**How to properly send a greeting message and common issues from customers**

*Posted on*[*July 12, 2018*](https://blog.botframework.com/2018/07/12/how-to-properly-send-a-greeting-message-and-common-issues-from-customers/)*by*[*The Bot Framework Team*](https://blog.botframework.com/author/botframeworkteam/)

**[Updated on 9/23/2019]**

*Please note: Recent updates to Bot Framework Direct Line have created a new developer experience when working with Direct Line or Bot Framework Web Chat. In short, when working with either of these products, when a token request is made that includes a user ID, a single conversation update event will occur that will include two user IDs, one for the user and one for the bot. You can read more about this new experience in this related blog, “*[*Sending Welcome Messages via Direct Line / Web Chat*](https://www.bing.com)*”.*

*Additionally, due to design changes between the v3 and v4 of the Bot Framework, this blog now only applies to the v3 SDK and v3 Web Chat (a.k.a. BotChat).*

A very common chat/conversational feature that bot developers need to implement, is how to create an initial welcome or greeting message when a user initially begins interacting with the bot (say, the first time they open web chat). One issue some customers have been having lately, is that when using trying the conversationUpdate, is that their bot behaves as expected during local testing when using the Emulator, however when the bot is running live, it does not work and user data cannot be submitted to the back channel. Typically what they’re trying to do, is submit some sort of locally stored information, say the user’s name, and then have the bot give a greeting to the user with the user’s name.

For example, in a web app with a bot, specific user information might be passed to the bot in the embedded client web code:

<script src="https://cdn.botframework.com/botframework-webchat/latest/botchat.js"></script>

<script>

BotChat.App({

directLine: { secret: \*\*\*\*\*\* },

user: { id: 'user123', name : 'johndoe' },

bot: { id: 'na' },

resize: 'detect'

}, document.getElementById("bot"));

document.oncontextmenu = function(){return false;};

</script>

Then, in the bot’s code, attempting to use the conversationUpdate property:

bot.on('conversationUpdate',(session,activity,message) => {

console.log('session.message.user.name = ' + session.message.user.name);

if(session.membersAdded){

session.membersAdded.forEach(function (identity) {

if(identity.id === session.address.bot.id){

bot.beginDialog(session.address,'Welcome');

//session.send(session.user.name)

//session.send('this is welcome message')

}

});

}

})

Using the above will result in an error in your bot code in a deployed bot.

You should not use conversationUpdate. Why?

ConversationUpdate is something sent by the Direct Line channel connector. Conversation Update is sent by the direct line connector once when the bot joins the conversation, and once when the user joins the conversation.

Knowing this, when a user first opens up an instance of WebChat, two conversationUpdate events are emitted:

One for when bot joins the conversation.

One for when a (human) user joins the conversation.

If you’re using WebChat or directline, the bot’s ConversationUpdate is sent when the conversation is created and the user sides’ ConversationUpdate is sent when they first send a message. When ConversationUpdate is initially sent, there isn’t enough information in the message to construct the dialog stack. The reason that this appears to work in the emulator, is that the emulator simulates a sort of pseudo DirectLine, but both conversationUpdates are resolved at the same time in the emulator, and this is not the case for how the actual service performs.

What this means, is that prompts do not work correctly as a response to ConversationUpdate. So, you should NOT be trying to send conversationUpdate from the client.

Workable Solution

So, now that we’ve clarified a little bit about how conversationUpdate works with respect to DirectLine, what can we do to solve this? In a lot of scenarios, we’d still like our bots to create some sort of custom prompt message to a user. Here’s a current solution that with various customers that we’re currently recommending:

Instead of conversationUpdate:

Post an event activity in client side code.

In your bot code, respond to the event with the desired welcome message.

Below is an example of embedded HTML for WebChat (client side code) where we are posting the event activity.

<!DOCTYPE html>

<html>

<head>

<link href="https://cdn.botframework.com/botframework-webchat/latest/botchat.css" rel="stylesheet" />

</head>

<body>

<div>

<div id="bot" />

</div>

<script src="https://cdn.botframework.com/botframework-webchat/latest/botchat.js"></script>

<script>

var user = {

id: 'user-id',

name: 'user name'

};

var botConnection = new BotChat.DirectLine({

token: '[DirectLineSecretHere]',

user: user

});

BotChat.App({

user: user,

botConnection: botConnection,

bot: { id: 'bot-id', name: 'bot name' },

resize: 'detect'

}, document.getElementById("bot"));

botConnection

.postActivity({

from: user,

name: 'requestWelcomeDialog',

type: 'event',

value: ''

})

.subscribe(function (id) {

console.log('"trigger requestWelcomeDialog" sent');

});

</script>

</body>

</html>

Keep note of the .postActivity call, with a property of type ‘event‘.

Below are two examples for how to handle the event from the bot side code, in for each Node.js and .NET Bot Builder SDK.

Bot Code (Node.js)

const connector = new builder.ChatConnector({

appId: 'MicrosoftAppIdHere',

appPassword: 'MicrosoftAppPasswordHere'

});

const server = restify.createServer();

server.listen(process.env.port || process.env.PORT || 3979, () => {

console.log(`${server.name} listening to ${server.url}`);

});

server.post('/api/messages', connector.listen());

var bot = new builder.UniversalBot(connector);

bot.on('event', function(message) {

if(message.name == 'requestWelcomeDialog'){

bot.beginDialog(message.address, '/');

}

});

This time, instead of using bot.on with conversationUpdate, we use bot.on ‘event‘ to trigger our chain of dialogs.

bot.dialog('/', [function(session, args, next) {

try {

builder.Prompts.choice(session, 'Hey, I'm a bot. Ask me about:', 'Item 1|Item 2|Other',

{listStyle: builder.ListStyle["button"]});

} catch (err) {}

},

function(session, results) {

switch (results.response.entity) {

case "Item 1":

session.replaceDialog("/Item1Dialog");

break;

case "Item 2":

session.replaceDialog("/Item2Dialog");

break;

case "Other":

session.replaceDialog("/Other");

break;

default:

session.replaceDialog("/");

break;

}

}

]);

bot.dialog('/Item1Dialog', [function(session, args, next) {

try {

session.send('in item 1 dialog');

} catch (err) {}

}]);

Bot Code (.NET)

The code snippet below is an example of how in MessagesController

private async Task HandleSystemMessage(Activity message)

{

await TelemetryLogger.TrackActivity(message);

if (message.Type == ActivityTypes.DeleteUserData)

{

// Implement user deletion here

// If we handle user deletion, return a real message

}

else if (message.Type == ActivityTypes.ConversationUpdate)

{

}

else if (message.Type == ActivityTypes.ContactRelationUpdate)

{

}

else if (message.Type == ActivityTypes.Typing)

{

// Handle knowing that the user is typing

}

else if (message.Type == ActivityTypes.Ping)

{

}

else if (message.Type == ActivityTypes.Event)

{

var eventActivity = message.AsEventActivity();

if (eventActivity.Name == "setUserIdEvent")

{

ConnectorClient connector = new ConnectorClient(new Uri(message.ServiceUrl));

var userId = message.From.Id;

var welcomeMessages = GreetingHelper.FormatWelcomeMessages(userProfile);

foreach (var welcomeMessage in welcomeMessages)

{

var reply = message.CreateReply(welcomeMessage);

await connector.Conversations.ReplyToActivityAsync(reply);

}

}

}

}

You’ll notice that the bulk of the logic is handled in the very last conditional statement, ‘else if (message.Type == ActivityTypes.Event)… ‘ .

**Summary**

We hope that this post has helped you to understand a little more about how DirectLine works. In this post we’ve provided a current recommended working solution which is how we’ve helped many customers to create custom greeting prompts for their bots. The key thing is to remember not to use conversationUpdate from your bot code, but rather submit an event activity from the client, and add some code to handle the event to your bot code. We’ll keep you posted in the future if things change.

Happy Making!

The Azure Bot Service Team.