Platform SDK Developer's Guide

Chat

4/13/2025

Chat

You can use the Web Media Platform SDK to write Java or .NET applications that use the Genesys Web Media Server's chat, email and voice callback protocols. These applications can range from the simple to the advanced.

This article shows how to implement the basic functions you will need to write a simple Web Media Server application. It provides code snippets to illustrate how the `FlexChat` protocol can be used to support a simple chat application.

Which Chat Protocol Should I Use?

One possible point of confusion when developing a Chat application is which protocol to use, because Platform SDK includes two different protocols: Basic Chat and Flex Chat.

Basic Chat is for agent applications. This protocol provides a persistent TCP connection for the duration of the chat session. With this protocol, a bi-directional connection is established only when required by the chat session, and is closed when the session is finished. Therefore the total number of connections is never larger than the number of active chat sessions. An unexpected loss of TCP connection is treated as a client disconnect by this protocol.

Flex Chat is specifically designed for customer-facing applications. This protocol is for one-directional messaging, with no push possible, and cannot be used to close the chat session directly.

Java

Importing the Web Media Protocols

Before using the Web Media Platform SDK, you will need to import the appropriate packages. Since we will be using the FlexChat protocol, we will use the following import statements:

```
[Java]
import com.genesyslab.platform.webmedia.protocol.*;
import com.genesyslab.platform.webmedia.protocol.flexchat.*;
import com.genesyslab.platform.webmedia.protocol.flexchat.events.*;
import com.genesyslab.platform.webmedia.protocol.flexchat.requests.*;
```

Setting Up Web Media Protocol Objects

When interacting with existing chat sessions, you will need to store session-specific details including a secure key and user ID. Additional objects that will be needed include a `FlexChatProtocol` object (for sending and receive messages) and a `FlexTranscript` object (used to store and interact with the chat transcript).

```
[Java]
private String mSecureKey = null;
private String mUserId = null;
private FlexTranscript mTranscript = null;
private FlexChatProtocol mFlexChatProtocol = null;
```

To use the Web Media Platform SDK, you first need to instantiate a protocol object by supplying information about the Web Media Server you want to connect with. This example specifies values for the name, host, and port values:

```
[Java]
mFlexChatProtocol = new FlexChatProtocol(new Endpoint("FlexChat", "<hostname>", <port>));
Thread mListenerThread = new ListenForEventsThread(mFlexChatProtocol);
```

Note that you have to provide a string when you create the `FlexChatProtocol` object. This string should be unique for each protocol used in your application. It might be a good idea to use the name of the server's application object from the configuration layer, which guarantees uniqueness as well as clearly identifying which server you are communicating with.

After instantiating the `FlexChatProtocol` object, you need to open the connection to the Web Media Server:

```
[Java]
mFlexChatProtocol.open();
```

Note that you should always use proper error handling techniques in your code, especially when working with the protocol connection. To save space, these error handling steps are not shown in this example.

Logging in to Chat Server

```
[Java]
// filter the request based on our configured application name
KeyValueCollection kvUserData = new KeyValueCollection();
kvUserData.addObject("FirstName", "John");
kvUserData.addObject("LastName", "Smith");
kvUserData.addObject("EmailAddress", "john.smith@email.com");
RequestLogin reqLogin = RequestLogin.create(strNickName, 0, kvUserData);
Message msg = mFlexChatProtocol.request(reqLogin);
```

After successfully logging in to Chat Server, a message is returned that includes some important information: the Secure Key and User ID values. You will use these values when sending messages to the Chat Server, so remember to keep track of them for later.

```
[Java]
if (msg != null && msg.messageId() == EventStatus.ID)
{
        EventStatus status = (EventStatus)msg;
        if (status.getRequestResult() == RequestResult.Success)
        {
                mSecureKey = status.getSecureKey();
                mUserId = status.getUserId();
        }
}
```

Updating a Chat Session

By creating a `RequestRefresh` object, you can either check for updates or send new text to an existing chat session. The following sample shows how to create a `RequestRefresh` object, send it to the Chat Server, and process the result.

```
[Java]
RequestRefresh reqRefresh = RequestRefresh.create(mUserId, mSecureKey,
        mTranscript.getLastPosition() + 1, MessageText.create("text", message));
Message msg = mFlexChatProtocol.request(reqRefresh);
if (msg != null && msg.messageId() == EventStatus.ID)
{
        EventStatus status = (EventStatus)msg;
        if (status.getRequestResult() == RequestResult.Success)
        {
                processTranscript(status.getFlexTranscript());
        }
}
```

Working with Restricted Characters

Due to server-side requirements, the XML-based Webmedia Platform SDK protocols (BasicChat, FlexChat, Callback and Email) do not support illegal characters in string values. See http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char for the allowable character range.

The Platform SDK protocols do not change user data by default, but the following options are available if you want to replace illegal characters:

(1) Include code in your application to configure the protocol connection. For example:

```
[Java]
PropertyConfiguration conf = new PropertyConfiguration();
conf.setBoolean(WebmediaChannel.OPTION_NAME_REPLACE_ILLEGAL_UNICODE_CHARS, true);
// "replacement" value is optional: if it is not specified - illegal characters will be
removed
conf.setOption(WebmediaChannel.OPTION_NAME_ILLEGAL_UNICODE_CHARS_REPLACEMENT, "?");
BasicChatProtocol protocol = new BasicChatProtocol(new Endpoint("chatServer", HOST, PORT,
conf));
protocol.open();
```

(2) Set specific JVM properties for the client application using `webmediaprotocol.jar`. For example:

```
[Java]
"-Dcom.genesyslab.platform.WebMedia.BasicChat.replace-illegal-unicode-chars=true"
```

or

```
[Java]
"-Dcom.genesyslab.platform.WebMedia.BasicChat.replace-illegal-unicode-chars=true
-Dcom.genesyslab.platform.WebMedia.BasicChat.illegal-unicode-chars-replacement=?"
```

Using JVM system properties will affect all protocol connections for the specified Webmedia protocol. Using specific connection configuration values will only affect the specified protocol instance(s), and will take priority over JVM settings.

If no replacement character or string is specified, then illegal characters will be removed (that is, replaced with an empty string).

Values are extracted independently for the two methods listed above. If you enable character replacement using the `PropertyConfiguration` class without specifying a replacement value, but a replacement value is already specified through the JVM system properties, then characters will be replaced without verifying the enabling option in the JVM properties. It is recommended to use both options while writing connection configuration code.

Logging out of a Chat Session

When a client is ready to log out from the existing chat session, build a `RequestLogout` object and send it to the Chat Server.

```
[Java]
RequestLogout reqLogout = RequestLogout.create(mUserId, mSecureKey,
        mTranscript.getLastPosition());
Message msg = mFlexChatProtocol.request(reqLogout);
if (msg != null && msg.messageId() == EventStatus.ID)
{
        EventStatus status = (EventStatus)msg;
        if (status.getRequestResult() == RequestResult.Success)
        {
                processTranscript(status.getFlexTranscript());
        }
}
```

Disconnecting from a Chat Server

Finally, when you are finished communicating with the Chat Server, you should close the connection to minimize resource utilization.

```
[Java]
mFlexChatProtocol.close();
```

.NET

Using the Web Media Protocols

Before using the Web Media Platform SDK, you should include `using` statements that allow access to types from the Platform SDK Commons and Web Media namespaces. For the `FlexChat` protocol, we use the following statements:

```
[C#]
using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Connection;
using Genesyslab.Platform.Commons.Protocols;

using Genesyslab.Platform.WebMedia.Protocols;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat.Events;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat.Requests;
```

Setting Up Web Media Protocol Objects

When interacting with existing chat sessions, you will need to store session-specific details including a secure key and user ID. Additional objects that will be needed include a `FlexChatProtocol` object (for sending and receive messages) and a `FlexTranscript` object (used to store and interact with the chat transcript).

```
[C#]
private string secureKey;
private string userId;
private FlexTranscript flexTranscript;
private FlexChatProtocol flexChatProtocol;
```

To use the Web Media Platform SDK, you first need to instantiate a `Protocol` object by supplying information about the Web Media Server you want to connect with. This example specifies values for the name, host, and port values:

```
[C#]
flexChatProtocol = new FlexChatProtocol(new Endpoint("Flex_Chat_Server", "<hostname>",
<port>));
```

Note that you have to provide a string when you create the `FlexChatProtocol` object. This string should be unique for each protocol used in your application. It might be a good idea to use the name of the server's application object from the configuration layer, which guarantees uniqueness as well as clearly identifying which server you are communicating with.

After instantiating the `FlexChatProtocol` object, you need to open the connection to the Web Media Server:

```
[C#]
flexChatProtocol.Open();
```

You should always use proper error handling techniques in your code, especially when working with the protocol connection. To save space, these error handling steps are not shown in this example.

Logging in to Chat Server

```
[C#]
// filter the request based on our configured application name
KeyValueCollection kvUserData = new KeyValueCollection();
```

```
kvUserData.Add("FirstName", "John");
kvUserData.Add("LastName", "Smith");
kvUserData.Add("EmailAddress", "john.smith@email.com");
RequestLogin reqLogin = RequestLogin.Create("reqLogin", 0, kvUserData);
EventStatus msg = this.flexChatProtocol.Request(reqLogin) as EventStatus;
```

After successfully logging in to Chat Server, a message is returned that includes some important information: the Secure Key and User ID values. You will use these values when sending messages to the Chat Server, so remember to keep track of them for later.

```
[C#]
if (msg != null && msg.Id == EventStatus.MessageId)
{
    if (msg.RequestResult == RequestResult.Success)
    {
        secureKey = msg.SecureKey;
        userId = msg.UserId;
    }
}
```

Updating a Chat Session

By creating a `RequestRefresh` object, you can either check for updates or send new text to an existing chat session. The following sample shows how to create a `RequestRefresh` object, send it to the Chat Server, and process the result.

```
[C#]
RequestRefresh reqRefresh = RequestRefresh.Create(
        userId, secureKey, flexTranscript.LastPosition + 1, MessageText.Create(""));
EventStatus msg = this.flexChatProtocol.Request(reqJoin) as EventStatus;
if (msg != null && msg.Id == EventStatus.MessageId)
{
        if (msg.RequestResult == RequestResult.Success)
        {
                ProcessTranscript(msg.FlexTranscript);
        }
}
```

Working with Restricted Characters

Due to server-side requirements, the XML-based Web Media Platform SDK protocols (`BasicChat`, `FlexChat`, `Callback` and Email) do not support illegal characters in string values. See http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char for the allowable character range.

The Platform SDK protocols do not change user data by default, but if you want to replace illegal characters then you can include code in your application to configure the protocol connection. For example:

```
[C#]
// Note: to use the PropertyConfiguration class, ensure that your using
// statements include Genesyslab.Platform.Commons.Connection
PropertyConfiguration conf = new PropertyConfiguration();
conf.SetBoolean(WebmediaChannel.OptionNameReplaceIllegalUnicodeChars, true);
```

```
// "replacement" value is optional: if it is not specified - illegal characters will be
removed
conf.SetOption(WebmediaChannel.OptionNameIllegalUnicodeCharsReplacement, "?");
BasicChatProtocol protocol = new BasicChatProtocol(new Endpoint("chatServer", HOST, PORT,
conf));
protocol.Open();
```

Using specific connection configuration values in this manner will only affect the specified protocol instance(s).

If no replacement character or string is specified, then illegal characters will be removed (that is, replaced with an empty string).

Logging out of a Chat Session

When a client is ready to log out from the existing chat session, build a `RequestLogout` object and send it to the Chat Server.

```
[C#]
RequestLogout reqLogout = RequestLogout.Create(userId, secureKey,
flexTranscript.LastPosition);
IMessage msg = flexChatProtocol.Request(reqLogout);
if (msg != null && msg.Id == EventStatus.MessageId)
{
    if ((msg as EventStatus).RequestResult == RequestResult.Success)
    {
                  ProcessTranscript((msg as EventStatus).FlexTranscript);
    }
}
```

Disconnecting from a Chat Server

Finally, when you are finished communicating with the Chat Server, you should close the connection to minimize resource utilization.

```
[C#]
flexChatProtocol.Close();
```