Sametime
Version 9.0

Sametime 9.0

Software Development Kit

Connect Web API 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 Sametime (program number 5725-M36) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Introduction

Overview

The IBM® Sametime® Connect Web API Toolkit is a web application programming interface (API) that provides an external interface to basic functionality of the IBM Sametime Client. It allows Web developers to Sametime-enable their Web pages and applications with "livenames." Web-based applications that integrate the Connect Web API are essentially able to proxy the functionality of the locally running Sametime Client (managing contacts, starting chats, presence status).

The Connect Web API Toolkit particularly differs from another Sametime Web-based toolkit, Sametime Links, in this respect. Whereas the Sametime Links Toolkit is self-contained by the use of JavaScript and applets, the Connect Web API Toolkit requires the client to be installed. It allows for a more seamless integration with the Sametime Client.

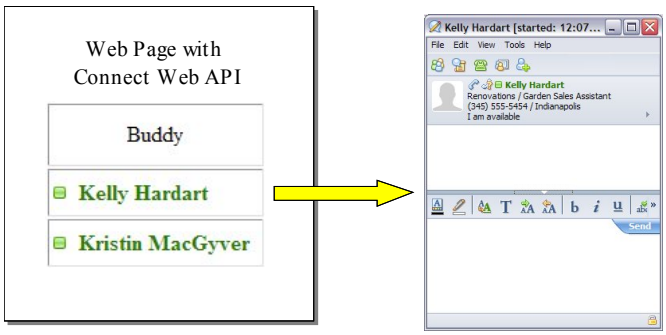


Figure 1: The Sametime Connect Web API interacts directly with the running Sametime client.

Additional information about Sametime is available at http://www.lotus.com/sametime.

Requirements

To work with the samples described in this guide and to create your own Sametime -enabled pages, you will need the following:

- IBM Sametime installed
- IBM Sametime Software Development Kit
- A client operating system supported by IBM Sametime Connect. See the release notes on your IBM Sametime server for a list of supported client operating systems.
- A text editor

Chapter 2. Getting Started

If you haven't yet installed the SDK, you can download it from the IBM developerWorkds Lotus toolkit download site at: http://www.ibm.com/developerworks/lotus/downloads/toolkits.html.

The Sametime Connect Web API Toolkit can be found in the `connectWebApi` directory of the client SDK; for example: `\sdk\client\connectWebApi`

The following outlines the Sametime Connect Web API Toolkit contents (other contents of the IBM Sametime SDK are not shown):

`\connectWebApi`

`\readme.txt`

Includes information about this toolkit including important last minute updates.

`\doc`

`\ConnectWebApiDevGuide.pdf` - *This development guide*

`\samples` - *Connect Web API Samples*

`\sample1.html` - *Basic static livename sample*

`\sample2.html` - *Basic dynamic livename sample*

`\sample3.html` - *Enhanced livename sample*

`\sample4.html` - *Contact action sample*

`\sample5.html` - Extending Recent Buddies sample

Chapter 3. Sametime Connect Web API

The Sametime Connect Web API is primarily a collection of servlet-based actions you can invoke on the client. The servlets themselves run in a web container on port 59449 within the Expeditor platform that the Sametime client uses. These actions can be invoked manually using forms in straight HTML, JavaScript posts, etc. They can also be invoked using Javascript wrapper functions available in `sametime.livename` objects.

You can also extend the functionality of the API by creating your own servlet-based actions using the `com.ibm.collaboration.realtime.webapi.function` extension point. The following sections will provide more detail on each option.

Note: The Sametime Connect client must be running and logged in for the web APIs to function.

Sametime Livenames

A Sametime livename is a UI representation of a contact. It displays the contact's online status using a status icon and bold coloring of the contact's display name (or nickname). Clicking on a Sametime livename created using the Connect Web API will initiate a chat with that contact.

Figure 2: Sametime livenames (active and away).

Note: The default Sametime livename implementation does not provide for right-click contextual menu options, but it could be easily enhanced to add them.

Adding livenames to your Web pages can be done in two ways:

1. Using a special element tag:

```
<a class="awareness" userId="<CONTACT ID>">
```

2. Constructing `sametime.livename` objects for every contact dynamically using JavaScript:

```
// Create an element to store the livename in

var contactElement = document.createElement("div");
```

```
contactElement.setAttribute("userId", "<CONTACT ID>");

contactElement.setAttribute("class", "awareness");

document.getElementsByTagName("body")[0].appendChild(contactElement);

// Create a sametime.livename

var contact = new sametime.livename("<CONTACT ID>", contactElement);
```

Sametime Livename Functions

If you have created your livenames using the JavaScript API, the following functions are then available to you in the `sametime.livename` object:

Table listing functions available in the sametime.livename object.

Name & Description	Return
start() Start watching awareness for this contact.	None
stop() Stop watching awareness for this contact.	None
getUsername() Returns the ID used to construct this livename.	ID for this livename object.
getElement() Returns the DOM element container for this livename.	DOM element container for this livename object.
runAction(action) Invokes the service indicated by the action parameter. The action parameter designates the context root of the action to run. A table of Sametime supported actions is below. You can also create your own action by writing an extension. See Chapter 4 for more details.	None

Working with Actions

Manually Invoking Actions

The Sametime servlet actions take the following form:

```
<BASE_SERVLET_URL>/<ACTION>?
userId=<USER_ID>[&<OPTIONALPARM>=<OPTIONALVALUE]
```

Where `<BASE_SERVLET_URL>` is:

```
http://localhost:59449/stwebapi/
```

For example, to open a chat with John whose ID is his e-mail address (john@acme.com):

```
http://localhost:59449/stwebapi/chat?userId=john@acme.com
```

It's also a good idea to pass in a timestamp parameter (time) as browsers may cache requests:

```
http://localhost:59449/stwebapi/chat?userId=john@acme.com&time=Tue%20Apr
%201%2012:52:50%20EDT%202008
```

Once you have the correct URL to use, you can either use this directly in the HTML as say an action to a form, or you can do a JavaScript or DOJO (AJAX) call to open the post.

Manually Invoking Actions with Callbacks

Some actions are simple and require just a post to invoke the action and then you're done – the `quickfind` action is an example of this. Other actions like `getstatus` require you to handle results that come back. This can be done by registering a callback function. You can manually register a callback function by adding your action post in a dynamic script tag append to the page, and then specifying the callback function name in an argument (`jsonp`) to the action. For example:

```
var getContactInfo = function(userid)
{
      var date = new Date();


   var url = sametime.servletUrl + "getstatus?userId=" + userid;

      // When the servlet returns, the callback
   // function (updateContactInfo) will be triggered.
      var head = document.getElementsByTagName("head")[0];
      var script = document.createElement("script");
      url =  url + "&jsonp=updateContactInfo&time=" + date.getTime();
      script.src = url;
      script.id = "stsamplescript" + userid;
      script.type = "text/javascript";
      head.appendChild(script);
```

```
}

var updateContactInfo = function(partner) {
      if (partner != null) {
            alert(partner.statusMessage);
   }
}
```

When the servlet returns, the `updateContactInfo` function will be called and the function argument (`partner`) will contain the results of the `getstatus` action in JSON format.

Invoking Actions using the Sametime Livename APIs (including callbacks)

The `sametime.livename` object provides wrapper functions for invoking actions. Some particular actions are wrapped with their own functions. The API also provides two generic wrappers for any action: `runAction(action)` and `runActionWithCallback(action, callback)`. These functions allow you to pass any action string (and callback function name) in and the API will handle constructing and invoking the URL and script tag append if necessary.

For example, to chat with John whose ID is his e-mail address (john@acme.com):

```
var contact = new sametime.livename("john@acme.com", contactElement);
contact.runAction("chat");
```

Or, to get some of John's status information:

```
var contact = new sametime.livename("john@acme.com", contactElement);
contact.runActionWithCallback("getstatus", "updateContactInfo");

var updateContactInfo = function(partner) {
      if (partner != null) {
            alert(partner.statusMessage);
   }
}
```

Sametime Supported Actions

NOTE: Many of these functions are disabled by default. To see the current state of each function, login to Sametime and go to http://localhost:59449/stwebapi/listservices in your browser. Functions that are disabled can be enabled by adding a preference to the plugin_customization.ini in the following format:

com.ibm.collaboration.realtime.webapi.<function>Enabled=true

For example, to enable the addgroup function you would add this preference:

com.ibm.collaboration.realtime.webapi.addgroupEnabled=true

To enabled all the Web API functions you can use the global override:

com.ibm.collaboration.realtime.webapi/enableAllWebApisOverride=true

Table listing parameters for Sametime supported actions.

Action (context root)	Parameters	Description
addgroup	None	Launches Add Group dialog.
addtolist	userId = User contact ID.	Launches the "Add to Sametime Contact List" dialog for the indicated user.
announce	userId = User contact ID or | delimited list of contact IDs	Send an announcement.
call	userId = User contact ID. number = The number you'd like to dial.	Start a telephony call with a user by userid or number. Requires telephony capabilities.
chat	userId = User contact ID.	Launches chat window with user.
getprivacylist	None	Returns the current local user's privacy list.

getstatus	userId = User contact ID
	
	jsonp = callback function.
		Returns full status information for a specified user or users in JavaScript Object Notation (JSON) format. The status information will include the following items if values can be obtained for them:
		`locationInfo`
		`countryName`
		`timeZoneId`
		`cityName`
		`phoneNumber`
		`postalCode`
		`stateName`
		`userDefinedLocationName`
		`resolvedName`
		`communityId`
		`gatewayCommunity`
		`hoverText`
		`displayName`
		`status`
		`isExternal`
		`statusMessage`
		`username`
		`contactId`
		`id`
		`lastChatTime`

getstatusshort	userId = User contact ID
	
	jsonp = callback function.
		Returns a subset of status information for a specified user or users in JSON format, namely:
		`status`
		`statusMessage`
		`username`

		displayName
instantshare	userId = User contact ID.	Start an Instant Share session with a user. Requires Sametime Advanced 8.0 or later.
listservices	callback = Callback function.	Returns a list of available actions in JSON format: name path
loggedin	fqdn = the fully qualified domain name of the community server	Returns whether the local user is logged in the given community server. If fqdn is not provided, it returns for the default community.
mystatus	jsonp = callback function.	Returns the current local user's status in JSON format. The status information will include the following items if values can be obtained for them: locationInfo countryName timeZoneId cityName phoneNumber postalCode stateName userDefinedLocationName resolvedName communityId gatewayCommunity hoverText displayName status isExternal statusMessage username contactId id lastChatTime
quickfind	None	Open a mini quickfind window.
remove	userId = User contact ID.	Remove user from the Contact List
rename	userId = The contact ID. nickname = The new nickname.	Rename user from the Contact List.
setstatus	status = The new status. message = Optional status message.	Set the status of the current user on the default community.
startmeeting	userId = User contact ID.	Starts an Instant Meeting with this user. Requires the server is configured for instant meetings.

NOTE: getbuddylist and getdirinfo have been removed permanently from the Web API in Sametime 8.5.1 due to potential security risks.

NOTE: Due to the ambiguous nature of the remove function, it has been disabled. The problem arises when a name appears in the buddylist more than once. In this case the client does not know which instance the user has selected and therefore which to remove, because such context is not preserved between web applications and the Sametime client.

NOTE: By default, all values returned by the functions are filtered such as any potentially harmful contents are removed. To disable the filter feature, set below preference to false.

com.ibm.collaboration.realtime.webapi/useActiveContentFilter=false

Servlet API and Extension Point

The Sametime Connect Web API can also be enhanced by creating your own custom servlet-based actions in your custom plug-ins. This can be done by implementing the `com.ibm.collaboration.realtime.webapi.function` extension point:

Web API Function

Identifier: com.ibm.collaboration.realtime.webapi.function

Since: Sametime 8.0.1

Description: Extension point for creating a new Sametime Connect Web API enabled function.

Configuration Markup:

```
<!ELEMENT extension (webapiFunction+)>

<!ATTLIST extension

point CDATA #REQUIRED

id    CDATA #IMPLIED

name  CDATA #IMPLIED>

<!ELEMENT webapiFunction (messageEvent | requestForwarder | usage)>

<!ATTLIST webapiFunction

id             CDATA #IMPLIED

path           CDATA #IMPLIED

label          CDATA #IMPLIED

labelNLS       CDATA #IMPLIED

labelNLSBundle CDATA #IMPLIED>
```

- **id** – Unique ID of this action.
- **path** - Path of the action. Example: ‘chat’.
- **label** - The display name of the service. Either label or labelNLS needs to be set. If labelNLS is set, it will be used over label. See labelNLS for more info.
- **labelNLS** - Get the display name for this api function from an org.eclipse.osgi.util.NLS object. The whole attribute value will be used to get the label string (e.g. com.ibm.collaboration.realtime.imhub.Messages.addContact). If the bundle containing this NLS isn't this plugin, then labelNLSBundle attribute needs to be set.
- **labelNLSBundle** - Required if labelNLS is set and you need to specify a different bundle to load the NLS.

```
<!ELEMENT messageEvent EMPTY>

<!ATTLIST messageEvent

code CDATA #REQUIRED>
```

- **code** – Unique message event code: The extension point will forward the http request and parameters to that message event. Required only if `RequestForwarder` is not defined.

```
<!ELEMENT requestForwarder EMPTY>

<!ATTLIST requestForwarder

class CDATA #IMPLIED>
```

- **class** – The RequestForwarder implementation servlet class.

```
<!ELEMENT usage (param+)>

<!ATTLIST usage

description CDATA #IMPLIED>
```

- **description** - A description of this service and parameters for documentation purposes.

```
<!ELEMENT param EMPTY>

<!ATTLIST param

name        CDATA #IMPLIED

description CDATA #IMPLIED

isPathInfo  (true | false) >
```

- **name** – Parameter name.
- **description** - Description of this parameter.

Servlets which extend this Web API Function extension point must implement the RequestForwarder interface:

```
com.ibm.collaboration.realtime.webapi.RequestForwarder
```

and in the particular the forward(…) method:

```
public void forward(HttpServletRequest request, HttpServletResponse
response);
```

Alternatively you can specify a unique message event code in your extension instead of a RequestForwarder servlet. In this case you would register a custom MessageHandlerListener and handle messages of this type appropriately.

Chapter 4. Sametime Connect Web API Toolkit by Example

The Sametime Connect API Toolkit contains a set of sample Web page applications. The sample applications are intended to demonstrate usage of the functionality available in the toolkit.

Basic Static Livename Sample

In this very basic sample, we will use the special element tags as described in Chapter 3 above with hard coded contact IDs.

This sample can be found in <SDK_HOME>/client/connectWebApi/samples/sample1.html

The first step is to import the main Sametime Connect Web API stylesheet:

```
<link rel="stylesheet" href="http://localhost:59449/stwebapi/main.css"
type="text/css" />
```

Next, we import the getStatus.js JavaScript include file. This includes many utility functions to deal with livename objects:

```
<script type="text/javascript"
src="http://localhost:59449/stwebapi/getStatus.js"></script>
```

Finally we can add HTML links for all of our contacts using the “awareness” class:

```
<a class="awareness" userId="<USER_ID>">Resolving contact, please
wait....</a>
```

Note: The link text is not strictly necessary. It will get replaced with the livename object.

To test this sample, edit the HTML directly and replace the instances of <USER_ID> with the actual contact IDs of your buddies.

The livename objects will display the status icon and the contact’s nickname as set in the client. When you click on a livename, it will automatically open a chat window in the Connect Client with the contact.

Basic Dynamic Livename Sample

This sample builds upon the basic static sample by providing a way to dynamically add contacts to your local buddy list. It uses the JavaScript APIs instead of the hard coded element tags.

This sample can be found in <*SDK_HOME*>/client/connectWebApi/samples/sample2.html

Here instead of a static table of links, we instead create a form where we can add buddies, and a table to hold the buddylist:

```
<form name="addContactForm">
   Enter user ID:
   <input type="text" name="buddyListUserId" id="buddyListUserId"/>
   <input type="button" value="Add to BuddyList"
         onClick="addLivenameToMyBuddiesTable(createLivename(documen
         t.getElementById('buddyListUserId').value));"/>
</form>
```

```
<table id="myBuddiesTable" border="1" cellpadding="3">
      <thead>
            <tr>
                  <td colspan="2" align="center"><B>My Buddies</B></td>
            </tr>
      </thead>
</table>
```

The sample has two JavaScript functions to dynamically add livenames to the table: `createLivename(userId)` and `addLivenameToMyBuddiesTable(livenameElement)`.

We will focus on the `createLivename(…)` function here.

First create an element container to hold the livename object, and set the userId and class attributes accordingly:

```
var livenameElement = document.createElement("div");
livenameElement.setAttribute("userId", userId);
livenameElement.setAttribute("class", "awareness");
```

Next create an actual livename object using the `sametime.livename` constructor which takes the userId and the containing element as arguments:

```
var livename = new sametime.livename(userId, livenameElement);
```

Finally, we tell the livename object to start watching awareness:

```
livename.start();
```

The result is a table full of livenames that display each contact's current status icon and nicknames as set in the client. If the API could not resolve a given user ID, then no livename will display.

Open the sample HTML in a text editor to view the full source.

Enhanced Livename Sample

This sample builds upon the dynamic sample by adding the ability to retrieve some status information for your contacts, namely their current status message and location information if available.

This sample can be found in <SDK_HOME>/client/connectWebApi/samples/sample3.html

In this sample we've added radio buttons next to each livename. When you select a radio button next to a livename, it will invoke the `getContactStatus` function. This function invokes the `getstatus` action using a callback as described in the **Manually Invoking Actions with Callbacks** section above:

```
// This function will trigger the call to retrieve the user's status (contact
information) from the Connect Client
var getContactStatus = function(userid)
{
      //    remove existing script if it is the same, to save memory
      var currentScript = document.getElementById("stscript" + userid);
      if(currentScript != null) {
            currentScript.parentNode.removeChild(currentScript);
      }

      var d = new Date();

      // The format of the URL to access is [servlet]/[action]?userId=[userid]
      var url = sametime.servletUrl + "getstatus?userId=" + userid;

      // Create a callback function (updateContactInfo) and add it to the page.
      // When the servlet returns, the callback function will be triggered.
      var head = document.getElementsByTagName("head")[0];
      var script = document.createElement("script");
      url =  url + "&jsonp=updateContactInfo&time=" + d.getTime();
      script.src = url;
      script.id = "stscript" + userid;
      script.type = "text/javascript";
      head.appendChild(script);
}
```

When the action returns, it will trigger the `updateContactInfo` function. This function checks the results and then retrieves the information we want to display from the object. Since the results are returned in JSON format (as defined in Chapter 3 above), we can just access methods on the returned result object:

```
var updateContactInfo = function(partner) {
      if (partner != null) {
            var infoHtml = "Status Message = " + partner.statusMessage + "<br>";
            if (partner.locationInfo.userDefinedLocationName != "") {
                  infoHtml = infoHtml + "Location = " +
partner.locationInfo.userDefinedLocationName + "<br>";
                  if (partner.locationInfo.phoneNumber != "") {
                        infoHtml = infoHtml + Phone = " +
partner.locationInfo.phoneNumber + "<br>";
```

```
                    }
                    if (partner.locationInfo.cityName != "") {
                          infoHtml = infoHtml + "    " +
                          "Address = " + partner.locationInfo.cityName + ", " +
partner.locationInfo.stateName + ", " + partner.locationInfo.postalCode + ", " +
partner.locationInfo.countryName;
                    }
              }
              document.getElementById("infoCell" + partner.contactId).innerHTML =
infoHtml;
        }
}
```

This results in an additional column next to the livename that includes the contacts current status message and location if available.

Open the sample HTML in a text editor to view the full source.

Contact Action Sample

The contact action sample builds upon the enhanced livename sample by adding some additional action buttons for each contact: Call, Instant Meeting and InstantShare. Each action is invoked using the Sametime Livename APIs. For example, we add the 'call' action by doing the following:

```
// Create a cell to hold the call action
var callCell = row.insertCell(2);
callCell.align = "center";
var callElement = document.createElement("input");
callElement.type = "button";
callElement.value = "Call";
callElement.name = "callCell" + userId;
callElement.id = "callCell" + userId;

// Set the action to be 'call':
callElement.onclick = function() { myBuddies[userId].runAction('call'); }

callCell.appendChild(callElement);
```

This sample can be found in <SDK_HOME>/client/connectWebApi/samples/sample4.html

Open the sample HTML in a text editor to view the full source.

Extending Recent Buddies

In our final sample, we will actually extend a sample in the Connect APIs Toolkit: Recent Buddies, and add a servlet-based action that we can access from a web page. The new action named 'recentbuddies' will return a list of recent buddy contact IDs.

The first step is to create a servlet in the Recent Buddies sample plug-in that implements the required `com.ibm.collaboration.realtime.webapi.RequestForwarder` interface:

```
package com.ibm.collaboration.realtime.sample.recentbuddies.servlet;

import com.ibm.collaboration.realtime.webapi.*;
…
…

public class RecentBuddyServlet implements RequestForwarder {
      public void forward(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        String pathInfo = request.getPathInfo();
        String responseString = "";

        // See if a callback function was declared
        String callback = request.getParameter("jsonp");

        // Set the status to be OK
        response.setStatus(HttpServletResponse.SC_OK);

        // Make sure the 'recentbuddies' action was specified
        if (pathInfo.startsWith("/recentbuddies") ||
pathInfo.startsWith("/recentbuddies"))
        {
            // Get the list of recent buddies
            List buddies = getRecentBuddies();
            Iterator i = buddies.iterator();

            // Cycle through each buddy and add their contact ID to the response
using a '|' delimiter
            while (i.hasNext())
            {
                  RecentBuddy buddy = (RecentBuddy) i.next();
                  responseString += buddy.getPerson().getContactId();
                  if (i.hasNext()) {
                        responseString += "|";
                  }
            }
        }

        // Format the response using the callback function name that was passed
in
        if ( (callback != null) && (!callback.equals("")) )
        {
            responseString = callback + "('" + responseString + "');";
        }
```

```
            response.setContentLength(responseString.length());
            response.getWriter().print(responseString);
            response.getWriter().flush();
            response.getWriter().close();
        }

        // Get a list of the recent buddies - this is a modified version of the
code that
        // can be found in RecentBuddyList.java. It uses the RecentBuddyListReader
to parse
        // the stored list of recent buddies.
        private List getRecentBuddies() {
              List recentBuddies = new ArrayList();
              try {
                    InputStream is;

                    if (new
File(RecentBuddiesPlugin.getRecentBuddiesFilespec()).exists()) {
                          is = new
FileInputStream(RecentBuddiesPlugin.getRecentBuddiesFilespec());
                          recentBuddies = new
RecentBuddyListReader().parseBuddyList(new InputSource(is));
                    }
              } catch (FileNotFoundException e) {
                    e.printStackTrace();
              }
              return recentBuddies;
        }
}
```

Next we register the `com.ibm.collaboration.realtime.webapi.function` extension point implementation in the Recent Buddies sample plugin.xml:

```
<extension
    point="com.ibm.collaboration.realtime.webapi.function">
    <webapiFunction
        id="com.ibm.collaboration.realtime.sample.recentbuddies"
        label="Recent Buddies"
        path="recentbuddies">
        <requestForwarder
class="com.ibm.collaboration.realtime.sample.recentbuddies.servlet.Recent
BuddyServlet"/>
    </webapiFunction>
</extension>
```

Now we update our HTML sample. In this sample, we've removed the table and form to add contacts. Instead we've added a function to the `onload` event of the page to manually invoke the '`recentbuddies`' action:

```
<body onload="loadRecentBuddies();">
```

where `loadRecentBuddies()` is:

```
var loadRecentBuddies = function() {
      var d = new Date();

      // The format of the URL to access is [servlet]/[action]?userId=[userid]
      var url = sametime.servletUrl + "recentbuddies/";

      // Create a callback function (updateRecentBuddies) and add it to the page.
      // When the servlet returns, the callback function will be triggered.
      var head = document.getElementsByTagName("head")[0];
      var script = document.createElement("script");
      url =  url + "?jsonp=updateRecentBuddies&time=" + d.getTime();
      script.src = url;
      script.id = "strecentbuddiesscript";
      script.type = "text/javascript";
      head.appendChild(script);
}
```

Finally, we implement our callback function to handle the results of the 'recentbuddies' action. We'll cycle through the IDs returned from the action and add livename objects to our page using functions we've defined in the previous samples:

```
var updateRecentBuddies = function(buddies) {
      if (buddies != null)
      {
            var myRecentBuddies=buddies.split("|") //split using | as delimiter
            // Now cycle through the list of contact IDs and add livename
objects to the page
            for (i = 0; i < myRecentBuddies.length; i++)
            {
                  addLivenameToMyBuddiesTable(myRecentBuddies[i]);
            }
      }
}
```

This sample HTML can be found in:

SDK_HOME/client/connectWebApi/samples/sample5.html

Open the sample HTML in a text editor to view the full source.

The Recent Buddies sample plug-in can be found in the Connect API Toolkit in the "samples" directory.

Appendix: Enabling HTTPS on the Sametime Web API

Support for HTTPS in the Web API comes from the Web Container in Lotus Expeditor. Standard instructions for enabling it are provided below. For more advanced options please refer to Expeditor Web Container documentation.

Note: Before beginning, make sure you have your keystore and truststore files ready. If using the SDK sample, your files should replace the ones found in the `com.ibm.collaboration.realtime.webapi.ssl.feature_version.jar`

Step 1 - Create the ssl.txt file

This file should contain the follow 5 parameters:T

```
com.ibm.ssl.clientAuthentication.59669=false
com.ibm.ssl.trustStorePassword.59669=plaintext_password
com.ibm.ssl.keyStorePassword.59669=plaintext_password
com.ibm.ssl.trustStore.59669=C\:/path_to_keystore/testkeystore.jks
com.ibm.ssl.keyStore.59669=C\:/path_to_truststore/testtruststore.jks
```

testkeystore.jks is your keystore file and *testtruststore*.jks is your truststore file. These files can exist anywhere in your files system. Pay close attention to the direction of slashes used in the paths. Using the wrong slashes will not work.
This is a temporary file that will be automatically deleted at a later step, so it is ok to include the plaintext password here. This file can be saved anywhere.

Note: If you are using the SDK sample, this file should replace the one found inside the `com.ibm.collaboration.realtime.webapi.ssl.feature_version.jar`

Step 2 - Update the rcpinstall.properties file of your Notes/Sametime install

`rcpinstall.properties` is located:

- Notes:

  Notes_location`\Notes\Data\workspace\.config`

- Sametime:

  `C:\Documents and Settings\`*user*`\Application Data\Lotus\Sametime\.config`

add the following 2 lines to this file:

```
-Dcom.ibm.pvc.webcontainer.port.secure=59669
-Dcom.ibm.pvc.webcontainer.ssl.configfile=C:/path_to_file/ssl.txt
```

Pay close attention to the direction of slashes used in the path. Using the wrong slashes will not work.

Alternatively, instead of modifying the `rcpinstall.properties` file directly, you can use the XPD install handler component to add these options through a feature packaged as a separate update site. An example of this is provided in the SDK.

Note: If you are using the SDK sample, these lines are found in the `handler.properties` file inside the `com.ibm.collaboration.realtime.webapi.ssl.feature_`*`version`*`.jar` and should be modified there. You will also need to modify the keystore and truststore file names found on the other two lines of this file. Those two lines are not necessary if you are just modifying the `rcpinstall.properties` file directly.

Some useful links that explain those steps in more detail:

- http://publib.boulder.ibm.com/infocenter/ledoc/v6r2/topic/com.ibm.rcp.tools.doc.admin/globalinstallhandler.html
- http://publib.boulder.ibm.com/infocenter/ledoc/v6r2/topic/com.ibm.rcp.tools.doc.admin/globalinstallhandler_adding.html

Step 3 - Enable the Web API (Notes only)

In Notes, the Web API feature is disabled by default. To enable it, find the following line in *`Notes_location`*`\framework\rcp\plugin_customization.ini` and change it to true:

```
com.ibm.collaboration.realtime.webapi/startWebContainer=true
```

Step 4 - Login to Sametime

Start the client and login to Sametime. Startup of the Web API feature is delayed for 15 seconds after you login to Sametime. After logging in and waiting 15 seconds, verify that the ssl.txt file has been deleted and a new webcontainer.properties files has been created. This new file will be created at this location:

- Notes:

  `Notes_location``\Notes\Data\workspace\.config`
- Sametime:

  `C:\Documents and Settings\`*`user`*`\Application Data\Lotus\Sametime\.config`

If this did not happen, verify that the path you specified to the `ssl.txt` is correct and that a `webcontainer.properties` file does not already exist. If one does exist, rename it to `webcontainer.properties.old` and try again.

Step 5 - Test that the Web API is running

Try to hit `https://localhost:59669/stwebapi/listservices` in a browser. If you are using test keystore and truststore files you may get a warning that the certificate is untrusted. Choose the option to add an exception for this certificate.

You can also verify that it is running by opening a command prompt or terminal window and typing `netstat -a` and then verifying that port 59669 is open and listening.

Troubleshooting

There are several reason why the Web API may be unreachable over HTTPS after configuration. Most often it is due to one of the issues listed below.

The ssl.txt file did not disappear

Did you wait 15 seconds after login? The Web API is on a 15 second delay from the time of successful login.

Is the path to the ssl.txt file correct? Verify that the path specified in the rcpinstall.properties file is correct. Relative paths are relative to the product install directory, not the directory containing the rcpinstall.properties file. Verify that slashes match the ones used in the example above.

Does a webcontainer.properties file already exist from a previous configuration? If so, delete this file and restart.

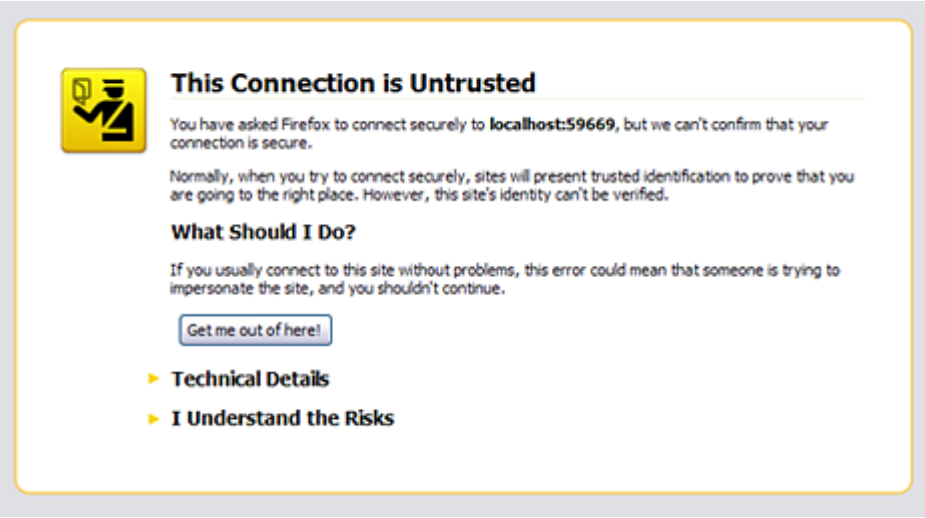
CWPWC0012E: Value null of the SSL property com.ibm.ssl.keyStore is not valid.

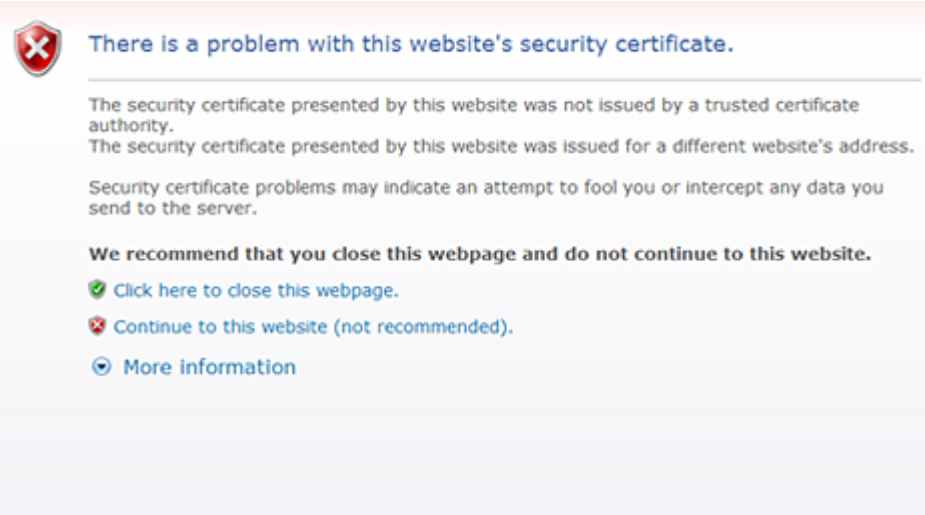
If this appears in the client trace log, it means the path of the certificate file in the `webcontainer.properties` file is incorrect. Do not modify the file directly, instead delete it and rerun the configuration steps.

Web Container SSL configuration has been tampered with and is now invalid

Something in the configuration process was corrupted. Delete the `webcontainer.properties` file and redo the configuration steps. If the error continues, the problem is probably the certificate files. Either recreate them or download them again from the original source.

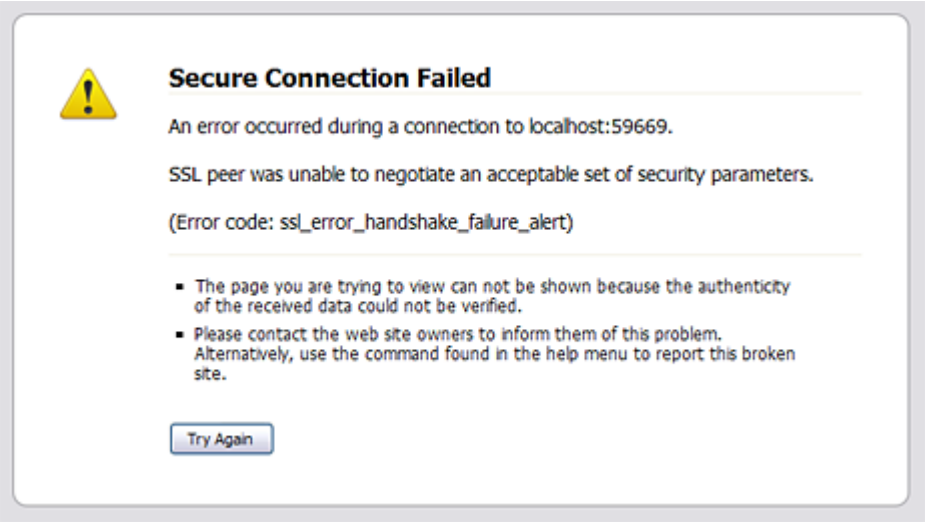
This Connection is Untrusted





This is not an error. This usually happens when working with a self-signed certificate. In Mozilla Firefox, click **I Understand the Risks** followed by the **Add Exception** button. In Microsoft Internet Explorer, just click **Continue to this website**.

Secure connection failed



This is usually caused by an error in the `ssl.txt`, and subsequently mirrored in the `webcontainer.properties` file. Verify that the following line is set to **false**, not true:

```
com.ibm.ssl.clientAuthentication.59669=false
```

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

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.