** = 0;  
  
 writeModifierKeyEvents(evt.**modifiers**);  
  
 **if** (evt.**x** < 0) evt.**x** = 0;  
 **if** (evt.**y** < 0) evt.**y** = 0;  
  
 **eventBuf**[**eventBufLen**++] = (**byte**) **PointerEvent**;  
 **eventBuf**[**eventBufLen**++] = (**byte**) **pointerMask**;  
 **eventBuf**[**eventBufLen**++] = (**byte**) ((evt.**x** >> 8) & 0xff);  
 **eventBuf**[**eventBufLen**++] = (**byte**) (evt.**x** & 0xff);  
 **eventBuf**[**eventBufLen**++] = (**byte**) ((evt.**y** >> 8) & 0xff);  
 **eventBuf**[**eventBufLen**++] = (**byte**) (evt.**y** & 0xff);  
 **if** (**pointerMask** == 0) {  
 writeModifierKeyEvents(0);  
 }  
  
 **os**.write(**eventBuf**, 0, **eventBufLen**);  
 }  
  
 **public void** sendCtrlAltDel() {  
 **try** {  
 Event ctrlAltDelEvent = **new** Event(**null**, 0, **null**);  
 ctrlAltDelEvent.**key** = 127;  
 ctrlAltDelEvent.**modifiers** = Event.***CTRL\_MASK*** | Event.***ALT\_MASK***;  
 ctrlAltDelEvent.**id** = Event.***KEY\_PRESS***;  
 writeKeyEvent(ctrlAltDelEvent);  
 ctrlAltDelEvent.**id** = Event.***KEY\_RELEASE***;  
 writeKeyEvent(ctrlAltDelEvent);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **void** writeKeyEvent(Event evt)  
 **throws** IOException  
 {  
 **int** key = evt.**key**;  
 **boolean** down = **false**;  
  
 **if** ((evt.**id** == Event.***KEY\_PRESS***) || (evt.**id** == Event.***KEY\_ACTION***))  
 down = **true**;  
  
 **if** ((evt.**id** == Event.***KEY\_ACTION***) || (evt.**id** == Event.***KEY\_ACTION\_RELEASE***)) {  
  
 **switch**(key) {  
 **case** Event.***HOME***: key = 0xff50; **break**;  
 **case** Event.***LEFT***: key = 0xff51; **break**;  
 **case** Event.***UP***: key = 0xff52; **break**;  
 **case** Event.***RIGHT***: key = 0xff53; **break**;  
 **case** Event.***DOWN***: key = 0xff54; **break**;  
 **case** Event.***PGUP***: key = 0xff55; **break**;  
 **case** Event.***PGDN***: key = 0xff56; **break**;  
 **case** Event.***END***: key = 0xff57; **break**;  
 **case** Event.***F1***: key = 0xffbe; **break**;  
 **case** Event.***F2***: key = 0xffbf; **break**;  
 **case** Event.***F3***: key = 0xffc0; **break**;  
 **case** Event.***F4***: key = 0xffc1; **break**;  
 **case** Event.***F5***: key = 0xffc2; **break**;  
 **case** Event.***F6***: key = 0xffc3; **break**;  
 **case** Event.***F7***: key = 0xffc4; **break**;  
 **case** Event.***F8***: key = 0xffc5; **break**;  
 **case** Event.***F9***: key = 0xffc6; **break**;  
 **case** Event.***F10***: key = 0xffc7; **break**;  
 **case** Event.***F11***: key = 0xffc8; **break**;  
 **case** Event.***F12***: key = 0xffc9; **break**;  
 **default**:  
 **return**;  
 }  
  
 } **else** {  
  
 **if** (key < 32) {  
 **if** ((evt.**modifiers** & Event.***CTRL\_MASK***) != 0) {  
 key += 96;  
 } **else** {  
 **switch**(key) {  
 **case** 8: key = 0xff08; **break**;  
 **case** 9: key = 0xff09; **break**;  
 **case** 10: key = 0xff0d; **break**;  
 **case** 27: key = 0xff1b; **break**;  
 }  
 }  
 } **else if** (key >= 127) {  
 **if** (key == 127) {  
 key = 0xffff;  
 } **else** {  
 **if** ((key < 0xff00) || (key > 0xffff))  
 **return**;  
 }  
 }  
 }  
  
 **eventBufLen** = 0;  
  
 writeModifierKeyEvents(evt.**modifiers**);  
  
 writeKeyEvent(key, down);  
  
 **if** (!down) {  
 writeModifierKeyEvents(0);  
 }  
  
 **os**.write(**eventBuf**, 0, **eventBufLen**);  
 }  
  
 **void** writeKeyEvent(**int** keysym, **boolean** down)  
 **throws** IOException  
 {  
 **eventBuf**[**eventBufLen**++] = (**byte**) **KeyEvent**;  
 **eventBuf**[**eventBufLen**++] = (**byte**) (down ? 1 : 0);  
 **eventBuf**[**eventBufLen**++] = (**byte**) 0;  
 **eventBuf**[**eventBufLen**++] = (**byte**) 0;  
 **eventBuf**[**eventBufLen**++] = (**byte**) ((keysym >> 24) & 0xff);  
 **eventBuf**[**eventBufLen**++] = (**byte**) ((keysym >> 16) & 0xff);  
 **eventBuf**[**eventBufLen**++] = (**byte**) ((keysym >> 8) & 0xff);  
 **eventBuf**[**eventBufLen**++] = (**byte**) (keysym & 0xff);  
 }  
 **int oldModifiers**;  
  
 **void** writeModifierKeyEvents(**int** newModifiers)  
 **throws** IOException  
 {  
 **if** ((newModifiers & Event.***CTRL\_MASK***) != (**oldModifiers** & Event.***CTRL\_MASK***))  
 writeKeyEvent(0xffe3, (newModifiers & Event.***CTRL\_MASK***) != 0);  
  
 **if** ((newModifiers & Event.***SHIFT\_MASK***) != (**oldModifiers** & Event.***SHIFT\_MASK***))  
 writeKeyEvent(0xffe1, (newModifiers & Event.***SHIFT\_MASK***) != 0);  
  
 **if** ((newModifiers & Event.***META\_MASK***) != (**oldModifiers** & Event.***META\_MASK***))  
 writeKeyEvent(0xffe7, (newModifiers & Event.***META\_MASK***) != 0);  
  
 **if** ((newModifiers & Event.***ALT\_MASK***) != (**oldModifiers** & Event.***ALT\_MASK***))  
 writeKeyEvent(0xffe9, (newModifiers & Event.***ALT\_MASK***) != 0);  
  
 **oldModifiers** = newModifiers;  
 }  
}

MindVNC.java

**package** mindbright.vnc;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.io.\*;  
  
**import** mindbright.ssh.\*;  
**import** mindbright.vnc.\*;  
  
**public class** MindVNC **extends** java.applet.Applet  
 **implements** java.lang.Runnable  
{  
 **boolean inAnApplet** = **true**;  
 **boolean separateFrame**;  
  
 Container **cont**;  
 **public static void** main(String[] argv) {  
 MindVNC v = **new** MindVNC();  
 v.**mainArgs** = argv;  
 v.**inAnApplet** = **false**;  
  
 v.**f** = **new** Frame(**"MindVNC v0.1"**);  
 v.**f**.add(**"Center"**, v);  
  
 v.init();  
 v.start();  
 }  
  
 Frame **f**;  
 String[] **mainArgs**;  
  
 String **sshHost**;  
 **int sshPort**;  
 String **vncHost**;  
 **int vncPort**;  
 String **sshUser**;  
 String **sshPasswd**;  
  
 ScrollPane **sp**;  
 vncCanvas **vc**;  
 **public** rfbProto **rfb**;  
 Thread **rfbThread**;  
 GridBagLayout **gridbag**;  
 Panel **buttonPanel**;  
 Button **disconnectButton**;  
 Button **optionsButton**;  
 Button **clipboardButton**;  
 Button **ctrlAltDelButton**;  
 authenticationPanel **authenticator**;  
  
 **public** optionsFrame **options**;  
 **public** clipboardFrame **clipboard**;  
  
 **public void** init() {  
  
 readParameters();  
  
 **if**(**inAnApplet** && **separateFrame**)  
 **cont** = **f** = **new** Frame(**"MindVNC v0.1"**);  
 **else  
 cont** = **this**;  
  
 **if**(**f** != **null**) {  
 **f**.addWindowListener(**new** WindowAdapter() {  
 **public void** windowClosing(WindowEvent e) { **f**.dispose(); **rfbThread**.stop();  
 **if**(!**inAnApplet**) System.*exit*(1); }  
 });  
 }  
  
  
 **options** = **new** optionsFrame(**this**);  
 **clipboard** = **new** clipboardFrame(**this**);  
 **authenticator** = **new** authenticationPanel();  
  
 **rfbThread** = **new** Thread(**this**);  
 **rfbThread**.start();  
 }  
  
 **public void** update(Graphics g) {  
 }  
  
 **public void** run() {  
  
 **gridbag** = **new** GridBagLayout();  
 **cont**.setLayout(**gridbag**);  
  
 **buttonPanel** = **new** Panel();  
 **buttonPanel**.setLayout(**new** FlowLayout(FlowLayout.***LEFT***, 0, 0));  
 **disconnectButton** = **new** Button(**"Disconnect"**);  
 **buttonPanel**.add(**disconnectButton**);  
 **optionsButton** = **new** Button(**"Options"**);  
 **buttonPanel**.add(**optionsButton**);  
 **clipboardButton** = **new** Button(**"Clipboard"**);  
 **buttonPanel**.add(**clipboardButton**);  
 **ctrlAltDelButton** = **new** Button(**"Send Ctrl-Alt-Del"**);  
 **buttonPanel**.add(**ctrlAltDelButton**);  
  
 GridBagConstraints gbc = **new** GridBagConstraints();  
 gbc.**gridwidth** = GridBagConstraints.***REMAINDER***;  
 gbc.**anchor** = GridBagConstraints.***NORTHWEST***;  
 **gridbag**.setConstraints(**buttonPanel**,gbc);  
 **cont**.add(**buttonPanel**);  
  
 **disconnectButton**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 buttonPressed(0);  
 }  
 });  
 **optionsButton**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 buttonPressed(1);  
 }  
 });  
 **clipboardButton**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 buttonPressed(2);  
 }  
 });  
 **ctrlAltDelButton**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 buttonPressed(3);  
 }  
 });  
  
  
 **while**(**true**) {  
  
 **disconnectButton**.disable();  
 **clipboardButton**.disable();  
 **ctrlAltDelButton**.disable();  
  
 **try** {  
 connectAndAuthenticate();  
  
 **rfb**.doProtocolInitialisation(**options**.encodings, **options**.nEncodings);  
  
 **vc** = **new** vncCanvas(**this**);  
  
 **sp** = **new** ScrollPane(ScrollPane.***SCROLLBARS\_AS\_NEEDED***);  
 gbc.**weightx** = 1.0;  
 gbc.**weighty** = 1.0;  
 gbc.**fill** = GridBagConstraints.***BOTH***;  
 **gridbag**.setConstraints(**sp**,gbc);  
 **cont**.add(**sp**);  
  
 **if**(**f** != **null**) {  
 **int** sbw = **sp**.getVScrollbarWidth();  
 **int** sbh = **sp**.getHScrollbarHeight();  
 Dimension max = Toolkit.*getDefaultToolkit*().getScreenSize();  
 Insets fIns = **f**.getInsets();  
 max.**width** -= (fIns.**left** + fIns.**right** + sbw);  
 max.**height** -= (fIns.**top** + fIns.**bottom** + sbh);  
 **int** w = max.**width** > **rfb**.**framebufferWidth** + 4 ? **rfb**.**framebufferWidth** + 4 : max.**width**;  
 **int** h = max.**height** > **rfb**.**framebufferHeight** + 4 ? **rfb**.**framebufferHeight** + 4 : max.**height**;  
 **sp**.setSize(**new** Dimension(w, h));  
 } **else** {  
 **sp**.setSize(getSize());  
 }  
 **sp**.add(**vc**);  
  
 **if**(**f** != **null**) {  
 **f**.setTitle(**rfb**.**desktopName**);  
 **f**.pack();  
 } **else** {  
 validate();  
 }  
  
 **disconnectButton**.enable();  
 **clipboardButton**.enable();  
 **ctrlAltDelButton**.enable();  
  
 **vc**.processNormalProtocol();  
 **cont**.remove(**sp**);  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 fatalError(e.toString());  
 }  
 }  
 }  
  
 **void** connectAndAuthenticate() **throws** IOException {  
  
 GridBagConstraints gbc = **new** GridBagConstraints();  
 gbc.**gridwidth** = GridBagConstraints.***REMAINDER***;  
 gbc.**anchor** = GridBagConstraints.***NORTHWEST***;  
 gbc.**weightx** = 1.0;  
 gbc.**weighty** = 1.0;  
 gbc.**ipadx** = 100;  
 gbc.**ipady** = 50;  
 **gridbag**.setConstraints(**authenticator**,gbc);  
 **cont**.add(**authenticator**);  
  
 validate();  
 **if**(**f** != **null**) {  
 **f**.pack();  
 **f**.show();  
 }  
  
 **boolean** authenticationDone = **false**;  
  
 **while** (!authenticationDone) {  
  
 **synchronized**(**authenticator**) {  
 **try** {  
 **authenticator**.wait();  
 } **catch** (InterruptedException e) {  
 }  
 }  
  
 **sshUser** = **authenticator**.sshUser.getText();  
 **sshPasswd** = **authenticator**.sshPassword.getText();  
 **vncHost** = **authenticator**.vncHost.getText();  
 **vncPort** = 5900;  
  
 **int** ix;  
 **if**((ix = **vncHost**.indexOf(**':'**)) != -1) {  
 **try** {  
 **vncPort** += Integer.*parseInt*(**vncHost**.substring(ix + 1));  
 } **catch** (NumberFormatException e) {  
 **authenticator**.retry(**"VNC host-string format error"**);  
 **continue**;  
 }  
 **vncHost** = **vncHost**.substring(0, ix);  
 }  
  
 **try** {  
 **rfb** = **new** rfbProto(**sshHost**, **sshPort**, **sshUser**, **sshPasswd**, **vncHost**, **vncPort**, **this**);  
 } **catch** (java.net.ConnectException e) {  
 System.***out***.println(**"Connect..."**);  
 **authenticator**.retry(**"VNC host, connection refused"**);  
 **continue**;  
 } **catch** (java.io.IOException e) {  
 System.***out***.println(**"IO..."**);  
 **authenticator**.retry(**"SSH error: "** + e.getMessage());  
 **continue**;  
 }  
  
 **if**(**rfb**.connectAndAuthenticate(**authenticator**))  
 **break**;  
 }  
  
 **cont**.remove(**authenticator**);  
 }  
  
 **int** getInt(**byte**[] b, **int** o) {  
 **int** i = ( ( b[o + 0] & 0xff ) << 24 ) |  
 ( ( b[o + 1] & 0xff ) << 16 ) |  
 ( ( b[o + 2] & 0xff ) << 8 ) |  
 ( b[o + 3] & 0xff );  
 **return** i;  
 }  
  
 **void** putInt(**byte**[] b, **int** o, **int** i) {  
 b[o + 0] = (**byte**) ( i >>> 24 );  
 b[o + 1] = (**byte**) ( i >>> 16 );  
 b[o + 2] = (**byte**) ( i >>> 8 );  
 b[o + 3] = (**byte**) i;  
 }  
  
 **public void** setCutText(String text) {  
 **try** {  
 **if** ((**rfb** != **null**) && **rfb**.**inNormalProtocol**) {  
 **rfb**.writeClientCutText(text);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
   
 **public void** buttonPressed(**int** buttonNum) {  
 **switch**(buttonNum) {  
 **case** 0:  
 **rfb**.close();  
 **rfb** = **null**;  
 **break**;  
 **case** 1:  
 **if** (**options**.isVisible()) {  
 **options**.hide();  
 } **else** {  
 **options**.show();  
 }  
 **break**;  
 **case** 2:  
 **if** (**clipboard**.isVisible()) {  
 **clipboard**.hide();  
 } **else** {  
 **clipboard**.show();  
 }  
 **break**;  
 **case** 3:  
 **rfb**.sendCtrlAltDel();  
 }  
 }  
   
 **public boolean gotFocus** = **false**;  
  
 **public boolean** gotFocus(Event evt, Object what) {  
 **gotFocus** = **true**;  
 **return true**;  
 }  
 **public boolean** lostFocus(Event evt, Object what) {  
 **gotFocus** = **false**;  
 **return true**;  
 }  
   
 **void** encryptBytes(**byte**[] bytes, String passwd) {  
 **byte**[] key = **new byte**[8];  
 passwd.getBytes(0, passwd.length(), key, 0);  
  
 **for** (**int** i = passwd.length(); i < 8; i++) {  
 key[i] = (**byte**)0;  
 }  
  
 DesCipher des = **new** DesCipher(key);  
  
 des.encrypt(bytes,0,bytes,0);  
 des.encrypt(bytes,8,bytes,8);  
 }  
  
   
 **public void** readParameters() {  
 **sshHost** = readParameter(**"sshhost"**, !**inAnApplet**);  
 **if** (**sshHost** == **null**) {  
 **sshHost** = getCodeBase().getHost();  
 **if** (**sshHost**.equals(**""**)) {  
 fatalError(**"Not able to determine sshhost"**);  
 }  
 }  
  
 String s = readParameter(**"sshport"**, **false**);  
 **if**(s != **null**)  
 **sshPort** = Integer.*parseInt*(s);  
 **else  
 sshPort** = SSH.***DEFAULTPORT***;  
  
 **try** {  
 **separateFrame** = (**new** Boolean(getParameter(**"sepframe"**))).booleanValue();  
 } **catch** (Exception e) {  
 **separateFrame** = **true**;  
 }  
  
 }  
  
 **public** String readParameter(String name, **boolean** required) {  
 **if** (**inAnApplet**) {  
 String s = getParameter(name);  
 **if** ((s == **null**) && required) {  
 fatalError(name + **" parameter not specified"**);  
 }  
 **return** s;  
 }  
  
 **for** (**int** i = 0; i < **mainArgs**.**length**; i += 2) {  
 **if** (**mainArgs**[i].equalsIgnoreCase(name)) {  
 **try** {  
 **return mainArgs**[i+1];  
 } **catch** (Exception e) {  
 **if** (required) {  
 fatalError(name + **" parameter not specified"**);  
 }  
 **return null**;  
 }  
 }  
 }  
 **if** (required) {  
 fatalError(name + **" parameter not specified"**);  
 }  
 **return null**;  
 }  
  
 **public void** fatalError(String s) {  
 System.***out***.println(s);  
  
 **if** (**inAnApplet**) {  
 **cont**.removeAll();  
 Label l = **new** Label(s);  
  
 setLayout(**new** FlowLayout(FlowLayout.***LEFT***, 30, 30));  
 **cont**.add(l);  
 validate();  
 Thread.*currentThread*().stop();  
 } **else** {  
 System.*exit*(1);  
 }  
 }  
}