Sametime
Version 9.0

Sametime 9.0
Software Development Kit
Community Server Toolkit Developer's Guide

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.

© Copyright IBM Corporation 2006, 2013.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. About this guide

This guide explains the IBM® Lotus® Sametime® Community Server Toolkit.

Intended Audience

This guide 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 side.

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

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

Chapter Title	**Description**
Chapter 1. About this guide	Provides an overview of this guide's contents.
Chapter 2: Introduction	Provides an overview of the Sametime Community Server Toolkit
Chapter 3: Server Application Service	Describes the service that allows a server application to log in and log out of the Sametime server and to register for specific service types
Chapter 4: Channel Service	Describes the service that provides the ability to communicate with other Sametime entities via a proprietary protocol
Chapter 5: Community Events Service	Describes the service that provides the ability to receive a variety of events from the community server
Chapter 6: General Awareness Service	Describes the service that provides the ability to change the online attributes of the server to which you are connected
Chapter 7: Places Admin Service	Describes the service that provides the ability to administer the Places server application
Chapter 8: Places Activity Service	Describes the service that provides the ability to write activity providers to a Sametime place
Chapter 9: Server Application Storage Service	Describes the service that stores user-related information as attributes directly on the Sametime server
Chapter 10: Light Login Service	Describes the service that allows a MUX application to multiplex a connection to the server on top of an existing physical connection
Chapter 11 Server Application Token Service	Describes the service that provides the ability to generate a token that can be used to log in as a user to Sametime
Chapter 12 Online Directory Service	Describes the service that allows you to locate users and groups in the community

Chapter 13 Telephony Presence Adapter	Describes the service that provides an easy-to-use infrastructure for interfacing between Sametime and third-party telephony applications.
Chapter 14 Cached Resolve component	
Appendix A: The Java Launcher	Describes this application provided by Sametime that registers Java applications as NT services.
Appendix B: Sametime Identifiers	Describes Sametime-supported identifiers.
Appendix C: Component Dependencies for Loading and Packaging	Describes component dependencies and restrictions and the various ways to load components.
Appendix D: Using the Sametime Java Toolkit	Describes how to use Sametime Java Toolkit components and services in your Community Server Toolkit applications.
Appendix E. Configuring the Sametime Directory to support Resolve by Telephony User ID	

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 Tutorial

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.

Accessing the toolkit page on the web

To install the toolkit, visit http://www.ibm.com/developerworks/lotus/products. On the Products page, click "Lotus Sametime". The Lotus Sametime page contains links to all the documentation and downloads. You can extract the files for this toolkit either on your local machine or (to make it available to other users) on your Sametime server.

To access the toolkit pages that include the toolkit documentation, samples, and binaries, open readme.html in the toolkit root folder. Assuming that the toolkit is installed in:
\<server data directory>\domino\html\ST*xx*CommServerToolkit

You can access the toolkit home page at http://<server hostname>/ST*xx*CommServerToolkit/readme.html.

Chapter 3. Server Application Service

The Server Application Service allows a server application to log in and log out of the Sametime server and to register for specific service types. The Server Application Service also provides the ability to send administrator messages.

Logging in as a server application

The Server Application Service's main task is to provide a way to log in to the Sametime community as a server application. This task is accomplished using the loginAsServerApp method. The application has to provide the login type for the login operation. The possible values are either STUserInstance.LT_SERVER_APP for regular server applications or STUserInstance.LT_MUX_APP for MUX applications. (For additional information about MUX applications, see Chapter 9, ____.)

During the login process, a server application can provide a list of service types that it supports. It is also possible to register to support additional service types at a later time by calling the serviceUp method. To unregister, use the serviceDown method.

In distributed environments, it is important to know whether the services provided by the application are global (known throughout the entire community) or local (known only to their connecting server). To declare a server application service as global, use a service type that begins with 1 (a hex number that begins with 8; for example, 0x80000010). To declare a service as local, use a service type that begins with 0.

Currently, the server allows only server applications to connect using a direct socket connection on Port 1516. Before calling loginAsServerApp, you must call the setConnectivity method to set the connectivity. Once logged in, you can log out at any time by calling the logout method.

Administration messages

A server application can send administrator messages by calling the adminMsg method.

Chapter 4. Channel Service

A channel is a virtual connection between two Sametime entities. All communication in Sametime uses channels. The Channel Service provides the ability to communicate with other Sametime entities via a proprietary protocol. Using this service you can:

- Create channels to other Sametime entities.
- Register to get notification when another entity tries to open a channel to you.
- Send and receive any kind of data through these channels.
- Close the channels.

Creating and addressing a channel

When creating a channel, you can address it three ways:

- **By login ID** – The channel is addressed to this specific login ID.
- **By user ID** – If the user has more than one login ID, the server will choose the preferred one and address the channel to this application.
- **By service type** – The channel is addressed to the Server Application that supports this service type.

Create a channel using the createChannel method of the Channel Service. You can address the channel by passing the service type parameter or the toID parameter. The other parameters are the protocol type, the protocol version, the encryption level, and the initial data to send.

Registering a channel

The createChannel method returns a Channel object. This object does not yet represent an open channel. Before opening the channel, a channel listener is registered to determine when this channel is actually opened. Register your listener using the addChannelListener method of the channel you have just created.

Sending and receiving messages through a channel

Once a channel is opened, you can use it to send any kind of data using the sendMsg method. This method accepts three parameters: the message type, the message data, and a flag that indicates if the message should be encrypted.

Note This flag is relevant only if the channel was encrypted when it was created.

A channelMsgReceived event is generated every time the other side sends a message. The ChannelEvent parameter contains the message type, the data, and a flag that indicates if the message was encrypted.

Accepting a channel from another Sametime entity

To get notification when another Sametime entity tries to open a channel to you, register a ChannelServiceListener to the Channel Service. Once the listener is registered, the service calls the channelReceived method whenever a channel is addressed to you.

In your implementation of channelReceived, you must specify in the Channel Service what to do with this channel by calling one of these three methods:

- accept(com.lotus.sametime.core.constants.EncLevel requestedLvl, byte[] data) – The channel should be accepted.

- **close(int reason, byte[] data)** – The channel should be destroyed.
- **pend()** – The channel should be put in a "pending" state; it can be accepted or destroyed later. If a program decides to put a channel in a pending state, it is then responsible for changing its state later by calling either accept or close. Channels should normally not be kept in a pending state.

Note If you did not call one of these functions within your implementation of channelReceived, the channel will be destroyed.

Chapter 5. Community Events Service

The Community Events Service provides the ability to receive a variety of events from the community server. Using this service, you can register to receive notifications about user status, user storage, user privacy, and available services.

Use the Community Events Service to register to get the following events:

- **User Login** – The service sends notification when each user logs in to Sametime.
- **User Login Failed** – The service sends notification of each failed attempt to log in to Sametime.
- **User Status** – The service sends notification every time a user's status changes.
- **User Privacy** – The service sends notification every time a user changes her privacy list.
- **User Online** – Any user can log in to Sametime from different applications. This service sends user online notification when the first application logs in, and it sends user offline notification when the last application logs out.
- **User Storage** – The service sends notification every time a user changes his storage.
- **Service Up/Down** – The service sends notification each time any service becomes available or unavailable in the Sametime domain.

Service Availability

Tracking the Community Events Service availability allows you to know:

- When you are receiving community event notifications in real time
- When you are not receiving any notifications due to network or server problems

To track this service availability, use ServiceAvailableListener, which provides two events: serviceAvailable and serviceUnavailable. To receive events whenever the service availability changes, register to this listener.

Event Listeners

Event listeners allow an object to be notified when an event occurs. The event listener is registered on the appropriate event source. The Community Server Toolkit contains the following event listeners:

- User Login Event Listener
- User Login Failed Event Listener
- User Status Event Listener
- User Privacy Event Listener
- User Online Event Listener
- User Storage Event Listener
- Service Up/Down Event Listener

User Login event listener

Login events are generated whenever a Sametime user logs in to Sametime.

Note A Sametime user can be logged in to Sametime several times (using different processes). Using the Community events allows you to receive a notification each time the user logs in.

STUserInstance is received with the login notification. STUserInstance has all the useful information on the user and on the specific login ID, such as login type and IP address.

To get the UserLogin notification, add a UserLoginListener to the service.

User Login Failed event listener

Login Failed events are generated whenever a Sametime user fails to log in to Sametime. This event can be used to track unauthorized login attempts to Sametime. The event includes information about the attempt, such as the IP address, the login type, and the reason for the failure.

To get the Login Failed notification, add a UserLoginFailedListener to the service.

User Status event listener

User Status events are generated whenever a user within the Sametime domain changes his status. The event is sent when the user goes online or offline and when he changes his online status (away, DND, and so on). The event provides information about the user and the new status.

To get the User Status notifications, add a UserStatusListener to the service.

User Privacy event listener

User Privacy events are generated whenever a user within the Sametime domain changes his privacy list. The event provides information about the user and the new privacy list.

To get the User Privacy notifications, add a UserPrivacyListener to the service.

User Online event listener

User online events are generated whenever a user within the Sametime domain goes online, such as when he logs in to Sametime using his first application. If the user logs in to Sametime again using an additional application, then the event is *not* sent. The event provides information about the user.

User offline events are generated when the last application of the user logs out.

To get the user online notifications, add a UserOnlineListener to the service.

User Storage event listener

User storage events are generated whenever one of the storage attributes of a Sametime user changes. The event gives information about the user.

To get the user storage notifications, add a UserStorageListener to the service.

Service Up/Down event listener

Service up events are generated whenever a server application registers a service type that was not registered before. Service down events are generated when a server application unregisters a service type, and no other server application supports this service type. The event gives information about the server application login ID and its supported services.

To get the Service up/down notifications, add a ServiceAvailableListener to the service.

Using the Community Events service

To use the Community Events Service:

1. Create a session and log in to Sametime as a server application.
2. Wait until the Community Events Service is available (add the relevant listener and wait for an event telling you that the service is available).
3. Add the specific listener to get the events.

See also

See the HackerCatcher Sample section in the *Sametime Community Server Toolkit Tutorial*, which demonstrates how to use the Community Events service.

Chapter 6. General Awareness Service

The General Awareness Service provides the ability to change the online attributes of the server to which you are connected. These attributes are stored in the server application responsible for awareness, such as the contact list. By adding a server to your watch list, you can receive notifications when attributes change.

For example, the Meeting Server in Sametime uses this technique to let the clients know the kinds of activities (such as audio and video) that are available. There is a list of predefined attributes; when an activity becomes available the Meeting Server sets the appropriate attribute. The client creates a watch list and adds the server to the list. When an attribute changes, the client UI changes to reflect the available meeting activities.

To get notifications about changes in a server's attributes, simply add the STServer object that represents the server you are interested in to a watch list. This process is similar to the process in the Sametime Java Toolkit (STWatchedObject from the Awareness service). Data associated with the object (an attribute) is stored in a key/value format where the value is a byte array, so that it can be used to store any kind of data.

The General Awareness Service supplies an interface for changing object attributes. Because the current version of this toolkit supports only one type of object (the server object), you can create, change, and destroy server attributes only. In future versions of the toolkit, this service will support other types of objects such as users and user-defined objects. Examples of user-defined objects are stock quotes and local weather conditions.

Creating and changing an attribute

You can create or change an attribute by using the changeAttr method of the service. If an attribute with the same key already exists in the Buddy List Server Application, the attribute is updated and a notification is sent to all subscribed users. If there is no attribute with the same key, the attribute is created and a notification is sent.

Removing an attribute

Use the removeAttr method of the service to remove an attribute from the Buddy List Server Application. An appropriate notification is sent to all subscribers.

Chapter 7. Places Admin Service

The Places Admin Service provides the ability to administer the Places server application. Use this service to create persistent places and control the characteristics and behavior of created places. For example, you can define the default number and capacity of sections in a place and the default activities in it. The Places Admin Service also provides a directory of existing places.

Persistent places

A Sametime virtual place usually gets destroyed immediately after the last user has left. A persistent place remains "alive" whether or not there are users in it. The only way to remove a persistent place is through a specific request or by rebooting the Places server application.

Use the createPersistentPlace and destroyPlace methods to create and destroy a persistent place. An appropriate notification is sent to all PlacesAdminListeners (as described later in this section).

Controlling places characteristics

Use the Places Admin Service to control some of the characteristics of places that are created:

- To set the number and capacity of sections to be created for every place of the given type, call the setDefaultSections method.
- To set the activity that will be automatically added to places of the given type, call the setDefaultActivity. Appropriate notifications will be sent to all PlacesAdminListeners (as described later in this section).
- To add activities to a place, call addActivity. To remove activities, call removeActivity. Appropriate notifications will be sent to all PlacesAdminListeners (as described later in this section).

PlacesAdminListener

The following responses to the operations described above are received through the PlacesAdminListener interface:

- **serviceAvailable/Unavailable** – Indicates the availability of the service. No administration operations should begin until this event is received.
- **placeCreated/Destroyed** – Received as an acknowledgment for the corresponding operations.
- **createPersistentPlaceFailed/destroyPlaceFailed** – Received if the corresponding operations fail.
- **defaultSectionSet** – Received if the setDefaultSections operation succeeds. If it fails, the setDefaultSectionsFailed event is generated.
- **defaultActivitySet** – Received if the setDefaultActivity operation succeeds. If it fails, the setDefaultActivityFailed event is generated.
- **activityAdded** – Received if the addActivity operation succeeds. If it fails, the addActivityFailed event is generated.
- **activityRemoved**. – Received if the removeActivity operation succeeds. If it fails, the removeActivityFailed event is generated.

PlacesDirectory

The Places Admin Service also provides a directory of places (both persistent and non-persistent) that exist on a single server. To start getting notifications on these places, call addPlaceDirectoryListener.

The notifications are as follows:

- **placesAdded** – New places were created.
- **placeDestroyed** – A place was destroyed.
- **placeUpdated** – A property of an existing place was changed.

In the above events, places are described with a PlaceInfo object, which contains the following parameters:

- **getName** – The place's unique name
- **getType** – The place type

Note A place is uniquely identified by both its name and its type.

- **getDisplayName** – The place presentable name
- **isPersistent** – Specifies if the place is persistent
- **isFull** – Specifies if the place has reached its capacity and can no longer accept users
- **isStageFull** – Specifies if the "stage" section has reached its capacity
- **getStageUserCount** – Specifies the number of users currently in the "stage" section
- **getUserCount** – Specifies the number of users in the place

Chapter 8. Places Activity Service

The Places Activity Service provides the ability to write activity providers to a Sametime place. An activity is an application that runs in a place and is shared by all of its members. Communication with users in the place can be implemented in a propriety protocol between a client application and the activity provider.

An activity provider in a place is allowed to perform various tasks that are not allowed for a regular user, making the activity provider a "super-user" of sorts. Examples of such tasks are monitoring all messages in the place, controlling the way users are located in a place, and changing the number and capacity of sections.

Creating an activity

An activity provider application is a server application that uses the PlacesActivityService to provide one or more activities to a place. When a server application logs in to Sametime, it must declare the list of service types it provides. See "Server Application Service". An activity provider declares the activity types it provides in exactly the same way.

An activity can be added to a place in three ways:

- Your application adds an activity by calling PlacesAdminService.addActivity. (For more information, see Chapter 7, ____.)
- The type of place being created includes a default activity. (For more information, see Chapter 7, ____.)
- A client application requests it by using the Sametime Java Toolkit Place.addActivity method.

In order for the activity provider application to receive such a request, it must register as an ActivityServiceListener to the activity service. Whenever a request for an activity of a type provided by the activity provider occurs, the activityRequested event is dispatched to that application. That event includes a Place object representing the virtual place (as described below in this section), and a MyActivity object representing the activity in that place.

Note Even if your application provides more than one activity in the same place, the same Place object will be used for all activities.

The activity provider application must respond to this event by calling either the acceptActivity or declineActivity method of the Activity Service. When accepting an activity, initial data can be given as well.

Places object model

The Activity Service API contains objects that correspond to the virtual entities in a Sametime place. The objects are Place, Section, Activity, UserInPlace, and MyActivity. Each object represents an activity provided by this application. All of these objects are defined to be place members; as such, they share common operations and events. Every member defines a listener interface from which it can receive events. An application that is interested in a particular member XXX should implement the XXXListener interface that is associated with it. The application should then register itself using the addXXXListener method of the desired member.

Place member interface

All Place objects implement the PlaceMemberT interface. This interface defines the following:

- Communication with each member of a place by sending text and data messages to the member
- Changing or removing the attributes of each member
- Listening to incoming and outgoing messages of this member

Note Place member attributes are limited to the scope of a specific place. Their lifetime is limited to the duration of the place. Place attributes have nothing to do with online attributes used by the Awareness Service (whose scope is the entire community) and with Storage Service attributes that are stored persistently on the Sametime server.

Sending Text and Data to a Place Member

Use the sendText and sendData methods of PlaceMember to send text and data to specific place members. By calling these methods on different place member objects, you can send a message to a specific user, to all users in a section, or to all users in a place. You can also use these methods to send messages to other activities in the place, since they are also place members themselves. If the message cannot be sent, you receive a sendFailed event in the appropriate listener.

Manipulating Attributes of Place Members

The attributes of a place member can be manipulated by calling the putAttribute or removeAttribute method. To retrieve the value of a known or heavy attribute, you can use the queryAttribute method. Every member's listener provides an attributeChanged event when an attribute is changed, or if an attribute is queried from a place member. An attributeRemoved event is generated if an attribute is removed from a place member.

These attribute methods for manipulating place members might not be authorized and can therefore fail. Trying to perform an unauthorized operation yields an XXXFailed event with the appropriate reason code.

Listening to Members' Messages

With an activity provider, you can listen to all the messages that are being delivered in a place.

- To get messages that are sent to a specific place member, add an IncomingMessageListener to that member. For example, to get messages that are sent to the entire place, add an IncomingMessageListener to the Place object. To get all the messages sent to the place, add that listener to all the members.
- To listen to outgoing messages of users and activities, add an OutgoingMessagesListener to the UserInPlace object or to the Activity-appropriate object.

Note Receiving messages that are sent to one of the activities you provide is performed in the same way. Just add an IncomingMessageListener to the MyActivity object representing the activity in question.

Place object

When one of the activities provided by your application is requested in a place, the activityRequested event of the ActivityServiceListener interface is called with the relevant Place object.

Operations on a Place

Once you have accepted a place, you can do the following:

- **Limit access to the place to a specified list of users –** Use the addAllowedUsers method. If this method is not called, any user can enter the place as long as he has the place name, type, and password (if applicable). If the operation succeeds, you receive the allowedUsersAdded event. If the Places Service does not accept this request, you receive an addAllowedUsersFailed event on the PlaceListener object.
- **Add sections to the place** – Use the addSections method and indicate the number of sections to add and their capacity. An acknowledgment event is dispatched to the registered PlaceListener to indicate if the operation succeeded.
- Get information about the place – Use the following methods:
  - **getMembers()** – Returns an enumeration you can use to review all the members of the place, such as users, sections, and activities

Note This method returns only the members for which you received an event. For example, if you are not listening to a particular section, you will not get users in that section.

- **getMyActivities()** – Returns an enumeration you can use to review all the activities that your application manages
- **getMyActivity(int type)** – Gets the MyActivity object that represents the managed activity of the given type

Receiving Place Events

You receive place events by implementing the PlaceListener and adding it to the place using the addPlaceListener method. After you accept a place, you receive sectionAdded and activityAdded events for each section and activity that already exist in the place. Once in the place, you receive updates on sections and activities being added and removed with the sectionAdded, sectionRemoved, activityAdded, and activityRemoved events.

If you are disconnected from the place, you receive a placeLeft event from the PlaceListener.

Section object

The Section object represents a section in the place. When you accept a place, you can receive a snapshot of all the sections of the place. You can also receive continuous updates on sections being added to or removed from the place by adding a PlaceListener to the place. (See the Receiving Place Events section.). In Sametime every place has three sections by default.

- To add sections to a place, use the addSections method of the Place object.
- To determine if a section is a stage section, use the isStage method.

Restricting the Section to Allowed Users

Use the Section object to restrict the section to a specified list of allowed users by calling the addAllowedUsers method. If the operation succeeds, you receive an allowedUsersAdded event of the SectionListener. You receive the addAllowedUsersFailed event if the allowed users could not be set. To determine the total number of possible users in the section, call the getCapacity method.

Receiving Notifications on Users Entering and Leaving Sections

Use SectionListener to receive notifications on users entering and leaving the section. You receive a usersEntered event each time users enter the section. The first time you add a listener to a section, you receive a usersEntered event with all the users who are currently in the section. You receive a userLeft event for each user who leaves the section.

Setting a Section's Capacity

A section can have a limited capacity of users. To set that capacity, call the setCapacity method of the Section object. To determine the capacity, use the getCapacity method. A capacity of zero means that there is no limit to the number of users that can be added to that section.

Activity object

The Activity object represents an activity in a place. Activities managed by your application are represented by a subclass of the Activity object called MyActivity.

Use the Activity object to retrieve information on the activity, such as the activity type, the activity data, and the identity of who added the activity.

UserInPlace object

This object represents a user in the place. UserInPlace also extends STUserInstance (which extends STUser) and therefore can be used easily with other toolkit services that expect an STUser object.

You can move a user from its current section to another section by using the moveToSection method of the UserInPlace object. Listen to UserInPlaceListener to find out if your request succeeded.

Communicating with users

An activity provider in a place usually prefers to have a propriety protocol of sorts with the users in the place. As described in previous sections of this chapter, this communication can be done easily in the following ways:

- To get messages that are sent to one of the activities you provide, add an IncomingMessageListener to the MyActivity object of that activity.
- To send messages to a user in the place, call the sendText or sendData methods of the UserInPlace class. You can also send a message to all the users in a section or in a place by calling the same methods on a Section object or the Place object, respectively.

Chapter 9. Server Application Storage Service

The Server Application Storage Service stores user-related information as attributes directly on the Sametime server. Your Sametime applications can use this service to access standard Sametime attributes, such as contact list and Sametime Connect preferences, or add their own application-specific attributes.

Do not confuse the attributes used in this service with the online attributes of the Awareness or Places Service. The Storage Service persistently stores user-related attributes, and only the owner user or another Server Application can retrieve the attribute values.

Querying stored attributes

To query an attribute of a user, you must have the attribute key and call the queryAttr or queryAttrList method and record the returned request ID. As a response to a query request, you receive an attrQueried event. Use the getRequestId method of the event object to compare the event request ID with the request ID you recorded previously. Use the getRequestResult method of the event object to find out if the query request was successful or not. The possible values are:

- **ST_OK** – All the requested attributes were retrieved successfully. Use the getAttrList method of the event object to get the list of successfully retrieved attributes.
- **STORAGE_ATTRIB_NOT_EXIST** – Some of the requested attributes were not retrieved because they had not been stored on the Sametime server. Use the getAttrList method of the event object to get the list of successfully retrieved attributes. Use the getFailedAttrKeys method of the event object to get the list of attribute keys that were not retrieved.
- **ST_FAIL** – None of the attributes was retrieved. Use the getFailedAttrKeys method of the event object to get the list of attribute keys that were not retrieved.

Receiving attribute updates

StorageServiceListener also contains the attrUpdated event. This event is generated if an attribute has been changed by a different login ID of the same user. In other words, the user is running more than one client, and one of the clients stored some attribute using the Storage Service. Use the getUpdatedKeys method of the event object to get the array of attribute keys that have changed.

Note The getUpdatedKeys method retrieves the attribute keys, not the actual values, of the changed attributes. Use the queryAttr method to retrieve the actual changed attribute values if they are needed.

Attributes are shared by all login IDs of the same user. The Storage Service stores only a single copy of each attribute on the Sametime server for each user.

Chapter 10. Light Login Service

The Light Login Service allows a MUX application to multiplex a connection to the server on top of an existing physical connection. Multiplexing provides the MUX application with the ability to log in on behalf of many users, without the overhead of creating separate connections (and connecting threads) for each user.

Using the Light Login service

In order to use the Light Login Service, an application has to be logged in to the server using the login type STUserInstance.LT_MUX_APP (to denote the fact that it is a MUX application). For each login ID, the application should create an STSession and use the Light Login Service to log in.

The Light Login Service provides methods for logging in using password or token authentication or for logging in anonymously. Each of these methods accepts a reference to the ServerAppService object that is responsible for the main login ID. These methods then log in using the same physical connection created by this main login ID.

Note A flag must be set in the Sametime.ini file in order to let an anonymous user log in to the community.

Chapter 11. Server Application Token Service

The Token Service provides the ability to generate a token that can be used to log in as a user to Sametime. As of version 8.5, the service provides an API that allows the generation of more than one token type.
This service is useful for an application that wants to log in on behalf of the user.

Using the Token service

The application should log in as a server application and get a token for each user who wants to be online. Alternatively, the application can ask for multiple tokens, and if the server supports multiple tokens, an array of tokens will be returned holding different token types. Then, using a generated token, it can log in as this user using the Light Login Service.

A token has a time limit, depending on the administrator configuration. It must therefore be used immediately after it is generated by this service.

The Token Service returns a Token object or several token objects, depending on the API used. There are 2 ways to login by token:

1. Use the Community Service method LoginByToken:
Use the token object's getLoginName and getTokenString methods to get the login name and token string to pass to the Community Service method loginByToken.

2. Use the Community Service method LoginByTokens:
Use one of the returned token objects to get the login Name and pass it along with the received array of tokens to the LoginByTokens method.

To get a login token using the TokenService:

Add a TokenServiceListener or a TokenServiceListener2 object by calling the addTokenServiceListener method of the Token Service.

Call the generateToken or generateTokens method of the Token Service, with the STUser object representing the user you want to get a token for as the parameter.

Receive the token in the tokenGenerated event of the TokenServiceListener object or in the tokensGenerated event of the TokenServiceListener2 object if multiple tokens were requested and the new listener type has been used.

Chapter 12. Online Directory Service

The Online Directory Service allows you to locate users and groups in the community. In a distributed environment, you may need to know the exact server to which a user is connected; for example, because you are accessing server side storage information for a specific user (see "Server Application Storage Service"). The storage service is provided locally and the call must be directed to the exact server to which the user is connected.

The online directory service also provides information such as whether or not a specific user is currently online. The service can also alert an application when a particular user comes online.

Locating users and groups

To locate a user or group in the community:

Create a Locator object.

Add a LocateListener to the Locator object.

Call Locator.locateUser (to find a specific user) or Locator.locateGroup (to find a specific group).

The locateUser method accepts an STUser object as its argument. The locateGroup method accepts an STGroup object as its argument.

Tip The most exact way of getting the STUser object of a user is to perform a resolve operation on the user using the Lookup Service provided with Java Client Toolkit and the Community Server Toolkit (the Community Server Toolkit extends the Java Client Toolkit and thus supports nearly all the client services). See the Java Client Toolkit documentation for information on using the Lookup Service.

Once the user is located, the userLocated event — or groupLocated event for a group — is dispatched through the registered LocateListener object. The retrieved information is returned in a LocatedUser object. From a LocatedUser object, you can get the ID of the server to which the user is connected and check to see if that user is currently online.

Requesting alerts

An application may need to know when an offline user becomes an online user. Although this can be done through the Community Events Service, that service allows you to get information only about all the users in the community. It doesn't allow you to watch a specific user. The Online Directory Service does.

To request an alert on a particular user in the community:

Create an Alerter object.

Add an AlertListener to the Alerter object.

Call Alerter.alert.

The alert method accepts an STUser object as an argument.

Tip The most exact way of getting the STUser object of a user is to perform a resolve operation on the user using the Lookup Service provided with both the Community Server Toolkit and the Java Client Toolkit. See the Java Client Toolkit documentation for information on using the Lookup Service.

Whenever this user comes online, the userOnline event is dispatched through the registered AlertListener. The retrieved information is returned in a LocatedUser object. From a LocatedUser object, you can get the ID of the server to which the user is connected and check to see if that user is currently online.

Chapter 13. Telephony Presence Adapter

The Telephony Presence Adapter provides an easy-to-use infrastructure for interfacing between Sametime and third-party telephony applications. The application that uses this component does not need to implement a Sametime Server Application. The Sametime Server Application is embedded inside the Telephony Presence Adapter.

After it has been adapted to the third-party telephony environment, the Telephony Presence Adapter publishes telephony status to the Sametime environment and gets notifications about the Sametime users' status changes that can be passed to the telephony environment. Specifically, the Telephony Presence Adapter gets publish and subscribe requests from the third-party implementation. It handles these requests and forwards them to the Sametime server. The Sametime server returns its notifications about user status changes to the Telephony Presence Adapter, which forward them to the third-party implementation.

Sametime 8.0-and-later versions provides the ability to publish telephony status for users that are both on-line and off-line in Sametime. Pre-Sametime 8.0 versions provide the ability to publish telephony status only for users who are on-line in Sametime.

Using the Telephony Presence Adapter

In order to send requests and receive responses, the third-party implementation should use two interfaces: 1) **TelephonyAdapterService**; and 2) **TelephonyAdapterListener**.

The **TelephonyAdapterService** includes the following services:

public void publishTelephonyStatus

(String telephonyUserId int telephonyStatus)

public void addWatch(String telephonyUserId)

public void removeWatch(String telephonyUserId)

public void setListener(TelephonyAdapterListener TelephonyAdapterListener)

public void initiate()

The **TelephonyAdapterListener** should be implemented by the developer in order to get the TelephonyAdapterService responses. It includes the following notifications:

public void sametimeStatusChanged(String telephonyUserId, int sametimeStatus)

public void communityAvailable()

public void communityNotAvailable()

public void errorEvent(TelephonyAdapterErrorEvent event)

The TelephonyAdapterErrorEvent encapsulates potential errors, such as:

Invalid telephony status

Invalid telephony user Id

The Telephony Presence Adapter is a singleton. The application should instantiate the Telephony Presence Adapter as follows:

Use getInstance() interface method to get a reference for the Telephony Presence Adapter.

Set itself as a listener in order to get the relevant events by setListener(this) method.

1. Start the Telephony Presence Adapter operation by using its initiate() method. The initiate method will activate all the adapter components, and create a session with each of the Sametime community servers.
2. When the Telephony Presence Adapter has at least one connection with a Sametime server it will send the CommunityAvailable event.
3. Upon receiving the CommunityAvailable event, the "addWatch", "removeWatch" and "PublishTelephonyStatus" events may be used.
4. Upon receiving the CommunityNotAvailable event all the previous requests are erased, and new requests should not be initiated until CommunityAvailable event is sent again by the Telephony Presence Adapter.

Please refer to the Telephony Presence Sample in the Toolkit tutorial.

Configuring the Telephony Presence Adapter

The Telephony Presence Adapter uses a configuration file named "st.telephony.adapter.properties".

1. Extract the file from the toolkit jar file.
2. Update it as follows:
3. Set the `connecting.server.dns` parameter to the qualified DNS name of any of the Sametime servers in the Sametime community.
4. Set the `support.offline.user.attributes` parameter to be false in cases where not **all** the Sametime servers in the community are of version 8.0 or later.
5. Save your changes and copy the configuration file to the local file system.
   Note that it should be in the expected classpath (see "___" below).

Configuring the Sametime directory

The Telephony Presence Adapter interfaces support the use of telephony user IDs. The Adapter handles resolving the user IDs to and from the Sametime ID. For this feature the Sametime directory should be configured to support resolve by the telephony user ID.

The directory configuration details are summarized in Appendix E.

Classpath considerations for the Telephony Presence Adapter application

The application's classpath should include the updated configuration file prior to the Sametime Community Server Toolkit jar.

Chapter 14. Cached Resolve component

The Cached Resolve component implements persistent cache of ST user resolved names. This component should be used by third party Sametime Server Applications.

A client application using the Cached Resolve Service APIs requests for user resolved name (Sametime ID). If a specific name is not represented in the cache, the component sends a resolve request to ST server. When resolve request is responded the component adds the resolved user name to the cache. For connecting to ST community the component uses a STSession object of the client (caller) application.

The caller application should log into ST community as a Server Application, and then should activate the necessary services (ServerAppService, CommunityEventsService, LookupService). The application should initialize the CachedResolveComp class instance and using an appropriate API should forward its own STSession object to the CachedResolve component. Once the CachedResolve has the STSession object of the Server Application, it may send resolve requests to ST server. The caller application should implement the CachedResolveListener interface for getting notifications on the persistent cache events.

Usage of the Cached Resolve component is effective for the applications managing a large amount of user names required being resolved. The component activates periodical synchronization process. It refreshes the cache (removes obsolete cache entries when it is required) and flushes the cache content from memory to disk. This operation is scheduled; it also may be started on demand using an appropriate API. The cache synchronization process is configured according to the st.cached.resolve.properties configuration file.

CachedResolveService APIs

The CachedResolveService service provides the following APIs:

initiate() – loads configuration file settings and initiates the component accordingly;

addServerSession(STSession serverSession, String serverName) - adds a Sametime server session. This session will be used for performing next resolve requests;

removeServerSession(STSession serverSession, String serverName) - removes a Sametime server session;

addListener(CachedResolveListener chacedResolveListener) - adds the specified CachedResolve service listener;

removeListener(CachedResolveListener chacedResolveListener) – removes the specified CachedResolve service listener;

resolveUser(String externalUserId, STEvent origEvent);

resolveUser(String externalUserId) - resolve user external name;

removeUser(String externalUserId, STEvent origEvent);

removeUser(String externalUserId, STEvent origEvent) - remove user from the cache if exist;

flushMemoryToFile() - stores the cache data to the persistent file;

loadFileToMemory() - loads the persistent cache data into the memory;

removeAllUsers() – cleans the cache up.

Sametime Server Applications may access the CachedResolveService APIs by retrieving of the CachedResolveComp singleton instance that implements the CachedResolveService interface:

CachedResolveService service = CachedResolveComp.getInstance();

To be notified on CachedResolve component events the application should implement the CachedResolveListener interface and register itself as a CachedResolve service events listener:

service.addListener(this);

Then the application has to initiate the component. The calls order is significant, since on the component initialization stage the listener may get some cache events (e.g., that notification on the cache file loading into the process memory):

service.initiate();

The application should connect to Sametime community as a Server Application and wait for the Login event from ST Community. When the application has been logged into the Community, it should forward its STSession object to the CachedResolve component:

service.addServerSession(_stSession, _serverName);

Once the application gets resolverAvailable() notification, it may send resolve requests to the CachedResolve component:

service.resolveUser("user1@lotus.com");

CachedResolveListener APIs

There are the following notifications in the CachedResolveListener interface:

nameResolved(String userExternalId, STId userId, STEvent origEvent) – user name is resolved;

resolverAvailable() - An informative event to inform the application that the initialization phase is completed (at least one STSession object is added to the component – see the *addServerSession* API). No special operation is required;

resolverNotAvailable() – Notifies that the community is no longer available. No special operation is required;

flushMemoryToFileCompleted() - Notifies the persistent file was loaded into memory successfully. No special action is required;

loadFileToMemoryCompleted() - Notifies that the persistent file was updated successfully. No special action is required;

errorEvent(CachedResolveErrorEvent event) - This method passes different errors to the application.

Possible error cases are listed below:

- ERROR_NULL_EXTERNAL_ID: Received external UserId is null
- INTERNAL_ERROR_RESOLVING_USER: CachedResolve Manager internal error while resolving a user.
- ERROR_NULL_STSESSION: The session for the resolver is null
- ERROR_FAILED_TO_LOAD_FILE: There was an error while loading the persistent file to memory
- ERROR_FAILED_TO_FLUSH_MEMORY: There was an error while updating the persistent file.
- ERROR_REMOVE_FAILED: The removal of user failed.
- ERROR_RESOLVER_NOT_AVAILABLE: The resolver is not available at the moment for performing resolve requests.
- ERROR_USER_RESOLVING_IN_PROCESS: The resolve request is pending more that the max-resolve-response-time.
- ERROR_FILE_NOT_CREATED_DUE_QUICK_STOP: The new persistent file is not created since resolved data is flushed to a new file when the loading of previous file is not finished yet.
- ERROR_CACHE_FILE_IO_OPERATION_FAILED: The persistent file IO error.
- ERROR_RESOLVING_NONE_UNIQUE_USER: User has a few entries in the organization Directory.
- ERROR_CACHE_FILE_CREATION: Error occurred during cache file creating.
- ERROR_CACHE_NOT_FOLDER: The cache folder name is not a folder name.
- ERROR_CACHE_FOLDER_NOT_READABLE: The cache folder is not readable.
- ERROR_CACHE_FOLDER_NOT_MODIFIABLE: The cache folder is not modifiable
- ERROR_NO_CACHE_FILE_FOUND: No cache files for loading found.
- ERROR_CACHE_VERSION_NOT_VERIFIED: The cache file version does not match the component version.

Cached Resolve component configuration

Cached Resolve component may be configured using **st.cached.resolve.properties** file. The component implementation for release 9.0 does not support online configuration parameters updating. The parameters are loaded at the component initialization time only. If the properties file is updated, the Server Application should be restarted to apply configuration changes for the persistent cache component.

Table of Cached Resolve components with descriptions, type, and default value.

Name	Type	Description	Default value
cache.max.entries	Number	The maximum number of entries to keep at the cache.	15000
cache.sync.period	Hours	The time period for activating the sync process. This process copies the cached entries from memory to disk, and clean obsolete entries if required.	24 Hours
cache.sync.start.time	Time	Alternative configuration parameter to set a specific start time for the synchronization process. The process will be activated every < cache.sync.period > hours.	00:00:00
cache.obsolete.time	Days	The "age" of not used entries that should be deleted. If the value is 0, the cache entries are never expired	0 Days
resolve.estimated.time	Milliseconds	The resolve request estimated time. Used for detecting of none responded resolve requests.	5000
resolve.notifications.start.time	Milliseconds	The time to wait on startup before sending the resolved failed status report.	1200000 (20 Min.)
resolve.timer.rate	Milliseconds	The time to wait between notifications about failed resolve status report.	300000 (5 Min.)
cache.persistent.file.num	Number	The number of old version of the cache persistent files that should be kept on the disk	2
cache.persistent.file.path	String	The folder name of the cached resolve persistent file.	No value, meaning, it would be the working directory

Appendix A. The Java Launcher

The Java Launcher is an application provided by Sametime that registers Java applications as Windows® services. When you write a Sametime server application on a Windows platform, you should implement it as a Windows service for easier control and management. IBM iSeries does not provide a Java Launcher application. For details about IBM iSeries applications, please see the section "Launching the Server Application."

System requirements

The Community Server Toolkit requirements are as follows:

- Sametime server
- Java Runtime Environment (JRE) 1.4.2 or later installed on the machine running the server application

Installation procedure - Windows

1. Copy the following files to a directory on the Sametime server:
    - **JavaLauncher.exe** – The application file
    - **stcommsrvrtk.jar** – Contains all the files that are needed to write a server application
1. Install the new server application as a service by executing the following command from the console:

```
JavaLauncher <options> service=ServiceName class=JavaClass [VM
              parameters: <-D|-X>JavaVMArg1 D| X>JavaVMArg2...]
               [Application parameters: JavaAppArg1 JavaAppArg2...]
                                                  [wrkdir=pathname]
```

You may specify the following command line arguments to JavaLauncher.exe:

Table listing command line arguments for JavaLauncher.exe.

<options>	• i – Install the service • r – Remove the service • c – Run as a console application
service	The name of the service
class	The name of the Java class that will be launched (the main class)
VM parameters	• D – Precedes arguments to be passed to the Java VM • X – Precedes arguments to be passed to the Java VM • wrkdir – Path to working directory. The winnt/system32 path is the default if none is specified **Note** All other arguments are passed to the Java application.

Example:

```
JavaLauncher -i service=JavaService class=MyClass -
      Djava.class.path=c:\myapp\l\myapp.jar apparg1 -
      wrkdir=c:\myapp\support
```

This code installs the coded Java application as a service:

- Starts the application by calling the MyClass main function
- Supplies the -D prefixed arguments to the JVM
- Changes the current directory to the specified working directory

Other arguments are supplied to the Java application's main class when the service starts.

Launching the server application

Windows applications

The two ways of launching the server application after it has been registered are:

- Start the server application by launching the installed service from the services panel of the operating system.
- Set up the service to launch automatically with the other Sametime services. To do this, add the following line to the StCommLaunch.dep file (located in the Sametime server's installation directory):

```
SERVERAPP 'service name', 'dependencies', SOFT.
```

The dependencies parameter lists all the other Sametime services that are required before launching this service.

Note After modifying the StCommLaunch.dep file, restart the Sametime server.

IBM iSeries (IBM i) applications

The two ways of launching the server application after it has been registered are:

- Start the Java server application manually by launching it from the console resulting from issuing STRQSH command on IBM iSeries.
- Set up the service to launch automatically with the other Sametime services.

To set up the service to launch automatically:

- Add a symbolic link to Yourpgm in /qibm/userdata/lotus/notes using the console resulting from issuing STRQSH command on IBM iSeries:

```
ADDLNK OBJ('/QIBM/PRODDATA/JAVA400/JDK13/BIN/JAVA')
NEWLNK('/qibm/UserData/LOTUS/NOTES/Yourpgm.PGM')
```

- Add an entry like the following to the meetingserver.ini file located in the Sametime server's data directory:

```
[SOFTWARE\Lotus\Sametime\MeetingServer\Services\Yourpgm]
Path=YOURPGM
Type=1
Enabled=1
StartOrderDependencies=ActivityProvider
RestartDependencies=
ArgString=com.lotus.sametime.yourprogramclass
StartWaitTime=0
StopWaitTime=0
```

Description=Your Sametime Program The above property keyword entries have the following meanings:

`[SOFTWARE\Lotus\Sametime\MeetingServer\Services\Yourpgm]`

This entry is a group entry similar to a registry group heading. "Yourpgm" can be set to any alphanumeric text string of 10 characters of less.

`Path=YOURPGM`

Set Path= an alphanumeric text string that matches the symbolic link name you used in the previous step (ADDLNK OBJ('/QIBM/PRODDATA/JAVA400/JDK13/BIN/JAVA') NEWLNK('/qibm/UserData/LOTUS/NOTES/YOURPGM.PGM')).

`Type=1`

Type 1 denotes a Java program running in its own process in the Sametime server subsystem. Using Type 1 will make it easier to deploy your application.

`Enabled=1`

To disable the application, set Enabled=0

`StartOrderDependencies=ActivityProvider`

This entry determines when your application will be started relative to other Sametime server applications. If your application has a dependency on other Community services, use ActivityProvider for this entry.

`RestartDependencies=`

This entry is required and should be entered exactly as shown.

`ArgString=com.lotus.sametime.yourprogramclass`

Denotes the Java class name that contains your main entry point.

`StartWaitTime=0`

This entry is required and should be entered exactly as shown.

`StopWaitTime=0`

This entry is required and should be entered exactly as shown.

`Description=Your Sametime Program`

Text description of the service's function.

2. In the [SOFTWARE\Lotus\Sametime\MeetingServer\Services] of meetingserver.ini, update the VM Arguments entry to include the classpath to your Java application: VMArguments=java qibm/proddata/lotus/sametime/java: $DOMINO_JAVA_PATH/notes.jar:/qibm/proddata/lotus/sametime/i18n.jar:…….

   The entry after the java keyword is used as the CLASSPATH environment variable for the Java JVM.

Note After modifying the meetingserver.ini file, restart the Sametime server.

Appendix B. Sametime identifiers

Sametime services use identifiers for many purposes. These purposes include:

- Service types supported by a server application
- Activity provider types
- Persistent user storage attribute keys

Sametime-enabled applications must not conflict with each other by using the same identifier for different purpose; for example, two separate companies could produce Sametime-enabled products that happen to use the same identifier, even though each identifier was intended for a different use. For this reason the following convention has been adopted for all Sametime identifier types:

- The range 0–100,000 is reserved for internal Sametime/Lotus/IBM use.
- Any third-party developers that want to allocate an identifier range for their Sametime-enabled applications must email to Sametime_ID_Request@lotus.com. An allocation of Sametime IDs will be sent by return email.

For a list of identifier types used in Sametime applications, refer to Appendix E of the Sametime Java Client Toolkit Developer's Guide.

Appendix C. Component Dependencies for Loading and Packaging

The first step in using the Sametime Community Server Toolkit is to create a Sametime session and load the components into it. This appendix describes component dependencies and restrictions and the various ways to load components.

Ways to load components

When you use the Sametime Community server Toolkit, you must explicitly define the names of the components to load. STSession.loadSemanticComponents and STSession.loadAllComponents load only client components, not Sametime Community Server components. Instead, use STSession.loadComponents or create the components directly.

Using STSession.loadComponents

STSession.loadComponents accepts an array of component names and loads them into the session. The names of the components must be the fully qualified names of the Java classes that implement the component interfaces. (See "Available Components," below for a list of these classes.)

For example:

```
STSession session = new STSession("A name");
String [] components = {
     ServerAppService.COMP_NAME,
     ActivityService.COMP_NAME
     };
session.loadComponents(components);
ActivityService as = (ActivityService)
     session.getCompApi(ActivityService.COMP_NAME);
```

Creating components directly

Another way to load components is to instantiate the component objects yourself, using the "new" operator with the component implementation class. To the component constructor, pass a reference to the containing Sametime session. The component will register itself to that session. If you keep a pointer to the component while loading it, you do not have to call STSession.getCompApi.

For example:

```
STSession session = new STSession("A name");
new STBase(session);
// Keep the reference to the Activity component,
// instead of calling 'getCompApi'.
ActivityService as = new ActivityComp(session);
```

Loading components from the Sametime Java client toolkit

If you need to use services from the Sametime Community Java Client Toolkit you can use the techniques described above, or you can use STSession.loadAllComponents or STSession.loadSemanticComponents instead. For more information, refer to the Sametime Java Toolkit documentation.

Restrictions

Note the following restrictions when loading components:

- Dependencies exist between components. When you load a component selectively, make sure to also load all the components it depends upon. For more details, see "Component Dependencies," below.
- The Community Services components must be loaded before the session is started.

Available components

The following table lists the available Server Toolkit Services and the classes that implement them. The implementation class can be passed to STSession.loadComponents or created directly.

Table listing Server Toolkit Services and associated classes.

Interface	Implementation Class
ServerAppService	com.lotus.sametime.community.STBase
ChannelService	com.lotus.sametime.community.STBase
CommunityEventsService	com.lotus.sametime.communityevents.CommunityEventsComp
GeneralAwarenessService	com.lotus.sametime.generalawareness.GeneralAwarenessComp
PlacesAdminService	com.lotus.sametime.placessa.PlacesAdminComp
PlacesActivityService	com.lotus.sametime.placessa.ActivityComp
SAppStorageService	com.lotus.sametime.storagesa.SAppStorageComp
LightLoginService	com.lotus.sametime.community.STBase
SATokenService	com.lotus.sametime.token.SATokenComp
OnlineDirectoryService	com.lotus.sametime.onlinedirectory.OnlineDirectoryComp

Component dependencies

Components are interdependent. When loading a component selectively, make sure to also load all the components it depends upon. The table below lists the component dependencies.

Table listing component dependencies.

Interface	Depends on
ServerAppService	
ChannelService	
CommunityEventsService	ServerAppService
GeneralAwarenessService	ServerAppService
PlacesAdminService	ServerAppService
PlacesActivityService	ServerAppService
SAppStorageService	ServerAppService, OnlineDirectoryService
LightLoginService	

SATokenService	ServerAppService
OnlineDirectoryService	ServerAppService

Appendix D. Using the Sametime Java Toolkit

The Sametime Community Server Toolkit and the Sametime Java Toolkit have the same architecture. Therefore, the services and UI components that are available from the Java Toolkit can also be used when developing applications with the Community Server Toolkit.

What is the Java toolkit?

The Sametime Java Toolkit is a collection of building blocks or components that developers use to build applications that influence the functionality and services provided by Lotus Sametime. These toolkit components can be used in any standard development environment that supports JDK 1.1.8 or above.

This toolkit is used by developers to embed Sametime-based services and functionality in Web applications. For example, a developer could use the toolkit to build a facility for live customer-service-style help in an online marketplace, or perhaps build awareness into a knowledge management application, or to bring application sharing into an e-business application.

The Java Toolkit also provides a standard user interface (UI) that can be used by developers to speed up the development time required to build a Sametime-enabled application. The standard UI is provided either by Toolkit UI components (similar to the Toolkit components providing the different Community and Meeting services) or by AWT dialogs and panels, which can be embedded inside any AWT Container.

Using the Java toolkit in server applications

Although you can use Java Toolkit components and services in your server applications, you can expect that some of them will behave differently when used in a server environment. For example:

- When using the AwarenessService in a server application, the server does not check the privacy settings of users before adding them to a watch list.
- If your server application exchanges instant messages with a Sametime Connect user, the name of the user will not appear in the message window.

For more information on the Java Toolkit, see the Sametime Java Toolkit Developer's Guide or the Sametime Java Toolkit Tutorial.

Appendix E. Configuring the Sametime Directory to Support Resolve by Telephony User ID

The Sametime community uses a directory to manage the users in the community. Sametime supports the use of LDAP and Domino directories. One of the Sametime services is resolving user IDs. The resolve process locates the user's entry in the directory being used by the Sametime server and returns the Sametime ID. To facilitate resolving telephony IDs, the directory must be configured to allow for search on an appropriate field.

Sametime configured with Domino directory

The resolve can search on one of the following fields:

FirstName

- LastName
- UserName: this field contains the user hierarchical name and any other variation
- ShortName/UserID field : the user's short name. This field is also the default for the mail file name and the userid file name

Any string written in the fields ShortName/UserID, UserName, FirstName LastName can be used as an input for the name lookup method.

Sametime configured with LDAP directory

If the provided string is formatted as a distinguished name (DN), Sametime finds it using a base-scope search. This kind of search is very efficient, because we tell the LDAP server exactly where to look in the directory tree. If the string is not formatted as a DN, Sametime issues a search operation using base, scope, and filter retrieved from the Sametime Configuration.

The attribute names are specified in the LDAP configuration document in StConfig.nsf.

This document includes the definitions of the filter used for searching the directory, the definition of the base DN, scope of search, and other attributes.

The search performed by the resolve can be customized by editing the LDAPserver document fields. For exmple, if we add the title field to the search filter :

(&(objectclass=organizationalPerson)(|(cn=%s)(givenname=%s)(sn=%s)(mail=%s) (title=%s)))

We can use the value stored in the title attribute for each entry in the LDAP server as an input for the resolve operation.

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.