Orange Networks GmbH
Sachsentor 26
21029 Hamburg

orange networks
IT-CONSULTING - IT-DEVELOPMENT

Developing Applications - Bots

Module X - Use Dialogs to simplify Bot development
Student Lab Manual

Orange Networks GmbH

Sachsentor 26
21029 Hamburg

Lyonel-Feininger-Strasse 28
81677 München

info@orange-networks.de
http://www.orange-networks.de
Telefon: +49 (0) 40 739 237 20
Telefax: +49 (0) 40 739 237 02

Geschäftsführer:
Oliver Mohr
Andreas Riedl
Matthias Seitle

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

Lab X: Use Dialogs to simplify Bot development

Introduction

Conversations happen within a context, and without it is easily to misunderstand and take one thing for another. And it's the same for bots. The phrase "list them" will have a different intent if you are communicating about available products, available services, contents of your basket.
As humans we expect that if we were talking about products and we state "list them" we want to list the products, requiring the user to always be explicit, i.e.:
-List the products,
-List the services,
-List the contents of my basket,

Is not natural!

The bot framework solves this by providing Dialogs, which allow us to model conversations and create context, i.e. each dialog fulfills one purpose and as such the phase "list" can have a specific meaning in each.
Technically speaking dialogs maintain conversation flow by being pushed on and popped off the stack during a conversation. The stack works like a normal LIFO stack; meaning, the last dialog added will be the first one to complete. Once a dialog completes then control is returned to the previous dialog on the stack.

Objectives

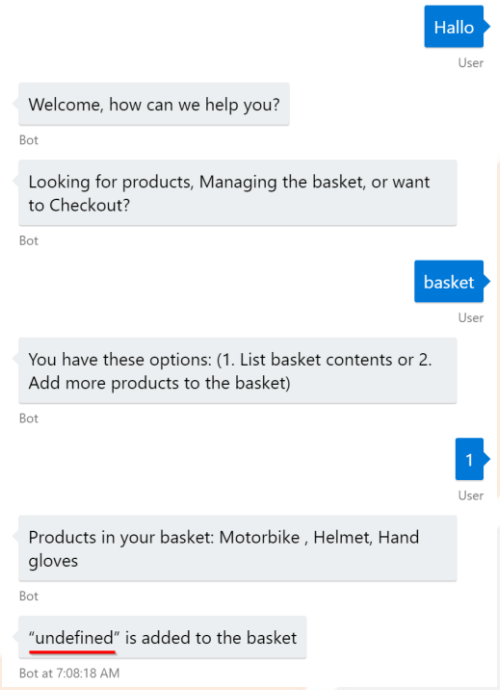
After completing this lab, you will be able to:
- work flawlessly with Dialogs,
- simplify code
- improve Chat UX
- Learn waterfall flow and dialogs.

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

Exercise 1: Fix the bug

Introduction

This exercise will show why working without Dialogs is very difficult and how quickly the codebase degrades.



Objectives

In the first exercise the objective is to:

- Fix bug so that we don't get "undefined" after listing the contents of a basket.
- Experience the un-maintainability of bot development without dialogs.

Task 1: Debug the app

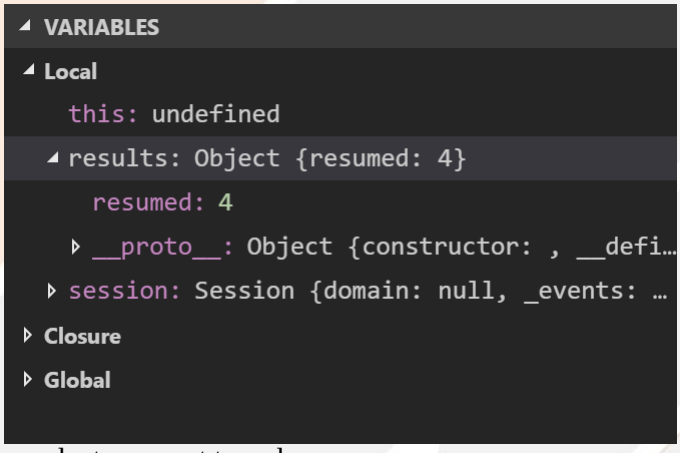
1. First of all open the folder "Begin_LabX" in Visual Studio Code.

2. Execute `` `npm install` `` to install dependencies.

3. In the file `app.js` add two breakpoints. In lines **49** and **63**.

4. To hit the breakpoint, add one or more products and then list the basket contents. (conversation flow is seen in image)

5. Now hitting the endpoint use the controls at the top of the VS Code IDE. With the button **Step Over** go line by line. At the **next()** call we are forwarded to the next section.



6. At the variables section we can see that results property doesn't have a response and hence it is undefined. Why are we expecting to have this property? We check the above method and we see that the same method could potentially receive the name of the product we want to order.

7. Great! Now we need another if statement to ensure if we are adding a product or listing products from basket.

So update the following code

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

```
session.userData.basket = session.userData.basket || []
session.userData.basket.push(results.response)
session.send(`"${results.response}" is added to the basket`)
session.endConversation()
```

To:

```
if (session.dialogData.actionChoice.includes("products")) {
    session.userData.basket = session.userData.basket || []
    session.userData.basket.push(results.response)
    session.send(`"${results.response}" is added to the basket`)
}
else if (session.dialogData.actionChoice.includes("basket")) {
    // ToDo
}
session.endConversation()
```

Bug Fixed! But this is the point where we start thinking. Do we want to keep on like this? Increasing the depth of if's in every subsequent method?

In programming this is known as the **Pyramid of doom** and should be avoided!

And it's another programming concept against us. **Rule of thumb**: if a condition is repeated in multiple methods it is telling us that it is an object on its own, that wants to break out.

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

Exercise 2: Migrating to Dialogs

Introduction

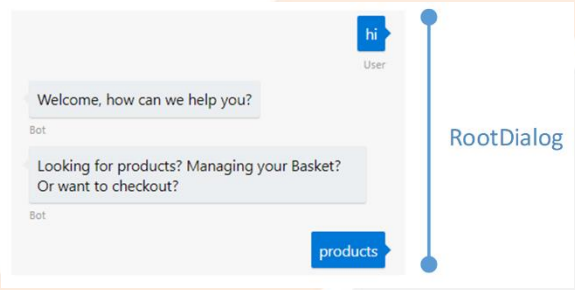
In this exercise we will migrate our solution to Dialogs.

Objectives

In the first exercise the objective is to:

- Reduce the if indentation pyramid.
- Refactor the code for simplification and easier to add functionality.
- Understand what dialogs are and how they work.

Task 1: Create the Products dialog



1. According to the picture this sequence happens in the first UniversalDialog also called as RootDialog. So let's leave that part unchanged. So we get to this code without changes:

```
if(results.response.toLocaleLowerCase().includes("products")) {
    session.dialogData.actionChoice = "products";
    builder.Prompts.choice(session, "You have these options:",
        ["Order products", "Remove products from basket"])
}
```

But from this point we are getting in the realm of Products. And this is what we need to do in a separate dialog.

```
if (results.response.toLocaleLowerCase().includes("products")) {
    session.beginDialog('/products');
}
```

2. Create the "Products" dialog.

```
bot.dialog('/products', [
    (session) => {
        builder.Prompts.choice(session, "You have these options:",
            ["Order products", "Remove products from basket"])
    },
])
```

3. Extend the products dialog by extracting the parts which are product related from the RootDialog:

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

```
bot.dialog('/products', [
    function (session) {
        builder.Prompts.choice(session, "You have these options:",
            ["Order products", "Remove products from basket"])
    },

    (session) => {
        builder.Prompts.text(session, "Name of the product you want to order:")
    },

    (session, results) => {
        session.userData.basket = session.userData.basket || []
        session.userData.basket.push(results.response)
        session.send(`"${results.response}" is added to the basket`)
        session.endDialog() //ToDo
    },
])
```

4. Extract the basket method (left to the reader)
5. Stub Checkout:

```
bot.dialog('/checkout', [
    (session) =>  {
        //ToDo
    },
])
```

In the end the RootDialog will contain only the logic to get us started into more specific dialogs.

```
var bot = new builder.UniversalBot(connector, [
    (session, args, next) => {
        session.send("Welcome, how can we help you?");
        next();
    },

    (session) => {
        builder.Prompts.text(session, `Looking for products, Managing the basket,
or want to Checkout?`);
    },

    (session, results) => {
        if (results.response.toLocaleLowerCase().includes("products")) {
```

```
            session.beginDialog('/products');
        }
        else if (results.response.toLocaleLowerCase().includes("basket")) {
            session.beginDialog('/basket');
        }
        else if (results.response.toLocaleLowerCase().includes("checkout")) {
            //ToDo
        }
    }
]).set('storage', inMemoryStorage);
```

In the end the RootDialog will contain only the logic to get us started into more specific dialogs.

Exercise 3: Using capabilities of Dialogs

Introduction

In this exercise we will add the possibility to remove products from basket. And that is possible in two conversation flows, to do it from products dialog and to do it from basket dialog.

Objectives

In the first exercise the objective is to:

- Add another dialog that deals with removing products.
- Learn and detect opportunities to use Dialogs.

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

Task 1: Create the RemoveProducts dialog

1. Add the code for RemoveProducts dialog

```
bot.dialog('/removeProducts', [
    (session) =>  {
        builder.Prompts.choice(session, "Select the product you want to remove:",
session.userData.basket, {
            listStyle: builder.ListStyle.button
        })
    },

    (session, results, next) =>  {
        session.userData.basket = session.userData.basket.filter(v => v !=
results.response.entity)
        session.send(`"${results.response.entity}" removed from your basket.`)
        next();
    },

    (session) => {
        session.endDialog()
    },
])
```

2. To have options more easier for user to select update all Prompts to contain ListStyle.button, e.g.:

```
builder.Prompts.choice(session, "You have these options:",
            ["List basket contents", "Remove products from basket"],
            {
                listStyle: builder.ListStyle.button
            })
```

3. Update basket dialog that upon receiving request on Remove Products from basket to replaceDialog:

```
if (results.response.entity === "List basket contents") {
            session.send('Products in your basket: ' +
session.userData.basket.join(", "))
            next();
        }
        else if (results.response.entity === "Remove products from basket") {
            session.replaceDialog('/removeProducts')
```

Orange Networks GmbH
Sachsentor 26
21029 Hamburg

```
        }
```

4. Do the same for Products dialog.