PONTON X/P 4.6 Adapter 2.0 Programming Guide

Version: 1

Datum: 17-Okt-2024

Table of Contents

1 Introduction

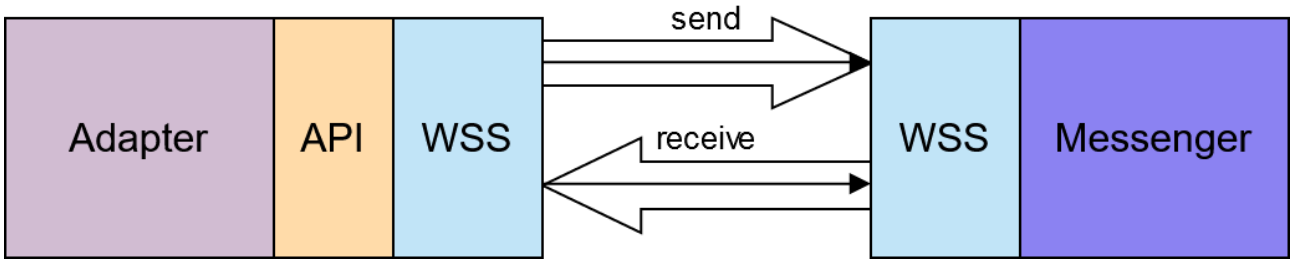
This document describes the basics how to write an adapter for the PONTON X/P Messenger.

If you plan to implement your own Adapter, you should be familiar with object-oriented programming, inheritance, interface implementation, and the Java programming language.

A Java library is provided that contains the classes that handle the communication with the PONTON X/P Messenger.

To use this library it is required to use at least Java version 11, no third party libraries are needed.

1.1 Message Flow



The communication between the PONTON X/P Messenger and an adapter is based on the HTTPS Websocket protocol.

An adapter establishes permanent Websocket connections to the Messenger. To send or receive messages the adapter must be authorized by the PONTON X/P Messenger.

2 SimpleAdapter

This example shows how to create a simple adapter to send and receive messages.

```
/*
 * Copyright 2020 by PONTON GmbH
 * Dorotheenstr. 64, 22301 Hamburg, Germany.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of
 * PONTON GmbH "Confidential Information". You shall not disclose
 * such Confidential Information and shall use it only in accordance with the
 * terms of the license agreement you entered into with PONTON.
 *
 */

import java.io.File;
import java.io.IOException;
import java.io.InputStream;

import de.ponton.xp.adapter.api.AdapterStatusRequestHandler;
import de.ponton.xp.adapter.api.ConnectionException;
import de.ponton.xp.adapter.api.MessageHandler;
import de.ponton.xp.adapter.api.MessengerConnection;
import de.ponton.xp.adapter.api.OutboundMessageStatusUpdateHandler;
import de.ponton.xp.adapter.api.TransmissionException;
import de.ponton.xp.adapter.api.domainvalues.AdapterInfo;
import de.ponton.xp.adapter.api.domainvalues.ConversationId;
import de.ponton.xp.adapter.api.domainvalues.InboundMetaData;
import de.ponton.xp.adapter.api.domainvalues.InboundStatusEnum;
import de.ponton.xp.adapter.api.domainvalues.MessageId;
import de.ponton.xp.adapter.api.domainvalues.MessageType;
import de.ponton.xp.adapter.api.domainvalues.MessengerInstance;
import de.ponton.xp.adapter.api.domainvalues.OutboundMetaData;
import de.ponton.xp.adapter.api.domainvalues.OutboundStatusEnum;
import de.ponton.xp.adapter.api.domainvalues.ReceiverId;
import de.ponton.xp.adapter.api.domainvalues.SenderId;
import de.ponton.xp.adapter.api.domainvalues.TransferId;
import de.ponton.xp.adapter.api.messages.InboundMessageStatusUpdate;
import de.ponton.xp.adapter.api.messages.OutboundMessage;
import de.ponton.xp.adapter.api.messages.OutboundMessageStatusUpdate;

public class SimpleAdapter {

    private final MessengerConnection messengerConnection;

    public SimpleAdapter() throws ConnectionException {
        final File adapterWorkFolder = new File("work");
        final AdapterInfo adapterInfo = AdapterInfo.newBuilder() //
                .setAdapterId("SimpleAdapterID") //
                .setAdapterVersion("1.0.0") //
                .build();

        messengerConnection = connectToMessenger(adapterInfo, adapterWorkFolder);
        startReceivingMessages();
    }

    private MessengerConnection connectToMessenger(AdapterInfo adapterInfo, File
adapterWorkFolder) throws ConnectionException {
        final int messengerPort = 2600;
        return MessengerConnection.newBuilder() //
                .setWorkFolder(adapterWorkFolder) //
                .setAdapterInfo(adapterInfo) //
                .addMessengerInstance(MessengerInstance.create("messenger.host",
messengerPort)) //
                .onMessageReceive(getMessageHandler()) //
```

```
                .onMessageStatusUpdate(getOutboundMessageStatusUpdateHandler()) //
                .onAdapterStatusRequest(getAdapterStatusRequestHandler()) //
                .build();
    }

    private void startReceivingMessages() throws TransmissionException {
        messengerConnection.startReception();
    }
```

Code Block 1 Create Messenger Connection

MessengerConnection ist the main class to send or receive messages. When an instance of the MessengerConnection is created, the adapter is connected to the messenger on host *messenger.host* and port *2600.*You can use this MessengerConnection instance to initialize your own beans to use it.

After initialization you have to call *startReception()* to inform the messenger, that the adapter is ready for using.

NOTE: MesengerConnection tries to reconnect to the messenger automatically, if the connection is lost.

```
    public void sendMessage(final InputStream inputStream, final SenderId senderId,
final ReceiverId receiverId) throws TransmissionException {
        final OutboundMessage outboundMessage = buildOutboundMessage(inputStream,
senderId, receiverId);
        final TransferId transferId =
messengerConnection.sendMessage(outboundMessage);
        // the transferId is needed to assign status updates to this send process
    }

    public OutboundMessageStatusUpdate sendMessageSynchronously(final InputStream
inputStream, final SenderId senderId, final ReceiverId receiverId) throws
TransmissionException {
        final OutboundMessage outboundMessage = buildOutboundMessage(inputStream,
senderId, receiverId);
        return messengerConnection.sendMessageSynchronously(outboundMessage);
    }

    private static OutboundMessage buildOutboundMessage(InputStream inputStream,
SenderId senderId, ReceiverId receiverId) {
        final OutboundMetaData outboundMetaData = OutboundMetaData.newBuilder()//
                .setSenderId(senderId)//
                .setReceiverId(receiverId)//
                .build();
        final OutboundMessage outboundMessage = OutboundMessage.newBuilder()//
                .setInputStream(inputStream)//
                .setOutboundMetaData(outboundMetaData)//
                .build();
        return outboundMessage;
    }
```

Code Block 2 Send Message

To send a message you have to build an *OutboundMessage,* which contains a message content as InputStream and optional meta data of the message (see *OutboundMetaData*), like *SenderId*, *ReceiverId*, *MessageId* and call one of the following send methods:

- *sendMessage(outboundMessage)* on *MessengerConnection* to send the message asynchronously. As result you get a *TransferId* to assign incoming status updates to this send process.

- *sendMessageSynchronously(outboundMessage)* on *MessengerConnection* to send the message synchronously. As result you get an *OutboundMessageStatusUpdate* with *TransferId* and the status of the process step in the Messenger (e.g. OutboundStatusEnum.PROCESSED_AND_QUEUED when message successfully processed).

```
    private OutboundMessageStatusUpdateHandler
getOutboundMessageStatusUpdateHandler() {
        return outboundMessageStatusUpdate -> {
            // the send process is finished.
            if (outboundMessageStatusUpdate.isFinal()) {
                // get transferId to reference the sent outbound message (see
sendMessage())
                final TransferId transferId =
outboundMessageStatusUpdate.getTransferId();

                // We can send the result of the sent message to backend.
                final OutboundStatusEnum outboundStatusEnum =
outboundMessageStatusUpdate.getResult();
                final String detailText =
outboundMessageStatusUpdate.getDetailText();
            }
        };
    }
```

Code Block 3 Handle Status Updates for Outbound Message

To receive status updates the *OutboundMessageStatusUpdateHandler* has to be implemented and set to the MessengerConnection. All status updates have to be assigned to the original send process by using the *TransferId*.

NOTE: the final flag informs you whether the original send process is completed. If the value of the final flag is *true* no other status updates will be sent to the adapter for the send process.

```
     private MessageHandler getMessageHandler() {
        return inboundMessage -> {
            // handle inbound message
            final InboundMetaData inboundMetaData =
inboundMessage.getInboundMetaData();
            final MessageId messageId = inboundMetaData.getMessageId();
            final ConversationId conversationId =
inboundMetaData.getConversationId();
            final SenderId senderId = inboundMetaData.getSenderId();
            final ReceiverId receiverId = inboundMetaData.getReceiverId();
            final MessageType messageType = inboundMetaData.getMessageType();

            try (final InputStream inputStream = inboundMessage.createInputStream())
{
                // process incoming message content
            } catch (final IOException ioe) {
                // handle IOException here
            }

            // create result for the inbound message
            final InboundMessageStatusUpdate inboundMessageStatusUpdate =
InboundMessageStatusUpdate.newBuilder() //
                    .setInboundMessage(inboundMessage) //
                    .setStatus(InboundStatusEnum.SUCCESS) //
                    .setStatusText("message successfully delivered to backend.") //
                    .build();
            return inboundMessageStatusUpdate;
        };
     }
```

Code Block 4 Receive Message

To receive messages *MessageHandler* has to be implemented and set to MessengerConnection. Each *InboundMessage* contains the message content, which can be read by calling *createInputStream()*, and meta data (see *InboundMetaData*), like *SenderId*, *ReceiverId*, *MessageType* and other.

If an incoming message was processed you have to return *InboundMessageStatusUpdate* with the status:

- SUCCESS - if the incoming message was successfully processed.
- REJECTED - if an error occurs while processing the message or forwarding it to a backend system.
- TEMPORARY_ERROR - if a temporary error occurs and the messenger has to send the message again.

```
    private AdapterStatusRequestHandler getAdapterStatusRequestHandler() {
        return () -> {
            final String adapterStatus = "Simple adapter is running and received 123
messages.";
            // the adapter status is shown by the messenger on the Adapter Monitor
            return adapterStatus;
        };
    }
 }
```

Code Block 5 Handle Adapter Status Request

To receive adapter status requests *AdapterStatusRequestHandler* has to be implemented and set to MessengerConnection. As adapter status you should to return a string with your own information, which is requested and shown by Messenger on the Adapter Monitor.

PONTON GmbH
Dorotheenstraße 64
22301 Hamburg
Germany

Web: http://www.ponton.de
LinkedIn https://www.linkedin.com/company/ponton-consulting/