# [Chatbots as middlemen](http://tomipaananen.azurewebsites.net/?p=1851)

Chatbots typically serve their customers on 1:1 basis. They are not unlike digital assistants (Cortana, Siri, Alexa etc.) except that a chatbot is usually designed to execute a small number of prefined tasks well and focus on a narrow subject like filling a pizza order for example.

[Building chatbots is easy](https://docs.botframework.com/en-us/csharp/builder/sdkreference/), but making them clever is more difficult. Despite all the analytics on user behavior, it is still impossible to anticipate every user reaction. As the technology, Conversation as a Platform (CaaP), evolves, creating more intelligent bots becomes easier and easier, but until Skynet grows self-aware humans still serve a purpose.

Imagine a customer service chat on a website. A bot can probably handle most of the problems a customer could have. For instance, implementing a simple FAQ bot is trivial using [Microsoft’s QnA Maker](https://www.microsoft.com/cognitive-services/en-us/qnamaker). Add some additional intellect including [natural language understanding service](https://www.luis.ai/) and [whatnot](https://www.microsoft.com/cognitive-services/en-us/apis) and you have an efficient customer service bot in your hands that 9 out of 10 customers are perfectly happy with. But for that one customer, you might want to consider a fallback: Let the human – in this case a customer service agent – take over to ensure customer satisfaction.

As long as you have the human labour, implementing this isn’t rocket science. What you need to do is as follows:

1. Make sure your bot keeps track of all the individuals the bot sees (but remember privacy policies!)
2. Make sure your bot also keeps track of itself. This might sound weird at first, but I’ll make the reason apparent soon.
3. Design the handover scenario. It could be based on [sentiment analysis](https://www.microsoft.com/cognitive-services/en-us/text-analytics-api) or simply a request of help by the user.
4. Implement the message relaying logic (don’t worry – there are samples available!)

## How and why to keep track of people and bots

By keeping track I mean collecting the contact information of a user (and the bot – I’ll explain later) from the bot’s perspective. You can’t send a post card to a person without knowing his or her address. The same applies to the bot framework: You cannot send a message to a user without knowing the IDs of the user and the conversation. What you’ll need at least are:

* [Service URL](https://docs.botframework.com/en-us/csharp/builder/sdkreference/dc/d2f/class_microsoft_1_1_bot_1_1_connector_1_1_activity.html#ad151704b4dbe85c03ba3e52fa61004ad),
* [Channel account ID](https://docs.botframework.com/en-us/csharp/builder/sdkreference/dd/def/class_microsoft_1_1_bot_1_1_connector_1_1_channel_account.html#a0b89cf01fdd73cbc00a524dce9e2ad1a) (think of this as user ID) and
* [Conversation account ID](https://docs.botframework.com/en-us/csharp/builder/sdkreference/d8/d48/class_microsoft_1_1_bot_1_1_connector_1_1_conversation_account.html#af580076af58e4d42bbe352ccb753f892) (think of this as conversation ID)

The aforementioned details may be enough, but this depends on the channel (Skype, MS Teams, Slack etc.) You might as well store all the details as shown in the following tables.

| Table 1. Identities in **Skype** (all values are of type string). | | |
| --- | --- | --- |
|  | **Me** | **Bot** |
| ServiceUrl | https://skype.botframework.com | https://skype.botframework.com |
| ChannelId | skype | skype |
| ChannelAccount.Id | 29:1byUvXHHhinNxwnPCHh4MPhpfiJUbadX\_Y3\_sTkBspdiSke8sX\_Ps6riTYRVez5jT | 28:f99fa2c3-8834-418e-b293-039205238055 |
| ChannelAccount.Name | Tomi Paananen | Intermediator Bot Sample |
| ConversationAccount.Id | 29:1byUvXHHhinNxwnPCHh4MPhpfiJUbadX\_Y3\_sTkBspdiSke8sX\_Ps6riTYRVez5jT | 29:1byUvXHHhinNxwnPCHh4MPhpfiJUbadX\_Y3\_sTkBspdiSke8sX\_Ps6riTYRVez5jT |
| ConversationAccount.Name | *(N/A in direct conversation)* | *(N/A in direct conversation)* |

The values above are from a direct conversation in Skype between my bot and I. As you can see the channel account ID (read: my user ID) and the conversation account ID match, but that isn’t necessarily the case in other channels.

| Table 2. Identities in **Slack**. | | |
| --- | --- | --- |
|  | **Me** | **Bot** |
| ServiceUrl | https://slack.botframework.com | https://slack.botframework.com |
| ChannelId | slack | slack |
| ChannelAccount.Id | U1F3JK9A9:T1F248PJ8 | B2NSU1D4Z:T1F248PJ8 |
| ChannelAccount.Name | tomi | intermediatorbot |
| ConversationAccount.Id | B2NSU1D4Z:T1F248PJ7:C3B1ZK5D0 | B2NSU1D4Z:T1F248PJ7:C3B1ZK5D0 |
| ConversationAccount.Name | bottest | bottest |

So why do we need the bot’s identity stored too? As you can see, the same bot has a different identity on different channel and conversation. When we send a message to a user, we need to specify who the message is from, and some channel, for example Slack, doesn’t allow you to send messages from bots that aren’t actually there. So in order to relay a message from a user to another on another channel (e.g. Skype to Slack) we need to know and use the bot’s identity in Slack in the **from** field.

Briefly about the technical implementation: All the activities flow through [the MessagesController class](https://github.com/tompaana/intermediator-bot-sample/blob/master/IntermediatorBotSample/Controllers/MessagesController.cs) in a bot built with C# and that’s the ideal place to keep track of everything. As for bots, the bot is always the receiving party when it gets a new activity, and that’s how you store the bot identities. See [Sending and Receiving Activities](https://docs.botframework.com/en-us/csharp/builder/sdkreference/routing.html) for more information.

Finally, store the records of the users and the bot somewhere in web e.g. [Azure Table storage service](https://docs.microsoft.com/en-us/azure/storage/storage-dotnet-how-to-use-tables). ***Note:***[*My sample*](https://github.com/tompaana/intermediator-bot-sample) *stores the data locally (in memory), which is never, ever a good idea, because bots are essentially web apps and can have multiple instances!*

### Comparison to Node.js

The essentials for relaying messages are the same whether you are building your bot using C# or Node.js SDKs. However, there are differences between the SDKs and some things are handled differently.

| Table 3. Node.js counterparts for establishing user/bot identity. | | |
| --- | --- | --- |
| **C#** | **Node.js** | **Node.js example** |
| [Activity](https://docs.botframework.com/en-us/csharp/builder/sdkreference/dc/d2f/class_microsoft_1_1_bot_1_1_connector_1_1_activity.html).ServiceUrl | [IChatConnectorAddress](https://docs.botframework.com/en-us/node/builder/chat-reference/interfaces/_botbuilder_d_.ichatconnectoraddress.html).serviceUrl | session.message.address.serviceUrl |
| Activity.ChannelId | [IAddress](https://docs.botframework.com/en-us/node/builder/chat-reference/interfaces/_botbuilder_d_.iaddress.html).channelId | session.message.address.channelId |
| [ChannelAccount](https://docs.botframework.com/en-us/csharp/builder/sdkreference/dd/def/class_microsoft_1_1_bot_1_1_connector_1_1_channel_account.html).Id | [IIdentity](https://docs.botframework.com/en-us/node/builder/chat-reference/interfaces/_botbuilder_d_.iidentity.html).id | session.message.address.bot.id session.message.address.user.id |
| ChannelAccount.Name | IIdentity.name | session.message.address.bot.name session.message.address.user.name |
| [ConversationAccount](https://docs.botframework.com/en-us/csharp/builder/sdkreference/d8/d48/class_microsoft_1_1_bot_1_1_connector_1_1_conversation_account.html).Id | IIdentity.id | session.message.address.conversation.id |
| ConversationAccount.Name | IIdentity.name | session.message.address.conversation.name |

## Samples

* [Intermediator Bot Sample](https://github.com/tompaana/intermediator-bot-sample) built with **C#**
* [Bot Hand Off sample](https://github.com/palindromed/Bot-HandOff) built with **Node.js** by Hannah Krager