



Copyright and Trademark Information

Disclaimer

THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS DOCUMENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS DOCUMENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS DOCUMENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.

Licensed Materials - Property of IBM

©Copyright IBM Corporation 2002, 2004 All rights reserved.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GS ADP Schedule Contract with IBM Corp.

Lotus Software

IBM Software Group

One Rogers Street

Cambridge, MA 02142

List of Trademarks

IBM, the IBM logo, 1-2-3, AIX, AS/400, DB2, Domino, Domino Designer, iNotes, iSeries, Lotus, Lotus Notes, MQSeries, Netfinity, Notes, QuickPlace, Sametime, SmartSuite, S/390, Tivoli, WebSphere, and Word Pro are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Pentium is a trademark of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Third Party Notices

For the XSL and XML Parser and Processor

The Apache Software License, Version 1.1

Copyright (c) 1999-2000 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 "Xerces" 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 and was originally based on software copyright (c) 1999, International Business Machines, Inc., <http://www.ibm.com>. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

For DSIG base64

COPYRIGHT 1995 BY: MASSACHUSETTS INSTITUTE OF TECHNOLOGY (MIT), INRIA

This W3C software is being provided by the copyright holders under the following license. By obtaining, using and/or copying this software, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee or royalty is hereby granted, provided that the full text of this NOTICE appears on ALL copies of the software and documentation or portions thereof, including modifications, that you make.

THIS SOFTWARE IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION,

COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDERS WILL BEAR NO LIABILITY FOR ANY USE OF THIS SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

For STLport

License Agreement

Boris Fomitchev grants Licensee a non-exclusive, non-transferable, royalty-free license to use STLport and its documentation without fee.

By downloading, using, or copying STLport or any portion thereof, Licensee agrees to abide by the intellectual property laws and all other applicable laws of the United States of America, and to all of the terms and conditions of this Agreement.

Licensee shall maintain the following copyright and permission notices on STLport sources and its documentation unchanged :

Copyright 1999,2000 Boris Fomitchev

This material is provided "as is", with absolutely no warranty expressed or implied. Any use is at your own risk.

Permission to use or copy this software for any purpose is hereby granted without fee, provided the above notices are retained on all copies. Permission to modify the code and to distribute modified code is granted, provided the above notices are retained, and a notice that the code was modified is included with the above copyright notice.

The Licensee may distribute binaries compiled with STLport (whether original or modified) without any royalties or restrictions.

The Licensee may distribute original or modified STLport sources, provided that:

The conditions indicated in the above permission notice are met; The following copyright notices are retained when present, and conditions provided in accompanying permission notices are met :

Copyright 1994 Hewlett-Packard Company

Copyright 1996,97 Silicon Graphics Computer Systems, Inc.

Copyright 1997 Moscow Center for SPARC Technology.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Moscow Center for SPARC Technology makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright 2001 by STLport

For MD5 hash

Copyright (C) 1990, RSA Data Security, Inc. All rights reserved. License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.

For Log4J Logging

The Apache Software License, Version 1.1 at <http://www.apache.org/LICENSE>, 24 May 2002

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:

6. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
7. 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.
8. 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.
9. 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.
10. 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.

Table of Contents

About This Tutorial	1
Chapter 1 Introduction.....	3
Chapter 2 HackersCatcher Sample	4
Overview	4
Implementing the Login Listener	6
Implementing the CommunityEventsService Listener	6
Implementing the UserLoginFailedListener Listener	7
Main Implementation.....	7
Building and Running	7
Chapter 3 OfflineMessages Sample	8
Overview	8
Implementing the Channel Listener	11
Implementing the Community Events Service	12
UserHandler Object – Message Handling	12
Implementing the Login Listener	14
Implementing the IM Listener.....	15
Chapter 4 PlacesLogger Sample	16
Overview	16
Running the Sample.....	16
How It Works.....	17

About This Tutorial

Intended Audience

This tutorial is for the IBM® Lotus® Instant Messaging and Web Conferencing (Lotus Sametime®) Community Server Toolkit.

Note Throughout this guide, the IBM Lotus Instant Messaging and Web Conferencing product is referred to as “Sametime,” and this and other toolkits are referred to as “Sametime” toolkits.

This tutorial is intended for Java™ developers who have used the Sametime Java Toolkit and want to use the Sametime Community Server Toolkit to enhance application logic on the server.

This guide does not include information about Sametime Java programming or Java programming in general. For more information on Sametime Java programming, see the Sametime Java Client Toolkit documentation. For more information on the Java language and Java programming, see <http://java.sun.com/>.

Requirements

The Sametime Community Server Toolkit can be used in any JDK 1.1 or higher Java development environment.

The toolkit is targeted for use with the following Sametime servers:

- Windows – 2.5 and above.
- iSeries – 3.1 and above

Although applications developed with this toolkit will work when run on a Sametime 2.x or later server, toolkit services that require features new to this release will not function. In particular, the code examples in the toolkit should be run on the latest version of the Sametime server.

How to Use this Guide

This guide contains the following main sections.

Chapter title	Description
Chapter 1: Introduction	Provides an overview of the Sametime Community Server Toolkit
Chapter 2: HackersCatcher sample	Demonstrates how to build the HackersCatcher sample
Chapter 3: OfflineMessages sample	Demonstrates how to build the OfflineMessages sample
Chapter 4: PlacesLogger sample	Demonstrates how to build the PlacesLogger sample

Guide Conventions

The following conventions are used in this guide:

- Sample code is in the `Courier New` font.
- Sample code that has been added to a previous sample step is in bold **Courier New**.

Related Documents

- *Sametime Community Server Toolkit Developer's Guide*

Additional Information

Additional information can be found on the following websites:

<http://www.lotus.com/sametime>
<http://www.ibm.com/developerworks/lotus>

You may also want to read “Working with the Sametime Client Toolkits,” available at the IBM Redbooks site (<http://www.redbooks.ibm.com>).

Chapter 1 Introduction

The Sametime Community Server Toolkit is one member of a comprehensive, Java-based application SDK that developers can use to embed real-time capabilities into e-business applications. This server toolkit is a collection of components used by developers to build applications that affect the functionality and services provided by a Sametime server. These components can be used in any standard development environment that supports JDK 1.1.

This tutorial demonstrates implementing the Sametime Community Server Toolkit components through these sample applications:

- [The HackersCatcher sample](#) – Notifies of “login failed” events through the implementation of the the UserLoginFailedListener Listener.
- [The OfflineMessages sample](#) – Allows online users to send messages to other users who are currently offline using the UserHandler object. Those messages are then sent to their intended recipients when they come online.
- [The PlacesLogger sample](#) – Using the ActivityService component, monitors the life cycle of a certain type of place and lists some of their properties in a swing-based table.

Chapter 2 HackersCatcher Sample

Overview

This sample demonstrates how to build the HackersCatcher application. The HackersCatcher sample logs in to the Sametime community as a server application. It then subscribes to get “login failed” events from the community. For demonstration purposes, the information on each “login failed” attempt is sent to the console but that action could easily be changed to some other action.

Running the Sample

To use the HackersCatcher sample:

1. Add stcommsrvtk.jar to your class path to enable using the Community Server Toolkit.
2. Open your server. (To run a server application from your machine, first configure the Sametime server.)

Note You will not be able to log in to the Sametime server as a server application if the IP address of your machine is not in the server’s trusted IP list.

To log in to the Sametime server as a server application, perform one of the following:

- Add your IP to the server trusted IP list. To do so, open stconfig.nsf and add the IP to the CommunityTrustedIPS field in the CommunityConnectivity document. Separate multiple IP addresses with commas or semicolons.
- Configure the server to accept all IPs as trusted (a less secure option):
 - a. On the server machine, open the Sametime.ini file that is located in the Sametime folder.
 - b. Add the following line to the Debug section of the Sametime.ini file:

```
[Debug]
...
VPS_BYPASS_TRUSTED_IPS=1
```

Imported Packages

The five packages used in this sample are:

```
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.types.*;
import com.lotus.sametime.community.*;
import com.lotus.sametime.communityevents.*;
import java.net.InetAddress;
```

Class Members

The table below identifies the three class members used in the HackersCatcher sample.

Class Member	Description
STSession m_session	Represents the Sametime session.
ServerAppService m_saService	Represents the server application service and enables logging in to the server as a server application.
CommunityEventsService m_ceService	Represents the CommunityEvents service and provides the “login failed” notifications.

Initializing and Logging In to Sametime

Add a “run” method to your class. This method has one parameter, which is the server name. The code for this method is as follows:

```
private void run(String serverName)
{
    try
    {
        m_session = new STSession("" + this );

        String [] compNames = { ServerAppService.COMP_NAME,
            CommunityEventsService.COMP_NAME };
        m_session.loadComponents( compNames );

        m_saService = (ServerAppService)
            m_session.getCompApi(ServerAppService.COMP_NAME);
        m_ceService = (CommunityEventsService)
            m_session.getCompApi(CommunityEventsService.COMP_NAME);
    }
    catch (DuplicateObjectException e)
    {
        System.out.println("STSession or Components created
            twice.");
    }

    m_session.start();

    m_saService.addLoginListener( this); short loginType =
    STUserInstance.LT_SERVER_APP; m_saService.loginAsServerApp(
    serverName, loginType, "Hacker Catcher", null);
}
```

As shown above, the STSession object is created, the needed components are loaded within a try-catch block, and the session is started. Finally, you add yourself as a listener to login/logout events and log in to the community as a server application with the login type LT_SERVER_APP.

Implementing the Login Listener

In the “run” method described above, the sample class was added as a login listener to `m_saService`. To implement the `m_saService` interface:

1. Declare a class to implement `LoginListener`:

```
public class HackersCatcher implements LoginListener
```

2. Add the two methods of `LoginListener` to this new class:

```
public void loggedIn(LoginEvent event)
{
    m_ceService.addCommunityEventsServiceListener(this);
}
public void loggedOut(LoginEvent event)
{
    m_ceService.removeCommunityEventsServiceListener(this);
}
```

To get notifications on the availability of the `CommunityEventsService`, add the sample class as a `CommunityEventsServiceListener` interface in the implementation.

Implementing the CommunityEventsService Listener

Add the `CommunityEventsServiceListener` to the implemented interface list of this class.

The `CommunityEventsService` listener interface has two methods:

- `serviceAvailable`
- `serviceUnavailable`

To implement the `CommunityEventsServiceListener`, add the following:

```
public void serviceAvailable(CommunityEventsServiceEvent event)
{
    m_ceService.addUserLoginFailedListener(this);
}

public void serviceUnavailable(CommunityEventsServiceEvent event)
{
    m_ceService.removeLoginFailedListener(this);
}
```

To receive “login failed” events in the implementation, add the sample class as a `LoginFailed` Listener.

Implementing the UserLoginFailedListener Listener

Add the UserLoginFailedListener to the implemented interface list of this class. This interface has only one method. It prints information on the failed login attempt.

```
public void userLoginFailed(UserLoginFailedEvent event)
{
    String s = new String("login failed:");
        s += " Name=" + event.getLoginName();
        s += ", ip=" + event.getLoginIp();
        s += ", type=" +
Integer.toHexString(event.getLoginType());
        s += ", reason=" +
Integer.toHexString(event.getReason());

        System.out.println(s);
}
```

Main Implementation

In Main, call the run method and pass it the server name from the command line: public static void main(String[] args)

```
{
    if ( args.length != 1 )
    {
        System.out.println("Usage: HackersCatcher
serverName");
        System.exit(0);
    }

    new HackersCatcher().run(args[0]);
}
```

Building and Running

Now you can build the program. It should compile with no errors.

After building, run the program from the command line for Windows platforms, or from the command line resulting from issuing STRQSH command on IBM iSeries, passing the server name. After the console is ready, open Sametime Connect and try to log in to the server with an incorrect password. You should see an entry for the failed login attempt on your program's console.

Chapter 3 OfflineMessages Sample

Overview

This sample demonstrates how to build an Offline Messages Server application by using the various server toolkit components. The application allows clients to send messages to a user who is currently offline. The message is saved and then transferred to its intended recipient when that user goes online.

The Offline Messages Server application works as follows:

- When a message to a user who is offline is received, the message is saved together with details about both the sender and the intended receiver. The server application then waits for the receiver to log in.
- When the receiver logs in to Sametime, the server application logs in on behalf of the sender, sends the message, and immediately logs out.

The receiver gets the message as if the sender sent it directly. If the receiver responds, the response is sent back to the original sender and not to the server application.

OfflineMessages Sample Content

The sample contains four files.

File name	Description
OfflineMessagesSA	The server application that listens to messages from clients and to users that log in to the community
UserHandler	An object that handles a single message to an offline user
Client	A sample contact list client that is able to send (but not receive) offline messages
MessageFrame	The frame on which clients can write a message

Running the Sample

To use the OfflineMessages sample:

1. Add stcommsrvrtk.jar and commres.jar to your class path to enable using the Community Server Toolkit.
2. Open your server.
To run a server application from your machine, first configure the Sametime server. Refer to the HackersCatcher Sample chapter of this tutorial for more information.
3. From a command prompt for Windows platforms or from the command line resulting from issuing STRQSH command on IBM iSeries, first run the server application of the sample. For example:

```
java -cp .:[server toolkit jar file location] OfflineMessagesSA [server name]
```
4. Run the sample client (which is like any other Sametime Java Toolkit Client). For example:

```
java -cp .:[java toolkit jar files];[resource jar file] Client [user name] [password] [server name].
```

For IBM iSeries, run the sample client on a Windows client machine or use a Sametime Connect client for this step.
5. Send a message to a user who is currently offline.
6. The offline user can then log in with a regular Sametime Connect client to receive the message.

OfflineMessages Sample Components

These components are used in this sample:

Component	Description
ServerAppService	Used for logging in to the Sametime community as a server application
ChannelService	Used for receiving channels from clients
CommunityEventsService	Used for getting notifications on users that log in to the Sametime community
LightLoginService	Used for logging in on behalf of the message sender
SATokenService	Used for getting the sender login token
InstantMessagingService	A client toolkit component used to send an instant message (IM)

Initializing the Sample

Look at the constructor of the OfflineMessage Server application class.

Follow these steps:

1. Create a session of components:

```
public OfflineMessagesSA(String hostName)
{
    // Create and load the session of components.
    try
    {
        m_session = new STSession("OfflineMessages");
        String [] compNames = { ServerAppService.COMP_NAME,
            CommunityEventsService.COMP_NAME,
            SATokenService.COMP_NAME };
        m_session.loadComponents( compNames );

        m_session.start();
    }
    catch(DuplicateObjectException e)
    {
        e.printStackTrace();
        exit();
    }

    m_hostName = hostName;
}
```

2. Address the session to get a reference for each component that will be used later. These components are the Community Events Service and the Channel Service. Also, add the sample class as a listener to each of the components to receive notifications.

```
// Get a reference to the needed components.
m_commEvents =
(CommunityEventsService)m_session.getCompApi(CommunityEventsService
.COMP_NAME);
m_commEvents.addCommunityEventsServiceListener(this);

ChannelService channelService =

(ChannelService)m_session.getCompApi(ChannelService.COMP_NAME);
channelService.addChannelServiceListener(this);
}
```

Logging In to Sametime

Use the login method to log in to Sametime. For information about the login process, refer to the HackersCatcher Sample chapter of this tutorial.

Implementing the Channel Listener

Upon logging in, a server application declares the type of service it provides. When clients want to communicate with the application, they create and open a channel providing that same service type.

The notification about a new incoming channel is received in the channelReceived event.

The server application unwraps the message information and creates a UserHandler object for the message. The message is kept in the watchedUsers hashtable. Since you do not need to respond to the client, close the channel.

The following is the implementation of the Channel Listener:

```
public void channelReceived(ChannelEvent event)
{
    // Get the incoming data.
    Channel cnl = event.getChannel();
    try
    {
        NdrInputStream inStream = new NdrInputStream(cnl.getCreateData());

        STId receiverId = new STId(inStream);
        String receiverName = inStream.readUTF();
        String message = inStream.readUTF();

        STUser receiver = new STUser(receiverId, receiverName, "");
        UserHandler handler = new UserHandler(m_session, m_hostName,

cnl.getRemoteInfo(), receiver, message);

        m_watchedUsers.put(receiverId.getId(), handler);
    }
    catch (IOException e)
    {
        e.printStackTrace();
        cnl.close(STError.ST_INVALID_ID_DATA, null);
        return;
    }
    // No need for the channel anymore.
    cnl.close(STError.ST_OK, null);
}
```

Implementing the Community Events Service

The Community Events Service is used to get notifications on users who log in to Sametime. The following code implements the Channel Listener:

```
m_commEvents.addUserLoginListener(this);
```

When a user logs in to Sametime, you receive a `userLoggedIn` event. If the user is in your watched users list, the `receiverOnline` method is invoked on the `UserHandler` object of that message.

```
public void userLoggedIn (UserLoginEvent event)
{
    // Check if we are interested in this user.
    STUser user = event.getUserInstance();
    Object o = m_watchedUsers.remove(user.getId().getId());
    if (o != null)
    {
        ((UserHandler)o).receiverOnline();
    }
}
```

UserHandler Object – Message Handling

The `UserHandler` object is used for handling a single message to an offline user. As described earlier, the server application creates one `UserHandler` object for every incoming message and invokes the `receiverOnline` method when it receives notification that the receiver has logged in.

When the receiver logs in, the `UserHandler` does the following:

- Uses the **SATokenService** to get the sender's login token
- Uses the **LightLoginService** to perform a virtual login on behalf of the sender
- Uses the **InstantMessagingService** to send an IM to the receiver with the message
- Uses the **LightLoginService** to log out

Initialization

```
public UserHandler(STSession saSession, String serverName, STUser
sender,STUser receiver, String message)
{
    m_sender    = sender;
    m_receiver  = receiver;
    m_message   = message;
    m_serverName = serverName;
```

```
// We need the sender's token for login.
m_tokenService
    =
    (SATokenService)saSession.getCompApi(SATokenService.COMP_NAME);
    m_tokenService.addTokenServiceListener(this);

m_mainLogin
    =
    (ServerAppService)saSession.getCompApi(ServerAppService.COMP_NAME);
}
```

The session of the server application is `saSession`. The `mainLogin` variable holds the `ServerAppService` instance used by the server application to log in. This instance is used later for the Light Login Service.

ReceiverOnline Implementation

This method is called when the offline receiver logs in to Sametime.

To log in on behalf of the sender, create another session of components to differentiate between the login IDs. The server application Token Service is used to get the sender's login token. See the *Sametime Community Server Toolkit Developer's Guide* for more information.

```
void receiverOnline()
{
    // First, create a session for the user.
    try
    {
        m_session = new STSession("OfflineMessageUser" + this);
        String [] compNames = { LightLoginService.COMP_NAME,
            InstantMessagingService.COMP_NAME };

        m_session.loadComponents( compNames );

        m_session.start();
    }
    catch(DuplicateObjectException e)
    {
        e.printStackTrace();
        return;
    }

    m_loginService =
    (LightLoginService)m_session.getCompApi(LightLoginService.COMP_NAME
    );

    m_imService =
    (InstantMessagingService)m_session.getCompApi(InstantMessagingService.COMP_NAME);

    // Now, try to get the sender's token.
    m_tokenService.generateToken(m_sender);
}
```

Implementing the Token Service Listener

If a token for the sender was successfully generated, try to log in to Sametime on behalf of the sender:

```
public void tokenGenerated(TokenEvent event)
{
    // Now, we login as the sender.

    m_loginService.setLoginType(STUserInstance.LT_LIGHT_CLIENT_USER);
    m_loginService.addLoginListener(this);

    // A light client has to provide its IP explicitly, giving null
    as IP
    // prevents the server from disconnecting the real sender.
    Token token = event.getToken();
    m_loginService.loginByToken(token.getLoginName(),
    token.getTokenString(), m_mainLogin,
    null);
}
```

Note When logging in using the Light Login Service, a reference to the main login ID and the IP address must be provided. The Sametime server automatically disconnects every additional login occurrence of a user that was made from a different machine. An IP address is not given to avoid having the server disconnect the real sender.

Implementing the Login Listener

After logging in to the Sametime server, send the waiting message to its recipient. Use the IM service of the client toolkit. Refer to the *Sametime Java Toolkit Tutorial* and the *Sametime Java Toolkit Developer's Guide* for more information.

```
public void loggedIn(LoginEvent event)
{
    System.out.println("SA: Handler LoggedIn");

    // Finally we can send the message.
    Im im = m_imService.createIm(m_receiver, EncLevel.ENC_LEVEL_ALL,
    1);
    im.addImListener(new ImHandler());
    im.open();
}
```

Note Use 1 as the **ImType** to integrate with a Sametime Connect client that uses that type.

Implementing the IM Listener

Use an inner class to listen for the IM events since you do not need them all. If the IM was created, the message text is sent and the user is immediately logged out.

```
class ImHandler extends ImAdapter
{
    public void imOpened(ImEvent event)
    {
        //Send the message and leave asap.
        event.getIm().sendText(true, m_message);
        m_loginService.logout();
        cleanUp();
    }

    public void openImFailed(ImEvent event)
    {
        System.out.println("SA HANDLER: Couldn't open IM " +
            event.getReason());
        cleanUp();
    }
}
```

Cleaning up

It is important to completely remove the session once it is no longer used so it can be eliminated during “garbage collection”(the Java Virtual Machine identifies objects that are no longer in use and reclaims the memory). Use the following cleanup method to remove the session:

```
private void cleanUp()
{
    m_session.stop();
    m_session.unloadSession();
}
```

The Sample Client

The sample client uses the Sametime Awareness Tree UI component to create a simple contact list that can send offline messages. Refer to the *Sametime Java Toolkit Tutorial* and *Sametime Java Toolkit Developer's Guide* for more information. For IBM iSeries, the sample client must be run on a Sametime client machine.

Note that the sample client can only send offline messages, not receive them. After you run this sample and send an offline message, log in with a regular Sametime Connect client to receive the message.

Chapter 4 PlacesLogger Sample

Overview

The PlacesLogger sample application monitors the life cycle of a certain type of place and lists some of their properties in a swing-based table. This sample demonstrates the basic use of the Places server components and serves as a starting point for developers who want to develop more elaborate applications. This sample is not supported on IBM iSeries since it requires native AWT GUI support. To run on iSeries, the PlacesHandler.java and LogFrame.java samples would need to be changed to store/retrieve the log information to file instead of a Swing-based table.

PlacesLogger Sample Content

The following table identifies the files contained in the sample.

File Name	Description
<code>PlacesLogger.java</code>	This file contains the main application that logs in to the community, registers your activity as one that will be added to places of the desired type, and listens to the newly created places.
<code>PlacesHandler.java</code>	This file creates a wrapper on top of each place that monitors the properties of the place and updates the output frame when required.
<code>LogFrame.java</code>	This file creates a swing-based table that implements the Graphical User Interface (GUI) for the sample.

Running the Sample

To use the PlacesLogger sample:

1. Add `stcommsrvrtk.jar` to your class path to enable using the Community Server Toolkit.
2. Open your server.
To run a server application from your machine, first configure the Sametime server. Refer to the HackersCatcher Sample chapter of this tutorial for more information.
3. Point a command prompt to the classes' directory and type:
`java - classpath [path for the server toolkit package] PlacesLogger [serverName]`

- Using the Sametime Connect client, create a meeting of any type on the Sametime server and see the results in the Logger.

PlacesLogger Sample Components

The following table identifies the components used in this sample.

Component	Description
ServerAppService	Used for logging in to the Sametime community
PlacesAdminService	Used for registering an activity as one that should always be added to places of a specific type
ActivityService	Used for registering as an activity provider in the place service and for receiving notifications on newly created places

How It Works

Every server application must first log in to the Sametime community by using the login method.

An activity can be added to a place in two ways. A user can explicitly request the activity in that place, or it can be registered as a “default activity” of a certain place type. A default activity is automatically added to any place of that type whenever the place is created.

For a logger application use the default activity. To do so, use the `setDefaultActivity` method of the `PlacesAdminService`. Look at the implementation of the `serviceAvailable` event to see how it is done.

```
public void serviceAvailable(PlacesAdminEvent event)
{
    // Set our activity as a default activity for the place
    type we
    // are interested in.
    System.out.println("AdminService available");
    m_adminService.setDefaultActivity(PLACE_TYPE,
    ACTIVITY_TYPE, null);
}
```

Now listen to places being created. Whenever an activity is requested by a place, the `ActivityServiceListener.activityRequested` method is called. The activity provider application should determine whether to accept the request or not.

```
public void activityRequested(ActivityServiceEvent event)
{
    // A place was created and our activity is requested.
    Place place = event.getPlace();

    System.out.println("Place: " + place.getName());
    m_activityService.acceptActivity(event.getMyActivity(), null);
}
```

```
        new PlaceHandler(event.getMyActivity(), m_frame);
    }
```

Look at the `activityRequested` implementation in the sample. Since there is no reason to decline such a request, accept it immediately. Also create a `PlaceHandler` object for every new place. The place handler will monitor the place for specific events.

In this sample, you monitor three properties of a place: its name, the time of its existence, and whether certain activities were added to it. You start a timer on the constructor and then stop it whenever you receive a `PlaceLeft` event (which should only happen if the place has been destroyed). You also listen to the `activityAdded` event to get the activities in the place. All the changes are displayed in the `LogFrame`, which uses standard swing components.

Note A place that is empty of users gets destroyed automatically after a timeout of three (3) minutes. To establish a different timeout, set the `VPPLACE_DEFAULT_TTL` flag in the `Sametime.ini` file to the desired time.

To monitor the activities that are added to the place:

```
/**
 * An activity was added to the place.
 */
public void activityAdded(PlaceEvent event)
{
    int type = event.getActivityType();
    switch (type)
    {
        case CHAT_ACTIVITY:
            m_hasChat = new Boolean(true);
            break;
        case AUDIO_ACTIVITY:
            m_hasAudio = new Boolean(true);
            break;
        case VIDEO_ACTIVITY:
            m_hasVideo = new Boolean(true);
            break;
        case SCREEN_SHARE_ACTIVITY:
            m_hasShare = new Boolean(true);
            break;
        case WHITEBOARD_ACTIVITY:
            m_hasWb = new Boolean(true);
            break;
    }
    m_frame.refreshAll();
}
```

To monitor the time that the place exists:

```
public PlaceHandler(MyActivity activity, LogFrame frame)
{
    .
    .
    m_time = new Time(m_frame, this);
}
```

The `Time` class uses a thread to update the frame's table every second.