**modified: sip-communicator/src/net/java/sip/communicator/SipCommunicator.java**

```
@ SipCommunicator.java:2 @
/* ================================================================
 * The Apache Software License, Version 1.1
 *  * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
@ SipCommunicator.java:65 @ import java.lang.reflect.*;
import java.net.*;
import java.util.*;
import java.awt.*;

import net.java.sip.communicator.common.*;
import net.java.sip.communicator.common.Console;
import net.java.sip.communicator.gui.*;
@ SipCommunicator.java:75 @ import net.java.sip.communicator.media.event.*;
import net.java.sip.communicator.sip.*;
import net.java.sip.communicator.sip.event.*;
import net.java.sip.communicator.sip.security.*;

import java.io.IOException;

import net.java.sip.communicator.plugin.setup.*;
import net.java.sip.communicator.sip.simple.*;

@ SipCommunicator.java:118 @ public class SipCommunicator

            guiManager = new GuiManager();
            mediaManager = new MediaManager();
            sipManager = new SipManager();
            sipManager = new SipManager(guiManager);
            simpleContactList = new SimpleContactList();

            guiManager.addUserActionListener(this);
@ SipCommunicator.java:139 @ public class SipCommunicator
        }
    }

    /**
     *
     */
    public void launch()
    {
        try {
@ SipCommunicator.java:213 @ public class SipCommunicator
                    exc);
            }


            /**
             * Dummy call to forward to ensure functionality
             */
//            try {
//              sipManager.forward("dfs");
//            }
//            catch (CommunicationsException exc) {
//                console.error(
//                        "An exception occurred while trying to set call forwarding, exc");
//                    console.showException(
//                        "Failed to set call forwarding!\n"
//                        + exc.getMessage() + "\n"
//                        + "This is a warning only. The phone would still function",
//                        exc);
//                }


//            try {
//              sipManager.block("bfs");
//            }
//            catch (CommunicationsException exc) {
//                console.error(
//                        "An exception occurred while trying to set call forwarding, exc");
//                    console.showException(
//                        "Failed to set call forwarding!\n"
//                        + exc.getMessage() + "\n"
//                        + "This is a warning only. The phone would still function",
//                        exc);
//                }

            boolean startSimple = false;
            try {
                startSimple = Boolean.valueOf(Utils.getProperty(
@ SipCommunicator.java:401 @ public class SipCommunicator
        }
    }

    /**
     * TODO: Implement for UserForwardRequest
     * TODO: Also implement handleDisableForwardEvent
     */
    public void handleForwardRequest(UserForwardEvent evt)
    {
        try {
            console.logEntry();
            String forwardee = (String) evt.getSource();

                try {
                sipManager.forward(forwardee);
            }
            catch (CommunicationsException exc) {
                console.error(
                        "An exception occurred while trying to set call forwarding, exc");
                    console.showException(
                        "Failed to set call forwarding!\n"
                        + exc.getMessage() + "\n"
                        + "This is a warning only. The phone would still function",
                        exc);
                }

        }
        finally {
            console.logExit();
        }
    }

    /**
     * TODO: Implement for UserBlockRequest
     * TODO: Also implement handleUnblockEvent
     */
    public void handleBlockRequest(UserBlockEvent evt)
    {
        try {
            console.logEntry();
            String blockee = (String) evt.getSource();

                try {
                sipManager.block(blockee);
            }
            catch (CommunicationsException exc) {
                console.error(
                        "An exception occurred while trying to set blocking, exc");
                    console.showException(
                        "Failed to set blocking!\n"
                        + exc.getMessage() + "\n"
                        + "This is a warning only. The phone would still function",
                        exc);
                }
        }
        finally {
            console.logExit();
        }

    }

    public void handleDisableForwardRequest(UserDisableForwardEvent evt){
```

```
        try {
                console.logEntry();
                sipManager.unforward();
        }
        catch (CommunicationsException exc) {
            console.error(
                    "An exception occurred while trying to disable call forwarding, exc");
                console.showException(
                    "Failed to disable call forwarding!\n"
                    + exc.getMessage() + "\n"
                    + "This is a warning only. The phone would still function",
                    exc);
            }
        finally {
            console.logExit();
        }
    }


    public void handleUnblockRequest(UserUnblockEvent evt){
        try {
                console.logEntry();
            String blockee = (String) evt.getSource();

                try {
                sipManager.unblock(blockee);
            }
            catch (CommunicationsException exc) {
                console.error(
                        "An exception occurred while trying to set blocking, exc");
                    console.showException(
                        "Failed to set blocking!\n"
                        + exc.getMessage() + "\n"
                        + "This is a warning only. The phone would still function",
                        exc);
                }
        }
        finally {
            console.logExit();
        }
    }

    public void handleDialRequest(UserCallInitiationEvent evt)
    {
        try {
@ SipCommunicator.java:535 @ public class SipCommunicator
            console.logExit();
        }
    }


    public void handleHangupRequest(UserCallControlEvent evt)
    {
        try {
            console.logEntry();
            /**
             * TODO: Send the call time to the server
             */
            sipManager.endCall(evt.getAssociatedInterlocutor().getID());
            //no further action should be taken here. Close
            //(mediaManager.closeStream guiManager.removeInterlocutor)
@ SipCommunicator.java:804 @ public class SipCommunicator
            console.logExit();
        }
    }

    public void blocking(BlockEvent evt){
        try {
            console.logEntry();
            String blockee = evt.getReason().split(":")[1].split("@")[0];
            guiManager.phoneFrame.blockPanel.remove(guiManager.phoneFrame.unblockButton);
            guiManager.phoneFrame.unblockButton.addItem(blockee);
            guiManager.phoneFrame.blockPanel.add(guiManager.phoneFrame.unblockButton, BorderLayout.WEST);
        }
        finally {
            console.logExit();
        }
    }

    public void unblockingGui(BlockEvent evt){
        try{
                console.logEntry();
                String unblock_usr = evt.getReason().split(":")[1].split("@")[0];

                if(!"Blocked People".equals(unblock_usr)){
                        console.debug("Se gamaw" + unblock_usr);
                        guiManager.phoneFrame.blockPanel.remove(guiManager.phoneFrame.unblockButton);
                        guiManager.phoneFrame.unblockButton.setSelectedItem("Blocked People");
                        guiManager.phoneFrame.unblockButton.removeItem(unblock_usr);

                        guiManager.phoneFrame.blockPanel.add(guiManager.phoneFrame.unblockButton, BorderLayout.WEST);

                }
        }
        finally{
                console.logExit();
        }
    }

    public void forwardedOKGuiChange(ForwardEvent evt){
        try {
            console.logEntry();
            if (evt.getReason() != "None"){
                guiManager.phoneFrame.frwl.setText("Forward to: " + evt.getReason().split("@")[0].split(":")[1]);
            }else{
                guiManager.phoneFrame.frwl.setText("Forward to: " + evt.getReason());
            }
        }
        finally {
            console.logExit();
        }

    }

    public void unregistered(RegistrationEvent evt)
    {
@ SipCommunicator.java:1160 @ public class SipCommunicator

        return SubscriptionAuthorizationResponse.createResponse(response);
    }
}
        }
```

**modified: sip-communicator/src/net/java/sip/communicator/gui/AuthenticationSplash.java**

```
@ AuthenticationSplash.java:90 @ public class AuthenticationSplash
     */
    private ResourceBundle resources = null;


    /**
     * Path to the image resources
     */
@ AuthenticationSplash.java:465 @ public class AuthenticationSplash
        else if (cmd.equals(CMD_LOGIN)) {
            userName = userNameTextField.getText();
            password = passwordTextField.getPassword();
            System.out.println(new String(password));
        }
        setVisible(false);
        dispose();
```

**modified: sip-communicator/src/net/java/sip/communicator/gui/GuiManager.java**

```
@ GuiManager.java:63 @ package net.java.sip.communicator.gui;
import java.util.*;
```

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import net.java.sip.communicator.common.*;
import net.java.sip.communicator.common.Console;
import net.java.sip.communicator.gui.event.*;

import java.awt.SystemColor;

import javax.swing.plaf.metal.MetalLookAndFeel;

import net.java.sip.communicator.gui.plaf.SipCommunicatorColorTheme;

import java.awt.event.KeyEvent;
import java.io.*;

import net.java.sip.communicator.media.JMFRegistry;
import net.java.sip.communicator.plugin.setup.*;
import net.java.sip.communicator.gui.imp.*;
@ GuiManager.java:112 @ public class GuiManager
        initLookAndFeel();
    }

    private PhoneFrame       phoneFrame   = null;
    public PhoneFrame       phoneFrame   = null;
    private ContactListFrame contactList  = null;
    private ConfigFrame      configFrame  = null;
    private ArrayList        listeners    = null;
    public ContactListFrame  blocked      = null;
    private AlertManager     alertManager = null;

/** @todo remove after testing */
@ GuiManager.java:147 @ public class GuiManager
        contactList   = new ContactListFrame();
        configFrame   = new ConfigFrame(phoneFrame);
        listeners     = new ArrayList();
        blocked       = new ContactListFrame();
        alertManager  = new AlertManager();
        logoPanel     = new JPanel(new FlowLayout(FlowLayout.CENTER));
        interlocutors = new InterlocutorsTableModel();
@ GuiManager.java:289 @ public class GuiManager
    {
//        phoneFrame.contactBox.setEnabled(enabled);
        phoneFrame.dialButton.setEnabled(enabled);
        phoneFrame.forwardButton.setEnabled(enabled);
        phoneFrame.blockButton.setEnabled(enabled);
        phoneFrame.unblockButton.setEnabled(enabled);
        phoneFrame.disableForwardButton.setEnabled(enabled);
        phoneFrame.hangupButton.setEnabled(enabled);
        phoneFrame.answerButton.setEnabled(enabled);
    }
@ GuiManager.java:359 @ public class GuiManager
            ( (UserActionListener) listeners.get(i)).handleDialRequest(commEvt);
        }
    }

    void forwardButton_actionPerformed(EventObject evt)
    {
        //TODO temporarily close alerts from here.
        console.logEntry();
        alertManager.stopAllAlerts();
        String usr_who_forwards = phoneFrame.contactBox.getSelectedItem().toString();
        if (usr_who_forwards == null || usr_who_forwards.trim().length() < 1) {
            return;
        }
        UserForwardEvent frwEvt = new UserForwardEvent(usr_who_forwards);
        for (int i = listeners.size() - 1; i >= 0; i--) {
            ( (UserActionListener) listeners.get(i)).handleForwardRequest(frwEvt);
        }
        console.logExit();
    }

    void blockButton_actionPerformed(EventObject evt)
    {
        //TODO temporarily close alerts from here.
        alertManager.stopAllAlerts();
        String block_usr = phoneFrame.contactBox.getSelectedItem().toString();
        if (block_usr == null || block_usr.trim().length() < 1) {
            return;
        }
        UserBlockEvent blockEvt = new UserBlockEvent(block_usr);
        for (int i = listeners.size() - 1; i >= 0; i--) {
            ( (UserActionListener) listeners.get(i)).handleBlockRequest(blockEvt);
        }
    }

    void unblockButton_actionPerformed(EventObject evt)
    {
        //TODO temporarily close alerts from here.
        alertManager.stopAllAlerts();
        String unblock_usr = phoneFrame.unblockButton.getSelectedItem().toString();
        if (unblock_usr == null || unblock_usr.trim().length() < 1) {
            return;
        }
        UserUnblockEvent unblEvt = new UserUnblockEvent(unblock_usr);
        for (int i = listeners.size() - 1; i >= 0; i--) {
            ( (UserActionListener) listeners.get(i)).handleUnblockRequest(unblEvt);
        }

    }

    void disableforwardButton_actionPerformed(EventObject evt)
    {
        //TODO temporarily close alerts from here.
        alertManager.stopAllAlerts();
        UserDisableForwardEvent stpEvt = new UserDisableForwardEvent("unforward");
        for (int i = listeners.size() - 1; i >= 0; i--) {
            ( (UserActionListener) listeners.get(i)).handleDisableForwardRequest(stpEvt);
        }
    }

    void hangupButton_actionPerformed(ActionEvent evt)
    {
@ GuiManager.java:700 @ public class GuiManager
            hangupButton_actionPerformed(evt);
            }
        });
        phoneFrame.forwardButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent evt)
            {
                forwardButton_actionPerformed(evt);
            }
        });
        phoneFrame.disableForwardButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent evt)
            {
                disableforwardButton_actionPerformed(evt);
            }
        });
        phoneFrame.blockButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent evt)
            {
                blockButton_actionPerformed(evt);
            }
        });
        phoneFrame.unblockButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent evt)
            {
                unblockButton_actionPerformed(evt);
            }
        });
        phoneFrame.addWindowListener(new WindowAdapter()
```

```
        {
            public void windowClosing(WindowEvent evt)
@ GuiManager.java:762 @ public class GuiManager
                                    String userName,
                                    char[] password)
    {
        console.logEntry();
        if (authenticationSplash != null)
            authenticationSplash.dispose();
        authenticationSplash = new AuthenticationSplash(phoneFrame, true);
@ GuiManager.java:775 @ public class GuiManager
        if(realm != null)
            authenticationSplash.realmValueLabel.setText(new String(realm));
        authenticationSplash.show();
        console.debug(authenticationSplash.userName);
        console.debug(new String(authenticationSplash.password));
        console.logExit();
    }

    public String getAuthenticationUserName()
```

---

**modified: sip-communicator/src/net/java/sip/communicator/gui/PhoneFrame.java**

---

```
@ PhoneFrame.java:76 @ import javax.swing.border.*;
 * @version 1.1
 *
 */
class PhoneFrame
public class PhoneFrame
    extends JFrame
{
    BorderLayout borderLayout1 = new BorderLayout();
@ PhoneFrame.java:98 @ class PhoneFrame
    GridLayout gridLayout1 = new GridLayout();
    Border border6;
    Border border7;
    Border border71;
    Border border8;
    JTable participantsTable = new JTable(30, 3);
    JMenuBar jMenuBar1 = new net.java.sip.communicator.gui.MenuBar();
@ PhoneFrame.java:114 @ class PhoneFrame
    Border border14;
    BoxLayout boxLayout21 = new BoxLayout(videoPane, BoxLayout.Y_AXIS);
    Border border15;
    public String frwed = "None yet";

    private GuiManager guiManCallback = null;
    BorderLayout borderLayout4 = new BorderLayout();
    BorderLayout borderLayout41 = new BorderLayout();

    /**
     * Dial GUI
     */
    JPanel dialPanel = new JPanel();
    JButton dialButton = new JButton();
    JComboBox contactBox = new JComboBox();


    /**
     * Call Forward GUI
     */
    JPanel forwardPanel = new JPanel();
    JButton forwardButton = new JButton();
    JButton disableForwardButton = new JButton();
    JComboBox forwardBox = new JComboBox();

    //BorderLayout borderLayoutDialForward = new BorderLayout();
    //JPanel dialAndForwardPanel = new JPanel();

    /**
     * Block & Unblock GUI
     */
    public JPanel blockPanel = new JPanel();
    JButton blockButton = new JButton();

    String[] blockedpeople = {"Blocked People"};
    public JComboBox unblockButton = new JComboBox(blockedpeople);

    JPanel forwardl = new JPanel();
    public JLabel frwl = new JLabel();
    Font myFont = new Font("Serif", Font.ITALIC | Font.BOLD, 12);

    BorderLayout borderLayoutDialForwardBlock = new BorderLayout();
    JPanel dialForwardBlockPanel = new JPanel();

    public PhoneFrame(GuiManager guiManCallback) //throws HeadlessException
    {
        try {
@ PhoneFrame.java:173 @ class PhoneFrame
            e.printStackTrace();
        }
    }

    private void setupDialAndForwardPanel() throws Exception{
        borderLayoutDialForwardBlock.setHgap(10);
        dialForwardBlockPanel.setLayout(borderLayoutDialForwardBlock);
        dialForwardBlockPanel.setLayout(borderLayoutDialForwardBlock);
        dialForwardBlockPanel.setBorder(border8);
        dialForwardBlockPanel.setBorder(border8);

        borderLayout4.setHgap(10);
        dialPanel.setLayout(borderLayout4);
        dialPanel.setBorder(border7);
        dialButton.setEnabled(false);
        dialButton.setMnemonic('D');
        dialButton.setText("Dial");

        contactBox.setBorder(null);
        contactBox.setDebugGraphicsOptions(0);
        contactBox.setActionMap(null);
        contactBox.setEditable(true);

        borderLayout41.setHgap(10);
        forwardPanel.setLayout(borderLayout41);
        forwardPanel.setBorder(border71);

        borderLayout41.setHgap(10);
        forwardl.setLayout(borderLayout41);
        forwardl.setBorder(border71);

        forwardButton.setEnabled(false);
        forwardButton.setMnemonic('F');
        forwardButton.setText("Forward");
        disableForwardButton.setEnabled(false);
        disableForwardButton.setMnemonic('D');
        disableForwardButton.setText("Disable Forward");


        frwl.setText("Forward to: "+frwed);
        frwl.setFont(myFont);

        blockButton.setEnabled(false);
        blockButton.setMnemonic('B');
        blockButton.setText("Block");
        unblockButton.setEnabled(false);
        //unblockButton.setRenderer(new MyComboBoxRenderer("Unblock"));
        unblockButton.setSelectedIndex(0);
        //unblockButton.addIt

        this.getContentPane().add(dialForwardBlockPanel, BorderLayout.NORTH);
        dialForwardBlockPanel.add(dialPanel, BorderLayout.NORTH);

        dialForwardBlockPanel.add(forwardPanel, BorderLayout.CENTER);
        dialForwardBlockPanel.add(blockPanel, BorderLayout.SOUTH);

        dialPanel.add(dialButton, BorderLayout.EAST);
        dialPanel.add(contactBox, BorderLayout.CENTER);
```

```
        forwardPanel.add(forwardButton, BorderLayout.EAST);
        forwardPanel.add(disableForwardButton, BorderLayout.WEST);
        forwardPanel.add(frwl, BorderLayout.SOUTH);

        blockPanel.add(blockButton, BorderLayout.EAST);
        blockPanel.add(unblockButton, BorderLayout.WEST);


    }

    private void jbInit() throws Exception
    {
@ PhoneFrame.java:255 @ class PhoneFrame
        border5 = BorderFactory.createEmptyBorder(5, 5, 5, 5);
        border6 = BorderFactory.createEmptyBorder(5, 5, 5, 5);
        border7 = BorderFactory.createEmptyBorder(5, 5, 5, 5);
        border71 = BorderFactory.createEmptyBorder(5, 5, 5, 5);
        border8 = BorderFactory.createEmptyBorder(5, 5, 5, 5);
        border9 = BorderFactory.createEmptyBorder(10, 10, 10, 10);

        border14 = BorderFactory.createEmptyBorder(4, 0, 0, 0);
        this.getContentPane().setLayout(borderLayout1);
        splitPane.setOrientation(JSplitPane.VERTICAL_SPLIT);
@ PhoneFrame.java:296 @ class PhoneFrame
        registrationLabel.setText("Not Registered");
        statusPanel.setLayout(borderLayout5);
//        participantsTable.setMinimumSize(new Dimension(45, 300));
        borderLayout4.setHgap(10);
        dialPanel.setLayout(borderLayout4);
        dialPanel.setBorder(border7);
        dialButton.setEnabled(false);
        dialButton.setMnemonic('D');
        dialButton.setText("Dial");
        contactBox.setBorder(null);
        contactBox.setDebugGraphicsOptions(0);
        contactBox.setActionMap(null);
        contactBox.setEditable(true);



        this.getContentPane().add(splitPane, BorderLayout.CENTER);
        splitPane.add(controlPanel, JSplitPane.BOTTOM);
        splitPane.add(videoPane, JSplitPane.TOP);
@ PhoneFrame.java:323 @ class PhoneFrame
        participantsScroll.setViewportView(participantsTable);
        statusPanel.add(registrationLabel, BorderLayout.WEST);
        statusPanel.add(registrationAddressLabel, BorderLayout.CENTER);
        this.getContentPane().add(dialPanel, BorderLayout.NORTH);
        dialPanel.add(dialButton, BorderLayout.EAST);
        dialPanel.add(contactBox, BorderLayout.CENTER);

        setupDialAndForwardPanel();

//        splitPane.setDividerLocation(200);
    }

@ PhoneFrame.java:338 @ class PhoneFrame
//            System.exit(0);
//        }
//    }


}
```

**modified: sip-communicator/src/net/java/sip/communicator/gui/event/UserActionListener.java**

```
@ UserActionListener.java:75 @ public interface UserActionListener
    extends java.util.EventListener
{
    public void handleDialRequest(UserCallInitiationEvent evt);

    public void handleDisableForwardRequest(UserDisableForwardEvent evt);

    public void handleForwardRequest(UserForwardEvent evt);

    public void handleBlockRequest(UserBlockEvent evt);

    public void handleUnblockRequest(UserUnblockEvent evt);

    public void handleHangupRequest(UserCallControlEvent evt);
```

**added: sip-communicator/src/net/java/sip/communicator/gui/event/UserBlockEvent.java**

```
@ UserBlockEvent.java:4 @
/* ====================================================================

 * The Apache Software License, Version 1.1

 *

 * Copyright (c) 2000 The Apache Software Foundation.  All rights

 * reserved.

 *

 * Redistribution and use in source and binary forms, with or without

 * modification, are permitted provided that the following conditions

 * are met:

 *

 * 1. Redistributions of source code must retain the above copyright

 *    notice, this list of conditions and the following disclaimer.

 *

 * 2. Redistributions in binary form must reproduce the above copyright

 *    notice, this list of conditions and the following disclaimer in

 *    the documentation and/or other materials provided with the

 *    distribution.

 *

 * 3. The end-user documentation included with the redistribution,

 *    if any, must include the following acknowledgment:

 *       "This product includes software developed by the

 *        Apache Software Foundation (http://www.apache.org/)."

 *    Alternately, this acknowledgment may appear in the software itself,

 *    if and wherever such third-party acknowledgments normally appear.

 *

 * 4. The names "Apache" and "Apache Software Foundation" must

 *    not be used to endorse or promote products derived from this

 *    software without prior written permission. For written

 *    permission, please contact apache@apache.org.

 *

 * 5. Products derived from this software may not be called "Apache",

 *    nor may "Apache" appear in their name, without prior written
```

```
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
package net.java.sip.communicator.gui.event;

import java.util.*;

/**
 * <p>Title: SIP COMMUNICATOR</p>
 * <p>Description:JAIN-SIP Audio/Video phone application</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * @version 1.1
 *
 */
public class UserBlockEvent
    extends EventObject
{
    public UserBlockEvent(String blocked)
    {
        super(blocked);
    }
}
\ No newline at end of file
```

**added: sip-communicator/src/net/java/sip/communicator/gui/event/UserDisableForwardEvent.java**

```
@ UserDisableForwardEvent.java:4 @
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
```

```
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
package net.java.sip.communicator.gui.event;

import java.util.*;

/**
 * <p>Title: SIP COMMUNICATOR</p>
 * <p>Description:JAIN-SIP Audio/Video phone application</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * @version 1.1
 *
 */

public class UserDisableForwardEvent
extends EventObject{

        public UserDisableForwardEvent(String unforward){
                super(unforward);
        }
}
\ No newline at end of file
```

**added: sip-communicator/src/net/java/sip/communicator/gui/event/UserForwardEvent.java**

```
@ UserForwardEvent.java:4 @
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
```

```
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
package net.java.sip.communicator.gui.event;

import java.util.*;

/**
 * <p>Title: SIP COMMUNICATOR</p>
 * <p>Description:JAIN-SIP Audio/Video phone application</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * @version 1.1
 *
 */

public class UserForwardEvent
extends EventObject{

        public UserForwardEvent(String usr_who_forwards){

                super(usr_who_forwards);
        }
}
\ No newline at end of file
```

**added: sip-communicator/src/net/java/sip/communicator/gui/event/UserUnblockEvent.java**

```
@ UserUnblockEvent.java:4 @
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
```

```
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
package net.java.sip.communicator.gui.event;


import java.util.*;


/**
 * <p>Title: SIP COMMUNICATOR</p>
 * <p>Description:JAIN-SIP Audio/Video phone application</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * @version 1.1
 *
 */
public class UserUnblockEvent
    extends EventObject
{
    public UserUnblockEvent(String unblock)
    {
        super(unblock);
    }
}
\ No newline at end of file
```

**modified: sip-communicator/src/net/java/sip/communicator/sip/Call.java**

```
@ Call.java:106 @ public class Call
     */
    private Request initialRequest = null;
    private String callState = "";

    /**
     * Time of connection so that billing can occur
     */
    private long timeOfConnection = 0;
    private long timeOfDisconnection = 0;

    //Event Management
    ArrayList listeners = new ArrayList();
    public String getState()
@ Call.java:149 @ public class Call
        return remoteSdpDescription;
    }

    void setState(String newStatus)
    void setState(String newStatus)
    {
        try
        {
@ Call.java:160 @ public class Call

            if( console.isDebugEnabled() )
                console.debug("setting call status to "+newStatus);


            /**
```

```
             * When the call changes to connected we must start the time counter
             */
            if(newStatus.equals(Call.CONNECTED) && !callState.equals(Call.CONNECTED)){
                timeOfConnection = System.currentTimeMillis();

            }

            String oldStatus = callState;
            this.callState = newStatus;
            fireCallStatusChangedEvent(oldStatus);
@ Call.java:240 @ public class Call
    {
        this.initialRequest = request;
    }

    Request getInitialRequest()
    {
        return this.initialRequest;
    }

    String getDialogID()
    {
@ Call.java:273 @ public class Call
            ( (CallListener) listeners.get(i)).callStateChanged(evt);
        }
    }

        public long getTimeOfConnection() {
                return timeOfConnection;
        }

        public void setTimeOfConnection(long timeOfConnection) {
                this.timeOfConnection = timeOfConnection;
        }
}
\ No newline at end of file
```

**added: sip-communicator/src/net/java/sip/communicator/sip/CallBlockProcessing.java**

```
@ CallBlockProcessing.java:4 @
package net.java.sip.communicator.sip;

import java.text.ParseException;
import java.util.ArrayList;

import javax.sip.ClientTransaction;
import javax.sip.InvalidArgumentException;
import javax.sip.TransactionUnavailableException;
import javax.sip.address.Address;
import javax.sip.address.URI;
import javax.sip.header.CSeqHeader;
import javax.sip.header.CallIdHeader;
import javax.sip.header.ContactHeader;
import javax.sip.header.ExpiresHeader;
import javax.sip.header.FromHeader;
import javax.sip.header.MaxForwardsHeader;
import javax.sip.header.ToHeader;
import javax.sip.message.Request;
import javax.sip.message.Response;

import net.java.sip.communicator.common.Console;

public class CallBlockProcessing {
        private static final Console console =
                Console.getConsole(CallBlockProcessing.class);
        private SipManager sipManCallback = null;
        private Request blockRequest = null;
        private boolean forwardeeURI = false;

        CallBlockProcessing(SipManager sipManCallback)
    {
        this.sipManCallback = sipManCallback;
    }

    void setSipManagerCallBack(SipManager sipManCallback)
    {
        this.sipManCallback = sipManCallback;
    }

    void blockOK(ClientTransaction clientTransaction, Response response)
    {
        try {
            console.logEntry();

            // Get the request that was accepted
            Request request = clientTransaction.getRequest();

            // Save the forwardee URI
            URI forwardeeURI = request.getRequestURI();

            // Fire forwarded ok to pass the information to all listeners
            sipManCallback.fireBlocked(forwardeeURI.toString());

        }
        finally {
            console.logExit();
        }
    }

    void unblockOK(ClientTransaction clientTransaction, Response response)
    {
        try {
            console.logEntry();

            // Get the request that was accepted
            Request request = clientTransaction.getRequest();

            // Save the forwardee URI
            URI forwardeeURI = request.getRequestURI();

            // Fire forwarded ok to pass the information to all listeners
            sipManCallback.fireUnblocked(forwardeeURI.toString());

        }
        finally {
            console.logExit();
        }
    }



        synchronized void block(String registrarAddress, int registrarPort,
            String registrarTransport, int expires, String addressToBlock, String blockOrUnblock) throws
           CommunicationsException
        {
          try
          {
              console.logEntry();
              console.debug("Address to block: " + addressToBlock);

              //Get the From Header and address
              FromHeader fromHeader = sipManCallback.getFromHeader();
              Address fromAddress = fromHeader.getAddress();
              console.debug("From Header: " + fromHeader);

              /**
               * TODO: GUI CALL Fix later
               */
              //sipManCallback.fireRegistering(fromAddress.toString());

              //Make the blockee address in a URI
          URI requestURI = ProcessingUtilities.createUriFromAddress(addressToBlock, sipManCallback, console);
              CallIdHeader callIdHeader = sipManCallback.sipProvider.getNewCallId();
              CSeqHeader cSeqHeader = ProcessingUtilities.safeCSeqHeader(1, Request.UPDATE, sipManCallback, console);
              ToHeader toHeader = ProcessingUtilities.headerFromAddress(fromAddress, sipManCallback, console);
              ArrayList viaHeaders = sipManCallback.getLocalViaHeaders();
              MaxForwardsHeader maxForwardsHeader = sipManCallback.
```

```
                getMaxForwardsHeader();

            /**
             * Make the request and then add an expires and a contact header
             */
            Request request = null;
            if(blockOrUnblock == "block"){
                    try {
                        request = sipManCallback.messageFactory.createRequest(requestURI,
                            "BLOCK ",
                            callIdHeader,
                            cSeqHeader, fromHeader, toHeader,
                            viaHeaders,
                            maxForwardsHeader);
                    }
                    catch (ParseException ex) {
                        console.error("Could not create the register request!", ex);
                        //throw was missing - reported by Eero Vaarnas
                        throw new CommunicationsException(
                            "Could not create the register request!",
                            ex);
                    }
            }
            else{
                try {
                        request = sipManCallback.messageFactory.createRequest(requestURI,
                            "UNBLOCK ",
                            callIdHeader,
                            cSeqHeader, fromHeader, toHeader,
                            viaHeaders,
                            maxForwardsHeader);
                    }
                    catch (ParseException ex) {
                        console.error("Could not create the register request!", ex);
                        //throw was missing - reported by Eero Vaarnas
                        throw new CommunicationsException(
                            "Could not create the register request!",
                            ex);
                    }
            }

            //Expires Header
            ExpiresHeader expHeader = null;
            for (int retry = 0; retry < 2; retry++) {
                try {
                    expHeader = sipManCallback.headerFactory.createExpiresHeader(
                        expires);
                }
                catch (InvalidArgumentException ex) {
                    if (retry == 0) {
                        expires = 3600;
                        continue;
                    }
                    console.error(
                        "Invalid registrations expiration parameter - "
                        + expires,
                        ex);
                    throw new CommunicationsException(
                        "Invalid registrations expiration parameter - "
                        + expires,
                        ex);
                }
            }
            request.addHeader(expHeader);
            //Contact Header should contain IP - bug report - Eero Vaarnas
            ContactHeader contactHeader = sipManCallback.
                getRegistrationContactHeader();
            request.addHeader(contactHeader);

            console.debug(request);

            //Transaction
            ClientTransaction blockTrans = null;
            try {
                blockTrans = sipManCallback.sipProvider.getNewClientTransaction(
                    request);
            }
            catch (TransactionUnavailableException ex) {
                console.error("Could not create a block transaction!\n"
                              + "Check that the Registrar address is correct!",
                              ex);
                //throw was missing - reported by Eero Vaarnas
                throw new CommunicationsException(
                    "Could not create a block transaction!\n"
                    + "Check that the Registrar address is correct!");
            }
            try {
                blockTrans.sendRequest();
                if( console.isDebugEnabled() )
                    console.debug("sent request= " + request);
                //[issue 2] Schedule re registrations
                //bug reported by LynlvL@netscape.com

                //scheduleReRegistration( registrarAddress, registrarPort,
                //            registrarTransport, expires);

            }
            //we sometimes get a null pointer exception here so catch them all
            catch (Exception ex) {
                console.error("Could not send out the block request!", ex);
                //throw was missing - reported by Eero Vaarnas
                throw new CommunicationsException(
                    "Could not send out the block request!", ex);
            }
            console.debug(blockTrans);
            this.blockRequest = request;

          }
          finally
          {
              console.logExit();
          }

        }

}
```

**added: sip-communicator/src/net/java/sip/communicator/sip/CallForwardProcessing.java**

```
@ CallForwardProcessing.java:4 @
package net.java.sip.communicator.sip;

import java.text.ParseException;
import java.util.ArrayList;
import java.util.Timer;

import javax.sip.ClientTransaction;
import javax.sip.InvalidArgumentException;
import javax.sip.TransactionUnavailableException;
import javax.sip.address.Address;
import javax.sip.address.SipURI;
import javax.sip.address.URI;
import javax.sip.header.CSeqHeader;
import javax.sip.header.CallIdHeader;
import javax.sip.header.ContactHeader;
import javax.sip.header.ExpiresHeader;
import javax.sip.header.FromHeader;
import javax.sip.header.MaxForwardsHeader;
import javax.sip.header.ToHeader;
import javax.sip.message.Request;
import javax.sip.message.Response;

import net.java.sip.communicator.common.Console;

public class CallForwardProcessing {

        private static final Console console =
```

```
            Console.getConsole(CallForwardProcessing.class);
    private SipManager sipManCallback = null;
    private Request forwardRequest = null;
    private boolean forwardeeURI = false;

    CallForwardProcessing(SipManager sipManCallback)
{
    this.sipManCallback = sipManCallback;
}

void setSipManagerCallBack(SipManager sipManCallback)
{
    this.sipManCallback = sipManCallback;
}

void forwardOK(ClientTransaction clientTransaction, Response response)
{
    try {
        console.logEntry();

        // Get the request that was accepted
        Request request = clientTransaction.getRequest();

        // Save the forwardee URI
        URI forwardeeURI = request.getRequestURI();

        // Fire forwarded ok to pass the information to all listeners
        sipManCallback.fireEnabledForward(forwardeeURI.toString());

    }
    finally {
        console.logExit();
    }
}

void unforwardOK(ClientTransaction clientTransaction, Response response)
{
    try {
        console.logEntry();

        // Get the request that was accepted
        Request request = clientTransaction.getRequest();

        // Save the forwardee URI
        URI forwardeeURI = request.getRequestURI();

        // Fire forwarded ok to pass the information to all listeners
        sipManCallback.fireDisabledForward("None");

    }
    finally {
        console.logExit();
    }
}

/**
 * Create URI from Address
 * @param addressToForward
 * @return
 * @throws CommunicationsException
 */
URI createUriFromAddress(String addressToForward) throws CommunicationsException
{
    URI requestURI;
    try {
        requestURI = sipManCallback.addressFactory.createURI(addressToForward);
    }
    catch (ParseException ex) {
        console.error(addressToForward + " is not a legal SIP uri!", ex);
        throw new CommunicationsException(addressToForward +
                                          " is not a legal SIP uri!", ex);
    }
    return requestURI;
}


    synchronized void forward(String registrarAddress, int registrarPort,
        String registrarTransport, int expires, String addressToForward) throws
       CommunicationsException
    {
      try
      {
          console.logEntry();
          console.debug("Address to forward: " + addressToForward);

          //From
          FromHeader fromHeader = sipManCallback.getFromHeader();
          Address fromAddress = fromHeader.getAddress();
          console.debug("From Header: " + fromHeader);

          /**
           * TODO: GUI CALL Fix later
           */
          //sipManCallback.fireRegistering(fromAddress.toString());
          CallIdHeader callIdHeader = sipManCallback.sipProvider.getNewCallId();
          CSeqHeader cSeqHeader = ProcessingUtilities.safeCSeqHeader(1, Request.UPDATE, sipManCallback, console);
          ToHeader toHeader = ProcessingUtilities.headerFromAddress(fromAddress, sipManCallback, console);
          ArrayList viaHeaders = sipManCallback.getLocalViaHeaders();
          MaxForwardsHeader maxForwardsHeader = sipManCallback.
              getMaxForwardsHeader();
          Request request = null;

          /**
           * Check if forward or unforward
           */
          if (addressToForward != null) {
                  URI requestURI = ProcessingUtilities.createUriFromAddress(addressToForward, sipManCallback, console);

                  /**
                   * Make the request and then add an expires and a contact header
                   */

                  try {
                      request = sipManCallback.messageFactory.createRequest(requestURI,
                          "FORWARD ",
                          callIdHeader,
                          cSeqHeader, fromHeader, toHeader,
                          viaHeaders,
                          maxForwardsHeader);
                  }
                  catch (ParseException ex) {
                      console.error("Could not create the register request!", ex);
                      //throw was missing - reported by Eero Vaarnas
                      throw new CommunicationsException(
                          "Could not create the register request!",
                          ex);
                  }
          }
          else{
              URI requestURI = fromAddress.getURI();

                  /**
                   * Make the request and then add an expires and a contact header
                   */
                  try {
                      request = sipManCallback.messageFactory.createRequest(requestURI,
                          "UNFORWARD ",
                          callIdHeader,
                          cSeqHeader, fromHeader, toHeader,
                          viaHeaders,
                          maxForwardsHeader);
                  }
                  catch (ParseException ex) {
                      console.error("Could not create the register request!", ex);
                      //throw was missing - reported by Eero Vaarnas
                      throw new CommunicationsException(
                          "Could not create the register request!",
                          ex);
```

```
                }
            }

            //Expires Header
            ExpiresHeader expHeader = null;
            for (int retry = 0; retry < 2; retry++) {
                try {
                    expHeader = sipManCallback.headerFactory.createExpiresHeader(
                        expires);
                }
                catch (InvalidArgumentException ex) {
                    if (retry == 0) {
                        expires = 3600;
                        continue;
                    }
                    console.error(
                        "Invalid registrations expiration parameter - "
                        + expires,
                        ex);
                    throw new CommunicationsException(
                        "Invalid registrations expiration parameter - "
                        + expires,
                        ex);
                }
            }
            request.addHeader(expHeader);
            //Contact Header should contain IP - bug report - Eero Vaarnas
            ContactHeader contactHeader = sipManCallback.
                getRegistrationContactHeader();
            request.addHeader(contactHeader);

            console.debug(request);



            //Transaction
            ClientTransaction forwardTrans = null;
            try {
                forwardTrans = sipManCallback.sipProvider.getNewClientTransaction(
                    request);
            }
            catch (TransactionUnavailableException ex) {
                console.error("Could not create a forward transaction!\n"
                              + "Check that the Registrar address is correct!",
                              ex);
                //throw was missing - reported by Eero Vaarnas
                throw new CommunicationsException(
                    "Could not create a forward transaction!\n"
                    + "Check that the Registrar address is correct!");
            }
            try {
                forwardTrans.sendRequest();
                if( console.isDebugEnabled() )
                    console.debug("sent request= " + request);
                //[issue 2] Schedule re registrations
                //bug reported by LynlvL@netscape.com

                //scheduleReRegistration( registrarAddress, registrarPort,
                //            registrarTransport, expires);

            }
            //we sometimes get a null pointer exception here so catch them all
            catch (Exception ex) {
                console.error("Could not send out the forward request!", ex);
                //throw was missing - reported by Eero Vaarnas
                throw new CommunicationsException(
                    "Could not send out the forward request!", ex);
            }
            console.debug(forwardTrans);
            this.forwardRequest = request;

          }
          finally
          {
              console.logExit();
          }

        }

//      synchronized void unforward(String registrarAddress, int registrarPort,
//            String registrarTransport, int expires) throws
//         CommunicationsException
//      {
//        try
//        {
//            console.logEntry();
//
//
//            //From
//            FromHeader fromHeader = sipManCallback.getFromHeader();
//            Address fromAddress = fromHeader.getAddress();
//            console.debug("From Header: " + fromHeader);
//            console.debug("From Address: " + fromAddress.getURI());
//
//            /**
//             * TODO: GUI CALL Fix later
//             */
//            //sipManCallback.fireRegistering(fromAddress.toString());
//            URI requestURI = fromAddress.getURI();
//            CallIdHeader callIdHeader = sipManCallback.sipProvider.getNewCallId();
//            CSeqHeader cSeqHeader = ProcessingUtilities.safeCSeqHeader(1, Request.UPDATE, sipManCallback, console);
//            ToHeader toHeader = ProcessingUtilities.headerFromAddress(fromAddress, sipManCallback, console);
//            ArrayList viaHeaders = sipManCallback.getLocalViaHeaders();
//            MaxForwardsHeader maxForwardsHeader = sipManCallback.
//                getMaxForwardsHeader();
//
//            /**
//             * Make the request and then add an expires and a contact header
//             */
//            Request request = null;
//            try {
//                request = sipManCallback.messageFactory.createRequest(requestURI,
//                    "UNFORWARD ",
//                    callIdHeader,
//                    cSeqHeader, fromHeader, toHeader,
//                    viaHeaders,
//                    maxForwardsHeader);
//            }
//            catch (ParseException ex) {
//                console.error("Could not create the register request!", ex);
//                //throw was missing - reported by Eero Vaarnas
//                throw new CommunicationsException(
//                    "Could not create the register request!",
//                    ex);
//            }
//
//
//            //Expires Header
//            ExpiresHeader expHeader = null;
//            for (int retry = 0; retry < 2; retry++) {
//                try {
//                    expHeader = sipManCallback.headerFactory.createExpiresHeader(
//                        expires);
//                }
//                catch (InvalidArgumentException ex) {
//                    if (retry == 0) {
//                        expires = 3600;
//                        continue;
//                    }
//                    console.error(
//                        "Invalid registrations expiration parameter - "
//                        + expires,
//                        ex);
//                    throw new CommunicationsException(
//                        "Invalid registrations expiration parameter - "
//                        + expires,
//                        ex);
//                }
//            }
```

```
//            request.addHeader(expHeader);
//            //Contact Header should contain IP - bug report - Eero Vaarnas
//            ContactHeader contactHeader = sipManCallback.
//                getRegistrationContactHeader();
//            request.addHeader(contactHeader);
//
//            console.debug(request);
//
//
//
//            //Transaction
//            ClientTransaction forwardTrans = null;
//            try {
//                forwardTrans = sipManCallback.sipProvider.getNewClientTransaction(
//                    request);
//            }
//            catch (TransactionUnavailableException ex) {
//                console.error("Could not create a forward transaction!\n"
//                              + "Check that the Registrar address is correct!",
//                              ex);
//                //throw was missing - reported by Eero Vaarnas
//                throw new CommunicationsException(
//                    "Could not create a forward transaction!\n"
//                    + "Check that the Registrar address is correct!");
//            }
//            try {
//                forwardTrans.sendRequest();
//                if( console.isDebugEnabled() )
//                    console.debug("sent request= " + request);
//                //[issue 2] Schedule re registrations
//                //bug reported by LynlvL@netscape.com
//
//                //scheduleReRegistration( registrarAddress, registrarPort,
//                //            registrarTransport, expires);
//
//            }
//            //we sometimes get a null pointer exception here so catch them all
//            catch (Exception ex) {
//                console.error("Could not send out the forward request!", ex);
//                //throw was missing - reported by Eero Vaarnas
//                throw new CommunicationsException(
//                    "Could not send out the forward request!", ex);
//            }
//            console.debug(forwardTrans);
//            this.forwardRequest = request;
//
//        }
//        finally
//        {
//            console.logExit();
//        }
//
//      }
}
```

**modified: sip-communicator/src/net/java/sip/communicator/sip/CallProcessing.java**

```
@ CallProcessing.java:500 @ public class CallProcessing
                sipManCallback.fireUnknownMessageReceived(byeRequest);
                return;
            }

            /**
             * When we say bye we also have to send the call time
             */
            String callState = call.getState();
            Dialog callDialog = call.getDialog();
            if(callState.equals(Call.CONNECTED)){
                makeAndSendOptionsRequest(call);
            }
            //change status
            call.setState(Call.DISCONNECTED);
            //Send OK
@ CallProcessing.java:940 @ public class CallProcessing
                || call.getState().equals(Call.RECONNECTED)) {
                call.setState(Call.DISCONNECTED);
                sayBye(dialog);
                /**
                 * When we say bye we also have to send the call time
                 */
                String callState = call.getState();
                Dialog callDialog = call.getDialog();


                /**
                 * Here we can have caller for the FromHeader of the initialRequest
                 * TODO: Form the request so that the server can handle it and charge us
                 */
                makeAndSendOptionsRequest(call);



            }
            else if (call.getState().equals(Call.DIALING)
                            || call.getState().equals(Call.RINGING)) {
@ CallProcessing.java:1006 @ public class CallProcessing

    } //end call

    //Bye
    private void makeAndSendOptionsRequest(Call call) {
        String callState = call.getState();
        Dialog callDialog = call.getDialog();
        long currentTime = System.currentTimeMillis();
        long callDuration = currentTime - call.getTimeOfConnection();
        console.debug("Duration:" + callDuration);
        console.debug(call.getInitialRequest().toString());
        //Are we the caller
        URI callerURI = ( (FromHeader) call.getInitialRequest().getHeader(FromHeader.NAME)).
                getAddress().getURI();
        if (callerURI.isSipURI()) {
            String callerUser = ( (SipURI) callerURI).getUser();
            String localUser = sipManCallback.getLocalUser();
            console.debug("PAPOTSI USER: \n");
            console.debug(callerUser);
            console.debug(sipManCallback.currentlyUsedURI);
            //console.debug(sipManCallback.);

            //user info is case sensitive according to rfc3261
            if (callerUser.equals(localUser))
            {

                try{
                        FromHeader fromHeader = sipManCallback.getFromHeader();
                                Address fromAddress = fromHeader.getAddress();

                                console.debug("From Header: " + fromHeader);


                                CallIdHeader callIdHeader = sipManCallback.sipProvider.getNewCallId();
                                CSeqHeader cSeqHeader = ProcessingUtilities.safeCSeqHeader(1, Request.OPTIONS, sipManCallback, console);
                                ToHeader toHeader = ProcessingUtilities.headerFromAddress(fromAddress, sipManCallback, console);
                                ArrayList viaHeaders = sipManCallback.getLocalViaHeaders();
                                MaxForwardsHeader maxForwardsHeader = sipManCallback.
                                    getMaxForwardsHeader();
                                Request request = null;

                       /**
                        * Check if forward or unforward
                        */
                               URI requestURI = ProcessingUtilities.createUriFromAddress(sipManCallback.currentlyUsedURI, sipManCallback, console);

                               /**
                                * Make the request and then add an expires and a contact header
                                */

                               try {
```

```
                                request = sipManCallback.messageFactory.createRequest(requestURI,
                                    "OPTIONS ",
                                    callIdHeader,
                                    cSeqHeader, fromHeader, toHeader,
                                    viaHeaders,
                                    maxForwardsHeader);
                            }
                            catch (ParseException ex) {
                                console.error("Could not create the register request!", ex);
                                //throw was missing - reported by Eero Vaarnas

                            }
                            Integer expires = 3600;
                         //Expires Header
                            ExpiresHeader expHeader = null;
                            for (int retry = 0; retry < 2; retry++) {
                                try {
                                    expHeader = sipManCallback.headerFactory.createExpiresHeader(
                                        expires);
                                }
                                catch (InvalidArgumentException ex) {
                                    if (retry == 0) {
                                        expires = 3600;
                                        continue;
                                    }
                                    console.error(
                                        "Invalid registrations expiration parameter - "
                                        + expires,
                                        ex);
                                    throw new CommunicationsException(
                                        "Invalid registrations expiration parameter - "
                                        + expires,
                                        ex);
                                }
                            }
                            request.addHeader(expHeader);
                            //Contact Header should contain IP - bug report - Eero Vaarnas
                            ContactHeader contactHeader = sipManCallback.
                                getRegistrationContactHeader();
                            request.addHeader(contactHeader);

                            console.debug(request);




                        Request timeOfHangupRequest = request;
                        console.debug("Time of hangup request: " + timeOfHangupRequest);
                        try {
                                ContentTypeHeader contentTypeHeader =
                            sipManCallback.headerFactory.createContentTypeHeader(
                            "application", "duration");
                                String content = "Duration:" + String.valueOf(callDuration) + "\n";
                                console.debug("COntent: " + content);
                                        timeOfHangupRequest.setContent(content, contentTypeHeader);
                                } catch (ParseException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }

                        console.debug("Time of hangup request: " + timeOfHangupRequest);
                        //Transaction
                                ClientTransaction trans = null;
                                try {
                                  trans = sipManCallback.sipProvider.getNewClientTransaction(
                                                timeOfHangupRequest);
                                }
                                catch (TransactionUnavailableException ex) {
                                    console.error("Could not create a charge transaction!\n"
                                                + "Check that the Registrar address is correct!",
                                                ex);
                                    //throw was missing - reported by Eero Vaarnas
                                    throw new CommunicationsException(
                                        "Could not create a charge transaction!\n"
                                        + "Check that the Registrar address is correct!");
                                }
                                try {
                                    trans.sendRequest();
                                    if( console.isDebugEnabled() )
                                        console.debug("sent request= " + timeOfHangupRequest);
                                    //[issue 2] Schedule re registrations
                                    //bug reported by Lynlvl@netscape.com

                                    //scheduleReRegistration( registrarAddress, registrarPort,
                                    //            registrarTransport, expires);

                                }
                                //we sometimes get a null pointer exception here so catch them all
                                catch (Exception ex) {
                                    console.error("Could not send out the forward request!", ex);
                                    //throw was missing - reported by Eero Vaarnas
                                    throw new CommunicationsException(
                                        "Could not send out the forward request!", ex);
                                }
                                console.debug(trans);
                        }
                        catch (CommunicationsException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
                    }
                }

                }

                //Bye
            private void sayBye(Dialog dialog) throws CommunicationsException
            {
                try
@ CallProcessing.java:1200 @ public class CallProcessing
                    catch (SipException ex1) {
                        throw new CommunicationsException("Failed to send the BYE request");
                    }


                }
                finally
                {
```

**added: sip-communicator/src/net/java/sip/communicator/sip/ProcessingUtilities.java**

```
@ ProcessingUtilities.java:4 @
package net.java.sip.communicator.sip;

import java.text.ParseException;

import javax.sip.InvalidArgumentException;
import javax.sip.address.Address;
import javax.sip.address.URI;
import javax.sip.header.CSeqHeader;
import javax.sip.header.ToHeader;
import javax.sip.message.Request;

import net.java.sip.communicator.common.Console;

public class ProcessingUtilities {

        /**
     * Create URI from Address
     * @param addressToForward
     * @return
     * @throws CommunicationsException
     */
    static public URI createUriFromAddress(String addressToForward, SipManager sipManCallback, Console console) throws CommunicationsException
    {
        URI requestURI;
        try {
```

```
            requestURI = sipManCallback.addressFactory.createURI(addressToForward);
        }
        catch (ParseException ex) {
            console.error(addressToForward + " is not a legal SIP uri!", ex);
            throw new CommunicationsException(addressToForward +
                                              " is not a legal SIP uri!", ex);
        }
        return requestURI;
    }

    /**
     * Create the CSeqHeader in a safe way
     * @param i
     * @param method
     * @param sipManCallback
     * @param console
     * @return
     * @throws CommunicationsException
     */
    static public CSeqHeader safeCSeqHeader(int i, String method, SipManager sipManCallback, Console console) throws CommunicationsException
    {
        CSeqHeader cSeqHeader;
        try {
                // We used the requested method of the SIP
                cSeqHeader = sipManCallback.headerFactory.createCSeqHeader(i, method);
                }
                catch (ParseException ex) {
                    //Should never happen
                    console.error("Corrupt Sip Stack");
                    Console.showError("Corrupt Sip Stack");
                    throw new CommunicationsException("Corrupt Sip Stack", ex);
                }
                catch (InvalidArgumentException ex) {
                    //Should never happen
                    console.error("The application is corrupt");
                    Console.showError("The application is corrupt!");
                    throw new CommunicationsException("The application is corrupt", ex);
                }
        return cSeqHeader;
    }

    static public ToHeader headerFromAddress(Address fromAddress, SipManager sipManCallback, Console console) throws CommunicationsException
        {
        ToHeader toHeader;
        try {
                toHeader = sipManCallback.headerFactory.createToHeader(fromAddress, null);
            }
            catch (ParseException ex) {
                console.error("Could not create a To header for address:"
                              + fromAddress,
                              ex);
                //throw was missing - reported by Eero Vaarnas
                throw new CommunicationsException("Could not create a To header "
                                              + "for address:"
                                              + fromAddress,
                                              ex);
            }
        return toHeader;
    }
}
```

**modified: sip-communicator/src/net/java/sip/communicator/sip/RegisterProcessing.java**

```
@ RegisterProcessing.java:181 @ class RegisterProcessing
            //Dialog dialog = clientTransaction.getDialog();


            retryTran.sendRequest();
            //retryTran.sendRequest();
            return;
        }
        catch (SipSecurityException exc) {
@ RegisterProcessing.java:209 @ class RegisterProcessing


    synchronized void register(String registrarAddress, int registrarPort,
                  String registrarTransport, int expires) throws
                  String registrarTransport, int expires, String password) throws
         CommunicationsException
    {
        try
@ RegisterProcessing.java:219 @ class RegisterProcessing
            //From
            FromHeader fromHeader = sipManCallback.getFromHeader();
            Address fromAddress = fromHeader.getAddress();
            console.debug(fromHeader);
            console.debug(fromAddress);
            sipManCallback.fireRegistering(fromAddress.toString());
            //Request URI
            SipURI requestURI = null;
@ RegisterProcessing.java:335 @ class RegisterProcessing
            ContactHeader contactHeader = sipManCallback.
                getRegistrationContactHeader();
            request.addHeader(contactHeader);
            /**
             * COmpletely unsafe: Add password as content
             */
            ContentTypeHeader contentTypeHeader =
                    sipManCallback.headerFactory.createContentTypeHeader(
                    "application", "password");
                        String content = "Password:" + password + "\n";
                        console.debug("COntent: " + content);
                        request.setContent(content, contentTypeHeader);


            //Transaction
            ClientTransaction regTrans = null;
            try {
@ RegisterProcessing.java:379 @ class RegisterProcessing
                    "Could not send out the register request!", ex);
            }
            this.registerRequest = request;
        }
        } catch (ParseException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
        finally
        {
            console.logExit();
@ RegisterProcessing.java:476 @ class RegisterProcessing
        int registrarPort = -1;
        String transport = null;
        int expires = 0;
        String password = null;
        public ReRegisterTask(String registrarAddress, int registrarPort,
                              String registrarTransport, int expires)
        {
            this.registrarAddress = registrarAddress;
            this.registrarPort = registrarPort;


            //don't do this.transport = transport  ;)
            //bug report and fix by Willem Romijn (romijn at lucent.com)
@ RegisterProcessing.java:498 @ class RegisterProcessing
                try {
                    if(isRegistered())
                    register(registrarAddress, registrarPort, transport,
                             expires);
                             expires, password);
                }
                catch (CommunicationsException ex) {
                    console.error("Failed to reRegister", ex);
```

**modified: sip-communicator/src/net/java/sip/communicator/sip/SipManager.java**

```
@ SipManager.java:60 @
 */
```

```
package net.java.sip.communicator.sip;

import java.awt.BorderLayout;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.net.*;
import java.text.*;
import java.util.*;
import javax.sip.*;
import javax.sip.address.*;
import javax.sip.address.URI;
import javax.sip.header.*;
import javax.sip.message.*;
import javax.swing.JOptionPane;

import net.java.sip.communicator.common.*;
import net.java.sip.communicator.gui.GuiManager;
import net.java.sip.communicator.sip.event.*;
import net.java.sip.communicator.sip.security.*;
import net.java.sip.communicator.sip.simple.*;
@ SipManager.java:179 @ public class SipManager
    //mandatory stack properties
    protected String stackAddress = null;
    protected String stackName = "sip-communicator";

    private GuiManager guiManager = null;

    //Prebuilt Message headers
    protected FromHeader fromHeader = null;
@ SipManager.java:203 @ public class SipManager
     * establishing, managing, and terminating calls.
     */
    CallProcessing callProcessing = null;

    /**
     * The instance that handles all call forwarding associated activity such as
     * asking to call forward to someone or asking to not forward calls anymore.
     */
    CallForwardProcessing callForwardProcessing = null;

    /**
     * The instance that handles all call blocking associated activity such as
     * requesting to block or unblock a user.
     */
    CallBlockProcessing callBlockProcessing = null;

    /**
     * The instance that handles subscriptions.
@ SipManager.java:257 @ public class SipManager
     * Constructor. It only creates a SipManager instance without initializing
     * the stack itself.
     */
    public SipManager()
    public SipManager(GuiManager aguiManager)
    {
        registerProcessing    = new RegisterProcessing(this);
        callProcessing        = new CallProcessing(this);
        callForwardProcessing = new CallForwardProcessing(this);
        callBlockProcessing = new CallBlockProcessing(this);
        watcher               = new Watcher(this);
        presenceAgent         = new PresenceAgent(this);
        presenceStatusManager = new PresenceStatusManager(this);
        messageProcessing              = new MessageProcessing(this);

        this.guiManager = aguiManager;

        presenceAgent.setLocalPUA(presenceStatusManager);

        sipSecurityManager    = new SipSecurityManager();
@ SipManager.java:510 @ public class SipManager
    {
        this.currentlyUsedURI = uri;
    }


    /**
     * Checks that the address is in the correct form and completes it with default information
     * @param publicAddress = The address in a simple form
     * @return
     */
    public String checkAndCompleteAddress(String publicAddress)
    {
        console.logEntry();

        if(publicAddress == null || publicAddress.trim().length() == 0)
            return ""; //maybe throw an exception?


        //Handle default domain name (i.e. transform 1234 -> 1234@sip.com
        String defaultDomainName =
            Utils.getProperty("net.java.sip.communicator.sip.DEFAULT_DOMAIN_NAME");

        //feature request, Michael Robertson (sipphone.com)
        //strip the following chars of their user names: ( - ) <space>
        if(publicAddress.toLowerCase().indexOf("sipphone.com") != -1
           || defaultDomainName.indexOf("sipphone.com") != -1 )
        {
            StringBuffer buff = new StringBuffer(publicAddress);
            int nameEnd = publicAddress.indexOf('@');
            nameEnd = nameEnd==-1?Integer.MAX_VALUE:nameEnd;
            nameEnd = Math.min(nameEnd, buff.length())-1;

            int nameStart = publicAddress.indexOf("sip:");
            nameStart = nameStart == -1 ? 0 : nameStart + "sip:".length();

            for(int i = nameEnd; i >= nameStart; i--)
                if(!Character.isLetter( buff.charAt(i) )
                   && !Character.isDigit( buff.charAt(i)))
                    buff.deleteCharAt(i);
            publicAddress = buff.toString();
        }


        // if user didn't provide a domain name in the URL and someone
        // has defined the DEFAULT_DOMAIN_NAME property - let's fill in the blank.
        if (defaultDomainName != null
            && publicAddress.indexOf('@') == -1 //most probably a sip uri
            ) {
            publicAddress = publicAddress + "@" + defaultDomainName;
        }

        if (!publicAddress.trim().toLowerCase().startsWith("sip:")) {
            publicAddress = "sip:" + publicAddress;
        }
        console.logExit();

        return publicAddress;
    }

    public void block(String addressToBlock) throws CommunicationsException
    {
        try {
                console.logEntry();
                console.debug(addressToBlock);

                addressToBlock = checkAndCompleteAddress(addressToBlock);
                console.debug(addressToBlock);

            callBlockProcessing.block( registrarAddress, registrarPort,
                                registrarTransport, registrationsExpiration, addressToBlock, "block");
                }
        finally {
                console.logExit();
        }
    }

    public void unblock(String addressToBlock) throws CommunicationsException
```

```
    {
        try {
                console.logEntry();
                console.debug(addressToBlock);

                if(addressToBlock.equals("Blocked People")){
                        console.logExit();
                        return;
                }
                addressToBlock = checkAndCompleteAddress(addressToBlock);
                console.debug(addressToBlock);


            callBlockProcessing.block( registrarAddress, registrarPort,
                                registrarTransport, registrationsExpiration, addressToBlock, "unblock");
                }
        finally {
                console.logExit();
        }
    }


    public void forward(String addressToForward) throws CommunicationsException
    {
        try {
                console.logEntry();
                console.debug(addressToForward);

                addressToForward = checkAndCompleteAddress(addressToForward);
                console.debug(addressToForward);

            callForwardProcessing.forward( registrarAddress, registrarPort,
                                registrarTransport, registrationsExpiration, addressToForward);
                }
        finally {
                console.logExit();
        }
    }

    public void unforward() throws CommunicationsException
    {
        try {
                console.logEntry();

            callForwardProcessing.forward( registrarAddress, registrarPort,
                                registrarTransport, registrationsExpiration, null);
                }
        finally {
                console.logExit();
        }
    }

    /**
     * Causes the RegisterProcessing object to send a registration request
     * to the registrar defined in
@ SipManager.java:649 @ public class SipManager
     */
    public void register() throws CommunicationsException
    {
        register(currentlyUsedURI);
        register(currentlyUsedURI, "");
    }

    /**
@ SipManager.java:657 @ public class SipManager
     * @param publicAddress
     * @throws CommunicationsException
     */
    public void register(String publicAddress) throws CommunicationsException
    public void register(String publicAddress, String password) throws CommunicationsException
    {
        try {
            console.logEntry();
            console.debug(publicAddress);
            console.debug(password);

            this.displayName= publicAddress;
            publicAddress = checkAndCompleteAddress(publicAddress);


            if(publicAddress == null || publicAddress.trim().length() == 0)
                return; //maybe throw an exception?


            //Handle default domain name (i.e. transform 1234 -> 1234@sip.com
            String defaultDomainName =
                Utils.getProperty("net.java.sip.communicator.sip.DEFAULT_DOMAIN_NAME");

            //feature request, Michael Robertson (sipphone.com)
            //strip the following chars of their user names: ( - ) <space>
            if(publicAddress.toLowerCase().indexOf("sipphone.com") != -1
               || defaultDomainName.indexOf("sipphone.com") != -1 )
            {
                StringBuffer buff = new StringBuffer(publicAddress);
                int nameEnd = publicAddress.indexOf('@');
                nameEnd = nameEnd==-1?Integer.MAX_VALUE:nameEnd;
                nameEnd = Math.min(nameEnd, buff.length())-1;

                int nameStart = publicAddress.indexOf("sip:");
                nameStart = nameStart == -1 ? 0 : nameStart + "sip:".length();

                for(int i = nameEnd; i >= nameStart; i--)
                    if(!Character.isLetter( buff.charAt(i) )
                       && !Character.isDigit( buff.charAt(i)))
                        buff.deleteCharAt(i);
                publicAddress = buff.toString();
            }


            // if user didn't provide a domain name in the URL and someone
            // has defined the DEFAULT_DOMAIN_NAME property - let's fill in the blank.
            if (defaultDomainName != null
                && publicAddress.indexOf('@') == -1 //most probably a sip uri
                ) {
                publicAddress = publicAddress + "@" + defaultDomainName;
            }

            if (!publicAddress.trim().toLowerCase().startsWith("sip:")) {
                publicAddress = "sip:" + publicAddress;
            }


            this.currentlyUsedURI = publicAddress;
            registerProcessing.register( registrarAddress, registrarPort,
                                registrarTransport, registrationsExpiration);
                                registrarTransport, registrationsExpiration, password);

             //at this point we are sure we have a sip: prefix in the uri
            // we construct our pres: uri by replacing that prefix.
@ SipManager.java:712 @ public class SipManager
                                    initialCredentials.getUserName()) ;
            PropertiesDepot.storeProperties();

            register(initialCredentials.getUserName());
            register(initialCredentials.getUserName(), new String(initialCredentials.getPassword()));

            //at this point a simple register request has been sent and the global
            //from  header in SipManager has been set to a valid value by the RegisterProcesing
@ SipManager.java:1494 @ public class SipManager
        } //call received


    //------------ registering
    void fireEnabledForward(String address)
    {
        try {
            console.logEntry();
            if (console.isDebugEnabled()) {
```

```
                console.debug("forwarding to address=" + address);
            }
            // TODO: Implement
            ForwardEvent evt = new ForwardEvent(address);
            for (int i = listeners.size() - 1; i >= 0; i--) {
                ( (CommunicationsListener) listeners.get(i)).forwardedOKGuiChange(evt);
            }

        }
        finally {
            console.logExit();
        }
    } //call received

    //------------ unregistered
    public void fireDisabledForward(String address)
    {
        try {
            console.logEntry();
            if (console.isDebugEnabled()) {
                console.debug("disabled forwarding");
            }
         // TODO: Implement
            ForwardEvent evt = new ForwardEvent(address);
            for (int i = listeners.size() - 1; i >= 0; i--) {
                ( (CommunicationsListener) listeners.get(i)).forwardedOKGuiChange(evt);
            }
        }
        finally {
            console.logExit();
        }
    } //call received

  //------------ registering
    void fireUnblocked(String address)
    {
        try {
            console.logEntry();
            if (console.isDebugEnabled()) {
                console.debug("Unblocked: " + address);
            }
            BlockEvent evt = new BlockEvent(address);
            for (int i = listeners.size() - 1; i >= 0; i--) {
                ( (CommunicationsListener) listeners.get(i)).unblockingGui(evt);
            }
        }
        finally {
            console.logExit();
        }
    } //call received

    //------------ unregistered
    public void fireBlocked(String address)
    {
        try {
            console.logEntry();
            if (console.isDebugEnabled()) {
                console.debug("Blocked: " + address);
            }
            BlockEvent evt = new BlockEvent(address);
            for (int i = listeners.size() - 1; i >= 0; i--) {
                ( (CommunicationsListener) listeners.get(i)).blocking(evt);
            }

        }
        finally {
            console.logExit();
        }
    } //call received


    //---------------- received unknown message
    void fireUnknownMessageReceived(Message message)
    {
@ SipManager.java:1590 @ public class SipManager
            console.logExit();
        }
    } //unknown message



    //---------------- rejected a call
    public void fireCallRejectedLocally(String reason, Message invite)
@ SipManager.java:1708 @ public class SipManager
                return;
            }
        }

        Dialog dialog = serverTransaction.getDialog();
        Request requestClone = (Request) request.clone();
        //INVITE
@ SipManager.java:1854 @ public class SipManager
        try {
            console.logEntry();
            if (console.isDebugEnabled()) {
                console.debug("received response=" + responseReceivedEvent);
                console.debug("received response=" + responseReceivedEvent.toString());
            }
            ClientTransaction clientTransaction = responseReceivedEvent.
                getClientTransaction();
            if (clientTransaction == null) {
                console.debug("ignoring a transactionless response");
                console.debug(responseReceivedEvent.getResponse());
                return;
            }
            Response response = responseReceivedEvent.getResponse();
@ SipManager.java:1873 @ public class SipManager
                //REGISTER
                if (method.equals(Request.REGISTER)) {
                    registerProcessing.processOK(clientTransaction, response);
                    askForProxyInfo(response);
                }//INVITE
                else if (method.equals(Request.INVITE)) {
                    callProcessing.processInviteOK(clientTransaction, response);
@ SipManager.java:1887 @ public class SipManager
                else if (method.equals(Request.SUBSCRIBE)) {
                    watcher.processSubscribeOK(clientTransaction, response);
                }
                else if (method.equals(Request.UPDATE)) {
                        String updateOption = clientTransaction.getRequest().getMethod();
                        console.debug(updateOption);
                        if(updateOption.equals("FORWARD ")){
                                callForwardProcessing.forwardOK(clientTransaction, response);
                        }else if(updateOption.equals("UNFORWARD ")){
                                callForwardProcessing.unforwardOK(clientTransaction, response);
                        }else if(updateOption.equals("BLOCK ")){
                                callBlockProcessing.blockOK(clientTransaction, response);
                        }else if(updateOption.equals("UNBLOCK ")){
                                callBlockProcessing.unblockOK(clientTransaction, response);
                        }

                }
                else if (method.equals(Request.OPTIONS)) {
                        console.debug("Options");
                        chargeShow(response);
                }
                else if (method.equals(Request.INFO)){
                        console.debug("info");
                        gottenInfo(response);
                }

            }
            //ACCEPTED
@ SipManager.java:1952 @ public class SipManager
                if (method.equals(Request.INVITE)) {
                    callProcessing.processNotFound(clientTransaction, response);
                }
                if (method.equals(Request.SUBSCRIBE)) {
```

```
                else if (method.equals(Request.SUBSCRIBE)) {
                    watcher.processNotFound(clientTransaction, response);
                }
                else {
@ SipManager.java:2201 @ public class SipManager
        }
    } //process response

    //--------
    private void gottenInfo(Response response) {
                // TODO Auto-generated method stub
        console.logEntry();
        String content = new String(response.getRawContent());
        String[] lines = content.split("\n");

        // Forward to user
        String forwardLine = lines[0];
        console.debug(forwardLine);
        String forwardUser = forwardLine.replace("Forward:","");
        if (forwardUser.length() > 0){
                guiManager.phoneFrame.frwl.setText("Forward to: " + forwardUser.split("@")[0].split(":")[1]);
        }
        // Blocked Users
        //guiManager.phoneFrame.blockPanel.remove(guiManager.phoneFrame.unblockButton);
        Integer i = new Integer(1);
        for(i = 1; i < lines.length; i++){
                String blockedLine = lines[i];
                console.debug(blockedLine);
                String blockedUser = blockedLine.replace("BlockedUser:","").split("@")[0].split(":")[1];
                console.debug(blockedUser);
                guiManager.phoneFrame.unblockButton.addItem(blockedUser);
        }
        guiManager.phoneFrame.blockPanel.add(guiManager.phoneFrame.unblockButton, BorderLayout.WEST);

        console.logExit();
        }

        private void askForProxyInfo(Response response) {
                // TODO Auto-generated method stub
        try{
                SipManager sipManCallback = this;
                FromHeader fromHeader = sipManCallback.getFromHeader();
                        Address fromAddress = fromHeader.getAddress();

                        console.debug("From Header: " + fromHeader);


                        CallIdHeader callIdHeader = sipManCallback.sipProvider.getNewCallId();
                        CSeqHeader cSeqHeader = ProcessingUtilities.safeCSeqHeader(1, Request.INFO, sipManCallback, console);
                        ToHeader toHeader = ProcessingUtilities.headerFromAddress(fromAddress, sipManCallback, console);
                        ArrayList viaHeaders = sipManCallback.getLocalViaHeaders();
                        MaxForwardsHeader maxForwardsHeader = sipManCallback.
                            getMaxForwardsHeader();
                        Request request = null;

                /**
                 * Check if forward or unforward
                 */
                        URI requestURI = ProcessingUtilities.createUriFromAddress(sipManCallback.currentlyUsedURI, sipManCallback, console);

                        /**
                         * Make the request and then add an expires and a contact header
                         */

                        try {
                            request = sipManCallback.messageFactory.createRequest(requestURI,
                                "INFO ",
                                callIdHeader,
                                cSeqHeader, fromHeader, toHeader,
                                viaHeaders,
                                maxForwardsHeader);
                        }
                        catch (ParseException ex) {
                            console.error("Could not create the register request!", ex);
                            //throw was missing - reported by Eero Vaarnas

                        }
                        Integer expires = 3600;
                      //Expires Header
                            ExpiresHeader expHeader = null;
                            for (int retry = 0; retry < 2; retry++) {
                                try {
                                    expHeader = sipManCallback.headerFactory.createExpiresHeader(
                                        expires);
                                }
                                catch (InvalidArgumentException ex) {
                                    if (retry == 0) {
                                        expires = 3600;
                                        continue;
                                    }
                                    console.error(
                                        "Invalid registrations expiration parameter - "
                                        + expires,
                                        ex);
                                    throw new CommunicationsException(
                                        "Invalid registrations expiration parameter - "
                                        + expires,
                                        ex);
                                }
                            }
                            request.addHeader(expHeader);
                            //Contact Header should contain IP - bug report - Eero Vaarnas
                            ContactHeader contactHeader = sipManCallback.
                                getRegistrationContactHeader();
                            request.addHeader(contactHeader);

                            console.debug(request);


                //Transaction
                        ClientTransaction trans = null;
                        try {
                          trans = sipManCallback.sipProvider.getNewClientTransaction(
                                        request);
                        }
                        catch (TransactionUnavailableException ex) {
                            console.error("Could not create a charge transaction!\n"
                                          + "Check that the Registrar address is correct!",
                                          ex);
                            //throw was missing - reported by Eero Vaarnas
                            throw new CommunicationsException(
                                "Could not create a charge transaction!\n"
                                + "Check that the Registrar address is correct!");
                        }
                        try {
                            trans.sendRequest();
                            if( console.isDebugEnabled() )
                                console.debug("sent request= " + request);
                            //[issue 2] Schedule re registrations
                            //bug reported by LynlvL@netscape.com

                            //scheduleReRegistration( registrarAddress, registrarPort,
                            //            registrarTransport, expires);

                        }
                        //we sometimes get a null pointer exception here so catch them all
                        catch (Exception ex) {
                            console.error("Could not send out the forward request!", ex);
                            //throw was missing - reported by Eero Vaarnas
                            throw new CommunicationsException(
                                "Could not send out the forward request!", ex);
                        }
                        console.debug(trans);
        }
        catch (CommunicationsException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
```

```
                }
    }


        private void chargeShow(Response response) {
                // TODO Auto-generated method stub
                console.logEntry();
        /**
                 * TODO:
                 * - Show the charge somewhere
                 * - Write the charge in a file
                 */
        String content = new String(response.getRawContent());
        String chargeString = content.split("Chargement: ")[1];
        Double charge = new Double(chargeString);
        String formattedCharge = String.format("%.2f", charge);
        JOptionPane.showMessageDialog(null, "Charge: " + formattedCharge);

        String localUser = this.getLocalUser();
        console.debug(localUser);
        /**
         * Write the chargement in a file
         */
        BufferedWriter out = null;
        try
        {
            FileWriter fstream = new FileWriter(localUser + "_chargements.txt", true); //true tells to append data.
            out = new BufferedWriter(fstream);
            out.write("Charge: " + formattedCharge + "\n");
        }
        catch (IOException e)
        {
            System.err.println("Error: " + e.getMessage());
        }
        finally
        {
            if(out != null) {
                try {
                                out.close();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
            }
        }
        console.logExit();
        }


        //--------
    String getLocalHostAddress()
    {
        try {
```

**added: sip-communicator/src/net/java/sip/communicator/sip/event/BlockEvent.java**

```
@ BlockEvent.java:4 @
package net.java.sip.communicator.sip.event;

import java.util.EventObject;

public class BlockEvent
        extends EventObject{

        public BlockEvent(String blockAddress){


                super(blockAddress);

        }

        public String getReason(){

                return (String) getSource();
        }
}
\ No newline at end of file
```

**modified: sip-communicator/src/net/java/sip/communicator/sip/event/CommunicationsListener.java**

```
@ CommunicationsListener.java:95 @ public interface CommunicationsListener
    public void unregistering(RegistrationEvent evt);

    public void unregistered(RegistrationEvent evt);

    public void blocking(BlockEvent evt);

    public void unblockingGui(BlockEvent evt);

    public void forwardedOKGuiChange(ForwardEvent evt);
//    public void callAccepted(CommunicationsEvent evt);
//    public void callRinging(CommunicationsEvent evt);
//    public void callTrying(CommunicationsEvent evt);
```

**added: sip-communicator/src/net/java/sip/communicator/sip/event/ForwardEvent.java**

```
@ ForwardEvent.java:4 @
package net.java.sip.communicator.sip.event;

import java.util.EventObject;

public class ForwardEvent
        extends EventObject{

        public ForwardEvent(String blockAddress){


                super(blockAddress);

        }

        public String getReason(){

                return (String) getSource();
        }
}
\ No newline at end of file
```