



Cisco JTAPI Examples

Introduction

This chapter provides the source code for `makecall`, the Cisco JTAPI program used to test the JTAPI installation. The `makecall` program is comprised of a series of programs written in Java using the Cisco JTAPI implementation.

This chapter contains the following sections:

- [makecall.java, page 3-2](#)
- [Actor.java, page 3-5](#)
- [Originator.java, page 3-8](#)
- [Receiver.java, page 3-10](#)
- [StopSignal.java, page 3-11](#)
- [Trace.java, page 3-12](#)
- [TraceWindow.java, page 3-13](#)

This Chapter also provides instructions on how to invoke `makecall`:

- [Running makecall, page 3-16](#)

makecall.java

```

/**
 * makecall.java
 *
 * Copyright Cisco Systems, Inc.
 *
 * Performance-testing application (first pass) for Cisco JTAPI
 * implementation.
 *
 * Known problems:
 *
 * Due to synchronization problems between Actors, calls may
 * not be cleared when this application shuts down.
 */

import com.ms.wfc.app.*;
import java.util.*;
import javax.telephony.*;
import javax.telephony.events.*;
import com.cisco.cti.util.Condition;

public class makecall extends TraceWindow implements ProviderObserver {

    Vector actors = new Vector ();
    Condition conditionInService = new Condition ();

    public makecall ( String [] args ) {

        this.setText ( "makecall" );
        this.show ();
        try {

            println ( "Initializing Jtapi" );
            int curArg = 0;
            String providerName = args[curArg++];
            String login = args[curArg++];
            String passwd = args[curArg++];
            int actionDelayMillis = Integer.parseInt ( args[curArg++] );
            String src = null;
            String dest = null;

            JtapiPeer peer = JtapiPeerFactory.getJtapiPeer ( null );
            if ( curArg < args.length ) {

                String providerString = providerName + ";login=" + login + ";passwd=" +
                passwd;
                println ( "Opening " + providerString + "...\\n" );
                flush ();
                Provider provider = peer.getProvider ( providerString );
                provider.addObserver ( this );
                conditionInService.waitTrue ();

                println ( "Constructing actors" );
                #if true
                for ( ; curArg < args.length; curArg++ ) {
                    if ( src == null ) {
                        src = args[curArg];
                    }
                    else {
                        dest = args[curArg];
                    }
                }
            }
        }
    }
}

```

```

actors.addElement (
new Receiver ( provider.getAddress ( dest ), this,
actionDelayMillis )
);
actors.addElement (
new Originator ( provider.getAddress ( src ), dest, this,
actionDelayMillis )
);
src = null;
dest = null;
}
}
if ( src != null ) {
println ( "Skipping last originating address \"" + src + "\"; no
destination specified" );
}
#else
dest = args[curArg++];
actors.addElement (
new Receiver ( provider.getAddress ( dest ), this, actionDelayMillis )
);
actors.addElement (
new OriginateNoDrop ( provider.getAddress ( args[curArg++] ), dest,
this, actionDelayMillis )
);
actors.addElement (
new OriginateNoDrop ( provider.getAddress ( args[curArg++] ),
args[curArg++], this, actionDelayMillis )
);
#endif
}

println ( "Starting actors" );
flush ();
Enumeration e = actors.elements ();
while ( e.hasMoreElements () ) {
Actor actor = (Actor) e.nextElement ();
actor.start ();
}
}
catch ( Exception e ) {
println ( "Caught exception " + e );
flush ();
}
}

public void dispose () {
println ( "Stopping actors" );
flush ();
setTrace ( false );
Enumeration e = actors.elements ();
while ( e.hasMoreElements () ) {
Actor actor = (Actor) e.nextElement ();
actor.stop ();
}
}

public static void main ( String [] args ) {
if ( args.length < 6 ) {
System.out.println ( "Usage: makecall <server> <login> <password> <delay>
<origin> <destination> ..." );
System.exit ( 1 );
}
Application.run ( new makecall ( args ) );
}

```

```
}  
  
public void providerChangedEvent ( ProvEv [] eventList ) {  
    if ( eventList != null ) {  
        for ( int i = 0; i < eventList.length; i++ ) {  
            if ( eventList[i] instanceof ProvInServiceEv ) {  
                conditionInService.set ();  
            }  
        }  
    }  
}
```

Actor.java

```
/**
 * Actor.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

public abstract class Actor implements CallControlCallObserver, Trace {

    private Tracetrace;
    private Objectobserved;
    private intactionDelayMillis;

    public Actor ( Trace trace, Object observed, int actionDelayMillis ) {
        this.trace = trace;
        this.observed = observed;
        this.actionDelayMillis = actionDelayMillis;
    }

    public final void start () {
        try {
            if ( observed != null ) {
                if ( observed instanceof Terminal ) {
                    println (
                        "Adding Call observer to terminal "
                        + ((Terminal)observed).getName ()
                    );
                    flush ();
                    ((Terminal)observed).addCallObserver ( this );
                }
                else if ( observed instanceof Address ) {
                    println (
                        "Adding Call observer to address "
                        + ((Address)observed).getName ()
                    );
                    flush ();
                    ((Address)observed).addCallObserver ( this );
                }
                else if ( observed instanceof Call ) {
                    println (
                        "Adding Call observer to call "
                        + observed
                    );
                    flush ();
                    ((Call)observed).addObserver ( this );
                }
                else {
                    throw new Exception ( "Object " + observed + " is not observable;
                    no observation is possible" );
                }
            }
            onStart ();
        }
        catch ( Exception e ) {
            println ( "Caught exception " + e );
        }
    }
}
```

```

}
finally {
flush ();
}
}

public final void stop () {
try {
onStop ();
if ( observed != null ) {
if ( observed instanceof Terminal ) {
println (
"Removing Call observer from terminal "
+ ((Terminal)observed).getName ()
);
flush ();
((Terminal)observed).removeCallObserver ( this );
}
else if ( observed instanceof Address ) {
println (
"Removing Call observer from address "
+ ((Address)observed).getName ()
);
flush ();
((Address)observed).removeCallObserver ( this );
}
else if ( observed instanceof Call ) {
println (
"Removing Call observer from call "
+ observed
);
flush ();
((Call)observed).removeObserver ( this );
}
else {
throw new Exception ( "Object " + observed + " is not observable;
no observation needs removal" );
}
}
}
catch ( Exception e ) {
println ( "Caught exception " + e );
}
finally {
flush ();
}
}

public final void callChangedEvent ( CallEv [] events ) {
//
// for now, all metaevents are delivered in the
// same package...
//
metaEvent ( events );
}

final void delay ( String action ) {
if ( actionDelayMillis != 0 ) {
println ( "Pausing " + actionDelayMillis + " milliseconds before " + action );
flush ();
try {
Thread.sleep ( actionDelayMillis );
}
}
catch ( InterruptedException e ) {}
}

```

```
}  
}  
  
protected abstract void metaEvent ( CallEv [] events );  
  
protected abstract void onStart ();  
protected abstract void onStop ();  
  
public final void print ( String string ) {  
    trace.print ( string );  
}  
public final void print ( char character ) {  
    trace.print ( character );  
}  
public final void print ( int integer ) {  
    trace.print ( integer );  
}  
public final void println ( String string ) {  
    trace.println ( string );  
}  
public final void println ( char character ) {  
    trace.println ( character );  
}  
public final void println ( int integer ) {  
    trace.println ( integer );  
}  
public final void flush () {  
    trace.flush ();  
}  
}
```

Originator.java

```

/**
 * originator.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.ms.com.*;

public class Originator extends Actor {

    Address srcAddress;
    String destAddress;
    int iteration;
    StopSignal stopSignal;

    public Originator ( Address srcAddress, String destAddress, Trace trace, int
    actionDelayMillis ) {
        super ( trace, srcAddress, actionDelayMillis ); // observe srcAddress
        this.srcAddress = srcAddress;
        this.destAddress = destAddress;
        this.iteration = 0;
    }

    protected final void metaEvent ( CallEv [] eventList ) {
        for ( int i = 0; i < eventList.length; i++ ) {
            try {
                CallEv curEv = eventList[i];

                if ( curEv instanceof CallCtlTermConnTalkingEv ) {
                    TerminalConnection tc =
                    ((CallCtlTermConnTalkingEv)curEv).getTerminalConnection ();
                    Connection conn = tc.getConnection ();
                    if ( conn.getAddress ().getName ().equals ( destAddress ) ) {
                        delay ( "disconnecting" );
                        println ( "Disconnecting Connection " + conn );
                        flush ();
                        conn.disconnect ();
                    }
                }
                else if ( curEv instanceof CallCtlConnDisconnectedEv ) {
                    Connection conn = ((CallCtlConnDisconnectedEv)curEv).getConnection ();
                    if ( conn.getAddress ().equals ( srcAddress ) ) {
                        stopSignal.canStop ();
                        makecall ();
                    }
                }
            }
            catch ( Exception e ) {
                println ( "Caught exception " + e );
            }
            finally {
                flush ();
            }
        }
    }
}

```

```
protected void makecall ()
throws ResourceUnavailableException, InvalidStateException,
    PrivilegeViolationException, MethodNotSupportedException,
    InvalidPartyException, InvalidArgumentException {
delay ( "making the next call" );
println ( "Making call #" + ++iteration + " from " + srcAddress + " to " +
destAddress );
flush ();
Call call = srcAddress.getProvider ().createCall ();
call.connect ( srcAddress.getTerminals ()[0], srcAddress, destAddress );
println ( "Done making call" );
flush ();
}
protected final void onStart () {
stopSignal = new StopSignal ();

try {
makecall ();
}
catch ( Exception e ) {
println ( "Caught exception " + e );
}
finally {
flush ();
}
}
protected final void onStop () {
stopSignal.stop ();
}
}
```

Receiver.java

```

/**
 * Receiver.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

public class Receiver extends Actor {
    Address address;
    StopSignal stopSignal;

    public Receiver ( Address address, Trace trace, int actionDelayMillis ) {
        super ( trace, address, actionDelayMillis );
        this.address = address;
    }

    protected final void metaEvent ( CallEv [] eventList ) {
        for ( int i = 0; i < eventList.length; i++ ) {
            TerminalConnection tc = null;
            try {
                CallEv curEv = eventList[i];

                if ( curEv instanceof CallCtlTermConnRingingEv ) {
                    tc = ((CallCtlTermConnRingingEv)curEv).getTerminalConnection ();
                    delay ( "answering" );
                    println ( "Answering TerminalConnection " + tc );
                    flush ();
                    tc.answer ();
                    stopSignal.canStop ();
                }
            }
            catch ( Exception e ) {
                println ( "Caught exception " + e );
                println ( "tc = " + tc );
            }
            finally {
                flush ();
            }
        }
    }

    protected final void onStart () {
        stopSignal = new StopSignal ();
    }

    protected final void onStop () {
        stopSignal.stop ();
    }
}

```

StopSignal.java

```
/**
 * StopSignal.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

class StopSignal {

    boolean stopping = false;
    boolean stopped = false;
    synchronized boolean isStopped () {
        return stopped;
    }
    synchronized boolean isStopping () {
        return stopping;
    }
    synchronized void stop () {
        if ( !stopped ) {
            stopping = true;
            try {
                wait ();
            }
            catch ( InterruptedException e ) {}
        }
    }
    synchronized void canStop () {
        if ( stopping = true ) {
            stopping = false;
            stopped = true;
            notify ();
        }
    }
}
```

Trace.java

```
/**
 * Trace.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

public interface Trace {

public void print ( String string );
public void print ( char character );
public void print ( int integer );
public void println ( String string );
public void println ( char character );
public void println ( int integer );
public void flush ();
}
```

TraceWindow.java

```
/**
 * TraceWindow.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.com.*;
import com.ms.win32.*;
import com.ms.win32.User32.*;

public class TraceWindow extends Form implements Trace {

    boolean traceEnabled = true;
    StringBuffer buffer = new StringBuffer ();

    public TraceWindow () {
        super ();

        // Required for Visual J++ Form Designer support
        initForm ();
        updateSize ();
    }

    private void updateSize () {
        Point dimensions = getSize ();
        Rectangle bounds = new Rectangle ( 0, 1, dimensions.x, dimensions.y );
        bounds.height -= 26;
        bounds.width -= 7;

        text.setBounds ( bounds );
    }

    /**
     * MyTextArea overrides dispose so it can clean up the
     * component list.
     */
    public void dispose() {
        super.dispose();
        components.dispose();
    }

    private void MyTextArea_layout(Object source, LayoutEvent e) {
        updateSize ();
    }

    /**
     * NOTE: The following code is required by the Visual J++ form
     * designer. It can be modified using the form editor. Do not
     * modify it using the code editor.
     */
    Container components = new Container();
    RichEdit text = new RichEdit();

    private void initForm() {
        this.setText("");
    }
}
```

```

this.setAutoScaleBaseSize(new Point(5, 13));
this.setSize(new Point(374, 245));
this.addOnLayout(new LayoutEventHandler(this.MyTextArea_layout));

text.setFont(Font.DEFAULT_GUI);
text.setForeground(Color.WINDOWTEXT);
text.setLocation(new Point(8, 8));
text.setSize(new Point(360, 232));
text.setTabIndex(0);
text.setText("");
text.setReadOnly(true);
text.setScrollBars(RichEditScrollBars.BOTH);
text.setWordWrap(false);

this.setNewControls(new Control[] {
text});
}

public final void print ( String str ) {
if ( traceEnabled ) {
buffer.append ( str );
}
}

public final void print ( char character ) {
if ( traceEnabled ) {
buffer.append ( character );
}
}

public final void print ( int integer ) {
if ( traceEnabled ) {
buffer.append ( integer );
}
}

public final void println ( String str ) {
if ( traceEnabled ) {
print ( str );
print ( '\n' );
}
}

public final void println ( char character ) {
if ( traceEnabled ) {
print ( character );
print ( '\n' );
}
}

public final void println ( int integer ) {
if ( traceEnabled ) {
print ( integer );
print ( '\n' );
}
}

public final void setTrace ( boolean traceEnabled ) {
this.traceEnabled = traceEnabled;
}

public final void flush () {
if ( traceEnabled ) {
text.invoke ( new MethodInvoker ( setSelText ) );
}
}

final void setSelText () {
if ( buffer.length () > 0 ) {
text.setSelText ( buffer.toString () );
if ( buffer.length () > 0 ) {
text.sendMessage ( wine.EM_SCROLLCARET, 0, 0 );
}
}
}

```

```
}  
buffer = new StringBuffer ();  
}  
}  
public final void clear () {  
this.setText("");  
}  
}
```

Running makecall

Invoking makecall

On the client workstation

From the Windows NT command line, navigate to the **makecall** directory where JTAPI Tools directory was installed and execute the following command:

```
jview makecall <server name> <login> <password> 1000 <device 1> <device2>
```

<server name> is the hostname or IP address of your Cisco CallManager and <device1> <device2> are directory numbers of IP phones. The phones must be part of the associated devices of a given user as administered in the Cisco CallManager's directory administration web page. The <login> and <password> are similarly as administered in the directory. This will test that you have installed and configured everything correctly. The application will make calls between the two devices with an action delay of 1000 msec until terminated.