Sametime
Version 9.0

Sametime 9.0
Software Development Kit
Community Server Toolkit Tutorial

Edition Notice

Note: Before using this information and the product it supports, read the information in "Notices."

This edition applies to version 9.0 of IBM Lotus Sametime (program number 5725-M36) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. About This Tutorial

Intended Audience

This tutorial is for the IBM® Lotus® Sametime® Community Server Toolkit.

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

For information about software requirements for the Sametime Community Server Toolkit, see releaseNotes.txt, included with the toolkit.

How to Use this Guide

This guide contains the following main sections.

Table listing chapter number and title with a description of each chapter.

Chapter title	Description
Chapter 2: Introduction	Provides an overview of the Sametime Community Server Toolkit
Chapter 3: HackersCatcher sample	Demonstrates how to build the HackersCatcher sample
Chapter 4: OfflineMessages sample	Demonstrates how to build the OfflineMessages sample
Chapter 5: PlacesLogger sample	Demonstrates how to build the PlacesLogger sample
Chapter 6: UserAttributeSA sample	Demonstrates how to use the UserAttributeSA service that enables a third-party server application to set attributes for a user
Chapter 7: Telephony Presence sample	Demonstrates how to build a telephony application that can publish telephony attributes, add and remove watch on a Sametime user, and get notifications about the user's status changes

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 2. 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. The toolkit can be used in any JDK 1.7 Java development environment. The code should be compiled with JDK 1.6 to allow running on Domino server, since Domino runs with 1.6 JRE.

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 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.
- **The UserAttributeSA sample** – Uses the UserAttributeSAService component to set and remove attributes associated with specific users. These attributes are visible to other users and server applications via the awareness service (i.e., buddy list), and look the same as the attributes set by the user.

Chapter 3. 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 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.)

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.

Table describing 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
   ```

1. 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 4. 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.

Table describing the four files used in the OfflineMessages sample.

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 Error: Reference source not found 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:

Table describing the components used in the OfflineMessages 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;
```

1. 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 Error: Reference source not found 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_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_NAM
E);

 // Now, try to get the sender's token.
 m_tokenService.generateTokens(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 tokensGenerated(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[] tokens = event.getTokens();
 m_loginService.loginByTokens(tokens[0].getLoginName(),
tokens, m_mainLogin,
 InetAddress.getByAddress(new byte[]{0, 0, 0, 0}));
}
```

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 5. 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.

Table describing the three files used in the PlacesLogger sample.

File Name	Description
`PlacesLogger.java`	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.
`PlacesHandler.java`	This file creates a wrapper on top of each place that monitors the properties of the place and updates the output frame when required.
`LogFrame.java`	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 Error: Reference source not found 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]

1. 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.

Table describing the components used in the PlacesLogger 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.

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, under the Debug section of 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.

Chapter 6. UserAttributeSA Sample

Overview

This sample demonstrates how to build a server application that can set and remove attributes associated with specific users. These attributes are visible to other users and server applications via the awareness service (i.e., buddy list), and look the same as the attributes set by the user/client.

This service enables the creation of a third-party server-side component that can contribute additional information about the current state of the user. For example, the component could indicate that the user is currently on the phone by receiving the information from its PBX.

ONLY server-side components have the permission to use this service. Client-side logins cannot change attributes for other users in the system.

UserAttributeSA Sample Content

The sample includes two files:

Table describing the two files used in the UserAttributeSA sample.

File name	Description
`UserAttribSASample`	This file includes Sample SA, which uses the UserAttributeSAService to manipulate the user's attributes
`ClientSample`	This file is a simple client application that views and enables the monitoring of the user's attributes. Together with the UserAttribSASample, ClientSample demonstrates how user attributes can be manipulated by a server application.

UserAttribSASample Structure

1. The sample imports the following Java Toolkit packages and classes:

```
import com.lotus.sametime.community.LoginEvent;
import com.lotus.sametime.community.LoginListener;
import com.lotus.sametime.community.ServerAppService;
import com.lotus.sametime.core.comparch.DuplicateObjectException;
import com.lotus.sametime.core.comparch.STSession;
import com.lotus.sametime.core.types.STAttribute;
import com.lotus.sametime.core.types.STUser;
import com.lotus.sametime.core.types.STUserInstance;
import com.lotus.sametime.lookup.LookupService;
import com.lotus.sametime.lookup.ResolveEvent;
import com.lotus.sametime.lookup.ResolveListener;
import com.lotus.sametime.lookup.Resolver;
import com.lotus.sametime.onlinedirectory.OnlineDirectoryService;
import com.lotus.sametime.userattributesa.UserAttributeSAEvent;
import com.lotus.sametime.userattributesa.UserAttributeSAListener;
import com.lotus.sametime.userattributesa.UserAttributeSAService;
import com.lotus.sametime.userattributesa.UserAttributeSAServiceEvent
```

2. The `_userAttribSvc` object provides access to the `UserAttributeSAService`.
3. UserAttribSASample implements the following interfaces:

Interface	Implementation
`UserAttributeSAListener`	Receives the UserAttributeSAService service events
`LoginListener`	Receives the login to Community events
`ResolveListener`	Receives events that verify whether or not the user name is resolved

How It Works

1. The main UserAttribSASample class method reads two necessary values from the input argument array:
   - Sametime Server name/IP address
   - User name whose attribute the UserAttribSASample is going to push
2. If both arguments are provided correctly, a new UserAttribSASample class object is created.
3. While a new UserAttribSASample class instance is being built, all required Sametime components are loaded:

```
session = new STSession("UserAttribSASample" + this);
String[] comps = { ServerAppService.COMP_NAME,
                   OnlineDirectoryService.COMP_NAME,
                   UserAttributeSAService.COMP_NAME,
                         LookupService.COMP_NAME };

_session.loadComponents(comps);

_session.start);
```

4. The newly created session starts and the UserAttribSASample object adds itself into both the `UserAttributeSAListener` and `LoginListener` lists. The UserAttribSASample object then logs into Sametime Community as a server application:

```
saSvc.loginAsServerApp(server, STUserInstance.LT_SERVER_APP,
                          "User Attributes SA", null);
```

5. If the object gets the `UserAttributeSAService` service available, it activates `LookupService`, adds itself to the `ResolveListener` list, and tries to resolve a user name provided as an input parameter:

```
public void serviceAvailable(UserAttributeSAServiceEvent event) {
      System.out.println("User Attributes SA Service Available");
      //Wait for service available before attempting to user the service

      //Resolve the user name into STUser object
      LookupService lookupSvc = (LookupService)
      _session.getCompApi(LookupService.COMP_NAME);
      Resolver resolver = lookupSvc.createResolver(true, false, true,
      false);

      resolver.addResolveListener(this);
      resolver.resolve(_userName);
   }
```

6. If the user name is resolved successfully (i.e., if the `resolved` event has occurred), then the class object is allowed to set/remove the user attribute. For example, the `778877` attribute ID is set:

```
public void resolved(ResolveEvent event) {
```

```
        //Inlcude a time stamp in the attribute content so we will //get
        an event each time it is updated
           java.util.Date d = new java.util.Date();
        STAttribute attr = new STAttribute(778877, "Some exciting new
        value" + ", Date: " + d.toString());
        System.out.println("User Resolved successfully, updating
        attribute: " + attr.getKey());
        _userAttribSvc.setAttribute((STUser) event.getResolved(), attr);

    }
```

7. `AttributeSet/attributeRemoved` events of the `UserAttributeSAService` service indicate that an attribute has been updated or removed successfully. `SetAttributeFailed` and `removeAttributeFailed` events point to the operation failure.
8. All the events of the required services are implemented via the `System.out.println` logging mechanism.

ClientSample Structure

The sample imports the following Java Toolkit packages and classes:

```
import com.lotus.sametime.awareness.AttributeEvent;
import com.lotus.sametime.awareness.AttributeListener;
import com.lotus.sametime.awareness.AwarenessService;
import com.lotus.sametime.awareness.WatchList;
import com.lotus.sametime.community.CommunityService;
import com.lotus.sametime.community.LoginEvent;
import com.lotus.sametime.community.LoginListener;
import com.lotus.sametime.core.comparch.DuplicateObjectException;
import com.lotus.sametime.core.comparch.STSession;
import com.lotus.sametime.core.types.STAttribute;
```

UserAttribSASample implements the following interfaces:

Table describing the interfaces used in the UserAttributeSA sample.

Interface	Implementation
`LoginListener`	Receives the login to Community events
`AttributeListener`	Receives the attribute change events

How It Works

1. The main method of the sample reads the required parameters from the argument array:
   - Sametime server name
   - User name
   - User password
2. If all the arguments are provided correctly, a new `ClientSample` object is created.
3. The `ClientSample` class constructor creates a new Sametime session object, loads all the necessary components, adds itself to the Listeners lists, and logs into Sametime Community as a client application:

```
_session = new STSession("ClientSample" + this);
_session.loadSemanticComponents();
_session.start();


CommunityService commSvc = (CommunityService)
_session.getCompApiCommunityService.COMP_NAME);
commSvc.addLoginListener(this);
commSvc.loginByPassword(host, loginName, password);
```

4. Receiving a `loggedIn` event indicates that the user successfully logged into Sametime Community. The `ClientSample` class activates `AwarenessService` and creates a filter of attributes whose status it wants to watch (i.e., `778877`). It then builds a watch list and adds itself to it:

```
public void loggedIn(LoginEvent event) {
      System.out.println("Client logged In");
      AwarenessService svc = (AwarenessService)
            _session.getCompApi(AwarenessService.COMP_NAME);
      //Set up the filter to include the attribute we want to watch
      int[] attrFilter = { 778877 };
```

```
        svc.addToAttrFilter(attrFilter);

        //Create a watch list and add our self to the watch list.
        WatchList watchList = svc.createWatchList();
        watchList.addAttrListener(this);

        watchList.addItem(event.getLogin().getMyUserInstance());
}
```

5. When the attribute below is changed by the `UserAttribSASample` application, the ClientSample class gets an `attrChanged` event. The event may deliver more than one change at a time. An updated attribute value is printed via the `System.out.println` logging mechanism.

This example shows that the use of a new `UserAttributeSAService` component does not require a code update on the client side.

Running the Sample

To use the UserAttributeSASample sample:

1. Add stcommsrvrtk.jar to your class path to enable using the Community Server Toolkit.

2. Open your server.
Note: To run a server application from your machine, first configure the Sametime server.

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 this, open stconfig.nsf and add your 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):

  On the server machine, open the Sametime.ini file located in the Sametime folder. Add the following line to the Debug section of the Sametime.ini file:

  ```
  [Debug]
  …
  VPS_BYPASS_TRUSTED_IPS=1
  ```

  a. To run the sample, start by running the client application. The application requires three parameters: host name, user name, and password.

  b. Now run the SA side that updates attribute `778877` for a specific client. The application requires two parameters: host name and user name.

  c. Once both applications are running you should see the client receiving an attribute update (via the system.out/logs) each time the SA is launched.

Chapter 7. Telephony Presence Sample

Overview

This sample demonstrates how to build a telephony application that can publish telephony attributes, add and remove watch on a Sametime user, and get notifications about the user's status changes.

Telephony Sample Content

The sample includes the following files:

Table describing the three files used in the Telephony Presence sample.

File name	Description
`TelephonySample`	This file includes a sample application that uses the Telephony Presence Adapter.
`RunSample.bat`	This file is a simple batch file for activating the sample program.
`logging.properties`	The logger properties file which is configured to display only the Telephony Presence Adapter log messages.

TelephonySample Structure

TelephonySample implements the following interfaces:

Table describing the interface used in the Telephony Presence sample.

Interface	Implementation
`TelephonyAdapterListener`	Receives the Telephony Presence Adapter events

How It Works

The TelephonySample instantiates the Telephony Presence Adapter as follows:

1. Create the telephony component:
telephonyComp = TelephonyAdapterComp.getInstance();
2. Add itself as a listener to receive sametimeStatusChanged events:
telephonyComp.setListener(this);
3. Initiate the component:
telephonyComp.initiate();

When the Sametime community is available and ready to get requests, the CommunityAvailable event is sent.

The **public void communityAvailable()** method is activated. When this event is received the telephony application might use any of the Telephony Presence Adapter services, for example:

1. Subscribe for user's status changes events:
_ telephonyComp.addWatch(USER_TELEPHONY_ID);
2. Publish user's telephony status:
telephonyComp.publishTelepohonyStatus(USER_TELEPHONY_ID,
TelephonyStatus.TELEPHONE_STATUS_BUSY);

The **public void sametimeStatusChanged(String telephonyUserId, int sametimeStatus)** method is activated when a watched user's status has been changed. It receives the user's telephony ID and the updated Sametime status.

Running the Sample

Customize the source

Change the USER_TELEPHONY_ID to a user ID which can be resolved by the Sametime environment.

Compile

Compile the TelephonySample.java to create the file TelephonySample.class

Set the telephony adapter properties

1. Extract the file **st.telephony.adapter.properties** from the server toolkit jar (stcommsrvrtk.jar).
2. Unmark and set the following attributes to match your environment:
   connecting.server.dns - the qualified DNS name of the Sametime server

Set the RunSample.bat

1. Set the JAVA_HOME variable to your Java home directory. (Java 1.4)
2. Set the ADAPTER_JAR variable to the server toolkit jar (stcommsrvrtk.jar).
3. Set the ADAPTER_PROPERTIES_FILE variable to the Telephony Presence Adapter configuration file. (st.telephony.adapter.properties).
4. Set the LOGGING_PROPERTIES_FILE variable to the Logger properties file (logging.properties).

Run

From the command line, activate RunSample.

Inspect the results

The sample application uses the Telephony Adapter Component to:

Add (and remove) watch on a user, and receive Sametime user status change notifications:
In order for the sample program to receive the sametimeStatusChanged events after adding the watch, please login as this user to the Sametime server and change his status. In any case, you will always receive his initial status.

Publish the user's telephony status to the Sametime server:
In order to verify that the telephony attribute of this user has changed, run the application ClientSample.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
5 Technology Park Drive
Westford Technology Park
Westford, MA 01886

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only. All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp.
Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

These terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM

AIX

DB2

DB2 Universal Database Domino

Domino

Domino Designer

Domino Directory

i5/OS

iSeries

Lotus

Notes

OS/400

Sametime

System i

WebSphere

AOL is a registered trademark of AOL LLC in the United States, other countries, or both.

AOL Instant Messenger is a trademark of AOL LLC in the United States, other countries, or both.

Google Talk is a trademark of Google, Inc, in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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