Platform SDK Developer's Guide

Bidirectional Messaging

4/3/2025

Bidirectional Messaging

The primary function of Platform SDK is to provide client protocols for communication with Genesys servers, where your applications would send requests for information and receive related events.

However, the introduction of bidirectional messaging provides an opportunity for client applications to send their own "events" and receive "requests" from the server. Reversing the direction of messages in this way allows your application to implement server side logic - which allows you to implement new servers or emulating existing servers.

Java

Existing Server Side Support

Releases of Platform SDK prior to 8.5.3 functionality allowed some server-side functionality to be implemented, but server-side code had to implement all custom preprocessing of incoming and outgoing messages. This could be inconvenient for server development, especially for complex protocols which act as containers for inner protocols.

To make this easier, the bidirectional messaging feature takes responsibility for preprocessing incoming and outgoing messages. For complex protocols, this increases transport protocol transparency and allows you to work with concrete protocols.

Bidirectional Messaging Support

Starting with Platform SDK 8.5.302, customized server channels are available for the following protocols:

Protocol	Client Handler Class Name
ConfServer	com.genesyslab.platform.configuration.protocol.ConfServerProtoco
Basic Chat	com.genesyslab.platform.webmedia.protocol.ChatProtocolListener
Flex Chat	com.genesyslab.platform.webmedia.protocol.ChatProtocolListener
Email	com.genesyslab.platform.webmedia.protocol.EmailProtocolListene
UCS	com.genesyslab.platform.contacts.protocol.UniversalContactServe
ESP Email	com.genesyslab.platform.webmedia.protocol.EspEmailProtocolListe
BasicChat + FlexChat	com.genesyslab.platform.webmedia.protocol.ChatProtocolListener
Callback	com.genesyslab.platform.webmedia.protocol.CallbackProtocolListe
ESP	com.genesyslab.platform.openmedia.protocol.ExternalServiceProto

Protocol	Client Handler Class Name
InteractionServer	com.genesyslab.platform.openmedia.protocol.InteractionServerPro

Important

Platform SDK provides class com.genesyslab.platform.webmedia.protocol.ChatProtocolListener that can recognize the dialect of BasiChat and FlexChat protocols.

Using Bidirectional Messaging

The following example shows how to create and use a custom server channel for UCS.

UniversalContactServerProtocolListener Example

```
ManagedConfiguration cfg = new ManagedConfiguration(new PropertyConfiguration());
cfg.setBoolean(UniversalContactServerProtocol.USE_UTF_FOR_RESPONSES, false); // do not change
string encoding
cfg.setBoolean(TKVCodec.UTF_STRING_KEY, false);
UniversalContactServerProtocolListener listener =
  new UniversalContactServerProtocolListener(new WildcardEndpoint(0, cfg)); // creates server
channel. Port is unknown before opened.
UniversalContactServerProtocol client = new UniversalContactServerProtocol(); // creates
client channel. Endpoint is unknown before server opens.
final AtomicReference messageReference = new AtomicReference();
try{
    listener.setClientRequestHandler(new ClientRequestHandler() {
        @Override
        public void processRequest(RequestContext context){
            try {
                Message msg = context.getRequestMessage();
                context.respond(msg); // return message to sender
                messageReference.set(msg); // save link to the received message
            }catch (Exception e){}
        }
    });
    listener.open();
    int port = listener.getLocalEndPoint().getPort();
    client.setEndpoint(new Endpoint("localhost",port, cfg));
    client.open();

    EventSearch request = EventSearch.create(); // create request
    DocumentList documentList = new DocumentList(); // fill request data
    DocumentData documentData = new DocumentData();
    KeyValueCollection data = new KeyValueCollection();
    data.addObject("E-mail","email@email.com");
    documentData.setDocumentIndex(0);
    documentData.setFields(data);
    documentList.add(documentData);
    request.setDocuments(documentList);
    client.send(request);
    Message received = client.receive(10000);
    /*
```

```
       Correct conditions:
         1. received!=null
         2. request.equals(received)
         3. received.equals(messageReference.get())
         4. received!=request
    */
}finally {
    client.close();
    if (listener.getState()== ChannelState.Opened)
        listener.close();
}
```

Handshake issues

Platform SDK offers no server-side handshake procedure. Even if most protocols have a simple "unconditional" registration, your application is responsible for validating the clients itself.

.NET

Existing Server Side Support

Releases of Platform SDK prior to 8.5.3 provide specific server channel classes for ESP and Custom Routing Protocol (`ExternalServiceProtocolListener` and `UrsCustomProtocolListener` respectively). There is also an unspecified base implementation of server channel with the `ServerChannel` class.

However, these classes have some restrictions, such as being unable to switch transport layers for incoming connections, that make them unusable with Web Media Protocols.

Bidirectional Messaging Support

Starting with Platform SDK release 8.5.201, an additional server channel class was available: `ServerChannel<T>` (where T extends the `ClientChannelHandler` abstract class).

Platform SDK also includes extensions of `ClientChannelHandler` for all supported protocols. These extensions have server side messaging logic, and are responsible for substitution of XML transport instead of binary transport for Web Media Protocols.

Platform SDK .NET 8.5.3 provides extensions for the following protocols:

Protocol	Client Handler Class Name
ConfServer	Genesyslab.Platform.Configuration.Protocols.ConfServerProtocol.Cl
Basic Chat	Genesyslab.Platform.WebMedia.Protocols.BasicChatProtocol.ClientH
Flex Chat	Genesyslab.Platform.WebMedia.Protocols.FlexChatProtocol.ClientH
Email	Genesyslab.Platform.WebMedia.Protocols.EmailProtocol.ClientHand

Protocol	Client Handler Class Name
Callback	Genesyslab.Platform.WebMedia.Protocols.CallbackProtocol.ClientHa
ESP	Genesyslab.Platform.OpenMedia.Protocols.ExternalServiceProtocol
InteractionServer	Genesyslab.Platform.OpenMedia.Protocols.InteractionServerProtoc
UCS	Genesyslab.Platform.Contacts.Protocols.UniversalContactServerPro
ESP Email	Genesyslab.Platform.WebMedia.Protocols.EspEmail.EspEmailProtoc
BasicChat + FlexChat	Genesyslab.Platform.WebMedia.Protocols.ChatServerClientHandler
Stat Server	Genesyslab.Platform.Reporting.Protocols.StatServerProtocol.Client
TServer	Genesyslab.Platform.Voice.Protocols.TServerProtocol.ClientHandler

Important

Platform SDK provides the `Genesyslab.Platform.WebMedia.Protocols.ChatServerClientHandler` class that can recognize the dialect of `BasicChat` and `FlexChat` protocols. Using this class together with `ServerChannel<T>` allows the following ChatServer logic.

Using Bidirectional Messaging

The following example shows how to create and use a custom server channel for Basic Chat Protocol.

ServerChannel<T> Example

```
var cfg = new ManagedConnectionConfiguration(null) { WrapUtfString = true, }; // allows to
use utf values in KvLists
server = new ServerChannel<BasicChatProtocol.ClientHandler>(new WildcardEndpoint(0,cfg)); //
creates server channel. Port is unknown before opened.
server.Received += (sender, args) =>
{
  var msgArg = args as MessageEventArgs;
  if (msgArg == null) return; // wrong arguments (in general it's impossible with
ServerChannel<>)
  var channel = sender as DuplexChannel;
  if (channel == null) return; // wrong client (in general it's impossible with
ServerChannel<>)
  var incomingMessage = msgArg.Message; // gets incoming message
  IMessage outgoingMessage= null;
  // TODO: handle incoming message...
  if (outgoingMessage!=null) channel.Send(outgoingMessage); // sends response to client
};
server.Open(); // opens server
client = new BasicChatProtocol(new Endpoint("localhost", (server.LocalEndPoint as
IPEndPoint).Port, cfg)); // create client
// client has to be created only after server will be opened for case of unknown port
client.AutoRegister = false; // skip handshake
client.Open(); // opens client

var msg = RequestMessage.Create("12345",Visibility.All, MessageText.Create(null));
client.Send(msg); // send message
```

```
var response = client.Receive(TimeSpan.FromSeconds(5)); // receive response
// TODO: handle response...
```

Handshake Issues

Platform SDK offers no server-side handshake procedure. Even if most protocols have a simple "unconditional" registration, your application is responsible for validating the clients itself.